Robust deep-reinforcement learning policies for mixed-autonomy traffic

Kathy Jang

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-252
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-252.html

December 8, 2021

Robust deep-reinforcement learning policies for mixed-autonomy traffic

M.S. Report Plan II

Author
Kathy Jang

Robust deep-reinforcement learning policies for mixed-autonomy traffic

by Kathy Jang

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Alexandre Bayen
Research Advisor

12/8/21

(Date)

* * * * * * *

Professor Jiantao Jiao
Second Reader

12/07/2021

(Date)

Acknowledgments

Many thanks to Eugene Vinitsky and Professor Bayen for their continuous mentorship, and all the various members of Bayen Lab that have made my time at Berkeley so memorable. I am eternally grateful for my family and their everlasting support, and dedicate to this to my dad, who always believed in me.

Contents

Chapter 1

Introduction

Transportation is a major source of US energy consumption and greenhouse gas emissions, accounting for 28% and 26% respectively. According to the bureau of transportation statistics, total road miles traveled is continuously increasing, growing at 2 to 3% per year between 2010 and 2014 while over the same period the total road length of the US transportation network remained unchanged. The increased road usage is coupled with an increase in congestion. Overall congestion delay in 2014 was 6.9 billion hours, an increase of 33% since 2000; the problem is even worse in metropolitan areas where travelers needed to allocate an additional 150% more travel time during peak periods to arrive on time. The congestion also has significant economic cost, totaling 160 billion dollars in 2014 [1]. Depending on their usage, automated vehicles have the potential to alleviate system level metrics such as *congestion*, *accident rates*, and *greenhouse gas emissions* through a combination of intelligent routing, smoother driving behavior, and faster reaction time [2].

Partially automated systems are predicted to increasingly populate roadways between 2020 and 2025 but will primarily be usable in high driving or high speed operations in light traffic. Hazard detection technology is not expected to be mature enough for full automation in the presence of general vehicles and pedestrians (i.e. heterogeneous fleets, manned/unmanned, bicycles, pedestrians, mixed use road-space etc.) until at least 2030. It takes 20 years for a vehicle fleet to turn over sufficiently which makes it likely that vehicles will be partially manned at least until 2050 [3].

Recently, the steady increase in usage of cruise control systems on the roadway offers an opportunity to study the optimization of traffic in the framework of *mixed-autonomy traffic*: traffic that is partially automated but mostly still consists of human driven vehicles. However, the control problems posed in this framework

are notoriously difficult to solve. Traffic problems, which often exhibit features such as time-delay, non-linear dynamics, and hybrid behavior, are challenging for classical control approaches, as microscopic traffic models are high complexity: discrete events (lane changes, traffic light switches), continuous states (position, speed, acceleration), and non-linear driving models. These complexities make analytical solutions often intractable. The variety and non-linearity of traffic often leads to difficult trade-offs between the fidelity of the dynamics model and tractability of the approach.

Classical control approaches: Classical control approaches have successfully solved situations in which the complexity of the problem can be reduced without throwing away key aspects of the dynamics. For example, there is a variety of analytical work on control of autonomous intersections with simple geometries. For mixed-autonomy problems, there have been significant classical controls based results for simple scenarios like vehicles on a ring [4] or a single lane of traffic whose stability can be characterized [5, 6]. A thorough literature review on coordinating autonomous vehicles in intersections, merging roadways, and roundabouts can be found in [**?**]. The classical control approaches described in this review can be broken down into reservation methods, scheduling, optimization with safety constraints, and safety maximization. Other approaches discussed in the review involve applications of queuing theory, game theory, and mechanism design.

However, as the complexity of the problem statement increases, classical techniques become increasingly difficult to apply. Shifting focus from simple scenarios to, for example, hybrid systems with coexisting continuous and discrete controllers, explicit guarantees for hand-designed controllers can become harder to find. Ultimately, when the complexity of the problem becomes too high, optimization-based approaches have been shown to be a successful approach in a wide variety of domains from robotics [7] to control of transportation infrastructure [8].

Deep reinforcement learning: *Deep reinforcement learning* (deep RL) has emerged as an effective technique for control in high dimensional, complex CPS systems. Deep RL has shown promise for the control of complex, unstructured problems as varied as robotic skills learning [9], playing games such as Go [10], and traffic light ramp metering [11]. Of particular relevance to this work, deep RL has been successful in training a single autonomous vehicle to optimize traffic flow in the presence of human drivers [12].

One key distinction in RL is whether the algorithm is model-free or model-based, referring to whether the algorithm is able to query a dynamics model in the computation of the control or the policy update. Model-free RL tends to outperform model-based RL if given sufficient optimization time, but requires longer training

times. Thus, model-free techniques are most effective when samples can be cheaply and rapidly generated. This often means that model-free RL works best in simulated settings where a simulation step can be made faster than real-time and simulation can be distributed across multiple CPUs or GPUs. A long-standing goal is to be able to train a controller in simulation, where model-free techniques can be used, and then use the trained controller to control the actual system.

Policy Transfer: Transfer of a controller from a training domain to a new domain is referred to as *policy transfer*. The case where the policy is directly transferred without any fine-tuning is referred to as *zero-shot policy transfer*. Zero-shot policy transfer is a difficult problem in RL, as the true dynamics of the system may be quite different from the simulated dynamics, an issue referred to as *model mismatch*. Techniques used to overcome this include adversarial training, in which the policy is trained in the presence of an adversary that can modify the dynamics and controller outputs and the policy must subsequently become robust to perturbations [13]. Other techniques to overcome *model mismatch* include re-learning a portion of the controller[14], adding noise to the dynamics model [15], and learning a model of the true dynamics that can be used to correctly execute the desired trajectory of the simulation-trained controller [16]. Efforts to overcome the reality gap have been explored in vision-based reinforcement learning [17] and in single AV systems [18].

Other challenges with policy transfer include *domain mismatch*, where the true environment contains states that are unobserved or different from simulation. For example, an autonomous vehicle might see a car color that is unobserved in its simulations and subsequently react incorrectly. Essentially, the controller overfits to its observed states and does not generalize. Domain mismatch can also occur as a result of imperfect sensing or discrepancies between the simulation and deployment environment. While in simulation it is possible to obtain perfect observations, this is not always the case in the real world. Small differences between domains can lead to drastic differences in output. For example, a slight geometric difference between simulation and real world could result in a vehicle being registered as being on one road segment, when it is on another. This could affect the control scheme in a number of ways, such as a premature traffic light phase change. Techniques used to tackle this problem include domain randomization [19], in which noise is injected into the state space to enforce robustness with respect to unobserved states.

Contributions and organization of the work: In this report we discuss two different approaches to policy transfer as well as the RL-traffic framework used to create these results:

- Chapter 2: Flow

- Chapter 3: Gaussian Policy Transfer
- Chapter 4: Adversarial Policy Transfer

Chapter 2 discusses an RL / driving framework whose involvement and development I have significantly contributed to, published about [20], and have used to produce the work discussed in Chapters 3 and 4.

Chapter 3 is first-author work that is published with authors from the University of Delaware at the International Conference on Cyber-Physical Systems and is drawn from the associated paper [21].

Chapter 4 is work that is published with authors from the University of Delaware at the International Conference on Control and Automation and is drawn from the associated paper [22]. While not listed as the first author, I am responsible for the RL work and chiefly responsible for the RL-related writing.

Chapter 2

Flow

For both chapters 3 and 4, we run our experiments in *Flow* [12], a library that provides an interface between a traffic microsimulator, SUMO [23], and three RL libraries, rllab [24] and RLlib [25], and Stable Baselines, which are centralized and distributed RL libraries respectively. *Flow* enables users to create new traffic networks via a Python interface, introduce autonomous controllers into the networks, and then train the controllers in a distributed system on the cloud via AWS EC2. To make it easier to reproduce our experiments or try to improve on our benchmarks, the code for *Flow*, scripts for running our experiments, and tutorials can be found at https://github.com/flow-project/flow. The remainder of this chapter discusses the experimental setup Flow has that is used to produce the work in both both chapters 3 and 4, and is organized as follows:

- Definition of Reinforcement Learning
- Reinforcement Learning in Traffic Control
- Policy Gradient Methods
- Car Following Models

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a form of machine learning which studies how an intelligent *agent* can perform optimal actions within the *environment* that it exists in. RL is formally described via a *Markov Decision Process* (MDP) [26]. The standard discounted, finite-horizon MDP can be defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma, T)$. The variables in the MDP are:

- $\mathcal{S}$ is a set of states (finite or infinite),
- $\mathcal{A}$ is a set of actions (finite or infinite),
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the transition probability distribution of transitioning from one state s to another state s' given action a,
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function,
- $\rho_0 : \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the probability distribution over start states,
- $\gamma \in (0, 1]$ is the discount factor, and
- T is the horizon.

