Agile Autonomous Driving using End-to-End Deep Imitation Learning

Yunpeng Pan*‡, Ching-An Cheng*, Kamil Saigol*, Keuntaek Lee†, Xinyan Yan*,
Evangelos A. Theodorou*, and Byron Boots*
*Institute for Robotics and Intelligent Machines, †School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, Georgia 30332–0250
‡JD.com American Technologies Corporation, Mountain View, California 94043
{ypan37,cacheng,kamilsaigol,keuntaek.lee,xyan43}@gatech.edu
evangelos.theodorou@gatech.edu, bboots@cc.gatech.edu

***Abstract*—We present an end-to-end imitation learning system for agile, off-road autonomous driving using only low-cost on-board sensors. By imitating a model predictive controller equipped with advanced sensors, we train a deep neural network control policy to map raw, high-dimensional observations to continuous steering and throttle commands. Compared with recent approaches to similar tasks, our method requires neither state estimation nor on-the-fly planning to navigate the vehicle. Our approach relies on, and experimentally validates, recent imitation learning theory. Empirically, we show that policies trained with online imitation learning overcome well-known challenges related to covariate shift and generalize better than policies trained with batch imitation learning. Built on these insights, our autonomous driving system demonstrates successful high-speed off-road driving, matching the state-of-the-art performance.**

I. Introduction

High-speed autonomous off-road driving is a challenging robotics problem [17, 30, 31] (Fig. 1). To succeed in this task, a robot is required to perform both precise steering and throttle maneuvers in a physically-complex, uncertain environment by executing a series of high-frequency decisions. Compared with most previously studied autonomous driving tasks, the robot here must reason about minimally-structured, stochastic natural environments and operate at high speed. Consequently, designing a control policy by following the traditional model-plan-then-act approach [17, 20] becomes challenging, as it is difficult to adequately characterize the robot's interaction with the environment *a priori*.

This task has been considered previously, for example, by Williams et al. [30, 31] using model-predictive control (MPC). While the authors demonstrate impressive results, their internal control scheme relies on expensive and accurate Global Positioning System (GPS) and Inertial Measurement Unit (IMU) for state estimation and demands high-frequency online replanning for generating control commands. Due to these costly hardware requirements, their robot can only operate in a rather controlled environment, which limits the applicability of their approach.

We aim to relax these requirements by designing a reflexive driving policy that uses only *low-cost, on-board* sensors (e.g. monocular camera, wheel speed sensors). Building on the success of deep reinforcement learning (RL) [15, 18],

Fig. 1: The high-speed off-road driving task.

we adopt deep neural networks (DNNs) to parametrize the control policy and learn the desired parameters from the robot's interaction with its environment. While the use of DNNs as policy representations for RL is not uncommon, in contrast to most previous work that showcases RL in simulated environments [18], our agent is a high-speed physical system that incurs real-world cost: collecting data is a cumbersome process, and a single poor decision can physically impair the robot and result in weeks of time lost while replacing parts and repairing the platform. Therefore, direct application of model-free RL techniques is not only sample inefficient, but costly and dangerous in our experiments.

These real-world factors motivate us to adopt *imitation learning* (IL) [23] to optimize the control policy instead. A major benefit of using IL is that we can leverage domain knowledge through *expert* demonstrations. This is particularly convenient, for example, when there already exists an autonomous driving platform built through classic system engineering principles. While such a system (e.g. [30]) usually requires expensive sensors and dedicated computational resources, with IL we can train a lower-cost robot to behave similarly, without carrying the expert's hardware burdens over to the learner. Here we assume the expert is given as a black box oracle that can provide the desired actions when queried, as opposed to the case considered in [9] where the expert can be modified to accommodate the learning progress.

In this work, we present an IL system for real-world high-speed off-road driving tasks. By leveraging demonstrations

TABLE I: Comparison of our method to prior work on IL for autonomous driving

Methods	Tasks	Observations	Action	Algorithm	Expert	Experiment
[1]	On-road low-speed	Single image	Steering	Batch	Human	Real &simulated
[23]	On-road low-speed	Single image & laser	Steering	Batch	Human	Real &simulated
[24]	On-road low-speed	Single image	Steering	Batch	Human	Simulated
[19]	Off-road low-speed	Left & right images	Steering	Batch	Human	Real
[32]	On-road unknown speed	Single image	Steering + break	Online	Pre-specified policy	Simulated
Our Method	Off-road high-speed	Single image + wheel speeds	Steering + throttle	Batch & online	Model predictive controller	Real & simulated

from an algorithmic expert, our system can learn a driving policy that achieves similar performance compared to the expert. The system was implemented on a 1/5-scale autonomous AutoRally car. In real-world experiments, we show the AutoRally car—without any state estimator or online planning, but with a DNN policy that directly inputs measurements from a low-cost monocular camera and wheel speed sensors—could learn to perform high-speed driving at an average speed of $\sim$6 m/s and a top speed of $\sim$8 m/s (equivalently 108 km/h and 144 km/h on a full-scale car), matching the state-of-the-art [31].

II. Related Work

End-to-end learning for self-driving cars has been explored since the late 1980s. The Autonomous Land Vehicle in a Neural Network (ALVINN) [23] was developed to learn steering angles directly from camera and laser range measurements using a neural network with a single hidden layer. Based on similar ideas, modern self-driving cars [19, 1, 24] have recently started to employ a batch IL approach: with DNN control policies, these systems require only expert demonstrations during the training phase and on-board measurements during the testing phase. For example, Nvidia's PilotNet [1, 2], a convolutional neural network that outputs steering angle given an image, was trained to mimic human drivers' reactions to visual input with demonstrations collected in real-world road tests.

