On Training Robust Policies for Flow Smoothing

Kanaad Parvate
Alexandre Bayen, Ed.
Pieter Abbeel, Ed.

Acknowledgement

The D-MALT work was a joint effort with Eugene Vinitsky, Yuqing Du, and Kathy Jang, advised by Profs. Pieter Abbeel and Alexandre Bayen. The flow smoothing analysis was a joint effort with Daniel Rothchild, Upadhi Vijay, Sulaiman Almatrudi and advised by Prof. Susan Shaheen.

I would like to specifically thank Eugene Vinitsky and Prof. Bayen for all of their support throughout the years. Additionally, I would like to thank all of my friends and professors who have made my time at Berkeley such a memorable experience. Go Bears!

Finally, to my parents. Without their constant support and love, none of this would have been possible. Thank you.

On Training Robust Policies for Flow Smoothing

by Kanaad Parvate

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:

[illegible]

Professor Alexandre Bayen
Research Advisor

May 28, 2020

(Date)

* * * * * * *

[illegible]

Professor Pieter Abbeel
Second Reader

May 28, 2020

(Date)

On Training Robust Policies for Flow Smoothing

Abstract

On Training Robust Policies for Flow Smoothing

by

Kanaad Parvate

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Flow smoothing, or the use specially designed controllers for automated vehicles to damp "stop-and-go" waves, has been shown to mitigate inefficiencies in human driving behavior, reducing system-wide fuel usage. Using the software framework, Flow, researchers can specify traffic scenarios and train flow smoothing controllers using deep reinforcement learning (RL). RL is a powerful technique for designing effective controllers, however, it tends to create controllers that overfit to their simulator: the resultant controllers are prone to failure when transferred to new settings.

We first present D-MALT: Diverse Multi-Adversarial Learning for Transfer. We explore the hypothesis that perturbing the dynamics of an environment with a diverse pool of adversaries can combat said failure by ensuring that the controller sees an informative set of possible perturbations. We demonstrate that the simple technique of rewarding adversaries for reaching different reward ranges is sufficient for constructing a set of highly diverse adversaries. Adversaries with the diversity reward find potential controller failure modes whereas we find that domain randomization, a standard transfer technique, fails to find these modes as the problem dimensionality increases. We show, in a variety of environments, that training against the diverse adversaries leads to better transfer than playing against a single adversary or using domain randomization.

Secondly, we detail a set of transfer tests and evaluations meant to assess the effectiveness of flow smoothing controllers and evaluate their performance in a variety of traffic scenarios. We present two such examples: varying the penetration rate of the AVs in the system and varying the aggressiveness of human drivers. As new flow smoothing controllers are developed in Flow, we expect D-MALT and other techniques for developing robustness to be paramount for successful transfer to physical settings. Analyzing the transfer performance of these controllers will be crucial to their ultimate deployment.

Contents

List of Figures

List of Tables

Acknowledgments

The D-MALT work was a joint effort with Eugene Vinitsky, Yuqing Du, and Kathy Jang, advised by Profs. Pieter Abbeel and Alexandre Bayen. The flow smoothing analysis was a joint effort with Daniel Rothchild, Upadhi Vijay, Sulaiman Almatrudi and advised by Prof. Susan Shaheen.

I would like to specifically thank Eugene Vinitsky and Prof. Bayen for all of their support throughout the years. Additionally, I would like to thank all of my friends and professors who have made my time at Berkeley such a memorable experience. Go Bears!

Finally, to my parents. Without their constant support and love, none of this would have been possible. Thank you.

Chapter 1

Introduction

Human driving behavior has shown to be unstable: even without the presence of outside perturbations, "stop-and-go" waves can form and propagate down a highway, causing cars to constantly accelerate and brake instead of traveling at a constant speed. As a result of this unnecessary acceleration and braking, traffic waves have been shown to increase fuel consumption by as much as 67% [32].

Flow smoothing is a technique that uses a type of adaptive cruise control in mixed-autonomy settings to dissipate these traffic waves, reducing the fuel wasted on unnecessary accelerating and braking. If a large enough fraction of vehicles ($\sim$ 5-30%) on the road have a flow-smoothing cruise controller enabled, traffic waves can be dissipated, even in regions of high congestion. Existing empirical evidence shows that flow smoothing can reduce real-world fuel consumption by 40% on a simple ringed road [32].

To facilitate the training and evaluation of controllers in more complex mixed-autonomy scenarios, researchers developed Flow [37], a framework interfacing an open source traffic microsimulator, SUMO [4] with state-of-the-art deep reinforcement learning (RL) algorithms. RL is a technique for developing controllers that has been shown to be successful in a wide variety of settings, including video games [24] [21], 3D locomotion [29], and manipulation [1]. RL policies are developed to maximize a target reward and work especially well when given access to direct sampling from a simulator.

Policies may seek to maximize average velocity, maintain a certain throughput, minimize travel time, reduce energy consumption, or otherwise manipulate some characteristic about the traffic at a system level. Flow has yielded state of the art controllers for a variety of traffic settings [36]. Controllers trained in Flow were even successfully transferred to physical, 1:25 scale cars in a toy-city environment [16].

In this report, we will (1) detail the Flow framework and outline the challenges in deploying policies trained in Flow, (2) present D-MALT: Diverse Multi-Adversarial Learning for Transfer, a contribution to the robustness literature that can be used in conjunction with Flow, (3) introduce transfer tests in Flow to the assess the robustness of controllers across several different metrics critical for deployment. This work serves as a stepping stone towards training driving policies in Flow and deploying them at-scale in the real world.

Chapter 2

Flow

Flow [37] is a framework for training controllers using deep reinforcement learning in mixed-autonomy traffic. In this section, we introduce flow smoothing, introduce Deep RL, detail the Flow framework, and present several challenges to deployment of controllers trained with Flow.

2.1 Flow Smoothing

On a given stretch of roadway, the average following distance between cars is determined by the density of vehicles on the road. When demand for the roadway increases, the density increases, and the average following distance decreases commensurately. As a result, traffic is forced to slow down in order to accommodate the shorter following distance: given human reaction time, the distance required to stop safely behind a braking vehicle increases with increasing speed, so a smaller gap between cars requires cars to be going slower.

In an ideal world, all the cars along a congested roadway would travel at the average speed; this would minimize accelerating and braking, thereby minimizing emissions. However, constant-speed flow is unstable in practice. As soon as one driver brakes, trailing cars also have to brake to compensate, eventually leading to traffic waves. In a traffic wave, cars speed up too much when there is space ahead of them and then are forced to brake once they meet slower traffic. This accelerating and braking phenomenon forms a positive feedback loop, and has been shown to lead to the spontaneous formation of stop-and-go traffic waves.

Flow smoothing is a technique used to smooth out these traffic waves: by accelerating less than a human would in the same situation, automated flow-smoothing vehicles can eliminate waves, leading to reduced energy consumption from all cars along the same stretch of road.

2.2 Deep Reinforcement Learning

Deep Reinforcement Learning (RL) aims develop a controller that operates on a Markov Decision Process [5], defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \rho_0, \gamma, H)$, where $\mathcal{S}$ is a (possibly infinite) set of states, $\mathcal{A}$ is a set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to R$ is the reward function, $\rho_0 : \mathcal{S} \Rightarrow \mathbb{R}_{\geq 0}$ is the initial state distribution, $\gamma \in (0, 1]$ is the discount factor, and H is the time horizon. For most traffic tasks in the real world, the MDP is only partially observable, requiring two extra components: Ω, a set of observations, and $\mathcal{S} \times \Omega \to \mathbb{R}_{\geq 0}$, a probability distribution over observations.

Throughout this work, we leverage policy gradient algorithms which aim to learn a stochastic policy $\pi_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$. Policies are parameterized by multilayer neural networks. Using samples from an environment, in this case the simulator, the policy is updated using gradient descent: tuples of $(s \in \mathcal{S}, a \in \mathcal{A}, s' \in \mathcal{S}, r)$ are collected by rolling out the policy over the horizon H, and the resulting samples are used to compute gradients.

2.3 Flow

Flow provides an interface between driving simulators and RL libraries. Specifically, it creates an environment interface, as defined by the popular library OpenAI Gym [7]. For each training iteration, batches of samples are collected by simulating the policy in the environment for a given number of timesteps. For each step, the simulator is queried for an observation, the policy is queried for an action given that observation, the simulator is stepped forward with that action. The resulting state is produced by the simulator, and the reward is calculated based on the new state. At the end of a rollout, the simulator is reset using the initialization procedure specified.

Figure 2.1 illustrates the modular design of Flow. Note that the learned actors use the environment interface to interact with the simulator. The policies controlling these actors are decentralized: each AV in the system is controlled by an identical policy, processing only local observations, and outputting an acceleration for itself. Flow handles the passing each AVs observation to the policy and correspondingly applying the actions individually. Flow further allows for the easy construction and design of the network, its dynamics, and pre-specified controllers that comprise of the other agents in the simulation.

Vehicle dynamics models

Traffic dynamics can be generally represented by *ordinary differential equation* (ODE) models known as *car following models* (CFMs). These models describe the longitudinal dynamics of human-driven vehicles, given only observations about itself and the vehicle preceding it. CFMs vary in terms of model complexity, interpretability, and their ability to reproduce prevalent traffic phenomena, including stop-and-go traffic waves. Separate models are necessary for modeling of a variety of additional complex traffic dynamics, including lane changing, merging, driving near traffic lights, and city driving.