The problem formulation in this project uses a fully observable MDP.

Every step in the RL algorithm involves an agent interacting with its environment. The agent receives sensory data or information about its environment in the form of a *state space*. It then uses this observation as an input to a function (a neural net in the case of deep RL), receiving a set of actions, which it then performs in the environment. After taking this action, the agent receives a reward from the environment, which informs the agent on the quality of its previous state-action (s, a) pair.

The goal of RL is to develop a series of actions such that the agent can achieve the highest possible cumulative, discounted reward: $R = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$ where r_t is the reward at time t. Optimizing over R produces an RL policy π, which maps state space information s to actions a. In deep RL, π is parameterized with the weights of a neural net. At one end the neural net contains an input layer which takes in $s \in \mathcal{S}$ as input, and at the other end it contains an output layer which yields $a \in \mathcal{A}$. Additional hidden layers, including affine transformations and non-linear activation functions, exist between the input and output. Due to this structure, neural nets, and thus deep RL, are capable handling large amounts of complexity.

2.2 Reinforcement Learning in Traffic Control

The data-rich combination of deep learning with *reinforcement learning* (RL) has overtaken years of classical control research in transportation. Combined, *deep reinforcement learning* (deep RL) has emerged as an effective form of traffic control. Wu demonstrated in [12] how an autonomous vehicle equipped with an RL-learned policy could effectively dissipate the same shockwaves Stern mitigated in 2017. The effectiveness of RL in traffic extends to a number of other traffic scenarios, such as figure-eight roads, bottlenecks, and merge networks [12, 27, 11, 20].

As control methods become increasingly complex, so does the importance of manual hyperparameter tuning. In order to reduce reliance on expert opinion and human subjectivity, methods have emerged that use intelligent control to determine hyperparameters for training. Work by Hutter et al. [28] demonstrates a Bayesian method for automated optimization of hyperparameter choices. Similar work is done via RL, using deep Q-network and evolutionary algorithms [29, 30].

2.3 Policy Gradient Methods

Policy gradient methods use Monte Carlo estimation to compute an estimate of the gradient of the expected discounted reward $\nabla_\theta R = \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$ where θ are the parameters of the policy π_θ. We perform repeated *rollouts*, in which the policy is used to generate the actions at each time step. At the end of the rollout, we have accumulated a state, action, reward trajectory $\tau = (s_0, a_0, r_0, \ldots, s_T)$. Policy gradient methods take in a set of these trajectories and use them to compute an estimate of the gradient $\nabla_\theta R$ which can be used in any gradient ascent-type method.

The particular policy gradient method used in this work is *Trust Region Policy Optimization* (TRPO) [31]. TRPO is a monotonic policy improvement algorithm, whose update step provides guarantees of an increase in the expected total reward. However, the exact expression for the policy update leads to excessively small steps so implementations of TRPO take larger steps by using a trust region. In this case, the trust region is a bound on the KL divergence between the old policy and the policy update. While not a true distance measure, a small KL divergence between the two policies suggests that the policies do not act too differently over the observed set of states, preventing the policy update step from sharply shifting the policy behavior.

2.4 Car Following Models

For our model of the driving dynamics, we used the *Intelligent Driver Model* [32] (IDM) that is built into the traffic microsimulator SUMO [23]. IDM is a microscopic car-following model commonly used to model realistic driver behavior. Using this model, the acceleration for vehicle α is determined by its bumper-to-bumper *headway* s_α (distance to preceding vehicle), the vehicle's own velocity v_α, and relative velocity Δv_α, via the following equation:

$$a_{\text{IDM}} = \frac{dv_\alpha}{dt} = a\left[1 - \left(\frac{v_\alpha}{v_0}\right)^\delta - \left(\frac{s^*(v_\alpha, \Delta v_\alpha)}{s_\alpha}\right)^2\right] \tag{2.1}$$

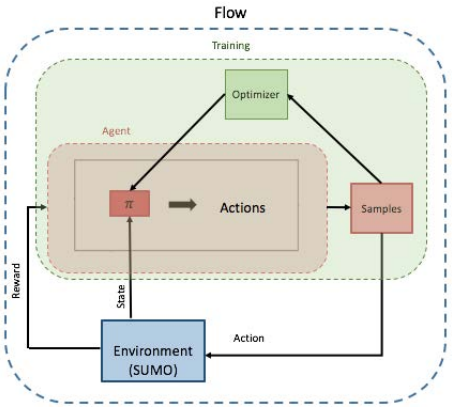


Figure 2.1: Diagram of the iterative process in *Flow*. Portions in red correspond to the controller and rollout process, green to the training process, and blue to traffic simulation.

where s^* is the desired headway of the vehicle, denoted by:

$$s^*(v_\alpha, \Delta v_\alpha) = s_0 + \max\left(0, v_\alpha T + \frac{v_\alpha \Delta v_\alpha}{2\sqrt{ab}}\right) \tag{2.2}$$

where $s_0, v_0, T, \delta, a, b$ are given parameters. Typical values for these parameters can be found in [32]; the values used in our simulations are given in Sec. 3.2.1. To better model the natural variability in driving behavior, we induce stochasticity in the desired driving speed v_0. For a given vehicle, the value of v_0 is sampled from a Gaussian whose mean is the speed limit of the lane and whose standard deviation is 20% of the speed limit.

Car following models are not inherently collision-free, we supplement them with a safe following rule: a vehicle is not allowed to take on velocity values that might lead to a crash if its lead vehicle starts braking at maximum deceleration. However, due to some uncertainty in merging behavior, there are still rare crashes that can occur in the system.

Fig. 2.1 describes the process of training the policy in *Flow*. The controller, here represented by policy π_θ, receives a state and reward from the environment and uses the state to compute an action. The action is taken in by the traffic microsimulator, which outputs the next state and a reward. The (state, next state, action, reward) tuple are stored as a sample to be used in the optimization step. After accumulating enough samples, the states, actions, and rewards are passed to the optimizer to compute a new policy.

Chapter 3

Gaussian Policy Transfer

In this work we use deep RL to train two autonomous vehicles to learn a classic form of control: ramp metering, in which traffic flow is regulated such that one flow of vehicles is slowed such that another flow can travel faster. While in the real world, ramp metering is controlled via metering lights, we demonstrate the same behavior using AVs instead of lights. Each RL vehicle interacts with sensors at each of the entrance ramps and is additionally able to acquire state information about vehicles on the roundabout, as well as state information about the other RL vehicle. By incorporating this additional sensor information, we attempt to learn a policy that can time the merges of the RL vehicles and their platoons to learn ramp metering behavior, which prevents energy-inefficient decelerations and accelerations. Being positioned at the front of a platoon of vehicles, each RL vehicle has the ability to control the behavior of the platoon of human-driven vehicles following it. The RL vehicle, also referred to in this work as an autonomous vehicle (AV), is trained with the goal of minimizing the average delay of all the vehicles in simulation.

Next, we show how we overcome the RL to real world reality gap and demonstrate RL's real world relevance by transferring the controllers to the University of Delaware's Scaled Smart City (UDSSC), a reduced-scale city whose dynamics, which include sensor delays, friction, and actuation, are likely closer to true vehicle dynamics. RL trained policies, which are learned in a simulation environment, can overfit to the dynamics and observed states of the simulator and can then fare poorly when transferred to the real world. The combination of model and domain mismatch contributes to this problem. We combine the ideas of domain randomization with adversarial perturbations to the dynamics and train a controller in the presence of noise in both its observations and actions. For reasons discussed in Sec. 3.3, we expect the addition of noise in both state and action to help account for both model and domain mismatch.

In this chapter we present the following results:

- The use of deep RL in simulation to learn an emergent metering policy.
- A demonstration that direct policy transfer to UDSSC leads to poor performance.
- A successful zero-shot policy transfer of the simulated policy to the UDSSC vehicles via injection of noise into both the state and action space.
- An analysis of the improvements that the autonomous vehicles bring to the congested roundabout.