Our problem differs substantially from these previous on-road driving tasks. We study autonomous driving on a fixed set of dirt tracks, whereas on-road driving must perform well in a larger domain and contend with moving objects such as cars and pedestrians. While on-road driving in urban environments may seem more difficult, our agent must overcome challenges of a different nature. It is required to drive at high speed, on dirt tracks, the surface of which is constantly evolving and highly stochastic. As a result, high-frequency application of both steering and throttle commands are required in our task, whereas previous work only focuses on steering commands [19, 2, 24]. A Dataset Aggregation (DAgger) [25] related online IL algorithm for autonomous driving was recently demonstrated in [32], but only considered simulated environments. A comparison of IL approaches to autonomous driving is presented in Table I.

Our task is similar to the task considered by Williams et al. [30, 31] and Drews et al. [5]. Compared with a DNN policy, their MPC approach has several drawbacks: computationally expensive optimization for planning is required to be performed online at high-frequency, which becomes repetitive for navigating the vehicle on a track after a few laps. In [30, 31], accurate GPS and IMU feedbacks are also required for state estimation, which may not contain sufficient information to contend with the changing environment in off-road driving tasks. While the requirement on GPS and IMU is relaxed by using a vision-based cost map in [5], a large dataset (300,000 images) was used to train the model, expensive on-the-fly planning is still required, and speed performance is compromised. In contrast to previous work, our approach offloads the hardware requirements to an expert. While the expert may use high-quality sensors and more computational power, our agent only needs access to cheap sensors and its control policy can run reactively in high frequency, without on-the-fly planning. Additionally, our experimental results match those in [30], and are faster and more data efficient than that in [5].

III. Imitation Learning for Autonomous Driving

In this section, we give a concise introduction to IL, and discuss the strengths and weakness of deploying a batch or an online IL algorithm to our task. Our presentation is motivated by the realization that the connection between online IL and DAgger-like algorithms [25] has not been formally introduced in continuous domains. To our knowledge, DAgger has only been used heuristically in these domains [26, 32]. Here we simplify the derivation of [25] and extend it to continuous action spaces as required in the autonomous driving task.

A. Problem Definition

To mathematically formulate the autonomous driving task, we consider a discrete-time continuous-valued RL problem. Let $\mathbb{S}$, $\mathbb{A}$, and $\mathbb{O}$ be the state, action, and the observation spaces. In our setting, the state space is unknown to the agent; observations consist of on-board measurements, including a monocular RGB image from the front-view camera and wheel speeds from Hall effect sensors; actions consist of continuous-valued steering and throttle commands.

The goal is to find a stationary, reactive policy[1] $\pi : \mathbb{O} \mapsto \mathbb{A}$ (e.g. a DNN policy) such that π achieves low accumulated cost over a finite horizon of length T,

$$\min_\pi J(\pi), \quad J(\pi) \coloneqq \mathbb{E}_{\rho_\pi}\left[\sum_{t=0}^{T-1} c(s_t, a_t)\right], \qquad (1)$$

in which $s_t \in \mathbb{S}$, $o_t \in \mathbb{O}$, $a_t \in \mathbb{A}$, and ρ_π is the distribution of trajectory $(s_0, o_0, a_0, s_1, \ldots, a_{T-1})$ under policy π. Here c is the instantaneous cost, which, e.g., encourages high speed driving while staying on the track. For notation: given a policy π, we denote π_o as the distribution of actions given observation

[1]While we focus on reactive policies in this section, the same derivations apply to history-dependent policies.

o, and $a = \pi(o)$ as the (stochastic) action taken by the policy. We denote $Q^t_\pi(s,a)$ as the Q-function at state s and time t, and $V^t_\pi(s) = \mathbb{E}_{a\sim\pi_s}[Q^t_\pi(s,a)]$ as its associated value function, where $\mathbb{E}_{a\sim\pi_s}$ is a shorthand of $\mathbb{E}_{o|s}\mathbb{E}_{a\sim\pi_o}$ denoting the expectation of the action marginal given state s.

B. Imitation Learning

Directly optimizing (1) is challenging for high-speed off-road autonomous driving. Since our task involves a physical robot, model-free RL techniques are intolerably sample inefficient and have the risk of permanently damaging the car when applying a partially-optimized policy in exploration. Although model-based RL may require fewer samples, it can lead to suboptimal, potentially unstable results, because it is difficult for a model that uses only on-board measurements to fully capture the complex dynamics of off-road driving.

Considering these limitations, we propose to solve for policy π by IL. We assume the access to an oracle policy or *expert* π^* to generate demonstrations during the training phase. This expert can rely on resources that are unavailable in the testing phase, like additional sensors and computation. For example, the expert can be a computationally intensive optimal controller that relies on exteroceptive sensors (e.g. GPS for state estimation), or an experienced human driver.

The goal of IL is to perform as well as the expert with an error that has at most linear dependency on T. Formally, we introduce a lemma due to Kakade and Langford [10] and define what we mean by an *expert*.

Lemma 1. *Define $d_\pi(s,t) = \frac{1}{T}d^t_\pi(s)$ as a generalized stationary time-state distribution, where d^t_π is the distribution of state at time t when running policy π. Let π and π' be two policies. Then*

$$J(\pi) = J(\pi') + \mathbb{E}_{s,t\sim d_\pi}\mathbb{E}_{a\sim\pi_s}[A^t_{\pi'}(s,a)] \tag{2}$$

where $A^t_{\pi'}(s,a) = Q^t_{\pi'}(s,a) - V^t_{\pi'}(s)$ is the (dis)advantage function at time t with respect to running π'.

Definition 1. A policy π^* is called an *expert* to problem (1) if $C_{\pi^*} = \sup_{t\in[0,T-1],s\in\mathbb{S}} \mathrm{Lip}\left(Q^t_{\pi^*}(s,\cdot)\right) \in O(1)$ independent of T, where $\mathrm{Lip}(f(\cdot))$ denotes the Lipschitz constant of function f and $Q^t_{\pi^*}$ is the Q-function at time t of running policy π^*.

Because $Q^t_{\pi^*}(s,a)$ is the accumulated cost of taking some action a at time t and then executing the expert policy π^* afterwards, the idea behind Definition 1 is that a reasonable expert policy π^* should perform stably under arbitrary action perturbation, regardless of where it starts.[2] As we will see in Section III-C, this requirement provides guidance for whether to choose batch learning vs. online learning to train a policy by imitation.

