Imitation Learning for Agile Autonomous Driving

Journal Title
XX(X):1–15

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Yunpeng Pan[1], Ching-An Cheng[1], Kamil Saigol[1], Keuntaek Lee[2], Xinyan Yan[1], Evangelos A. Theodorou[1], Byron Boots[1]

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.

Keywords

Imitation learning, autonomous driving, control

1 Introduction

High-speed autonomous off-road driving is a challenging robotics problem (Michels et al. 2005; Williams et al. 2016, 2017) (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 (Michels et al. 2005; Paden et al. 2016) 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. (2016, 2017) 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) (Levine et al. 2016; Volodymyr et al. 2015), 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

Figure 1. The high-speed off-road driving task.

to most previous work that showcases RL in simulated environments (Volodymyr et al. 2015), 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) (Pomerleau 1989) 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

[1]Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, Georgia 30332–0250
[2]School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332–0250

Table 1. Comparison of our method to prior work on IL for autonomous driving

Methods	Tasks	Observations	Action	Algorithm	Expert	Experiment
Bojarski et al. (2016)	On-road low-speed	Single image	Steering	Batch	Human	Real &simulated
Pomerleau (1989)	On-road low-speed	Single image & laser	Steering	Batch	Human	Real &simulated
Rausch et al. (2017)	On-road low-speed	Single image	Steering	Batch	Human	Simulated
Muller et al. (2006)	Off-road low-speed	Left & right images	Steering	Batch	Human	Real
Zhang and Cho (2016)	On-road unknown speed	Single image	Steering + break	Online	Pre-specified policy	Simulated
Yang et al. (2018)	On-road low/high speed	Single image + vehicle speed sequence	Steering + speed	Batch	Human	Simulated
Yu et al. (2017)	On-road low/high speed	Image sequence + GPS/IMU measurement	Steering + acceleration	Batch	Human	Simulated
Our Method	Off-road high-speed	Single image + wheel speeds	Steering + throttle	Online	Model predictive controller	Real & simulated

(e.g. (Williams et al. 2016)) 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 by Kahn et al. (2017) and Mordatch and Todorov (2014) 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 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 (Williams et al. 2017). A preliminary version of this paper was published as (Pan et al. 2018) in RSS 2018; the current version compliments the previous paper with additional details of the setup of IL problems and the system design.

2 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) (Pomerleau 1989) 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 (Muller et al. 2006; Bojarski et al. 2016; Rausch et al. 2017) 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 (Bojarski et al. 2016, 2017), 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 many previous work only focuses on steering commands (Muller et al. 2006; Bojarski et al. 2017; Rausch et al. 2017). In (Yang et al. 2018), a Convolutional Neural Network (CNN) + Long Short Term Memory network (LSTM) design was proposed to predict both steering angle and vehicle speed. In (Yu et al. 2017), a convolutional LSTM model is used to predict acceleration commands. However, these approaches are not exactly end-to-end because a lower-level control module is required to compute the throttle/brake commands. Furthermore, (Yu et al. 2017) requires GPS/IMU measurements which increase the hardware cost. A Dataset Aggregation (DAgger) (Ross et al. 2011) related online IL algorithm for autonomous driving was recently demonstrated in (Zhang and Cho 2016), but only considered simulated environments and used a rule-based policy as the expert. In comparison with the previous setups, our system uses an MPC expert that solves optimal control problems at a high frequency, rather than a human driver (Bojarski et al. 2016) or a simple rule-based policy (Zhang and Cho 2016), in order to provide timely feedback to contend with the stochasticity in high-speed dirt-track driving. A comparison of different IL approaches to autonomous driving is presented in Table 1.

Our task is similar to the task considered by Williams et al. (2016, 2017) and Drews et al. (2017). 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 (Williams et al. 2016, 2017), 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 (Drews et al. 2017), 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 off-loads 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 (Williams et al. 2016), and are faster and more data efficient than that in (Drews et al. 2017).

3 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 realizations that the connection between online IL and DAgger-like algorithms (Ross et al. 2011) has not been formally introduced in continuous domains,* and that the original derivation is more convoluted and does not convey important structural properties. Here we simplify the derivation of Ross et al. (2011) into a compact tutorial, and extend it to continuous action spaces as required in the autonomous driving task.

3.1 Problem Setup

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 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.

Our goal here is to find a stationary, reactive policy† $\pi : \mathbb{O} \mapsto \mathbb{A}$ (e.g. a DNN policy) such that π achieves low accumulated costs 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 $(s_0, o_0, a_0, s_1, \ldots, a_{T-1})$ generated by running policy π from a fixed initial state distribution. 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 o, and $a \sim \pi_o$ as the stochastic action taken by the policy. Moreover, we denote $Q_t^\pi(s,a)$ as the Q-function (i.e. state-action value function) at state s, action a, and time t, i.e., $Q_t^\pi(s,a) = \mathbb{E}_{\rho_\pi}[\sum_{\tau=t}^{t+T-1} c(s_\tau, a_\tau) | s_t = s, a_t = a]$; we denote $V_t^\pi(s) = \mathbb{E}_{a\sim\pi_s}[Q_t^\pi(s,a)]$ 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.