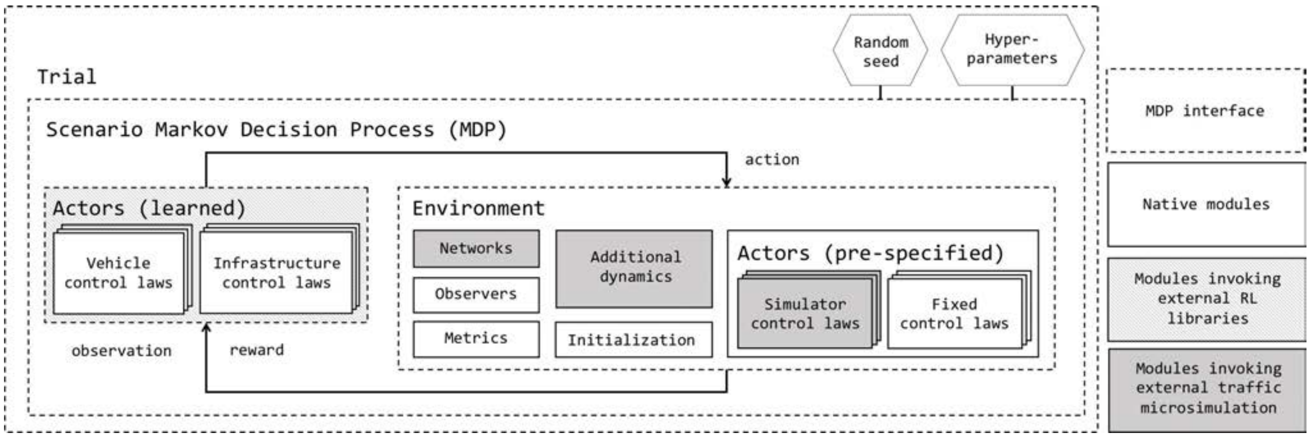


Figure 2.1: Overview of the Flow Framework from [37]. Specifically, Flow is comprised of several modules that aim to represent the traffic scenario as an MDP. The pre-defined actors and simulation dynamics attempt are configured to match a specific deployment environment. Additional dynamics may include logic for toll-metering, traffic lights, or other components of the traffic network.

2.4 Deployment Challenges

When designing these controllers for deployment in the real world, careful calibration of network dynamics and control boundaries is essential. However, in many facets of the simulation, large gaps will exist between the dynamics during training and the dynamics during deployment. To be deployed in real settings, controllers must be robust to a variety of inaccuracies, such as observational and actuation noise, fluctuations in density and speed of traffic, misspecified driving dynamics, or unpredictable behavior of human drivers.

Jang et al. attempted zero-shot transfer from simulation to real in a 1:25 scale toy city [16]. They found that transferring naively trained controllers from Flow directly to the physical environment was unsuccessful, leading to collisions. By introducing noise in both the actions and the states of the policy during training, the authors improved the policies performance in transfer. Though this simple trick may have been sufficient for the toy city, more sophisticated strategies will likely be necessary for a full scale deployment.

Secondly, though human models can be tuned to recreate macroscopic phenomenon such as stop-and-go-waves, the true behavior of humans may similarly cause a distributional shift that the controllers were not trained against. Furthermore, injecting noise or otherwise randomizing the behavior of the humans in a way that is realistic is a non-trivial problem.

The sim-to-real gap between the driving simulator used in Flow and a full-scale road network with actual autonomous vehicles must be bridged in order to successfully deploy learned policies on these fleets of vehicles.

Chapter 3

D-MALT: Diverse Multi-Adversarial Learning for Transfer

3.1 Introduction

Developing policies that work effectively across a wide range of potential deployment environments is one of the core problems of engineering. The complexity of the physical world means that the models used to design policies are often inaccurate. This leads to performance degradation when policies tuned to fit a particular model are deployed on the true model, also called *model mismatch*. In particular, applications such as robotics often employ physics simulators as models, but the mismatch in dynamics between the simulator and the real world results in underperforming controllers. This is known as the *reality gap*. In this work, we focus on mitigating the performance degradation that results from model mismatch.

Optimization-based policy design schemes are highly effective at taking advantage of the particulars of their simulator, often overfitting to the simulator model. In some extreme cases, policies have been shown to exploit bugs or unrealistic physics, learning policies that allow them to fly [17]. Some physical dynamics such as contact forces are inherently difficult to model, making accurate simulation nearly impossible. To make simulation-based reinforcement learning control design viable, it is necessary to develop techniques that can alleviate overfitting and guarantee robust performance at transfer time.

For model-free control problems, an effective technique to combat the reality gap is *domain randomization* [34], a procedure in which perturbations are randomly sampled for both the dynamics and the state space: the learned policy should perform well in expectation over all the dynamics instances. For example, [25] train a robot to push a puck to a goal while sampling random friction and mass values for the dynamics of the puck. Then, even if the base value of puck friction is not perfectly modelled in the simulator, the resultant policy will transfer well as long as the real puck's friction is near the range of sampled values.

However, simulators have an immense range of possible parameters to vary; domain randomization depends critically on a human designer picking appropriate parameters to randomize. In the puck example, the designers chose to neglect varying the friction of the arm joints, air resistance, actuation speeds, etc. When the variations between simulator and reality are obvious, picking the right randomizations is straightforward. However, when the model mismatch between simulator and reality is less obvious, it is possible to pick randomizations that entirely miss failure modes. Furthermore, because domain randomization uniformly samples over examples which are not aligned with any notions of task difficulty, robustness against hard examples is not guaranteed.

An alternate approach is to introduce a single perturbative adversary trained to minimize the agent's reward (**RARL**) [26]. Training against this adversary should yield a policy that approximately plays a minimax strategy. This policy will act as though the worst possible outcome could occur at any time, leading to lower performance if the transfer task turns out to have easier dynamics than the dynamics constructed by the adversary. Furthermore, as we discuss in Sec. 3.3, standard RL formulations are unlikely to actually yield a minimax optimal policy.

In this work, we contrast domain randomization and RARL with Diverse Multi-Adversarial Learning for Transfer (**D-MALT**), a procedure in which we deploy a set of *learned, diverse adversaries* that perturb the system dynamics. We set up the problem as a general-sum, multi-agent game in which each adversary is given a cumulative reward target that it tries to push the policy to achieve. Using multiple adversaries with different target rewards, we can construct a range of difficult and easy dynamics for the policy to solve. The agent must subsequently learn a policy that is robust to a range of very distinct tasks.

We show that adversaries trained with this simple reward are can catch catch failure modes missed by domain randomization and RARL. We use a multi-armed bandit task and a linear-quadratic regulator task to illustrate these failure modes and demonstrate that policies trained using D-MALT surpass both domain randomization and RARL on these baselines. Finally, we show in a Mujoco Hopper task that training against multiple diverse adversaries leads to a policy that matches the performance of a hand-designed domain randomization baseline.

The main contributions of this paper are the following:

- Formulating a simple reward that encourages adversary diversity and showing that this reward is sufficient to yield significant diversity.
- Showing that even on simple tasks, D-MALT can catch failure modes that domain randomization consistently misses.
- Demonstrating that playing against diverse adversaries is competitive against hand-designed domain randomization and can even exceed it on a holdout test set of transfer tasks.

3.2 Related Work

Our work builds on prior work on using domain randomization to learn robust policies, as well as existing work on learned domain randomizations. Domain randomization, without any fine-tuning in the real world, has been used for effective robot grasping [34], manipulating a puck to a goal [25], learning to fly a quadcopter [28], and manipulating a Rubik's cube [2].

We differ from these approaches in that our domain randomizations are learned, removing the need for a human to design the specific randomizations. Additionally, as we note in Sec. 3.3, domain randomization can yield uninformative samples in high-dimensional space. Unlike [34], we do not consider state space domain randomizations in this work.

Additionally, there is prior work on using one or more adversaries to attempt to learn a robust policy. In [20], the work closest to ours, they use a method called *Active Domain Randomization*, in which a set of diverse adversaries are used to generate the domain randomizations. Our primary improvement over this work is a series of simplifications: we replace their discriminator reward with a simple goal reward (described in Sec. 3.3) and show that there is no need to use a training strategy that couples the adversary policies in the gradient update; simple uncoupled policy gradient algorithms are sufficient and faster. Robust Adversarial Reinforcement Learning (RARL) [26] uses a single adversary to pick out friction forces to apply in simulation environments and trains the adversary against the policy in a zero-sum game to approximate a minimax formulation; we use this work as a baseline.

We also note a series of other approaches for learning policies that are robust to transfer, each of which our work is compatible with. Domain randomization is commonly used to generate tasks in meta learning, as in [23]; our approach can be substituted in to replace the domain randomization. [31] apply a penalty for adversaries generating trajectories that differ sharply from the base environment. [3] regularize the Lipschitz coefficient of the policy to ensure that the policy is smooth across jumps in state and observe that this appears to improve robustness and generalization. [27] point out that the use of neural networks can contribute to a loss of robustness, and that simpler linear controllers may help with transfer.

Other approaches to tackling sim-to-real include re-tuning the simulator using real-world data to increase its correspondence with reality [6, 33, 9], using the simulation to discover high level commands that are obeyed by a controller optimized for command-following in realistic dynamics [22, 10], using the simulator as a prior on the true model [11], and using a generative network to make reality look more like the simulator [15].

3.3 Methods

In this section, we discuss the baselines we compare against for our transfer tests: **a)** domain randomization, and **b)** single-adversary training in a minimax formulation (RARL). We outline weaknesses of both approaches. Finally, we propose D-MALT, a training scheme that aims to mitigate the identified weaknesses of our baselines.

Baselines

RARL: Robust Adversarial Reinforcement Learning

Our first baseline consists of a policy trained with RARL: a single adversary operating in a zero-sum game against the policy, as is done in [26]. The zero-sum game approximates a minimax game (i.e. if the policy reward is r_t the adversary reward will be $-r_t$); however, it is not necessarily equivalent to a minimax game, as RL optimizes the expected return rather than the minimum or maximum return. Training under this framework can provide a lower bound on performance; a policy playing a minimax strategy has a guaranteed minimum return as long as it is deployed in environments that do not apply perturbations exceeding possible adversarial actions. We note that a single policy playing the true minimax strategy would yield behavior that is maximally conservative. Because the policy always expects perturbations that yield the worst possible return, the policy plays conservatively on transfer tasks that might be easier than its training task.