[2]We define the expert here using an uniform Lipschitz constant because the action space in our task is continuous; for discrete action spaces, $\mathrm{Lip}\left(Q^t_{\pi^*}(s,\cdot)\right)$ can be replaced by $\sup_{a\in\mathbb{A}} Q^t_{\pi^*}(s,a)$ and the rest applies.

1) Online Imitation Learning: We now present the objective function for the online learning [27] approach to IL. Assume π^* is an expert to (1) and suppose $\mathbb{A}$ is a normed space with norm $\|\cdot\|$. Let $D_W(\cdot,\cdot)$ denote the Wasserstein metric [7]: for two probability distributions p and q defined on a metric space $\mathcal{M}$ with metric d,

$$D_W(p,q) \coloneqq \sup_{f:\mathrm{Lip}(f(\cdot))\le 1} \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)] \tag{3}$$

$$= \inf_{\gamma\in\Gamma(p,q)} \int_{\mathcal{M}\times\mathcal{M}} d(x,y)d\gamma(x,y), \tag{4}$$

where Γ denotes the family of distributions whose marginals are p and q. It can be shown by the Kantorovich-Rubinstein theorem that the above two definitions are equivalent [7]. These assumptions allow us to construct a surrogate problem, which is relatively easier to solve than (1). We achieve this by upper-bounding the difference between the performance of π and π' given in Lemma 1:

$$\begin{aligned}
&J(\pi) - J(\pi^*)\\
&= \mathbb{E}_{s,t\sim d_\pi}\left[\mathbb{E}_{a\sim\pi_s}[Q^t_{\pi^*}(s,a)] - \mathbb{E}_{a^*\sim\pi^*_s}[Q^t_{\pi^*}(s,a^*)]\right]\\
&\le C_{\pi^*}\mathbb{E}_{s,t\sim d_\pi}\left[D_W(\pi,\pi^*)\right]\\
&\le C_{\pi^*}\mathbb{E}_{s,t\sim d_\pi}\mathbb{E}_{a\sim\pi_s}\mathbb{E}_{a^*\sim\pi^*_s}[\|a-a^*\|],
\end{aligned} \tag{5}$$

where we invoke the definition of advantage function $A^t_{\pi^*}(s,a) = Q^t_{\pi^*}(s,a) - \mathbb{E}_{a^*\sim\pi^*_s}[Q^t_{\pi^*}(s,a^*)]$, and the first and the second inequalities are due to (3) and (4), respectively.

Define $\hat{c}(s,a) = \mathbb{E}_{a^*\sim\pi^*_s}[\|a-a^*\|]$. Thus, to make π perform as well as π^*, we can minimize the upper bound, which is equivalent to solving a surrogate RL problem

$$\min_\pi \mathbb{E}_{\rho_\pi}\left[\sum_{t=1}^T \hat{c}(s_t,a_t)\right]. \tag{6}$$

The problem in (6) is called the *online* IL problem. This surrogate problem is comparatively more structured than the original RL problem (1) [4], so we can adopt algorithms with provable performance guarantees. In this paper, we use the meta-algorithm DAgger [25], which reduces (6) to a sequence of supervised learning problems: Let $\mathcal{D}$ be the training data. DAgger initializes $\mathcal{D}$ with samples gathered by running π^*. Then, in the ith iteration, it trains π_i by supervised learning,

$$\pi_i = \arg\min_\pi \mathbb{E}_{\mathcal{D}}[\hat{c}(s_t,a_t)], \tag{7}$$

where subscript $\mathcal{D}$ denotes empirical data distribution. Next it runs π_i to collect more data, which is then added into $\mathcal{D}$ to train π_{i+1}. The procedure is repeated for $O(T)$ iterations and the best policy, in terms of (6), is returned. Suppose the policy is linearly parametrized. Since our instantaneous cost $\hat{c}(s_t,\cdot)$ is strongly convex, the theoretical analysis of DAgger applies. Therefore, together with the assumption that π^* is an expert, running DAgger to solve (6) finds a policy π with performance $J(\pi) \le J(\pi^*) + O(T)$, achieving our initial goal.

We note here the instantaneous cost $\hat{c}(s_t,\cdot)$ can be selected to be any suitable norm according the problem's property. In our off-road autonomous driving task, we find l_1-norm is preferable (e.g. over l_2-norm) for its ability to filter outliers in a highly stochastic environment.

2) Batch Imitation Learning: By swapping the order of π and π^* in the above derivation in (5), we can derive another upper bound and use it to construct another surrogate problem: define $\tilde{c}_\pi(s^*, a^*) = \mathbb{E}_{a\sim\pi_{s^*}}[\|a - a^*\|]$ and $C^t_\pi(s^*) = \text{Lip}(Q^t_\pi(s^*, \cdot))$, then

$$\begin{aligned} &J(\pi) - J(\pi^*) \\ &= \mathbb{E}_{s^*,t\sim d_{\pi^*}}\left[\mathbb{E}_{a\sim\pi_{s^*}}[Q^t_\pi(s^*, a)] - \mathbb{E}_{a^*\sim\pi^*_{s^*}}[Q^t_\pi(s^*, a^*)]\right] \\ &\le \mathbb{E}_{s^*,t\sim d_{\pi^*}}\mathbb{E}_{a^*\sim\pi^*_{s^*}}\left[C^t_\pi(s^*)\tilde{c}_\pi(s^*, a^*)\right]. \end{aligned} \tag{8}$$

where we use again Lemma 1 for the equality and the property of Wasserstein distance for inequality. The minimization of the upper-bound (8) is called the *batch* IL problem [24, 2]:

$$\min_\pi \mathbb{E}_{\rho_{\pi^*}}\left[\sum_{t=1}^T \tilde{c}_\pi(s^*_t, a^*_t)\right], \tag{9}$$

In contrast to the surrogate problem in online IL (6), batch IL reduces to a supervised learning problem, because the expectation is defined by a fixed policy π^*.