3.1 University of Delaware's Scaled Smart City (UDSSC)

The *University of Delaware's Scaled Smart City (UDSSC)* was used to validate the performance of the RL control system. UDSSC is a testbed (1:25 scale) that can help prove concepts beyond the simulation level and can replicate real-world traffic scenarios in a small and controlled environment. UDSSC uses a VICON camera system to track the position of each vehicle with sub-millimeter accuracy, which is used both for control and data collection. The controller for each vehicle is offloaded to a mainframe computer and runs on an independent thread which is continuously fed data from the VICON system. Each controller uses the global VICON data to generate a speed reference for the vehicles allowing for precise independent closed-loop feedback control. A detailed description of UDSSC can be found in [**?**]. To validate the effectiveness of the proposed RL approach in a physical environment, the southeast roundabout of the UDSSC was used (Fig. 3.1).

3.2 Experimental Setup

3.2.1 Simulation Details

To derive the RL policy, we developed a model of the roundabout highlighted in red in Fig. 3.1 in SUMO. The training of the model, shown in Fig. 3.2, included a single-lane roundabout with entry points at the northern and western ends. Throughout this work we will refer to vehicles entering from the western end as the *western platoon* and the north entrance as the *northern platoon*. The entry points of the model are angled slightly different as can be seen in Figs. 3.1 and 3.2.

The human-controlled vehicles operate using SUMO's built-in IDM controller, with several modified parameters. In these experiments, the vehicles operating with the

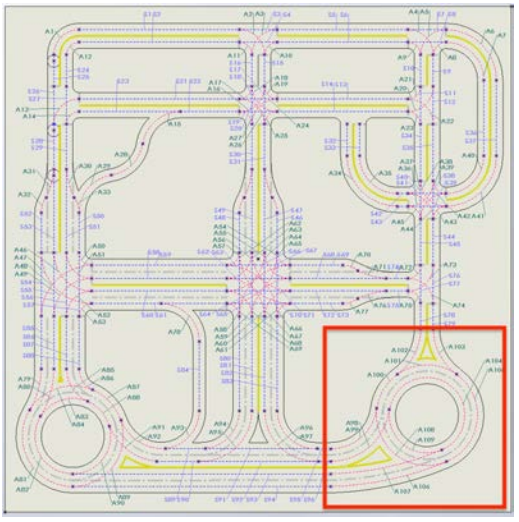


Figure 3.1: Diagram of the UDSSC road map, with the experimental zone highlighted in red.

IDM controller are run with $T = 1$, $a = 1$, $b = 1.5$, $\delta = 4$, $s_0 = 2$, $v_0 = 30$, and noise $= 0.1$, where T is a safe time headway, a is a comfortable acceleration in m/s^2, b is a comfortable deceleration, δ is an acceleration exponent, $s0$ is the linear jam distance, v_0 is a desired driving velocity, and noise is the standard deviation of a zero-mean normal perturbation to the acceleration or deceleration. Details of the physical interpretation of these parameters can be found in [32]. Environment parameters in simulation were set to match the physical constraints of UDSSC. These include: a maximum acceleration of $1\frac{\mathrm{m}}{\mathrm{s}^2}$, a maximum deceleration of $-1\frac{\mathrm{m}}{\mathrm{s}^2}$, and a maximum velocity of $15\frac{\mathrm{m}}{\mathrm{s}}$. The timestep of the system is set to 1.0 seconds.

Simulations on this scenario in the roundabout were executed across a range of different settings in terms of volume and stochasticity of inflows. In the RL policy implemented in UDSSC and discussed in 3.2.2, vehicles are introduced to the system via deterministic inflows from the northern and western ends of the roundabout using two routes: (1) the northern platoon enters the system from the northern inflow, merges into the roundabout, and exits through the western outflow and (2) the western platoon enters the system from the western inflow, U-turns through the roundabout, and exits through the western outflow. The western platoon consists of four vehicles total: three vehicles controlled with the IDM controller led by a vehicle running with the RL policy. The northern platoon consists of of three vehicles total: two vehicles controlled with the IDM controller led by a vehicle running with the RL policy. New platoons enter the system every 1.2 minutes, the rate of which is significantly sped up in simulation. These inflow settings are designed to showcase

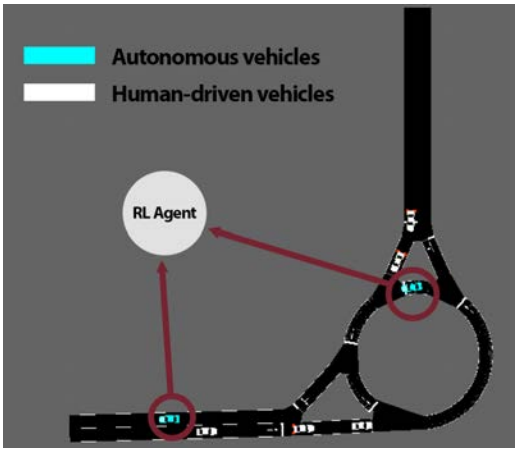


Figure 3.2: SUMO-generated network in UDSSC's roundabout. The blue vehicles are the AVs; they are both controlled by the RL policy. Videos of this policy in simulation are available at https://sites.google.com/view/iccps-policy-transfer.

the scenario where routes clash (Fig. 3.2).

3.2.2 UDSSC Setup

Each vehicle in UDSSC uses a saturated IDM controller to (1) avoid negative speeds, (2) ensure that the rear-end collision constraints do not become active, and (3) maintain the behavior of Eq. (2.1). Both the IDM and RL controllers provide a desired acceleration for the vehicles, which is numerically integrated to calculate each vehicle's reference speed.

Merging at the northern entrance of the roundabout is achieved by an appropriate yielding function. Using this function, the car entering the roundabout proceeds only if no other vehicle is on the roundabout at a distance from which a potential lateral collision may occur. Otherwise, the vehicle stops at the entry of the roundabout waiting to find a safe space to proceed.

To match the SUMO training environment, only one vehicle per path was allowed to use the RL policy. The paths taken by each vehicle are shown in Fig. 3.3. For each path, the first vehicle to enter the experimental zone was controlled by the RL policy; every subsequent vehicle runs with the saturated IDM controller. Once an active vehicle running with the RL policy exits the experimental zone, it reverts back to the IDM controller.

In the experiments, the vehicles operated in a predefined deterministic order, as

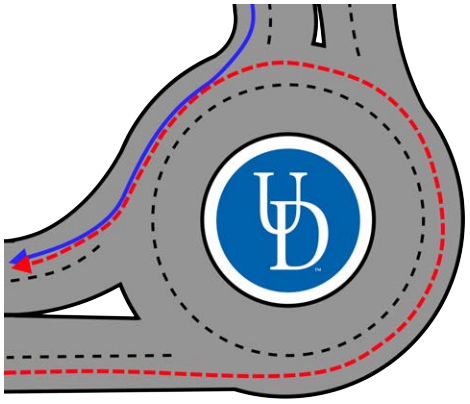


Figure 3.3: Visualization of the path taken on the UDSSC roundabout. The red route enters going east and exits going west; the blue route enters north and exits via the same west entrance as the red route.

described in 3.2.1. Four vehicles were placed just outside the experimental zone on the western loop, and three vehicles were placed in the same fashion near the northern entrance. Each vehicle platoon was led by a vehicle running with the RL policy, except in the baseline case where all vehicles used the saturated IDM controller. The experiment was executed with three variations: (1) the baseline case with all vehicles running with the IDM controller, (2) the case with a leader vehicle running with an RL policy trained in SUMO, and (3) the case where the leader vehicles running with an RL policy were trained in simulation with noise injected into their observations and accelerations.

3.2.3 Reinforcement Learning Structure

Action space

We parametrize the controller as a neural net mapping the observations to a mean and diagonal covariance matrix of a Gaussian. The actions are sampled from the Gaussian; this is a standard controller parametrization [33]. The actions are a two-dimensional vector of accelerations in which the first element corresponds to vehicles on the north route and the second element to the west route. Because the dimension of the action vector is fixed, there can only ever be 1 AV from the northern entry and 1 AV from the western entry. Two queues, one for either entryway, maintain a list of the RL-capable vehicles that are currently in the system. It should be noted that the inflow rates are chosen such that the trained policy never contains more than a queue of length 2. Platoons are given ample time to enter and exit the system before the next platoon arrives. This queue mechanism is designed to support the earlier

stages of training, when RL vehicles are learning how to drive, which can result in multiple sets of platoons and thus more than 2 RL vehicles being in the system at the same time. Control is given to vehicles at the front of both queues. When a vehicle completes its route and exits the experimental zone, its ID is popped from the queue. All other RL-capable vehicles are passed IDM actions until they reach the front of the queue. If there are fewer than two AVs in the system, the extra actions are simply unused.