We note that there is no guarantee of convergence of policy gradient methods to a Nash equilibrium in zero-sum games [19]. Empirically, policy gradient learning in zero-sum games often fails to find the Nash equilibrium [13]. Furthermore, it is common to parameterize a continuous action policy via a diagonal Gaussian distribution; this class of policies is insufficiently expressive to represent any Nash equilibrium that involves a multi-modal, stochastic adversary policy. Due to this, the adversary can only represent dynamical systems drawn from a unimodal distribution; playing against an adversary trained in the RL formulation is unlikely to actually yield the guaranteed lower bound performance that would occur if we learned a true minimax optimal policy.

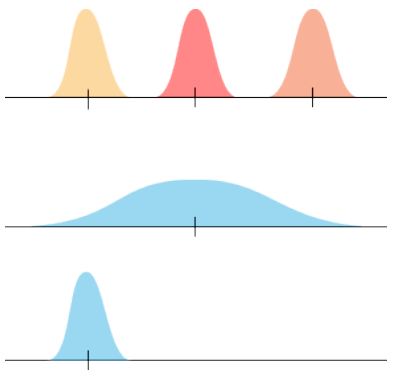


Figure 3.1: Comparison between the optimal adversary strategy, a trimodal distribution captured by three adversaries (shown on top), and some possible single adversary strategies (shown in blue) which can never express the required multi-modality.

Fig. 3.1 illustrates this issue; the upper figure illustrates a multi-modal strategy in which an adversary chooses with equal weight between three sets of actions. The unimodal, single adversary can only approximate this by either concentrating its weight on one of the actions or uniformly spreading out between the actions. One could attempt to resolve this by sampling uniformly over multiple adversaries, each playing a minimax strategy but this is likely to yield a degenerate case where some of the minimax adversary strategies collapse onto the same mode instead of covering all the modes. This is one of the reasons diversity is necessary; it helps to avoid mode collapse.

Domain Randomization

As a second baseline, we train policies in the presence of domain randomizations that are sampled uniformly at the beginning of each training rollout. For each experiment, we pick particular randomization parameters that seem to logically correlate with the set of transfer tasks that we eventually intend to deploy our policies in. However, since policy gradient methods aim to optimize the expected cumulative return across rollouts, we expect the resultant policy to perform well in *expectation* across the distribution of tasks returned by the domain randomization. Therefore, if the domain randomization is biased towards tasks that are “easy”, the policy will bias its behavior to favor improved performance on easy tasks.

As we show on two simple environments, domain randomization can be unintentionally biased towards tasks that are not particularly difficult, leading to a policy that performs poorly on hard examples. Without a deep understanding of the specific problem domain, there is not an obvious way to tune domain randomization to incorporate knowledge about the difficulty of the transfer tasks that the policy will be deployed on.

D-MALT: Diverse Adversarial Reinforcement Learning for Adaptation

In D-MALT, we allow a diverse set of adversaries to pick perturbations that the policy experiences during training time. Each adversary has an expected policy return target and achieves maximum return when the policy hits the return target. With this method, we address the solutions to both weaknesses mentioned above: 1) the presence of multiple adversaries should alleviate the issues of uni-modality discussed in Sec. 3.3, and 2) the reward target gives us a knob to tune the difficulty of the set of environments the policy experiences during training, allowing us to overcome the pitfall of domain randomization discussed in Sec. 3.3, where most of the sampled perturbations do not actually cause any degradation in the expected reward. In Sec. 3.3 we describe the formulation of the adversary reward that we use to tune diversity / task difficulty and in Sec. 3.3 we describe the training procedure of our algorithm.

Adversary reward function

In reinforcement learning (RL), the goal is to maximize the expected return. If a policy trained in one environment achieves the same expected return when transferred to another, we would say that the controller was robust to transfer even if the distribution of trajectories in the new environment was completely different. Thus, environments that directly cause a shift in reward present valuable information to the policy.

To encourage diversity in the observed rewards, we reward the adversaries for pushing the policy into trajectories that yield different expected returns. For notation, we define $r^i_{\text{adv}}(t)$ as the reward of adversary i at timestep t, $r_{\text{p}}(t)$ as the reward of the policy at timestep t and $R^t_{\text{p}} = \sum_{i=0}^{t} r_{\text{p}}(i)$ as the cumulative reward of the policy up to timestep t. Finally, we denote the maximum time-horizon of the problem as T.

We define two hyperparameters: R_{low}, the lowest target of the desired reward range and R_{high}, the highest target. Given N adversaries indexed from 1 to N, we uniformly divide the reward range between adversaries such that adversary i will have a desired total score of:

$$r^i_{\text{goal}} = R_{\text{low}} + i\frac{R_{\text{high}} - R_{\text{low}}}{N}$$

Because it is easier to learn a dense reward than a sparse reward, we allocate the total reward evenly among each timestep. Finally, since many RL problems have termination conditions, we want the adversary reward to be positive so it does not have an incentive to terminate the rollout. Putting these to components together, our adversary reward is:

$$r^i_{\text{adv}}(t) = \frac{r^i_{\text{goal}} - \left|r^i_{\text{goal}} - \frac{T}{t}R^t_{\text{p}}\right| + c}{T} \tag{3.1}$$

where c is a constant used to ensure that the reward is always positive. At each timestep, the adversary reward is maximized by moving the policy toward hitting the reward target exactly, treating the cumulative reward target as evenly divided amongst the timesteps.

Training procedure

In the multi-adversary formulation, we sample a new adversary at the start of each rollout, such that if the batch size is M and we have N adversaries, each adversary sees an average of $\frac{M}{N}$ samples. In contrast to [26] we do not alternate training between the policy and adversaries; every adversary that has acted at least once during an iteration receives a gradient update. We use Proximal Policy Optimization [30] to perform the optimization of our policies and adversaries. Since only one adversary is active in any given rollout, the total time cost of D-MALT is the normal cost of stepping the environment plus twice the amount of time needed to compute actions plus twice the amount of time needed to compute gradients.

3.4 Experiments

In this section we briefly characterize the Markov Decision Processes (MDPs), implementations of domain randomization, and transfer tests for each task. A more complete description of the state, action, and reward spaces of the MDP is available in the supplementary section.

Overview of the Environments

Linear systems

We use a linear system for demonstrating how domain randomization can fail to identify critical failure modes. The objective is to minimize the expected cumulative regret of a quadratic cost for an unknown linear system with the following dynamics:

$$\begin{gathered} x_{k+1} = -\alpha(I + \Delta)x_k + 2(1-\alpha)Ia_k \\ 0.5 < \alpha < 1 \\ |\Delta_{ij}| < \frac{2(1-\alpha)}{d} \end{gathered} \tag{3.2}$$

where x_k is the state of the system at time k, the policy action is a_k, Δ is a perturbation matrix either picked by an adversary or via domain randomization, and d is the state dimension. Every n_r we reset the system to prevent the state from going off to infinity in long-horizon tasks.

The value of α is chosen so that the base system without Δ is stable. By the Gershgorin Circle Theorem [14], the bounds on the individual elements of Δ mean that $\lambda_{\max} \leq 2(1-\alpha)$ where $\lambda_{\max}$ is the spectral radius of Δ. This implies that the adversary can always destabilize the uncontrolled system and that the policy can always stabilize the perturbed system.

The agent minimizes the cumulative quadratic regret $\sum_{t=0}^{T} \frac{J^*}{T} - (x_t^T x_t + 5u_t^T u_t)$ where $J^* = x_0^T P x_0$ is the optimal infinite horizon discrete time cost which can be found by solving the discrete time Algebraic Ricatti Equation to compute a value for P.

The agent's state space is all previous observed states and actions, concatenated together and padded with zeros; the reward is $r_t = \frac{J^*}{T} - (x_t^T x_t + 5u_t^T u_t)$. The adversary has a horizon of 1; its only state is x_0 and at the first timestep it outputs the entries of the matrix Δ.

For the domain randomization baseline we uniformly sample the entries of Δ from $\left[-\frac{2(1-\alpha)}{n}, \frac{2(1-\alpha)}{n}\right]^{n\times n}$. This corresponds to a naive domain randomization scheme. Another possible baseline would be to sample the eigenvalues of Δ directly and then apply a unitary similarity transformation to Δ; however, this approach would fail to construct systems with geometric multiplicity greater than 1. A researcher hoping to easily construct systems with non-trivial null-spaces might fall back on uniformly sampling the matrix entries. Thus, we used this as our baseline rather than directly sampling the matrix eigenvalues.

Multi-armed stochastic bandit

As a second illustrative example we examine a multi-armed stochastic bandit, a problem widely studied in the reinforcement learning literature. We construct a k-armed bandit where each arm i is parameterized by a Gaussian with unknown mean μ_i, bounded between $[-5, 5]$ and standard deviation σ_i, bounded between $[0.1, 1]$. The goal of the policy is to minimize total cumulative regret $R_T = T\max_i \mu_i - \mathbb{E}\left[\sum_{t=0}^{n} X(a_t)\right]$ over a horizon of T steps where $X(a_t) \sim \mathcal{N}(\mu_a, \sigma_a)$.

At each step, the policy is given an observation buffer of stacked frames consisting of all previous action-reward pairs padded with zeros to keep the length fixed. The adversary has a horizon of 1, and at timestep zero it receives an observation of 0 and outputs k means and standard deviations that are randomly shuffled and assigned to the arms. For our domain randomization baseline we uniformly sample the means and standard deviations of the arms.

Mujoco

We use the Mujoco Hopper [35] environment as a test of the efficacy of our method versus the baselines described in Sec. 3.3. Recalling that one of our goals is to minimize human involvement, we replace the adversary perturbation formulation of [26] with a more general perturbation in which the adversary applies an action that is added to the policy actions. The notion here is that the adversary action is passed through the dynamics function and represents a perturbed set of dynamics.