C. Comparison of Imitation Learning Algorithms

Comparing (5) and (8), we observe that in batch IL the Lipschitz constant $C^t_\pi(s^*)$, without π being an expert as in Definition 1, can be on the order of $T - t$ in the worst case. Therefore, if we take a uniform bound and define $C_\pi = \sup_{t\in[0,T-1],s\in\mathbb{S}} C^t_\pi(s)$, we see $C_\pi \in O(T)$. In other words, under the same assumption in online imitation (i.e. (8) is minimized to an error in $O(T)$), the difference between $J(\pi)$ and $J(\pi^*)$ in batch IL actually grows quadratically in T due to error compounding. This problem manifests especially in stochastic environments. Therefore, in order to achieve the same level of performance as online IL, batch IL requires a more expressive policy class or more demonstration samples. As shown in [25], the quadratic bound is tight.

Therefore, if we can choose an expert policy π^* that is stable in the sense of Definition 1, then online IL is preferred theoretically. This is satisfied, for example, when the expert policy is an algorithm with certain performance characteristics. On the contrary, if the expert is human, the assumptions required by online IL become hard to realize in real-road driving tasks. This is especially true in off-road driving tasks, where the human driver depends heavily on instant feedback from the car to overcome stochastic disturbances. Therefore, the frame-by-frame labeling approach [26], for example, can lead to a very counter-intuitive, inefficient data collection process, because the required dynamics information is lost in a single image frame. Overall, when using human demonstrations, online IL can be as bad as batch IL [13], simply due to inconsistencies introduced by human nature.

IV. The Autonomous Driving System

Building on the previous analyses, we design a system that can learn to perform fast off-road autonomous driving with only on-board measurements. The overall system architecture for learning end-to-end DNN driving policies is illustrated in Fig. 2. It consists of three high-level controllers (an expert, a learner, and a safety control module) and a low-level controller,

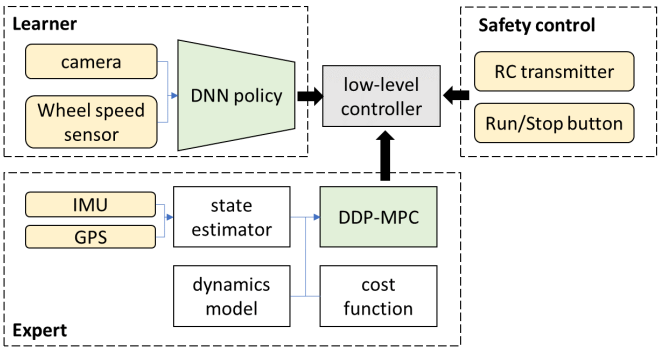


Fig. 2: System diagram.

which receives steering and throttle commands from the high-level controllers and translates them to pulse-width modulation (PWM) signals to drive the steering and throttle actuators of a vehicle.

On the basis of the analysis in Section III-C, we assume the expert is algorithmic and has access to expensive sensors (GPS and IMU) for accurate global state estimates[3] and resourceful computational power. The expert is built on multiple hand-engineered components, including a state estimator, a dynamics model of the vehicle, a cost function of the task, and a trajectory optimization algorithm for planning (see Section IV-A). By contrast, the learner is a DNN policy that has access to only a monocular camera and wheel speed sensors and is required to output steering and throttle command directly (see Section IV-B). In this setting, the sensors that the learner uses can be significantly cheaper than those of the expert; specifically on our experimental platform, the AutoRally car (see Section IV-C), the IMU and the GPS sensors required by the expert in Section IV-A together cost more than $6,000, while the sensors used by the learner's DNN policy cost less than $500. The safety control module has the highest priority among all three controllers and is used prevent the vehicle from high-speed crashing.

The software system was developed based on the Robot Operating System (ROS) in Ubuntu. In addition, a Gazebo-based simulation environment [12] was built using the same ROS interface but without the safety control module; the simulator was used to evaluate the performance of the software before real track tests.

A. An Algorithmic Expert: Model-Predictive Control

We use an MPC expert [22] based on an incremental Sparse Spectrum Gaussian Process (SSGP) [14] dynamics model (which was learned from 30 minute-long driving data) and an iSAM2 state estimator [8]. To generate actions, the MPC expert solves a finite horizon optimal control problem for *every* sampling time: at time t, the expert policy π^* is a locally optimal policy such that

$$\pi^* \approx \arg\min_\pi \mathbb{E}_{\rho_\pi}\left[\sum_{\tau=t}^{t+T_h} c(s_\tau, a_\tau)|s_t\right] \tag{10}$$

where T_h is the length of horizon it previews.

[3]Global position, heading and roll angles, linear velocities, and heading angle rate.

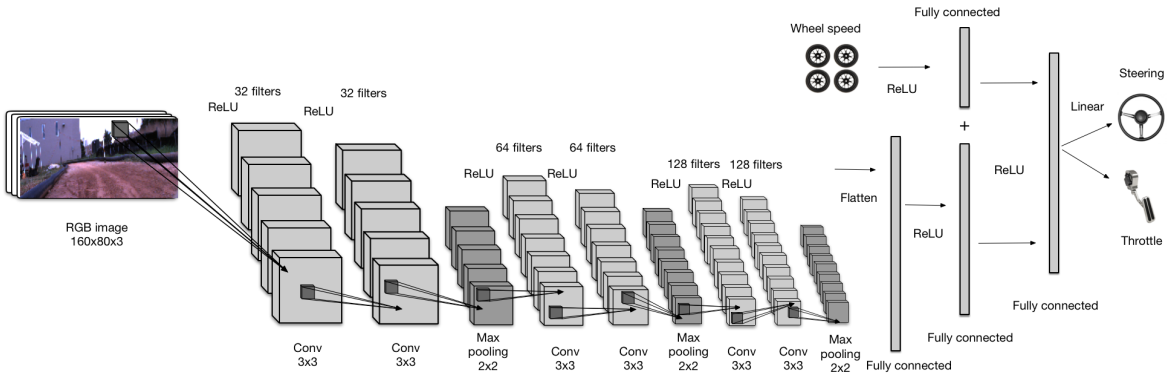


Fig. 3: The DNN control policy.