The dynamics model of the autonomous vehicles are given by the IDM described in sec. 2.4 subject to a minimum and maximum speed i.e.

$$v_j^{\text{IDM}}(t+\Delta t) = \max\left(\min\left(v_{AV}(t) + a_{IDM}\Delta t, v_j^{\max}(t)\right), 0\right) \tag{3.1}$$

where $v_j^{AV}(t)$ is the velocity of autonomous vehicle j at time t, a_{IDM} is the acceleration given by an IDM controller, Δt is the time-step, and $v_j^{\max}(t)$ is the maximum speed set by the city j. For the AVs, the acceleration a_t is straightforwardly added to the velocity via a first-order Euler integration step

$$v_j^{\text{AV}}(t+\Delta t) = \max\left(\min\left(v_j^{\text{AV}}(t) + a_t\Delta t, v_j^{\max}(t)\right), 0\right) \tag{3.2}$$

Observation space

For the purposes of keeping in mind physical sensing constraints, the state space of the MDP is partially observable. It is normalized to ± 1 and includes the following:

- The positions of the AVs.
- The velocities of the AVs.
- The distances from the roundabout of the 6 closest vehicles to the roundabout for both roundabout entryways.
- The velocities of the 6 closest vehicles to the roundabout for both roundabout entryways.
- Tailway and headway (i.e. distances to the leading and following vehicles) of vehicles from both AVs.
- Length of the number of vehicles waiting to enter the roundabout for both roundabout entryways.
- The distances and velocities of all vehicles in the roundabout.

This state space was designed with real-world implementation in mind, and could conceivably be implemented on existing roadways equipped with loop detectors,

sensing tubes, and vehicle-to-vehicle communication between the AVs. For a sufficiently small roundabout, it is possible that an AV equipped with enough cameras could identify the relevant positions and velocities of roundabout vehicles. Similarly, the queue lengths can be accomplished with loop detectors, and the local information of the AVs (its own position and velocity, as well as the position and velocity of its leader and follower) are already necessarily implemented in distance-keeping cruise control systems.

Action and State Noise

The action and state spaces are where we introduce noise with the purpose of training a more generalizable policy that is more resistant to the difficulties of cross-domain transfer. We train the policies in two scenarios, a scenario where both the action and state space are perturbed with noise and a scenario with no noise. This former setting corresponds to a type of *domain randomization*. In the noisy case, we draw unique perturbations for each element of the action and state space from a Gaussian distribution with zero mean and a standard deviation of 0.1. In the action space, which is composed of just accelerations, this corresponds to a standard deviation of $0.1\frac{\text{m}}{\text{s}^2}$. In the state space, which is normalized to 1, this corresponds to a standard deviation of 1.5 m/s for velocity-based measures. The real-life deviations of each distance-based state space element are described here: AV positions, and the tailways and headways of the AVs, deviate the most at 44.3 m, the large uncertainty of which results in a policy that plays it safe. The distance from the northern and western entryways respectively deviate by 7.43m and 8.66 m. The length of the number of vehicles waiting to enter the roundabout from the northern and western entryway respectively deviate by 1.6 and 1.9 vehicles.

These perturbations are added to each element of the action and state space. The elements of the action space are clipped to the maximum acceleration and deceleration of ± 1, while the elements of the state space are clipped to ± 1 to maintain normalized boundaries. Noisy action and state spaces introduce uncertainty to the training process. The trained policy must still be effective even in the presence of uncertainty in its state as well as uncertainty that its requested actions will be faithfully implemented.

Reward function

For our reward function we use a combination of the L2-norm of the velocity of all vehicles in the system and penalties discouraging standstills or low velocity travel.

$$r_t = \frac{\max\left(v_{\max}\sqrt{n} - \sqrt{\sum_{i=1}^{n}(v_{i,t} - v_{\max})^2}, 0\right)}{v_{\max}\sqrt{n}} - 1.5 \cdot \text{pen}_\text{s} - \text{pen}_\text{p} \tag{3.3}$$

where n is the number of all vehicles in the system, $v_{\max}$ is the maximum velocity of $15\frac{\text{m}}{\text{s}}$, $v_{i,t}$ is the velocity that vehicle v_i is travelling at at time t. The first term incentives vehicles to travel near speed $v_{\max}$ but also encourages the system to prefer a mixture of low and high velocities versus a mixture of mostly equal velocities. The preference for low and high velocities is intended to induce a platooning behavior. RL algorithms are sensitive to the scale of the reward functions; to remove this effect the reward is normalized by $v_{\max}\sqrt{n}$ so that the maximum reward of a time-step is 1.

This reward function also introduces 2 penalty functions, pen_s and pen_p. pen_s returns the number of vehicles that are traveling at a velocity of 0, and pen_p is the number of vehicles that are traveling below a velocity of 0.3 m/s. They are defined as:

$$\text{pen}_\text{s} = \sum_{i=1}^{n} g(i) \quad \text{where} \quad g(x) = \begin{cases} 0, & v_x \neq 0, \\ 1, & v_x = 0. \end{cases} \tag{3.4}$$

$$\text{pen}_\text{p} = \sum_{i=1}^{n} h(i) \quad \text{where} \quad h(x) = \begin{cases} 0, & v_x \geq 0.3, \\ 1, & v_x \leq 0.3 \end{cases} \tag{3.5}$$

These penalty functions are added to discourage the autonomous vehicle from fully stopping or adopting near-zero speeds. In the absence of these rewards, the RL policy learns to game the simulator by blocking vehicles from entering the simulator on one of the routes, which allows for extremely high velocities on the other route. This occurs because velocities of vehicles that have not yet emerged from an inflow do not register, so no penalties are incurred when the AV blocks further vehicles from entering the inflow.

3.2.4 Algorithm/simulation details

We ran the RL experiments with a discount factor of .999, a trust-region size of .01, a batch size of 20000, a horizon of 500 seconds, and trained over 100 iterations. The controller is a neural network, a *Gaussian multi-layer perceptron* (MLP), with hidden sizes of $(100, 50, 25)$ and a tanh non-linearity. The choice of neural network non-linearities, size, and type were picked based on traffic controllers developed in [20]. The states are normalized so that they are between 0 and 1 by dividing each states by its maximum possible value. The actions are clipped to be between -1 and 1. Both normalization and clipping occur after the noise is added to the system so that the bounds are properly respected.

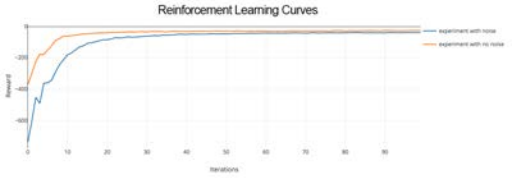


Figure 3.4: Convergence of the RL reward curve of an experiment with noised IDM, RL accelerations, and noisy state space

3.2.5 Code reproducibility

In line with open, reproducible science, the following codebases are needed to reproduce the results of our work. *Flow* can be found at `https://github.com/flow-project/flow`. The version of *rllab* used for the RL algorithms is available at `https://github.com/cathywu/rllab-multiagent` at commit number **4b5758f**. *SUMO* can be found at `https://github.com/eclipse/sumo` at commit number **1d4338ab80**.

3.2.6 Policy Transfer

The RL policy learned through Flow was encoded as the weights of a neural network. These weights were extracted from a serialized file and accessed via a Python function which maps inputs and outputs identical to those used in training. Separating these weights from rllab enables an interface for state space information from the UDSSC to be piped straight into the Python function, returning the accelerations to be used on the UDSSC vehicles. The Python function behaves as a control module within the UDSSC, replacing the IDM control module in vehicles operating under the RL policy.

The inputs to the RL neural network were captured by the VICON system and mainframe. The global 2D positions of each vehicle were captured at each time step. These positions were numerically derived to get each vehicle's speed and were compared to the physical bounds on the roadways to get the number of vehicles in each queue at the entry points. Finally, the 2D positions were mapped into the 1D absolute coordinate frame used during training. This array was passed into the RL control module as the inputs of the neural network.