Although it is standard to clip actions within some box, we clip the policy and adversary actions separately. Otherwise, a policy would be able to limit the effect of the adversary by always taking actions at the bounds of its clipping range.

The policy's state space consists of the previous h observed states and actions concatenated together and padded with zeros; the reward is a combination of distance travelled and a penalty for large actions. The adversary observes the same state as the agent; its actions are directly applied onto the policy action. Its reward function is given by Eq. 3.1.

For our domain randomization Hopper baseline, we use the following randomization scheme: at each rollout, we scale the friction of all joints by a single value uniformly sampled from [0.7, 1.3]. We also randomly scale the mass of the 'torso' link by a single value sampled from [0.7, 1.3].

Train-Validate-Test Split

Unlike standard RL, where the effectiveness of a set of hyperparameters directly corresponds to its performance during train time, we cannot simply use the cumulative return as the selection metric, as the optimal hyperparameters on the training task may not necessarily train the most robust policy.

We adopt the conventional train-validate-test split from supervised learning to evaluate different hyperparameters. We use the validation tasks (described in supplementary section) to select hyperparameters. For the chosen hyperparameters, we retrain the policy across ten seeds and report the resultant mean and standard deviation of the seeds across the validation tests. This tests for robustness of the proposed approach to initialization of the neural network. We also construct a holdout set of "hard" examples for each of the tasks that we report the mean and standard deviation for (std. deviations are reported in the supplementary section).

We use the following holdout test sets for the different environments.

- Linear: a set of systems in which the perturbation matrix always makes the system unstable.
- Bandit: variations of the std. deviations of the arms to be at their minimum and maximum values, as well as varying the maximal separation of the arms.
- Mujoco: combinations of low and high values of frictions for different combinations of legs and torso.

Exact details on the tests are in the supplementary section.

Code reproducibility

Details on the computational resources used for each environment, including the hyperparameter sweeping, is available in the supplemental material. Additionally, an estimate of the total cost of reproduction of this paper is provided in the supplementary section; it comes out to ($\approx$ \$100) on AWS EC2 when using spot instances.

All code used in this experiment is available at `Anonymized` and is frozen at git commit `Anonymized`. We have constructed a single file that can be used to rerun all of the experiments and a single file that can be used to reproduce all the graphs. For RL training, we use RLlib 0.8.0 [18] and AWS EC2 for all the computational resources.

3.5 Results and Discussion

Adversary Diversity

To understand whether D-MALT is sufficient to induce diversity in our adversaries, we visualize the eigenvalues from a 6-dimensional linear system system with five adversaries in Fig. 3.2, as well as the distribution of adversary arms for a 2-arm bandit in Fig. 3.3. For the linear system, the learned strategies of the adversary correspond to expected intuition: the adversaries trying to make the problem difficult (upper row) output eigenvalues that make the system unstable. As the adversaries shift to making the problem easier (lower row), they gradually shift to outputting eigenvalues that help stabilize the system. The diversity in the bandit cases is also clearly visible in Fig. 3.3. The average regret against the adversaries in the top row is lower, while the average regret against the bottom row is higher.

Analysis of validation and test set results

In the linear task, the policy trained with RARL always created an unstable system, leading to a policy unaware that systems could be stable. This subsequently resulted in poor transfer performance on the tests that primarily had stable eigenvalues (left and middle column in Fig. 3.4). This stands to reason; a policy trained against RARL has no notion of stable dynamics and will never learn that dynamics can naturally decay to zero. Conversely, domain randomization performed well on the stable example but performed poorly on the unstable

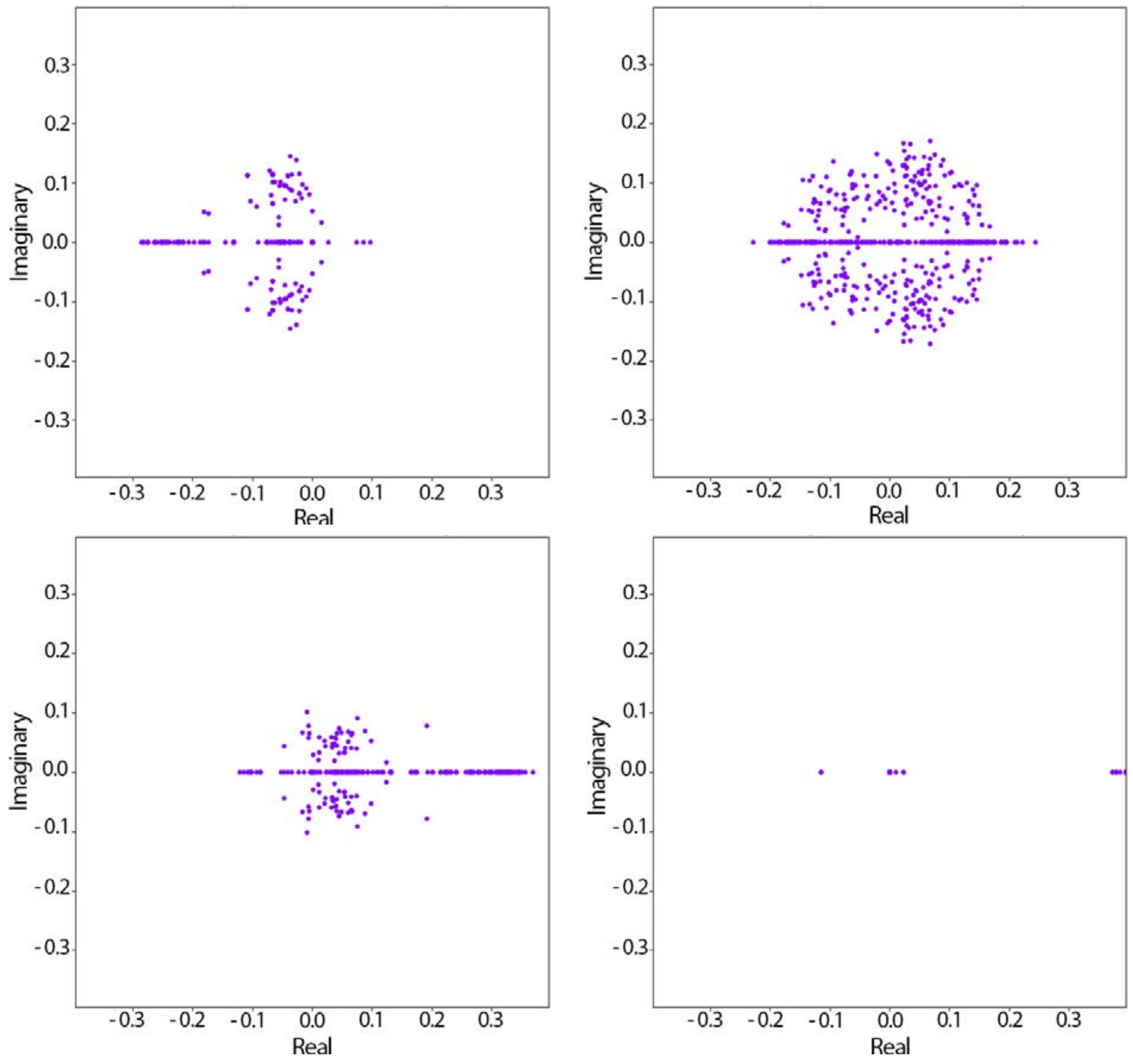


Figure 3.2: Visualizations of the eigenvalues of the matrices output by 4 adversaries from a 6-dimensional, 5 adversary run. For each adversary, we sample 100 actions. The upper row consists of adversaries whose goal was a low policy reward. The bottom row has adversaries with a higher policy goal.

examples in 4-dimensions. This can be understood by looking at Fig. 3.5; in 4 dimensions, random sampling of matrix entries failed to yield any unstable systems ($\lambda < 0.2$). The logic for picking the particular form of domain randomization discussed in Sec. 3.4 may have been compelling, but due to the unexpected clustering behavior of eigenvalues of high dimensional matrices, domain randomization fails to pick up on the critical notion that linear systems