The computation is realized by Differential Dynamic Programming (DDP) [29]: in each iteration of DDP, the system dynamics and the cost function are approximated quadratically along a nominal trajectory; then the Bellman equation of the approximate problem is solved in a backward pass to compute the control law; finally, a new nominal trajectory is generated by applying the updated control law through the dynamics model in a forward pass. Upon convergence, DDP returns a locally optimal control sequence $\{\hat{a}_t^*, ..., \hat{a}_{t+T_h-1}^*\}$, and the MPC expert executes the first action in the sequence as the expert's action at time t (i.e. $a_t^* = \hat{a}_t^*$). This process is repeated at every sampling time (see [21] for details).

In view of the analysis in Section III-B, we can assume that the MPC expert satisfies Definition 1, because it updates the approximate solution to the original RL problem (1) in high-frequency using global state information. However, because MPC requires replanning for every step, running the expert policy (10) online consumes significantly more computational power than what is required by the learner.

B. Learning a DNN Control Policy

The learner's control policy π is parametrized by a DNN containing $\sim$10 million parameters. As illustrated in Fig. 3, the DNN policy, consists of two sub-networks: a convolutional neural network (CNN) with 6 convolutional layers, 3 max-pooling layers, and 2 fully-connected layers, that takes 160×80 RGB monocular images as inputs,[4] and a feedforward network with a fully-connected hidden layer that takes wheel speeds as inputs. The convolutional and max-pooling layers are used to extract lower-dimensional features from images. The DNN policy uses 3×3 filters for all convolutional layers, and rectified linear unit (ReLU) activation for all layers except the last one. Max-pooling layers with 2×2 filters are integrated to reduce the spatial size of the representation (and therefore reduce the number of parameters and computation loads). The two sub-networks are concatenated and then followed by another fully-connected hidden layer. The structure of this DNN was selected empirically based on experimental studies of several different architectures.

In construction of the surrogate problem for IL, the action space $\mathbb{A}$ is equipped with $\|\cdot\|_1$ for filtering outliers, and the optimization problem, (7) or (9), is solved using ADAM [11], which is a stochastic gradient descent algorithm with an adaptive learning rate. Note while s_t or s_t^* is used in (7) or (9), the neural network policy does not use the state, but rather the synchronized raw observation o_t as input. Note that we did not perform any data selection or augmentation techniques in any of the experiments. [5] The only pre-processing was scaling and cropping of raw images.

C. The Autonomous Driving Platform

To validate our IL approach to off-road autonomous driving, the system was implemented on a custom-built, 1/5-scale autonomous AutoRally car (weight 22 kg; LWH 1m$\times$0.6m$\times$0.4m), shown in the top figure in Fig. 4. The car was equipped with an ASUS mini-ITX motherboard, an Intel quad-core i7 CPU, 16GB RAM, a Nvidia GTX 750 Ti GPU, and a 11000mAh battery. For sensors, two forward facing machine vision cameras,[6] a Hemisphere Eclipse P307 GPS module, a Lord Microstrain 3DM-GX4-25 IMU, and Hall effect wheel speed sensors were instrumented. In addition, an RC transmitter could be used to remotely control the vehicle by a human, and a physical run-stop button was installed to disable all motions in case of emergency.

In the experiments, all computation was executed on-board the vehicle in real-time. In addition, an external laptop was used to communicate with the on-board computer remotely via Wi-Fi to monitor the vehicle's status. The observations were sampled and action were executed at 50 Hz to account for the high-speed of the vehicle and the stochasticity of the environment. Note this control frequency is significantly higher than [2] (10 Hz), [24] (12 Hz), and [19] (15 Hz).

V. Experimental Setup

A. High-speed Driving Task

We tested the performance of the proposed IL system in Section IV in a high-speed driving task with a desired speed of 7.5 m/s (an equivalent speed of 135 km/h on a full-scale car). The performance index of the task was formulated as the

[4]The raw images from the camera were re-scaled to 160×80.

[5]Data collection or augmentation techniques such as [6, 1] can be used in conjunction with our method.

[6]In this work we only used one of the cameras.

Fig. 4: The AutoRally car and the test track.

cost function in the finite-horizon RL problem (1) with

$$c(s_t, a_t) = \alpha_1 c_{\text{pos}}(s_t) + \alpha_2 c_{\text{spd}}(s_t) + \alpha_3 c_{\text{slip}}(s_t) + \alpha_3 c_{\text{act}}(a_t), \tag{11}$$

in which c_{pos} favors the vehicle to stay in the middle of the track, c_{spd} drives the vehicle to reach the desired speed, c_{slip} stabilizes the car from slipping, and c_{act} inhibits large control commands (see [21] for details).

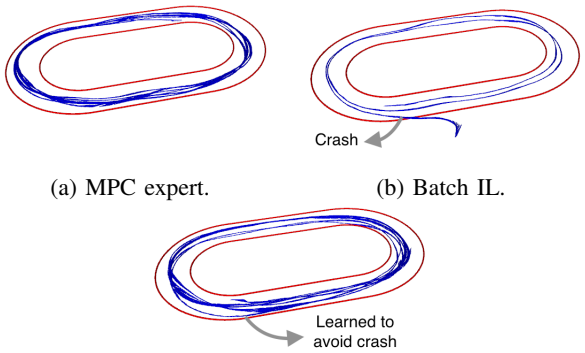
The goal of the high-speed driving task to minimize the accumulated cost function over one-minute continuous driving. That is, under the 50-Hz sampling rate, the task horizon was set to 60 seconds ($T = 3000$). The cost information (11) was given to the MPC expert in Fig. 2 to perform online trajectory optimization with a two-second prediction horizon ($T_h = 100$). In the experiments, the weighting in (11) were set as $\alpha_1 = 2.5$, $\alpha_2 = 1$, $\alpha_3 = 100$ and $\alpha_4 = 60$, so that the MPC expert in Section IV-A could perform reasonably well. The learner's policy was tuned by online/batch IL in attempts to match the expert's performance.