3.2.7 Results

In this section we present our results from a) training vehicular control policies via RL in simulation, and b) transferring the successful policy to UDSSC. Extended

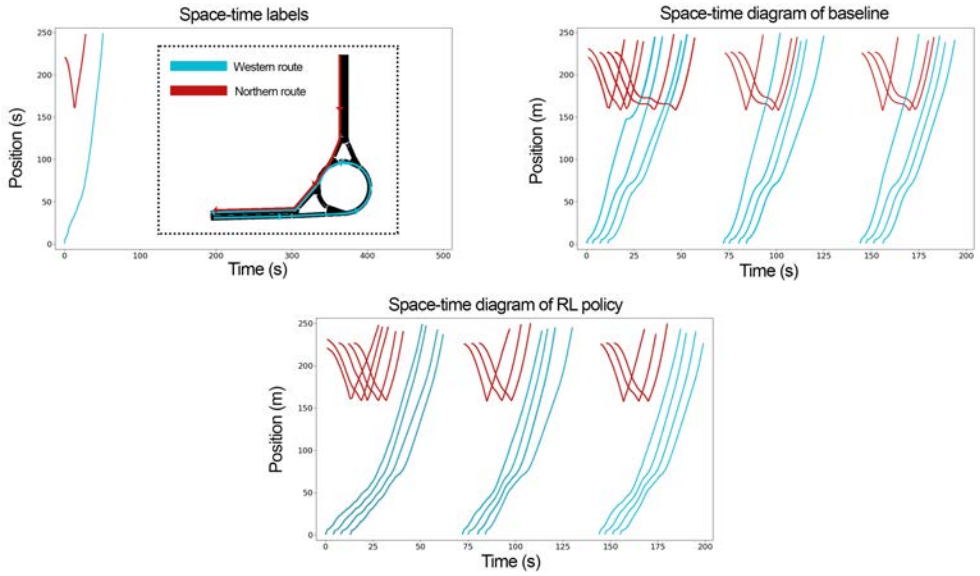


Figure 3.5: Space-time diagrams of the simulated baseline and RL policy. Each line corresponds to a vehicle in the system. Top: a guide to the color-scheme of the space time diagrams. The northern route is in red, the western route in blue. Middle: illustrates the overlap between the merging northern platoon and the western platoon. Bottom: The RL policy, depicted at the bottom successfully removes this overlap. Videos of this policy in simulation are available at https://sites.google.com/view/iccps-policy-transfer.

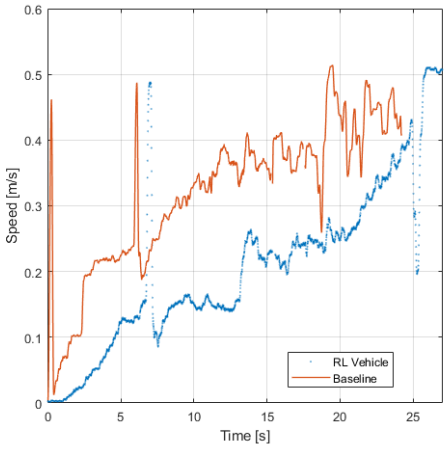


Figure 3.6: Comparison of the first vehicle on the southern loop for the baseline (IDM) and RL experiments. The RL vehicle starts off slower but eventually accelerates sharply once the northern platoon has passed.

Figure 3.7: Experiment with two platoons being led by RL vehicles (blue, circled).

work and videos of the policies in action are available at https://sites.google.com/view/iccps-policy-transfer.

Simulation results

Fig. 3.4 depicts the reward curve. The noise-injected RL policy takes longer to train than the noise-free policy and fares much worse during initial training, but converges to an almost identical final reward. In the simulations, videos of which are on the website, a ramp metering behavior emerges in which the incoming western vehicle learns to slow down to allow the vehicles on the north ramp to smoothly merge.

This ramp metering behavior can also be seen in the space-time diagrams in Fig. 3.5, which portrays the vehicle trajectories and velocities of each vehicle in the system. Western vehicles are depicted in blue and northern in red. Due to the overlapping routes, visible in Fig. 3.3, it was necessary to put a kink in the diagram for purposes of clarity; the kink is at the point where the northern and western routes meet. As can be seen in the middle figure, in the baseline case the two routes conflict as the northern vehicles aggressively merge onto the ramp and cut off the western platoon. Once the RL policy controls the autonomous vehicles, it slows down the western platoon so that no overlap occurs and the merge conflict is removed.

Transfer to UDSSC

The RL policies were tested under three cases in UDSSC: (1) the baseline case with only vehicles running with the IDM controller, (2) the case with a leader vehicle running with the RL policy trained in sumo without additional noise, and (3) the case where the leader vehicles running with the RL policy were trained with noise actively injected into their observations and accelerations. The outcomes of these trials are presented in Table 3.1.

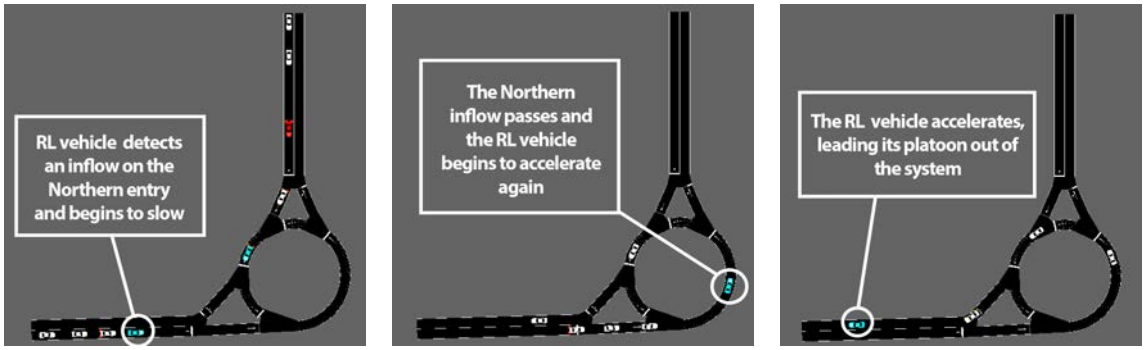


Figure 3.8: RL-controlled vehicle demonstrating smoothing behavior in this series of images. **First**: RL vehicle slows down in anticipation of a sufficiently short inflow from the north. **Second**: The northern inflow passes through the roundabout at high velocity. **Fourth**:The RL vehicle accelerates and leads its platoon away from the roundabout. Videos of this policy in simulation are available at https://sites.google.com/view/iccps-policy-transfer.

During the congestion experiment, the third case, in which noise was actively injected into the action and state space during training, successfully exhibits the expected behavior it demonstrated in simulation. In this RL controlled case, the transfer consistently showed successful ramp metering: the western platoon adopted a lower speed than the baseline IDM controller, as can be seen in the lower velocity of the RL vehicle in Fig. 3.6. This allowed the northern queue to merge before the western platoon arrived, increasing the overall throughput of the roundabout. This is closer to a socially optimal behavior, leading to a lower average travel time than the greedy behavior shown in the baseline scenario. No unexpected or dangerous driving behavior occurred.

This is in comparison with the second case, a policy trained on noiseless observations. In the second case, undesirable and unexpected deployment behavior suggests problems with the transfer process. Collisions occurred, sometimes leading to pile-ups, and platooning would frequently be timed incorrectly, such that, for example, only part of the Western platoon makes it through the roundabout before the Northern platoon cuts the Western platoon off. This indicates that the noise-injected policy is robust to transfer and resistant to domain and model mismatch.

Furthermore, the platoons led by RL vehicles trained with injected noise outperformed the baseline and noise-free cases. The results of these experiments, averaged over three trials with the RL vehicles, are presented in Table 3.1. This improvement was the outcome of a metering behavior learned by the western RL platoon leader. In the baseline case, the north and western platoons meet and lead to a merge conflict that slows the incoming western vehicles down. This sudden decrease in speed

	Avg. Vel.[m/s]	Avg. Time[s]	Max Time[s]
Baseline	0.26	15.71	23.99
RL	0.22	15.68	20.62
RL with Noise	0.23	14.81	18.68

Table 3.1: Results for the congestion experiment, the average and maximum times are averaged between three RL trials and a single baseline trial.

can be seen in the drop in velocity at 20 seconds of the baseline in Fig. 3.6.

Fig. 3.7 shows the experiment in progress, with the blue (circled) vehicles being RL vehicles trained under noisy conditions. The RL vehicle entering from the western (lower) entrance has performed it's metering behavior, allowing vehicles from the northern (upper) queue to pass into the roundabout before the western RL vehicle speeds up again. Videos of the emergent behavior can be found on the website.

3.3 Discussion

For the simulated environment, the choice of reward function, specifically, using the L2-norm rather than the L1-norm, encourages a more stable, less sparse solution. This makes evaluating the success of policy transfer more straightforward. Table 3.1 reports the results of the UDSSC experiments on three metrics:

- The average velocity of the vehicles in the system
- The average time spent in the system
- The maximum time that any vehicle spent in the system

Note, the system is defined as the entire area of the experiments, including the entrances to the roundabouts. The layer of uncertainty in the noise-injected policy aided with overcoming the domain and model mismatch between the simulation system and the UDSSC system. Thus, the noised policy was able to successfully transfer from simulation to UDSSC and also improved the average travel time by 5% and the maximum travel time by 22%. The noise-free policy did not improve on the average travel time and only improved the maximum travel time by 14%. Although we did not perform an ablation study to check whether both state space noise and action space noise were necessary, this does confirm that the randomization improved the policy transfer process.