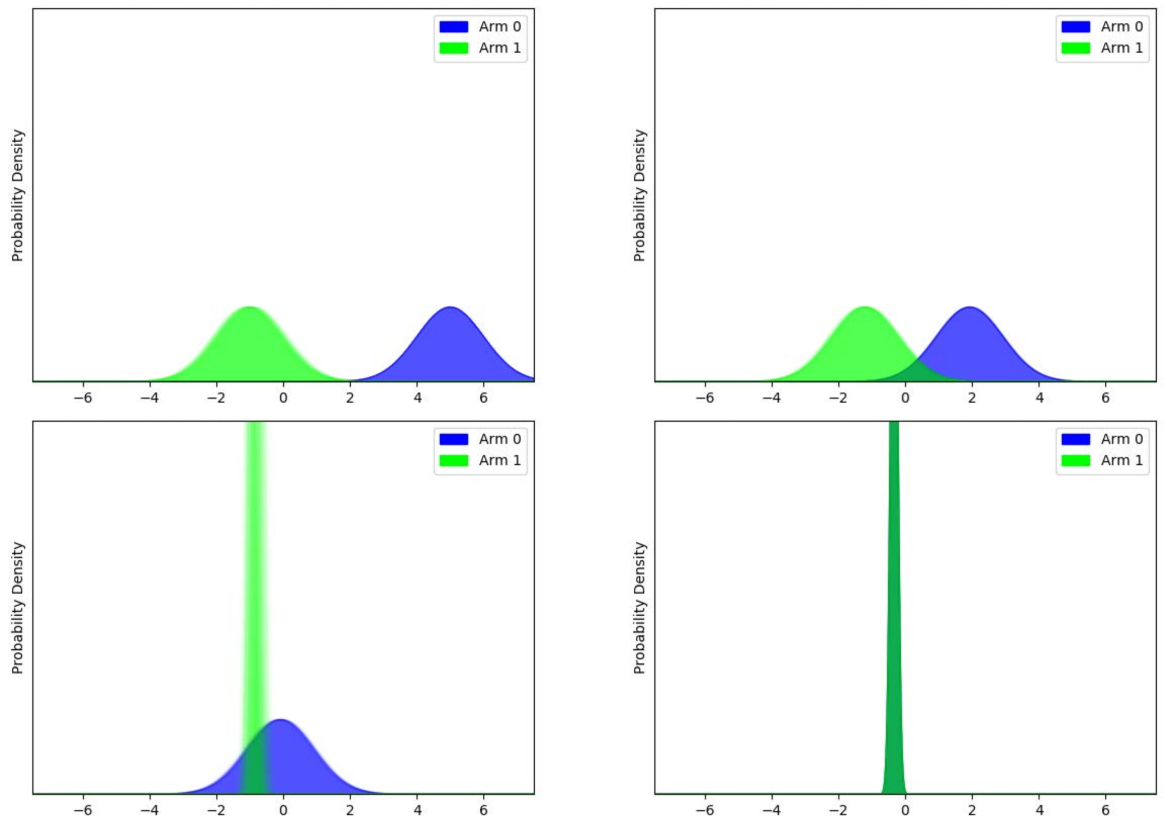


Figure 3.3: Visualizations of four of the adversaries for the 2-arm Gaussian bandit; we present 50 sampled arms for each adversary. Upper figures have a high regret target, lower figures try to decrease the regret. As the difficulty increases, the means get closer and the std. deviation decreases. On the easiest problem, the two arms are entirely overlapping.

can be unstable. In contrast, the D-MALT trained policy performed comparably or better across all of the tests.

In the bandit case, Fig. 3.7 illustrates why RARL does not yield good performance on the transfer tasks in Fig. 3.6. The adversary learns to output one nearly deterministic arm with a low reward and a high-reward arm with maximal standard deviation. On average, the policy has a 50% chance of receiving a reward of -10 on the first step; the optimal policy strategy in response is to randomly sample an arm at the first timestep and then only switch arms if it observes a negative reward. This strategy is clearly sub-optimal for other arm distributions and unsurprisingly performs poorly on the holdout tests. This is a failing of RARL parametrized by a diagonal Gaussian; it can only represent one set of possible dynamics. Our approach overcomes this by filling the space with many possible arm types from easy to hard, leading to a policy that learns a more general identification strategy. Fig. 3.3 shows that the set of arms yielded by the adversaries span a wide range of possible tasks:

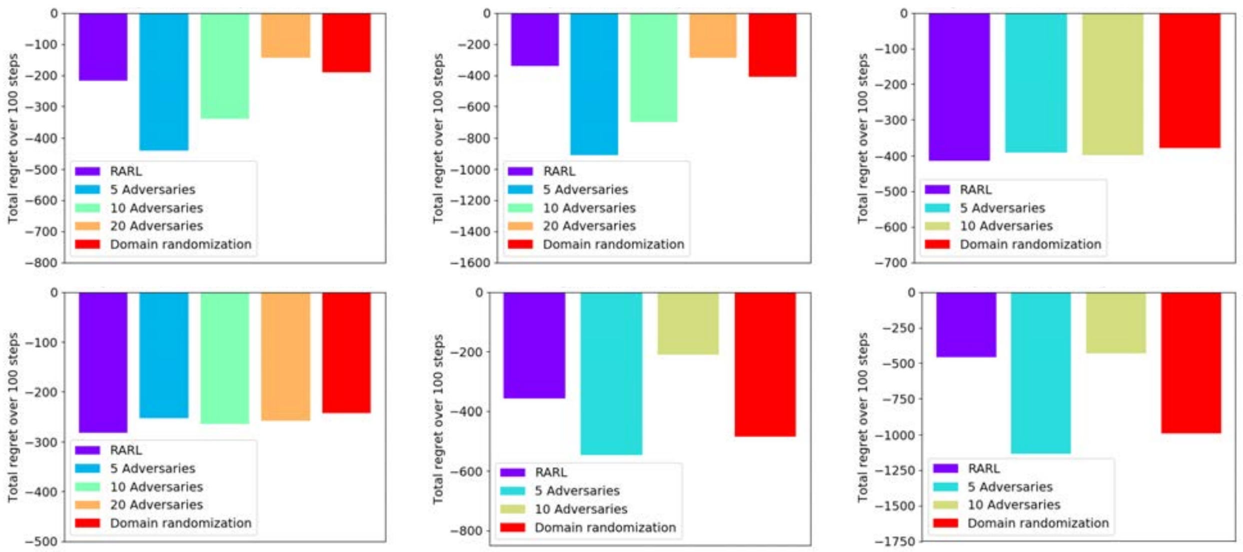


Figure 3.4: Regret results from the linear transfer tests. **Top row:** 2 dimensional system. **Bottom row:** 4 dimensional system. The left column depicts the performance of the policy on the unperturbed, stable system $x_{k+1} = -0.8Ix_k + Iu_k$. The middle column represents the regret on randomly sampled perturbation matrices with matrix entries bounded between $[-\frac{0.4}{\text{dim}}, \frac{0.4}{\text{dim}}]$. The right column represents the regret on sampled perturbation matrices that make the system unstable.

from arms that are well separated and easy to identify to arms that overlap significantly. Unsurprisingly, Fig. 3.6 shows that the D-MALT trained method sharply outperforms the RARL strategy.

Fig. 3.6 illustrates that domain randomization did well across most transfer tests. However, as predicted in Sec. 3.3, it struggled on certain "difficult" transfer tests. For example, our approach significantly outperformed domain randomization on the 5-arm Transfer Test F with means (-5, -5, -5, -5, 5) and standard deviations of 1.0 for each arm. We found that the most challenging adversaries often produced distributions that looked similar to Transfer Test F, giving the learner more opportunities to play against this distribution. As a result, D-MALT consistently outperformed domain randomization on the transfer tests with a sufficient number of adversaries to play against. Thus, given the same compute budget for a policy training with D-MALT vs. training against domain randomization, the policy training with D-MALT will likely produce a more general bandit strategy. This is particularly interesting for papers like [12], which directly use domain randomization to train a bandit strategy that appears to be competitive with UCB1 and Thomson sampling. It is possible that their strategy would not be competitive on instances that were not sampled uniformly.

Surprisingly, RARL training also failed to yield significant improved robustness over no

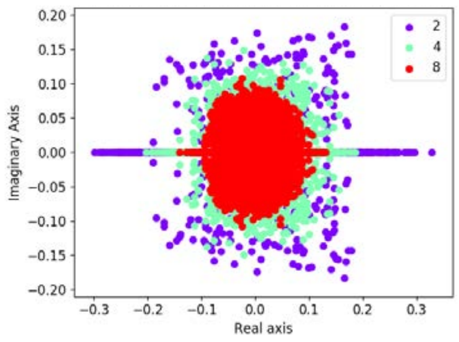


Figure 3.5: Eigenvalues of the perturbation matrices as we increase the dimension d. For each dimension we sample 500 matrices with elements bounded between $\left[\frac{-0.4}{d}, \frac{0.4}{d}\right]$ and plot the eigenvalues. Above dimension 2, the probability of sampling an eigenvalue that makes the uncontrolled system $x_{k+1} = -0.8x_k + \Delta$ unstable approaches 0.

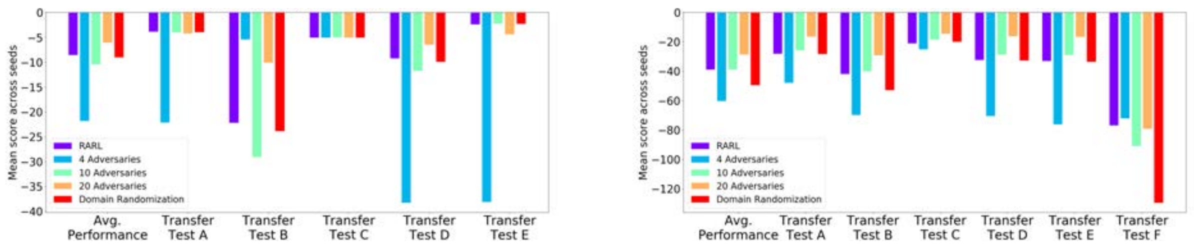


Figure 3.6: Performance of increasing number of adversaries vs. domain randomization on a holdout set of transfer tests. Left graph depicts 2-armed bandit, right depicts 5-armed bandit.

adversary on both the Hopper validation set (Fig. 3.8) and the holdout test set (Fig. 3.9). This is in contrast to the results in [26]. We surmise that the improved robustness of the policy trained against a single adversary in that work is due to their particular parameterization of the adversary; applying friction at the feet. Being robust to this particular adversary parameterization may happen to align very well with the set of transfer tests used in that work. However, the policy trained with D-MALT sharply outperforms the zero adversary case on both the validation and holdout test set.

Our results for D-MALT vs. domain randomization on Mujoco are less conclusive. We do not outperform domain randomization when tested on the same mass and friction grid that domain randomization was allowed to train on. We form our perturbations by adding the adversary actions to the policy actions; this cannot represent all possible dynamical systems. It is possible that the dynamical systems that are found by scaling the mass and friction coefficients of Hopper cannot be represented by the set of dynamical systems that can be

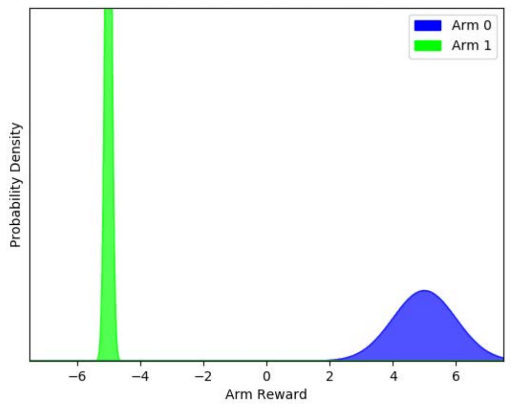


Figure 3.7: The reward distribution for the multi-armed bandit for the single adversary playing the zero-sum game. At the first timestep the policy has a 50% chance of recieving a high regret of -10.

formed by the adversary. However, on the holdout test set, we perform competitively with domain randomization: we outperform it on 5 tasks and underperform it on 3 tasks.