B. Test Track

All the experiments were performed on an elliptical dirt track, shown in the bottom figure of Fig. 4, with the AutoRally car described in Section IV-C. The test track was $\sim$3m wide and $\sim$30m long and built with fill dirt. Its boundaries were surrounded by soft HDPE tubes, which were detached from the ground, for safety during experimentation. Due to the changing dirt surface, debris from the track's natural surroundings, and the shifting track boundaries after car crashes, the track condition and vehicle dynamics can change from one experiment to the next, adding to the complexity of learning a robust policy.

C. Data Collection

Training data was collected in two ways. In batch IL, the MPC expert was executed, and the camera images, wheel speed readings, and the corresponding steering and throttle commands were recorded. In online IL, a mixture of the expert and learner's policy was used to collect training data (camera images, wheel speeds, and expert actions): in the ith iteration of DAgger, a mixed policy was executed at each time step $\hat{\pi}_i = \beta^i \pi^* + (1 - \beta^i)\pi_{i-1}$, where π_{i-1} is learner's DNN policy after $i - 1$ DAgger iterations, and β^i is the probability of executing the expert policy. The use of a mixture policy was suggested in [25, 3] for better stability. A mixing rate $\beta = 0.6$ was used in our experiments. Note that the probability of using the expert decayed exponentially as the number of DAgger iterations increased. Experimental data was collected on an outdoor track, and consisted of changing lighting conditions and environmental dynamics. In the experiments, the rollouts about to crash were terminated remotely by overwriting the autonomous control commands with the run-stop button or the RC transmitter in the safety control module; these rollouts were excluded from the data collection.



(a) MPC expert. (b) Batch IL.

(c) Online IL.

Fig. 5: Examples of vehicle trajectories, where online IL avoids the crashing case encountered by batch IL. (b) and (c) depict the test runs after training on 9,000 samples.

D. Policy Learning

In online IL, three iterations of DAgger were performed. At each iteration, the robot executed one rollout using the mixed policy described above (the probabilities of executing the expert policy were 60%, 36%, and 21%, respectively). For a fair comparison, the amount of training data collected in batch IL was the same as all of the data collected over the three iterations of online IL.

At each training phase, the optimization problem (7) or (9) was solved by ADAM for 20 epochs, with mini-batch size 64, and a learning rate of 0.001. Dropouts were applied at all fully connected layers to avoid over-fitting (with probability 0.5 for the firstly fully connected layer and 0.25 for the rest). See Section IV-B for details. Finally, after the entire learning session of a policy, three rollouts were performed using the learned policy for performance evaluation.

VI. Experimental Results

A. Empirical Performance

We first study the performance of training a control policy with online and batch IL algorithms. Fig. 5 illustrates the vehicle trajectories of different policies. Due to accumulating errors, the policy trained with batch IL crashed into the lower-left boundary, an area of the state-action space rarely explored in the expert's demonstrations. In contrast to batch IL, online IL successfully copes with corner cases as the learned policy occasionally ventured into new areas of the state-action space.

Fig. 6 shows the performance in terms of distance traveled without crashing[7] and Table II shows the statistics of the experimental results. Overall, DNN policies trained with both

[7]We used the safe control module shown in Fig. 2 to manually terminate the rollout when the car crashed into the soft boundary.

TABLE II: Test statistics. Total loss denotes the imitation loss in (6), which is the average of the steering and the throttle losses. Completion is defined as the ratio of the traveled time steps to the targeted time steps (3,000). All results here represent the average performance over three independent evaluation trials.

Policy	Avg. speed	Top speed	Training data	Completion ratio	Total loss	Steering/Throttle loss
Expert	6.05 m/s	8.14 m/s	N/A	100 %	0	0
Batch	4.97 m/s	5.51 m/s	3000	100 %	0.108	0.092/0.124
Batch	6.02 m/s	8.18 m/s	6000	51 %	0108	0.162/0.055
Batch	5.79 m/s	7.78 m/s	9000	53 %	0.123	0.193/0.071
Batch	5.95 m/s	8.01 m/s	12000	69 %	0.105	0.125/0.083
Online (1 iter)	6.02 m/s	7.88 m/s	6000	100 %	0.090	0.112/0.067
Online (2 iter)	5.89 m/s	8.02 m/s	9000	100 %	0.075	0.095/0.055
Online (3 iter)	6.07 m/s	8.06 m/s	12000	100 %	0.064	0.073/0.055

online and batch IL algorithms were able to achieve speeds similar to the MPC expert. However, with the same amount of training data, the policies trained with online IL in general outperformed those trained with batch IL. In particular, the policies trained using online IL achieved better performance in terms of both completion ratio and imitation loss.

In addition, we found that, when using online IL, the performance of the policy monotonically improves over iterations as data are collected, which is opposed to what was found by Laskey et al. [13]. The discrepancy can be explained with a recent theoretical analysis by Cheng and Boots [3], which provides a necessary and sufficient condition for the convergence of the policy sequence. In particular, the authors show that adopting a non-zero mixing (as used in our experiment) is sufficient to guarantee the convergence of the learned policy sequence. Our autonomous driving system is a successful real-world demonstration of this IL theory.

Finally, it is worth noting that the traveled distance of the batch learning policy, learned with 3,000 samples, was longer than that of other batch learning policies. This is mainly because this policy achieved better steering performance than throttle performance (cf. Steering/Throttle loss in Table II). That is, although the vehicle was able to navigate without crashing, it actually traveled at a much slower speed. By contrast, the other batch learning policies that used more data had better throttle performance and worse steering performance, resulting in faster speeds but also higher chances of crashing.