The UDSSC consistently reproduced the moderate metering behavior for the noise-injected policy, but did not do so for the policy that was trained without noise. As can be seen in the videos, the noise-free policy was not consistent and would only

irregularly reproduce the desired behavior, or meter dramatically to the point that average travel time increased. Overall the noised policy significantly outperformed the noise-free version.

However, we caution that in our testing of the policy transfer process on the UDSSC, we performed a relatively limited test of the effectiveness of the policy. The vehicles were all lined up outside the system and then let loose; thus, the tests were mostly deterministic. Any randomness in the tests would be due solely to randomness in the dynamics of the UDSSC vehicles and stochasticity in the transferred policy. In training, the acceleration of IDM vehicles are noised and can account for some stochasticity in the initial distribution of vehicles on the UDSSC. However, the trained policy was not directly given this inflow distribution at train time, so this does correspond to a separation between train and test sets.

There may be several reasons why the noise and action injection may have allowed for a successful zero shot transfer. First, because the action is noisy, the learned policy will have to learn to account for *model mismatch* in its dynamics: it cannot assume that the model is exactly the double-integrator that is used in the simulator. Subsequently, when the policy is transferred to an environment with both delay, friction, and mass, the policy sees the mismatch as just another form of noise and accounts for it successfully. The addition of state noise helps with domain randomization; although the observed state distributions of the simulator and the scaled city may not initially overlap, the addition of noise expands the volume of observed state space in the simulator which may cause the two state spaces to overlap. Finally, the addition of noise forces the policy to learn to appropriately filter noise, which may help in the noisier scaled city environment.

3.4 Conclusions

In this work, we demonstrated the real-world relevance of deep RL AV controllers for traffic control by overcoming the gap between simulation and the real world. Using RL policies, AVs in the UDSSC testbed successfully coordinated at a roundabout to ensure a smooth merge. We trained two policies, one in the presence of state and action space noise, and one without, demonstrating that the addition of noise led to a successful transfer of the emergent metering behavior, while the noise-free policy often over-metered, or failed, to meter at all. This implies that for non-vision based robotic systems with small action-dimension, small amounts of noise in state and action space may be sufficient for effective zero-shot policy transfer. As a side benefit, we also demonstrate that the emergent behavior leads to a reduction of 5% in average travel time and 22% max-travel time on the transferred network.

Ongoing work includes characterizing this result more extensively, evaluating the effectiveness of the efficiency of the policy against a wide range of vehicle spacing, platoon sizes, and inflow rates. In this context, there are still several questions we hope to address, as for example:

- Are both state and action space noise needed for effective policy transfer?
- What scale and type of noise is most helpful in making the policy transfer?
- Would selective domain randomization yield a less lossy, more robust transfer?
- Would adversarial noise lead to a more robust policy?
- Can we theoretically characterize the types of noise that lead to zero-shot policy transfer?

Finally, we plan to generalize this result to more complex roundabouts including many lanes, many entrances, and the ability of vehicles to change lanes. We also plan to evaluate this method of noise-injected transfer on a variety of more complex scenarios, such as intersections using stochastic inflows of vehicles.

Chapter 4

Adversarial Policy Transfer

In this chapter and sequel to Chapter 3, we demonstrate a zero-shot transfer of an autonomous driving policy from simulation to University of Delaware's scaled smart city with adversarial multi-agent reinforcement learning, in which an adversary attempts to decrease the net reward by perturbing both the inputs and outputs of the autonomous vehicles during training. We train the autonomous vehicles to coordinate with each other while crossing a roundabout in the presence of an adversary in simulation. The adversarial policy successfully reproduces the simulated behavior and incidentally outperforms, in terms of travel time, both a human-driving baseline and adversary-free trained policies. Finally, we demonstrate that the addition of adversarial training considerably improves the performance of the policies after transfer to the real world compared to Gaussian noise injection introduced in Chapter 3.

The contributions of this work are: (1) the introduction of Gaussian single-agent noise and adversarial multi-agent noise to learn traffic control behavior for an automated vehicle; (2) a comparison performance with noise injected into the action space, state space, and both; (3) the demonstration of real-world disturbances leading to poor performance and crashes for some training methods, and (4) experimental demonstration of how autonomous vehicles can improve performance in a mixed-traffic system.

The remainder of this chapter is organized as follows.

- Section 4.1 introduces the mixed-traffic roundabout problem and the implementation of the RL framework
- Section 4.2 presents the simulation results
- Section 4.3 discusses the policy transfer process along with the experimental results.

- Finally, we draw concluding remarks in Section 4.4.

4.1 Problem Formulation

To demonstrate the viability of autonomous RL vehicles in reducing congestion in mixed traffic, we implemented the scenario shown in Fig. 4.1. In this scenario, two groups of vehicles enter the roundabout stochastically, one at the northern end and one at the western end. In what follows, we refer to the vehicles entering from the north entry as the *northern group*, and to the vehicles entering from the west entry as the *western group*.

The baseline scenario consists of homogeneous human-driven vehicles using the IDM controller (2.1). The baseline is designed such that vehicles approaching the roundabout from either direction will clash at the roundabout. This results in vehicles at the northern entrance yielding to roundabout traffic, resulting in significant travel delays. The RL scenario puts an autonomous vehicle at the head of each group, which can be used to control and smooth the interaction between vehicles; these mixed experiments correspond to a $15\% - 50\%$ mixture of autonomous and human-driven vehicles.

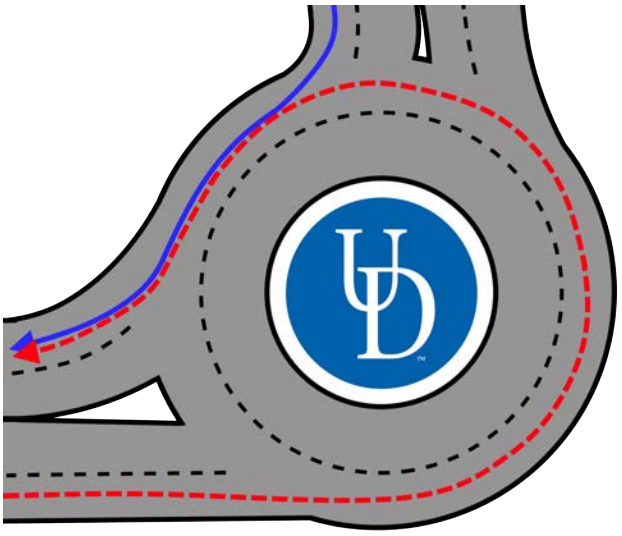


Figure 4.1: The routes taken by the northern (solid blue) and western (dashed red) groups through the roundabout.

4.1.1 Reinforcement Learning Structure

We categorize two sets of RL experiments that are used and compared in this work. We will refer to them as: *Gaussian single-agent*: A single-agent policy trained with Gaussian noise injected into the state and action space. *Adversarial multi-agent*: A multi-agent policy trained wherein a second agent provides selective adversarial noise to the learning agent. We discuss the particulars of these two methods in Sections 4.1.1 and 4.1.1. In this work, we deploy seven RL-trained policies, one of which is single-agent with no noise, three of which are Gaussian single-agent and the other three of which are adversarial multi-agent. All experiments follow the same setup. Inflows of stochastic length emerge at the northern and western ends of the roundabout. The size of the northern group will range from 2 to 5 cars, while the size of the western group ranges from 2 to 8. The length of these inflows will remain static across each rollout, and are randomly selected from a uniform distribution at the beginning of each new rollout.

Action Space

The actions are applied from a 2-dimensional acceleration vector, in which the first element is used to control the AV leading the northern group, and the second is used to control the AV leading the western group. If the AV has left the experiment, that element of the action vector is discarded.

State Space

The state space conveys the following information about the environment to the agent: the position, velocity, tailway, and headway of each AV and each vehicle in the roundabout, the distance from the roundabout entrances to the 6 closest vehicles, the velocities of the 6 closest vehicles to each roundabout entrance, the number of vehicles queued at each entrance, and the lengths of each inflow. All elements of the state space are normalized. The state space was designed with real-world implementation in mind and could contain any environmental factors that the simulation supports. As such, it is partially-observable to support modern sensing capabilities. All of these observations are reasonably selected and could be emulated in the physical world using sensing tools such as induction loops, camera sensing systems, and speedometers.