3.6 Conclusion and Future Work

In this work we demonstrated that a diverse set of adversaries can effectively train robust controllers that match and sometimes exceed those trained by domain randomization. Using two simple examples, a linear system and a bandit, we demonstrated that the standard technique of domain randomization can fail to yield risky dynamical systems that are easily identified by learned adversaries. We showed that the diverse adversaries can easily be learned using a simple reward function that rewards them for pushing the policy to a single reward target. Finally, using a Mujoco example, we demonstrated that this approach extended outside of our simple examples and can be competitive with domain randomization on tasks that were not hand-picked to illustrate failures of domain randomization. In future work we are interested in exploring the application of this approach to a wider range of environments.

Additionally, in this work we primarily focused on transfer in simulated tasks. In [1] the authors found that a learned set of adversaries did not help with transfer for a Rubik's cube manipulation task; we will conduct follow-up work to see if our approach to adversary training helped with an actual robotics task.

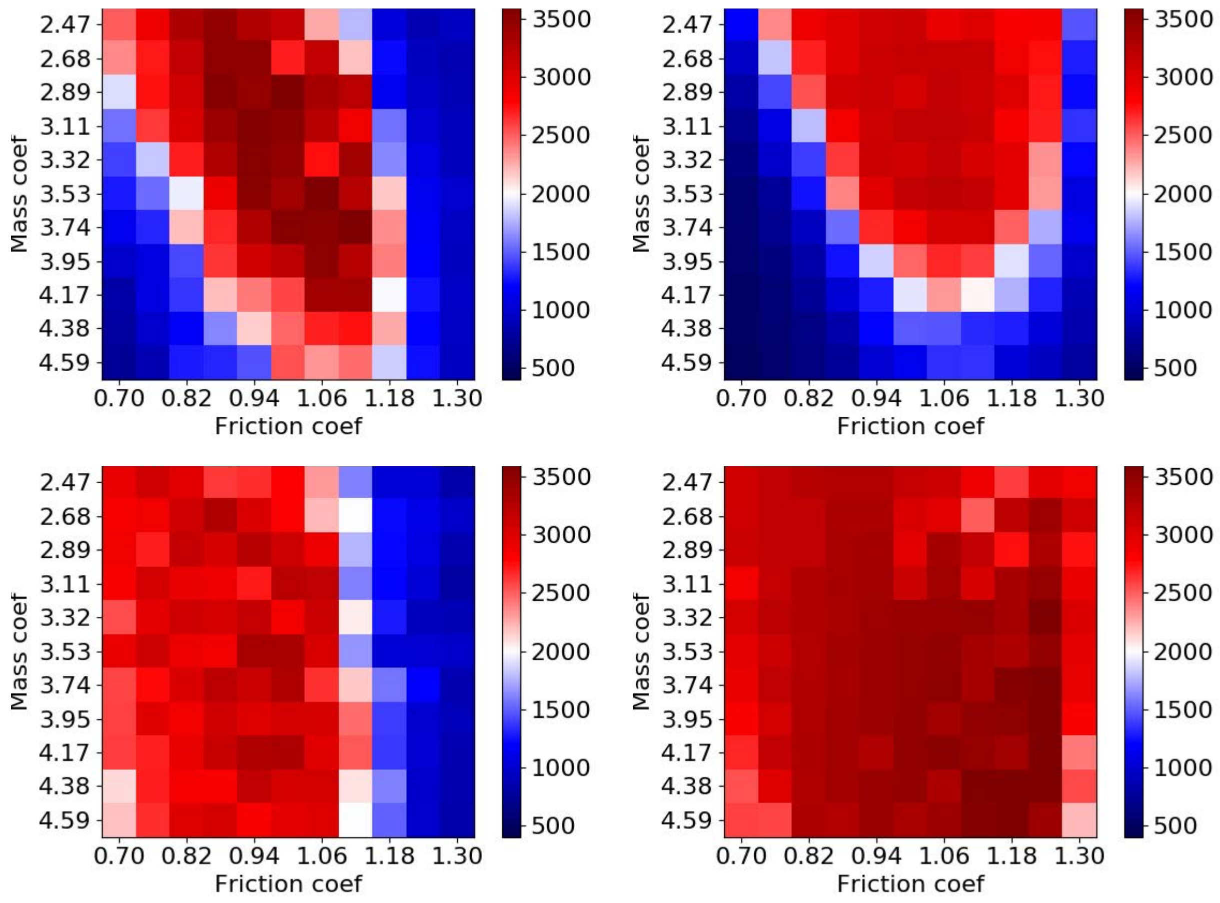


Figure 3.8: Comparison of the effects of different numbers of adversaries and domain randomization on a swept grid of friction and mass scaling coefficients for the best hyper-parameter for each training type. **Upper left:** 0 adversaries. **Upper right:** RARL. **Lower left:** 5 adversaries. **Bottom right:** Domain randomization.

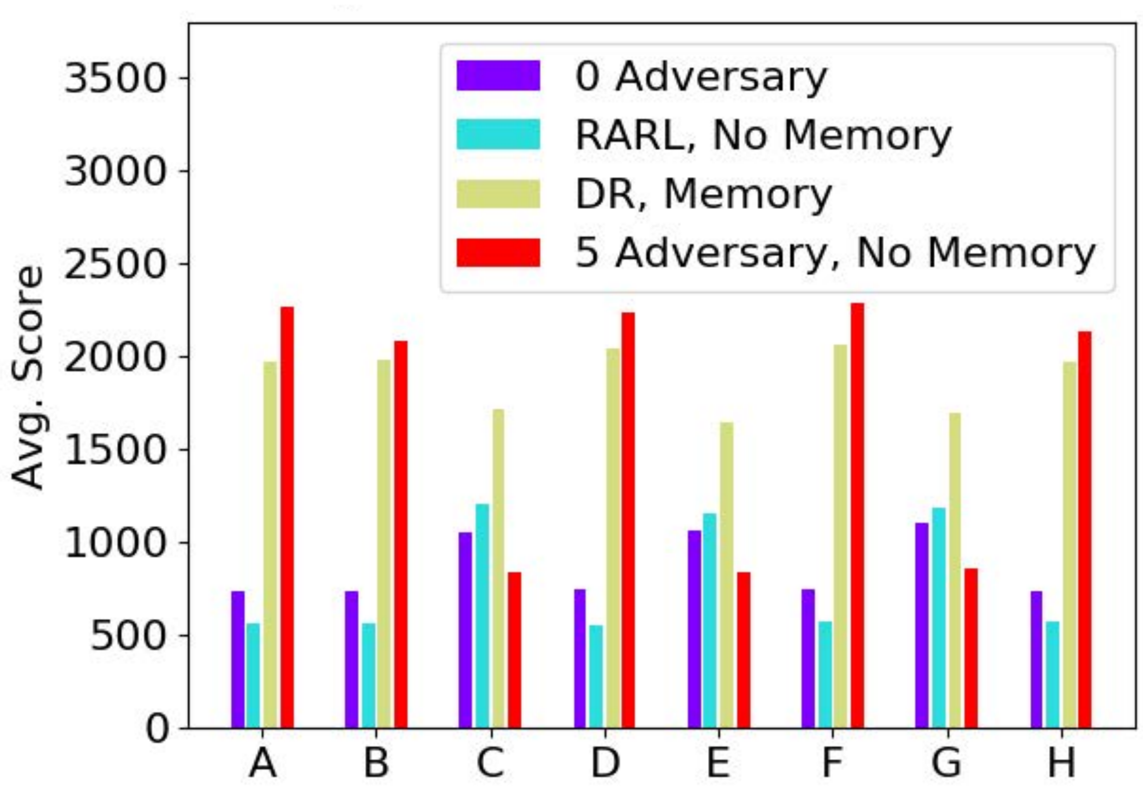


Figure 3.9: Best performing hyperparameters of different numbers of adversaries and domain randomization on the holdout tests, averaged across 10 seeds. The x-axis contains the labels of the corresponding tests described in the supplementary section. Bars labelled with memory correspond to 10 observations stacked together as the state.

Chapter 4

Assessing Controller Effectiveness in Flow

4.1 Introduction

In this section, we evaluate a flow smoothing controller's ability to reduce the energy consumption of all vehicles on a stretch of highway. Before being able to deploy this controller, appropriate transfer tests should evaluate whether the energy gains attained in the baseline setting are still applicable in a range of similar environments.

We outline the analysis around two of these transfer tests: penetration rate of AVs and increased aggressive driving by humans. Though we use a naive, hand-designed controller rather than a trained RL policy, the software components used to evaluate this controller are analogous with those used to evaluate RL controllers. As more sophisticated controllers are trained, this contribution to Flow allows researchers easily design analyze their robustness to a variety of perturbations.

Furthermore, when deploying these controllers, researchers must be aware of a number of other important system-level metrics such as total travel time and average velocity. We report empirical measured values for the various penetration rates tested, and note their connection to induced demand.

4.2 Experiment Scenario

We choose to analyze the *FollowerStopper* controller, designed by [32]. This controller aims to maintain a pre-tuned target speed, even if a larger gap opens up in front of it. By not "filling-the-gap," the controller uses the headway opened up in front of it as a damper to reduce wave instability. Note, this is not a controller trained with RL; however, this controller captures many of the important characteristics of general flow smoothing controllers, namely, the large headway of the flow smoothing vehicle and periods of time in which the flow smoothing vehicle is travelling slower than its neighbors.