B. Generalizability of the Learned Policy

To further analyze the difference between the DNNs trained using online and batch IL, we embed the data in a two-dimensional space using t-Distributed Stochastic Neighbor Embedding (t-SNE) [16], as shown in Fig. 7 and Fig. 8. These figures visualize the data in both batch and online IL settings, where "train" denotes the data collected to train the policies and "test" denotes the data collected to evaluate the performance of the final policies after the learning phase.[8]

We first observe in Fig. 7 that, while the wheel speed data have similar training and testing distributions, the image

[8]For the online setting, the train data include the data in all DAgger iterations; for the batch setting, the train data include the same amount of data but collected by the expert policy. The figures plot a subset of 3,000 points from each data set.

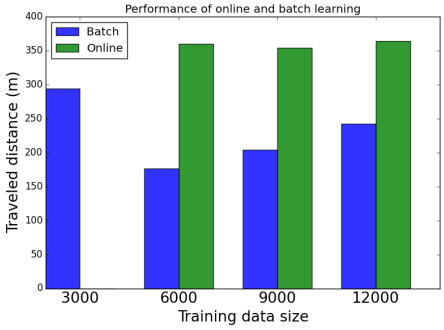
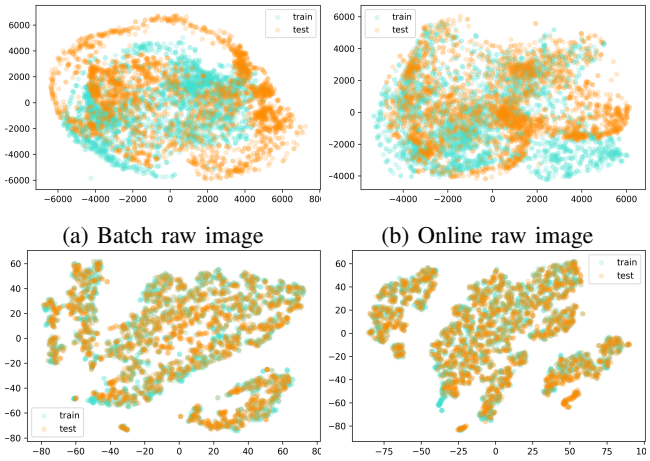


Fig. 6: Performance of online and batch IL in the distance (meters) traveled without crashing. The policy trained with a batch of 3,000 samples was used to initialize online IL.

distributions are fairly misaligned. The raw images are subject to changing lighting conditions, as the policies were executed at different times and days, and to various trajectories the robot stochastically traveled. Therefore, while the task (driving fast in the same direction) is *seemingly* monotone, it actually is not. More importantly, the training and testing images were collected by executing different policies, which leads to different distributions of the neural networks inputs. This is known as the *covariate shift* problem [28], which can significantly complicate the learning process.

The policy trained with online IL yet still demonstrated great performance in the experiments. To further understand how it could generalize across different image distributions, we embed its feature distribution in Fig. 8 (a) and (b).[9] Interestingly, despite the difference in the raw feature distributions in Fig. 7 (a) and (b), the DNN policy trained with online IL are able to map the train and test data to similar feature distributions, so that a linear combination (the last layer) of those features is sufficient to represent a good policy. On the contrary, the DNN policy trained with batch IL fails to learn a coherent feature embedding, as shown in Fig. 8 (c) and (d). This could explain the inferior performance of batch IL, and its inability to deal with the corner case in Fig. 5 (b). This evidence shows that our online learning system can alleviate the covariate shift issue caused by executing different policies at training and testing time.

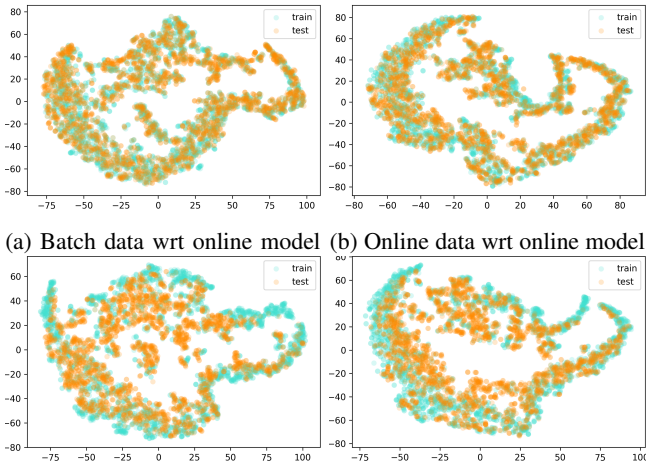
[9]The feature here are the last hidden layer of the neural network. The output layer is a linear function of the features.



(a) Batch raw image (b) Online raw image

(c) Batch wheel speed (d) Online wheel speed

Fig. 7: The distributions (t-SNE) of the raw images and wheel speed used as DNN policy's inputs (details in Section VI-B).



(a) Batch data wrt online model (b) Online data wrt online model

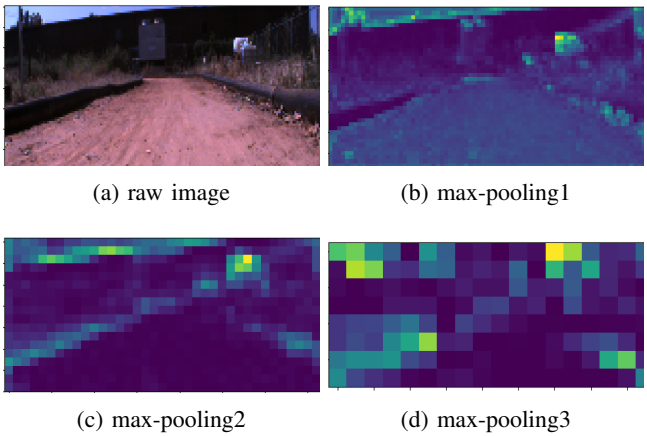
(c) Batch data wrt batch model (d) Online data wrt batch model

Fig. 8: The distributions (t-SNE) of the learned DNN feature in the last fully-connected layer (details are in Section VI-B).