Reward Function

The reward function used for all experiments minimizes delay and applies penalties for standstill velocities, near-standstill velocities, jerky driving, and speeding, i.e.,

$$r_t = 2 \cdot \frac{\max\left(v_{\max}\sqrt{n} - \sqrt{\sum_{i=1}^{n}(v_{i,t} - v_{\max})^2}, 0\right)}{v_{\max}\sqrt{n}} - p, \tag{4.1}$$

$$p = p_s + p_p + p_j + p_v. \tag{4.2}$$

where n is the total number of vehicles, p is the sum of four different penalty functions, p_s is a penalty for vehicles traveling at zero velocity, designed to discourage standstill; p_p penalizes vehicles traveling below 0.2 m/s, which discourages the algorithm from learning an RL policy which substitutes extremely low velocities to circumvent the zero-velocity penalty; p_j discourages jerky driving by maintaining a dynamic queue containing the last 10 actions and penalizing the variance of these actions; and p_v penalizes speeding.

Gaussian single-agent noise

Injecting noise directly to the state and action space has been shown to aid with transfer from simulation to real-world testbeds [21, 19]. In this method, which applies to three of the policies we deployed, each element of the state space was perturbed by a random number selected from a Gaussian distribution. Only two elements describing the length of the inflows approaching the merge were left unperturbed. Elements of the state space corresponding to positioning on the merge edge were perturbed from a Gaussian distribution with a standard deviation of 0.05. For elements corresponding to absolute positioning, the standard deviation was 0.02. All other elements used a standard deviation of 0.1. These values were selected to set reasonable bounds for the degree of perturbation in the real world. Each element of the action space was perturbed by a random number selected from a zero mean Gaussian distribution with 0.5 standard deviation.

Adversarial multi-agent noise

For the other three policies, we use a form of adversarial training to yield a policy resistant to noise [13]. This is a form of multi-agent RL, in which two policies are learned. Adversarial training pits two agents against each other in a zero-sum game. The first is structurally the same as the agent which is trained in the previous four policies. The second, *adversarial* agent has a reward function that is the negative of the first agent's reward; in other words, it is incentivized by the first agent's failure. The adversarial agent can attempt to lower the agent reward by perturbing elements of the action and state space of the first agent.

The adversarial agent's action space is a 1-dimensional vector of length 22, composed of perturbation values bound by $[-1, 1]$. The first two elements of the adversarial

action space are used to perturb the action space of the original agent's action space. Adversarial action perturbations are scaled by 0.1. Combining adversarial training with selective randomization, the adversarial agent has access to perturb a subset of the original agent's state space. The remaining 20 elements of the adversarial agent's action space are used to perturb 20 selective elements of the original agent's state space. Both the adversarial action and state perturbations are scaled down by 0.1. The selected elements that the adversary can perturb are the observed positions and velocities of both controlled AVs in the system and the observed distances of vehicles from the merge points.

4.2 Simulation Framework

4.2.1 Car Following Parameters

As introduced in Section 2.4, the human-driven vehicles in these simulations are controlled via IDM. Accelerations are provided to the vehicles via (2.1) and (2.2). Within these equations, s_0 is the minimum spacing or minimum desired net distance from the vehicle in front of it, v_0 is the desired velocity, T is the desired time headway, δ is an acceleration exponent, a is the maximum vehicle acceleration, and b is the comfortable braking deceleration.

Human-driven vehicles in the system operate using SUMO's built-in IDM controllers, which allows customization to the parameters described above. Standard values for these parameters as well as a detailed discussion on the experiments producing these values can be found in [32]. In these experiments, the parameters of the IDM controllers are defined to be $T = 1$ s , $a = 1$ m/s^2 , $b = 1.5$ m/s^2, $\delta = 4$, $s_0 = 2$ m, $v_0 = 30$ m/s. A noise parameter 0.1 was used to perturb the acceleration of the IDM vehicles.

Environment parameters in the simulation were set to match the physical constraints of the experimental testbed. These include: a maximum acceleration of 1 m/s^2, a maximum deceleration of -3 m/s^2, and a maximum velocity of 8 m/s. The timestep of the system is set to 1 s.

4.2.2 Algorithm/Simulation Details

We ran experiments with a discount factor of 0.999, a trust-region size of 0.01, a batch size of 20000, a horizon of 500 seconds, and trained over 100 iterations. The controller is a neural network, a *Gaussian multi-layer perceptron* with a tanh non-linearity, and hidden sizes of $(100, 50, 25)$. The choice of neural network non-linearities, size, and type were picked based on traffic controllers developed in [20].

The states are normalized so that they are between 0 and 1 by dividing each state by its maximum possible value. The agent actions are clipped to be between -3 and 1. Both normalization and clipping occur after the noise is added to the system so that the bounds are properly respected. The following codebases are needed to reproduce the results of our work. *Flow*[1], *SUMO*[2] and the version of *RLlib*[3] used for the RL algorithms is available on GitHub.

4.2.3 Simulation Results

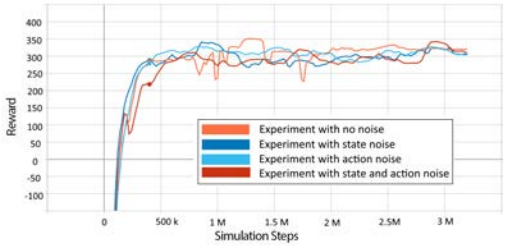


Figure 4.2: Convergence of the RL reward curves of the 3 Gaussian experiments and the noiseless policy.

Figure 4.3: Two RL-controlled AVs trained with adversarial multi-agent noise demonstrate emergent ramp metering behavior.

The reward curves of the Gaussian single-agent experiments are displayed in Fig. 4.2. These include the curves of the 3 experiments, which are trained with Gaussian noise injection, as well as one trained without any noise. In both the Gaussian single-agent and adversarial multi-agent experiments, the policy learns a classic form of traffic control: ramp-metering, in which one group of vehicles slows down to allow for another group of vehicles to pass. Despite the varying length of inflows from the two entries, policies consistently converge to demonstrate ramp-metering.

[1] https://github.com/flow-project/flow.
[2] https://github.com/eclipse/sumo at commit number **1d4338ab80**.
[3] https://github.com/flow-project/ray/tree/ray_master at commit number **ce606a9**.

4.3 Experimental Deployment

4.3.1 The University of Delaware's Scaled Smart City

University of Delaware's Scaled Smart City (UDSSC) is a 1:25 scale testbed designed to replicate real-world traffic scenarios and implement cutting-edge control technologies in a safe and scaled environment. UDSSC is a fully integrated smart city, which can be used to validate the efficiency of control and learning algorithms, including their performance on physical hardware. UDSSC utilizes high-end computers and a VICON motion capture system to simulate a variety of control strategies with as many as 35 scaled CAVs. For further information on the capabilities and features of the UDSSC, see [34].

UDSSC utilizes a multi-level control architecture to precisely position each vehicle using position feedback from a VICON motion capture system. High-level routing and desired velocity calculation is handled by the mainframe computer, as well as locating the vehicle relative to each street on the map. This information is sent to each CAV which then calculates its desired steering and velocity actions based on a Stanley Controller [35, eq. (9)], and the velocity control for each non-RL vehicle is specified by the IDM controller (2.1).

To implement the RL policy in UDSSC, the weights of the network generated by *Flow* were exported into a data file. This file was accessed through a Python script on the mainframe, which uses a ROS service to map the current state of the experiment into a control action for each RL vehicle. During the experiment, the RL vehicles took commands from this script as opposed to the IDM controller.

To generate a disturbance on the roundabout system, a random delay for when each group was released was introduced. This delay was uniformly distributed between 0 and 1 seconds for the western group and between 0 and 4 seconds for the northern group during UDSSC experiments. The size of each vehicle group was randomly selected from a uniform distribution for each trial.

4.3.2 Experimental Results

The data for each vehicle was collected through the VICON motion capture system and is presented in Table 4.1. The position of each car was tracked for the duration of each experiment, and the velocity of each car was numerically derived with a first order finite difference method.

For all trials, the RL vehicle exhibited the learned ramp metering behavior, where the western leader reduced its speed to avoid yielding by the northern group. The metering behavior was extreme for the Gaussian single-agent noise case, especially

Table 4.1: Experimental results for the baseline (no RL) case and each training method.