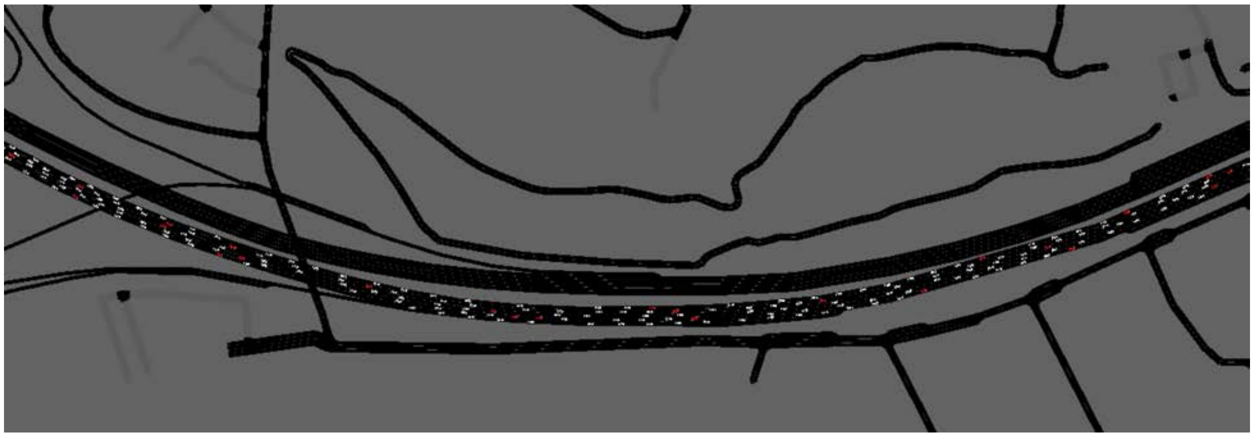


Figure 4.1: Screenshot of the simulated stretch of the I-210 in SUMO. Red cars represent AVs.

Using a calibrated model of the I-210, we simulate about five minutes of traffic flow across the highway using SUMO. The simulation is initialized with free-flowing traffic. Figure 4.1 shows the simulated highway soon after initialization.

We use the fuel consumption models built into SUMO to estimate the fuel usage for each of the vehicles on the highway. Specifically, we report both the total fuel usage of the vehicles and compute the average instantaneous MPG of a vehicle throughout its duration in the simulation.

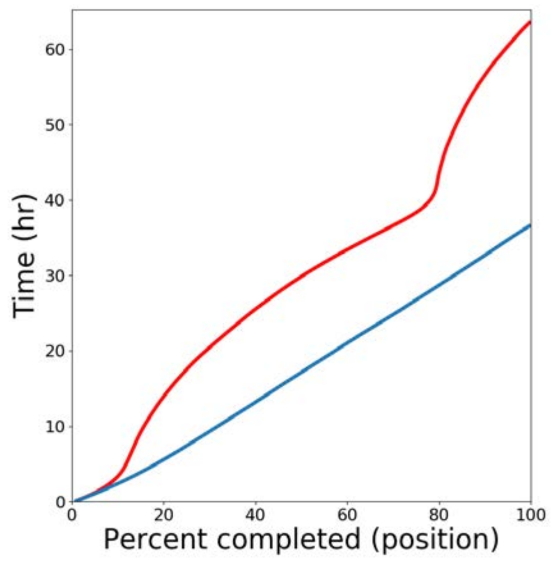
4.3 Empirical Results

In this section we detail the results of the flow smoothing experiments.

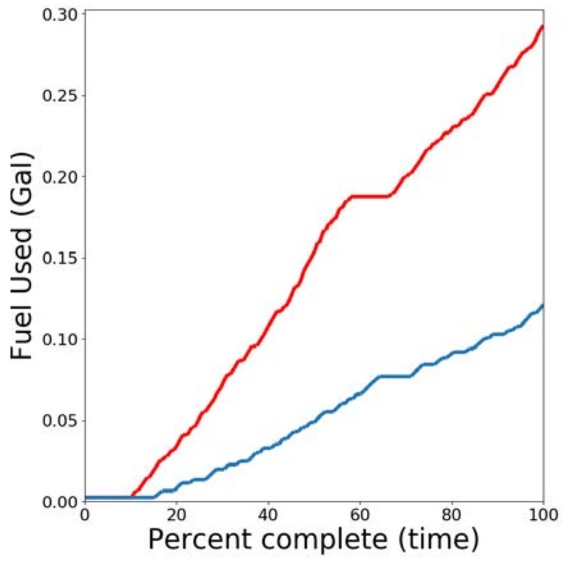
Effectiveness of Flow Smoothing

We find that deploying flow smoothing vehicles onto the I-210 yields savings in both time and fuel usage. Figure 4.2 shows the traces of vehicles in two different scenarios: a baseline scenario without flow smoothing, and a scenario with flow smoothing vehicles present. Notably, the vehicle in the flow smoothing scenario traverses the stretch of highway in less time, as it does not have to pass through a the traffic wave and slow down. Furthermore, the vehicle uses less fuel, as it does not have to decelerate and accelerate through the wave.

To illustrate the effects of flow smoothing in aggregate, Table 4.1 reports the average MPG of all non-AV simulated cars as a function of the penetration rate of flow-smoothing AVs. As expected, increasing the penetration rate of flow smoothing vehicles dramatically increases the average fuel economy of the human drivers, by almost a factor of 2. This



(a) Space—Time plot for two vehicles. Note the waves the red vehicle passes through 60% of the way through.



(b) Fuel usage plot for two vehicles.

Figure 4.2: Time and fuel savings for two randomly selected vehicles, **in a baseline scenario**, and **with flow smoothing present.**

Penetration	0%	10%	20%	30%
MPG	19 ± 2	25 ± 2	34 ± 4	36 ± 0

Table 4.1: Fuel economy achieved by the simulated human drivers as a function of the penetration rate of flow-smoothing AVs.

doubling of fuel economy is broadly in line with empirical results of [32], who found that fuel consumption decreased by 40% with flow smoothing present.

Sensitivity to Aggressive Human Drivers

There are a number of potential mismatches between realistic human driving behavior and the models used in SUMO; we chose to analyze the effect of increasing the aggressiveness of human drivers when making lane changes. Because flow smoothing controllers aim to open large headway as a method of damping waves, we suspected that vehicles that lane change more frequently and into smaller gaps may reduce the effectiveness of the flow smoothing controller.

We designed an "aggressive" human driver model with the same car-following controller as the other humans in the simulation but design a lane changing controller that encourages them to change lanes more aggressively. We then ran experiments to measure the MPG of the non-AVs as above, varying the penetration rate of flow smoothing vehicles and aggressive drivers.

Figure 4.3 shows the results from this two-dimensional sensitivity analysis. Error bars listed represent one standard deviation around the mean MPG value obtained over 12 runs with different random seeds. The left-most column of the figure demonstrates the effect of introducing flow smoothing when no aggressive drivers are present. As the human drivers make increasingly aggressive lane changes, the positive effects of flow smoothing are attenuated. At 50% aggressive driver penetration, the benefits of flow smoothing are almost entirely eliminated.

These results highlight a central challenge of designing flow smoothing controllers: traffic microsimulations have accurate enough human driver models to reproduce macroscopic traffic phenomena such as traffic waves, but the effectiveness of flow smoothing depends on the microscopic movements of individual cars (in this case, lane changes). Modeling these microscopic movements based on real data is a challenging and unsolved problem. To our knowledge, we are the first to examine how the effectiveness of flow smoothing degrades as a function of driver behavior, and we argue that sensitivity analyses like these are crucial for the successful development of flow smoothing controllers. Before we can be confident that flow smoothing microsimulations accurately reflect the effects that flow smoothing would have on a real road, it will be necessary to measure how human drivers actually behave in a deployment region, and simulate whether drivers like those will hinder flow smoothers.

Travel Time and System Average Velocity

Improved travel time or average speeds are of particular interest to researchers, as the increase in demand caused by improved travel times or average speed could cause induce demand, defined as additional trips that are created because the system is now more desirable to travelers. The resulting increase in travel could offset the fuel savings gains generated by flow smoothing vehicles.