C. The Neural Network Policy

Compared with hand-crafted feature extractors, one main advantage of a DNN policy is that it can learn to extract both low-level and high-level features of an image and automatically detect the parts that have greater influence on steering and throttle. We validate this idea by showing in Fig. 9 the averaged feature map at each max-pooling layer (see Fig. 3), where each pixel represents the averaged unit activation across different filter outputs. We can observe that at a deeper level, the detected salient features are boundaries of the track and parts of a building. In contrast, grass and dirt contribute little.

We also analyze the importance of incorporating wheel speeds in our task. We compare the performance of the policy based on our DNN policy and a policy based on only the CNN subnetwork (without wheel-speed inputs) in batch IL. The data was collected in accordance with Section V-C. Fig. 10 shows the batch IL loss in (9) of different network architectures. The full DNN policy in Fig. 3 achieved better performance



(a) raw image (b) max-pooling1

(c) max-pooling2 (d) max-pooling3

Fig. 9: The input RGB image and the averaged feature maps for each max-pooling layer.

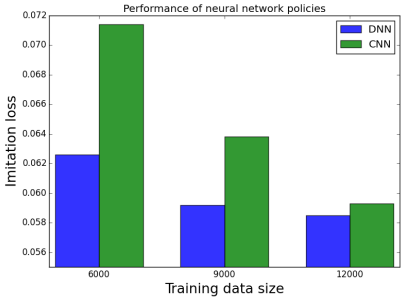


Fig. 10: Performance comparison between our DNN policy and its CNN sub-network in terms of batch IL loss, where the horizontal axis is the size of data used to train the neural network policies.

consistently. While images contain position and orientation information, it is insufficient to infer velocities, which are a part of the (hidden) vehicle state. Therefore, we conjecture state-of-the-art CNNs (e.g. [2]) cannot be directly used to perform both lateral and longitudinal controls, as we do here. By contrast, while without a recurrent architecture, our DNN policy learned to combine wheel speeds in conjunction with CNN to infer hidden state and achieve better performance.

VII. Conclusion

We introduce an end-to-end system to learn a deep neural network control policy for high-speed driving that maps raw on-board observations to steering and throttle commands by mimicking a model predictive controller. In real-world experiments, our system was able to perform fast off-road navigation autonomously using a low-cost monocular camera and wheel speed sensors. We also provide an analysis of both online and batch IL frameworks, both theoretically and empirically and show that our system, when trained with online IL, learns generalizable features that are more robust to covariate shift than features learned with batch IL.

Acknowledgements

This work was partially supported by NSF NRI Awards 1637758 and 1426945.

References

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*, 2017.

[3] Ching-An Cheng and Byron Boots. Convergence of value aggregation for imitation learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1801–1809, 2018.

[4] Ching-An Cheng, Xinyan Yan, Nolan Wagener, and Byron Boots. Fast policy learning through imitation and reinforcement. 2018.

[5] Paul Drews, Grady Williams, Brian Goldfain, Evangelos A Theodorou, and James M Rehg. Aggressive deep driving: Model predictive control with a cnn cost model. *arXiv preprint arXiv:1707.05303*, 2017.

[6] A René Geist, Andreas Hansen, Eugen Solowjow, Shun Yang, and Edwin Kreuzer. Data collection for robust end-to-end lateral vehicle control. In *ASME 2017 Dynamic Systems and Control Conference*, pages V001T45A007–V001T45A007. American Society of Mechanical Engineers, 2017.

[7] Alison L Gibbs and Francis Edward Su. On choosing and bounding probability metrics. *International Statistical Review*, 70(3):419–435, 2002.

[8] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2): 216–235, 2012.

[9] Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Plato: Policy learning using adaptive trajectory optimization. In *IEEE International Conference on Robotics and Automation*, pages 3342–3349, 2017.

[10] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, volume 2, pages 267–274, 2002.

[11] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference onIntelligent Robots and Systems, 2004.(IROS 2004)*, volume 3, pages 2149–2154. IEEE, 2004.

[13] Michael Laskey, Caleb Chuck, Jonathan Lee, Jeffrey Mahler, Sanjay Krishnan, Kevin Jamieson, Anca Dragan, and Ken Goldberg. Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations. *arXiv preprint arXiv:1610.00850*, 2016.

[14] Miguel Lázaro-Gredilla, Joaquin Quiñonero Candela, Carl Edward Rasmussen, and Aníbal R. Figueiras-Vidal. Sparse spectrum gaussian process regression. *Journal of Machine Learning Research*, 11(Jun):1865–1881, 2010.

[15] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(1):1334–1373, January 2016.

[16] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[17] Jeff Michels, Ashutosh Saxena, and Andrew Y Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *International Conference on Machine learning*, pages 593–600, 2005.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[19] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in Neural Information Processing Systems*, pages 739–746, 2006.

[20] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1 (1):33–55, 2016.

[21] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile off-road autonomous driving using end-to-end deep imitation learning. *arXiv preprint arXiv:1709.07174*, 2017.

[22] Yunpeng Pan, Xinyan Yan, Evangelos A. Theodorou, and Byron Boots. Prediction under uncertainty in sparse spectrum Gaussian processes with applications to filtering and control. In *International Conference on Machine Learning*, pages 2760–2768, 2017.

[23] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*, pages 305–313, 1989.

[24] Viktor Rausch, Andreas Hansen, Eugen Solowjow, Chang Liu, Edwin Kreuzer, and J. Karl Hedrick. Learning a deep neural net policy for end-to-end control of autonomous vehicles. In *IEEE American Control Conference*, 2017.

[25] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, volume 1, page 6, 2011.

[26] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shau-

rya Shankar, Andreas Wendel, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive uav control in cluttered natural environments. In *IEEE International Conference on Robotics and Automation*, pages 1765–1772, 2013.

[27] Shai Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

[28] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[29] Yuval Tassa, Tom Erez, and William D Smart. Receding horizon differential dynamic programming. In *Advances in Neural Information Processing Systems*, pages 1465–1472, 2008.

[30] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *IEEE International Conference on Robotics and Automation*, pages 1433–1440, 2016.

[31] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721. IEEE, 2017.

[32] Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016.