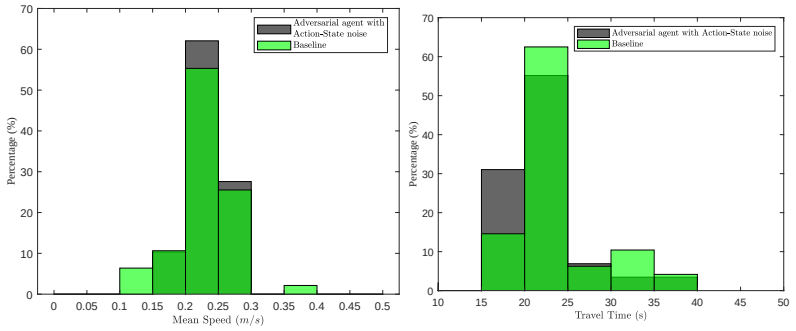
Training	Mean Time (s)	Mean Speed (m/s)	Trials	% Time Saved	Crashes
Baseline	23.6	0.23	47	-	0
Adversarial Multi-Agent					
Action-State	22.1	0.24	29	+6.3	0
Action	22.1	0.23	23	+6.4	0
State	21.4	0.24	26	+9.6	10
Gaussian Single-Agent					
Action-State	25.8	0.21	26	-9.2	0
State	23.0	0.23	18	+2.6	0
Action	23.1	0.22	37	+2.4	0
Noiseless	22.8	0.23	32	+3.5	0

when noise was added to the action and state together. This excessive metering significantly reduced the average speed and increased travel delay, as seen in Table 4.1. The adversarial multi-agent training significantly outperformed the Gaussian single-agent and tended to leave only a single vehicle yielding at the northern entrance. This strategy led to a travel time reduction for the northern group without a significant delay in western vehicles. The adversarial multi-agent case with noise injected only into the state accelerated especially fast and led to several catastrophic accidents between the two RL vehicles. Finally, for small numbers of vehicles, the adversarial multi-agent trained controllers appeared to exhibit an emergent zipper merging behavior.[4]

Relative frequency histograms of average travel time and mean speed for the adversarial multi-agent case with noise injected into both action and state versus baseline scenarios are overlaid in Fig. 4.4. Over all trials, the adversarial case had a higher relative frequency of shorter travel time compared to the baseline scenario. Average travel time for 30% of adversarial scenarios, lies in the range $[15\text{s}, 20\text{s}]$ comparing to 15% for the baseline scenarios.

From Table 4.1, we can see the average speed for baseline and the adversarial multi-agent with noise in action-state are nearly the same. However, in Fig. 4.4, we can see that approximately 8% of trials in the baseline scenarios have an average speed between 0.1 m/s and 0.15 m/s. Furthermore, there are some trials that the average speed of the baseline scenario is between 0.35 m/s and 0.4 m/s. On the other hand,

[4]Videos of the experiment and supplemental information can be found at: `https://sites.google.com/view/ud-ids-lab/arlv`.



(a) (b)

Figure 4.4: A relative frequency histogram for a the mean speed and b travel time of each vehicle for the baseline and adversarial multi-agent scenarios with noise injected in action and state.

the average speed for the adversarial multi-agent with noise in action and state varies less, and near 65% of trials have the average speed between 0.2 m/s and 0.25 m/s compared to 55% in the baseline scenarios.

4.4 Conclusion

In this work, we developed a zero-shot transfer of an autonomous driving policy directly from simulation to the UDSSC testbed. Even under stochastic, real-world disturbances, the adversarial multi-agent policy improved system efficiency by reducing travel time and average speed for most vehicles.

As we continue to investigate approaches for policy transfer, some potential directions for future research include: multi-agent adversarial noise with multiple adversaries, tuning to determine which elements of the state space are most suitable for perturbations, tuning injected noise to maximize policy robustness, larger, more complex interactions, such as intersections, or merging at highway on-ramps, and longer tests involving corridors with multiple bottlenecks.

Bibliography

[1] U. DOT, "National transportation statistics," *Bureau of Transportation Statistics, Washington, DC*, 2016.

[2] Z. Wadud, D. MacKenzie, and P. Leiby, "Help or hindrance? the travel, energy and carbon impacts of highly automated vehicles," *Transportation Research Part A: Policy and Practice*, vol. 86, pp. 1–18, 2016.

[3] S. E. Shladover, "Connected and automated vehicle systems: Introduction and overview," *Journal of Intelligent Transportation Systems*, no. just-accepted, pp. 00–00, 2017.

[4] S. Cui, B. Seibold, R. Stern, and D. B. Work, "Stabilizing traffic flow via a single autonomous vehicle: Possibilities and limitations," in *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pp. 1336–1341, IEEE, 2017.

[5] G. Orosz, "Connected cruise control: modelling, delay effects, and nonlinear behaviour," *Vehicle System Dynamics*, vol. 54, no. 8, pp. 1147–1176, 2016.

[6] D. Swaroop and J. K. Hedrick, "String stability of interconnected systems," *IEEE transactions on automatic control*, vol. 41, no. 3, pp. 349–357, 1996.

[7] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous Robots*, vol. 40, no. 3, pp. 429–455, 2016.

[8] L. Li, Y. Lv, and F.-Y. Wang, "Traffic signal timing via deep reinforcement learning," *IEEE/CAA Journal of Automatica Sinica*, vol. 3, no. 3, pp. 247–254, 2016.

[9] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 3389–3396, IEEE, 2017.

[10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[11] F. Belletti, D. Haziza, G. Gomes, and A. M. Bayen, "Expert level control of ramp metering based on multi-task deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[12] C. Wu, A. Kreidieh, K. Parvate, E. Vinitsky, and A. M. Bayen, "Flow: Architecture and benchmarking for reinforcement learning in traffic control," *arXiv preprint arXiv:1710.05465*, 2017.

[13] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," *arXiv preprint arXiv:1703.02702*, 2017.

[14] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," *arXiv preprint arXiv:1610.04286*, 2016.

[15] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *arXiv preprint arXiv:1710.06537*, 2017.

[16] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv preprint arXiv:1610.03518*, 2016.

[17] M. Mueller, A. Dosovitskiy, B. Ghanem, and V. Koltun, "Driving policy transfer via modularity and abstraction," in *Conference on Robot Learning*, pp. 1–15, IEEE, 2018.

[18] Z. Xu, C. Tang, and M. Tomizuka, "Zero-shot deep reinforcement learning driving policy transfer for autonomous vehicles based on robust control," in *International Conference on Intelligent Transportation Systems*, pp. 2865–2871, IEEE, 2018.

[19] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pp. 23–30, IEEE, 2017.

[20] E. Vinitsky, A. Kreidieh, L. Le Flem, N. Kheterpal, K. Jang, F. Wu, R. Liaw, E. Liang, and A. M. Bayen, "Benchmarks for reinforcement learning in mixed-autonomy traffic," in *Conference on Robot Learning*, pp. 399–409, IEEE, 2018.

[21] K. Jang, E. Vinitsky, B. Chalaki, B. Remer, L. Beaver, A. Malikopoulos, and A. Bayen, "Simulation to scaled city: zero-shot policy transfer for traffic control via autonomous vehicles," in *2019 International Conference on Cyber-Physical Systems*, (Montreal, CA), 2019.

[22] B. Chalaki, L. E. Beaver, B. Remer, K. Jang, E. Vinitsky, A. M. Bayen, and A. A. Malikopoulos, "Zero-shot autonomous vehicle policy transfer: From simulation to real-world via adversarial learning," 2019.

[23] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, pp. 128–138, December 2012.

[24] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, pp. 1329–1338, 2016.

[25] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica, "Ray rllib: A composable and scalable reinforcement learning library," *arXiv preprint arXiv:1712.09381*, 2017.

[26] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

[27] C. Wu, A. Kreidieh, E. Vinitsky, and A. M. Bayen, "Emergent behaviors in mixed-autonomy traffic," in *Conference on Robot Learning*, pp. 398–407, 2017.

[28] F. Hutter, J. Lücke, and L. Schmidt-Thieme, "Beyond manual tuning of hyperparameters," *KI-Künstliche Intelligenz*, vol. 29, no. 4, pp. 329–337, 2015.

[29] S. Hansen, "Using deep q-learning to control optimization hyperparameters," *arXiv preprint arXiv:1602.04062*, 2016.

[30] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, pp. 1–5, 2015.

[31] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, pp. 1889–1897, 2015.

[32] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[33] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems*, pp. 1071–1079, 2014.

[34] L. E. Beaver, B. Chalaki, A. M. Mahbub, L. Zhao, R. Zayas, and A. A. Malikopoulos, "Demonstration of a Time-Efficient Mobility System Using a Scaled Smart City," *Vehicle System Dynamics*, vol. 58, no. 5, pp. 787–804, 2020.

[35] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *2007 American Control Conference*, pp. 2296–2301, July 2007.