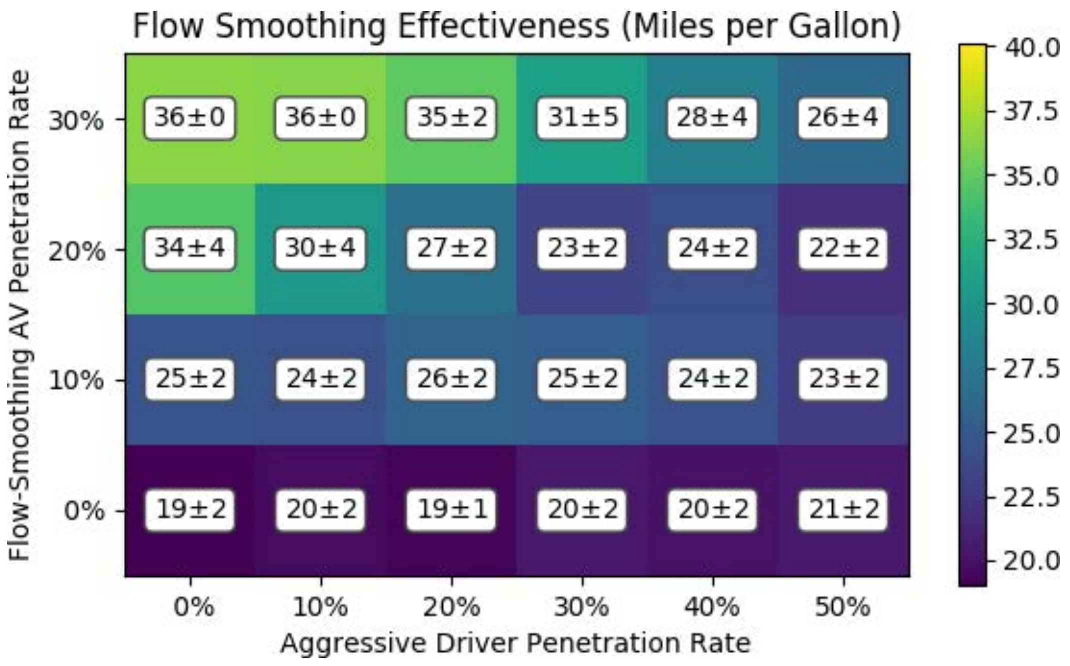
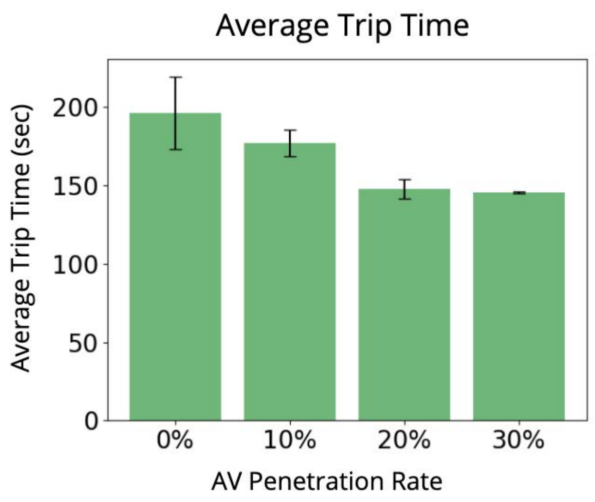


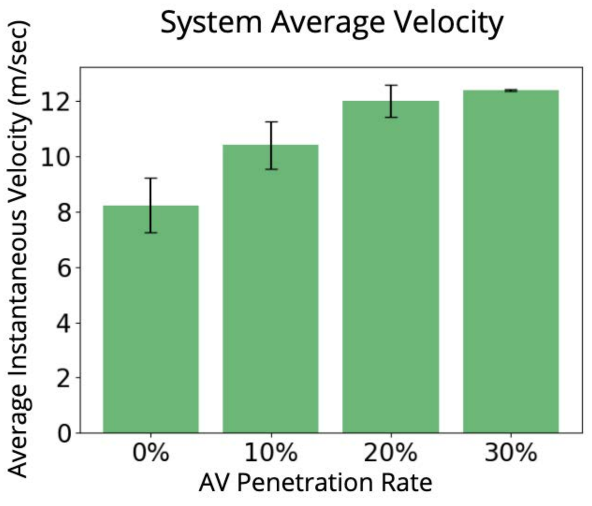
Figure 4.3: Average fuel economy achieved by simulated human drivers as a function of flow smoothing penetration rate and aggressive driver penetration rate. Each cell lists the average and standard deviation of the distribution of MPG obtained when running the simulations with 12 different random seeds.

We therefore present both the system trip time improvement and average speed improvement in Figure 4.4. In our simulations, we find a nearly 25% decrease in trip time and a 50% increase in average speeds as the penetration rate of flow smoothing vehicles increases to 30%.

In order to estimate the potential for induced demand, we refer to [8], who comprehensively model short- and long-term induced demand effects. These authors find that, in the long term, "every 10% increase in travel speeds is associated with a 6.4% increase in VMT" [8]. Applying this number directly to flow smoothing is difficult, since the projects used to obtain this elasticity figure were all highway widening projects. However, without prior flow smoothing deployments to refer to, this is likely as accurate an estimate of induced demand as we can feasibly make. In our case, average travel speeds increase by 50% for 30% flow smoother penetration. Referring to [8], this suggests that we would see an increase in VMT of about 30%, far less than the 100% increase in VMT that would be needed to offset the fuel savings at 30% flow smoothing penetration.



(a) Average trip time vs. penetration of flow smoothers. Flow smoothing reduces travel times due to smoother flow of traffic.



(b) Average speed vs. penetration of flow smoothing analysis. Vehicles move faster when flow smoothing is deployed.

Figure 4.4: When flow smoothing is deployed, average trip times through the deployment area decrease, and average speed of all vehicles increases. These phenomena suggest that flow smoothing will cause induced demand.

4.4 Conclusion and Future Work

We have constructed a set of transfer tests to assess the effectiveness of the flow smoothing controller under varying penetration rates and aggressive driver percentages. The changes to Flow that enabled these transfer tests can be used to design a number of different scenarios to assess a controllers effectiveness in other regimes.

In D-MALT, we perform a train-validate-test analysis to choose hyperparameters and algorithms. We anticipate that this contribution will be applied in a similar fashion to gauge the effectiveness and choose hyperparameters for various training strategies.

It would be further interesting to apply the adversarial behavior directly to the behavior of the human vehicles in the simulation. This formulation is very enticing, as traditional methods of generating robustness, such as domain randomization, struggle to characterize the range of potential driving behaviors of a human. Rather, by allowing the adversaries to learn or modifying existing human driving policies, we can potentially generate robustness in the trained controllers without explicitly characterizing a range of behaviors.

Bibliography

[1] Ilge Akkaya et al. "Solving Rubik's Cube with a Robot Hand". In: *arXiv preprint arXiv:1910.07113* (2019).

[2] OpenAI: Marcin Andrychowicz et al. "Learning dexterous in-hand manipulation". In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20.

[3] Reda Bahi Slaoui et al. "Robust Domain Randomization for Reinforcement Learning". In: *arXiv preprint arXiv:1910.10537* (2019).

[4] Michael Behrisch et al. "SUMO–simulation of urban mobility: an overview". In: *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind. 2011.

[5] Richard Bellman. "A Markovian decision process". In: *Journal of Mathematics and Mechanics* (1957), pp. 679–684.

[6] Konstantinos Bousmalis et al. "Using simulation and domain adaptation to improve efficiency of deep robotic grasping". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 4243–4250.

[7] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[8] Robert Cervero. "Road expansion, urban growth, and induced travel: A path analysis". In: *Journal of the American Planning Association* 69.2 (2003), pp. 145–163.

[9] Yevgen Chebotar et al. "Closing the sim-to-real loop: Adapting simulation randomization with real world experience". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8973–8979.

[10] Paul Christiano et al. "Transfer from simulation to real world through learning deep inverse dynamics model". In: *arXiv preprint arXiv:1610.03518* (2016).

[11] Mark Cutler and Jonathan P How. "Efficient reinforcement learning for robots using informative simulated priors". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 2605–2612.

[12] Yan Duan et al. "RL2: Fast Reinforcement Learning via Slow Reinforcement Learning". In: *arXiv preprint arXiv:1611.02779* (2016).

[13] Jakob Foerster et al. “Learning with opponent-learning awareness”. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2018, pp. 122–130.

[14] Semyon Aranovich Gershgorin. “Uber die abgrenzung der eigenwerte einer matrix”. In: 6 (1931), pp. 749–754.

[15] Stephen James et al. “Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12627–12637.

[16] Kathy Jang et al. “Simulation to scaled city: zero-shot policy transfer for traffic control via autonomous vehicles”. In: *CoRR* abs/1812.06120 (2018). arXiv: `1812.06120`. URL: `http://arxiv.org/abs/1812.06120`.

[17] Victoria Krakovna. *Specification gaming examples in AI*. 2018. URL: `https://vkrakovna.wordpress.com/2018/04/02/specification-gaming-examples-in-ai/`.

[18] Eric Liang et al. “RLlib: Abstractions for distributed reinforcement learning”. In: *arXiv preprint arXiv:1712.09381* (2017).

[19] Eric Mazumdar and Lillian J Ratliff. “On the convergence of gradient-based learning in continuous games”. In: *arXiv preprint arXiv:1804.05464* (2018).

[20] Bhairav Mehta et al. “Active Domain Randomization”. In: *arXiv preprint arXiv:1904.04762* (2019).

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

[22] Igor Mordatch et al. “Combining model-based policy search with online model learning for control of physical humanoids”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 242–248.

[23] Anusha Nagabandi et al. “Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning”. In: *CoRR* abs/1708.02596 (2017). arXiv: `1708.02596`. URL: `http://arxiv.org/abs/1708.02596`.

[24] OpenAI. *OpenAI Five*. https://blog.openai.com/openai-five/. 2018.

[25] Xue Bin Peng et al. “Sim-to-real transfer of robotic control with dynamics randomization”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–8.

[26] Lerrel Pinto et al. “Robust adversarial reinforcement learning”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2817–2826.

[27] Aravind Rajeswaran et al. “Towards generalization and simplicity in continuous control”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 6550–6561.

[28] Fereshteh Sadeghi and Sergey Levine. "Cad2rl: Real single-image flight without a single real image". In: *arXiv preprint arXiv:1611.04201* (2016).

[29] John Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

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

[31] Hiroaki Shioya, Yusuke Iwasawa, and Yutaka Matsuo. "Extending robust adversarial reinforcement learning considering adaptation and diversity". In: (2018).

[32] 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.

[33] Jie Tan et al. "Sim-to-real: Learning agile locomotion for quadruped robots". In: *arXiv preprint arXiv:1804.10332* (2018).

[34] Josh Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 23–30.

[35] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.

[36] Eugene Vinitsky et al. "Benchmarks for reinforcement learning in mixed-autonomy traffic". In: *Proceedings of The 2nd Conference on Robot Learning*. Ed. by Aude Billard et al. Vol. 87. Proceedings of Machine Learning Research. PMLR, 2018, pp. 399–409. URL: `http://proceedings.mlr.press/v87/vinitsky18a.html`.

[37] Cathy Wu et al. "Flow: Architecture and benchmarking for reinforcement learning in traffic control". In: *arXiv preprint arXiv:1710.05465* (2017).