3.2 Goal of 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* $\pi^\star$ 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.

We wish to design an IL algorithm that can train a policy π to perform as well as the expert $\pi^\star$ with an error that has at most *linear* dependency on the time horizon of the problem T. Specifically, it is desired that $J(\pi) \leq J(\pi^\star) + O(T\epsilon)$, where ϵ denotes potential errors due to approximation and optimization at each time step. This requirement is motivated by the long problem horizon in autonomous driving tasks. Suppose otherwise an algorithm returns a policy that has performance only as $J(\pi) \leq J(\pi^\star) + O(T^2\epsilon)$; the applicability of the algorithm to tasks that involve a long problem horizon would be limited, because learning imperfection in every time step would accumulate quickly and hurt the policy performance.

3.3 Admissible Experts

It turns out this linear error dependency is not feasible unless we make some assumptions on the quality of expert policies. Therefore, in this paper, we consider a qualification on the properties of the expert policies below.‡

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

Let us give some intuition about what Definition 1 means. Recall by definition $Q_t^{\pi^\star}(s,a)$ is the accumulated cost of taking some action a at time t and then executing the expert policy $\pi^\star$ afterwards. The idea behind Definition 1 is that a reasonable expert policy $\pi^\star$ should perform stably under arbitrary action perturbation, regardless of where it starts.

In the following, we will assume that we have access to an admissible expert policy in IL. This assumption rules out the situation of learning from arbitrarily bad expert policies; in other words, we consider sub-optimal experts that have at least non-trivial performance with respect to (1). Later in Section 3.5.3, we will discuss that if the expert is not admissible, both batch and online IL learning approaches could incur the non-desirable superlinear error dependency.

*Before the conference version of this paper (Pan et al. 2018) was published, DAgger had only been used heuristically in these domains (Ross et al. 2013; Zhang and Cho 2016)

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

‡This assumption was implicitly made by Ross et al. (2011) to derive the linear dependency bound; here we make the assumption explicit, and define the admissible experts using an uniform Lipschitz constant because the action space in our task is continuous; for discrete action spaces, $\mathrm{Lip}\left(Q_t^{\pi^\star}(s,\cdot)\right)$ can be replaced by $\sup_{a\in\mathbb{A}} Q_t^{\pi^\star}(s,a)$ and the rest applies (cf. (Ross et al. 2011)). The Lipschitz condition applies only to the expected behaviors (which makes it more likely to be true), which is supported empirically by our experimental results.

However, when the expert policy is indeed admissible, we will show that the property delineated in Definition 1 provides a guidance for whether to choose batch learning vs. online learning to train a policy by imitation.

3.4 Performance Difference

As the goal of IL is to bound the performance difference between the learner π and the expert $\pi^\star$, we need a way to express the difference $J(\pi) - J(\pi^\star)$. A useful tool is the Performance Difference Lemma below, which was due to Kakade and Langford (2002).

Lemma 1. *Define $d^\pi(s,t) = d_t^\pi(s)$ as an unnormalized 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 arbitrary 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'.

Lemma 1 gives a closed-form expression of the performance difference based on structural properties of Markov decision process. Precisely, it can be read in two ways:

$$J(\pi) - J(\pi^\star) = \mathbb{E}_{s,t\sim d^\pi}\mathbb{E}_{a\sim\pi_s}[A_t^{\pi^\star}(s,a)] \tag{3}$$
$$= -\mathbb{E}_{s,t\sim d_{\pi^\star}}\mathbb{E}_{a\sim\pi_s^\star}[A_t^{\pi}(s,a)] \tag{4}$$

giving a duality relationship of the performance difference. First, (3) says that the performance difference is equivalent to the degree the learner policy π is better than the expert policy $\pi^\star$ in expectation over the states visited by the learner, because the advantage function $A_t^{\pi^\star}(s,a)$ in essence measures how an action a is better than the expert policy $\pi^\star$ at state s. Alternatively, (4) says that the performance difference is equivalent to the degree the expert policy $\pi^\star$ is worse than the learner policy π in expectation over the states the expert visits. In short, these two perspectives show that the performance difference can be upper bounded in terms of either the state-action distribution of the learner or that of the expert. We will show that the choice of perspective, (3) or (4), is the bifurcation point that leads to the online approach or the batch approach to IL.

In order to derive upper bounds of (3) and (4), let us review a basic statistical distance, Wasserstein metric (Gibbs and Su 2002), for distributions with continuous random variables, because the action space of (1) is continuous. We will use this distance to derive upper bounds of (3) and (4); we note that other statistical distances, such as KL-divergence, can also be adopted naturally. For two probability distributions p and q on a metric space $\mathcal{M}$ with metric d, the Wasserstein metric $D_W(\cdot,\cdot)$ is defined as

$$D_W(p,q) \coloneqq \sup_{f:\mathrm{Lip}(f(\cdot))\le 1} \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)] \tag{5}$$
$$= \inf_{\gamma\in\Gamma(p,q)} \int_{\mathcal{M}\times\mathcal{M}} d(x,y)d\gamma(x,y), \tag{6}$$

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 (Gibbs and Su 2002). For our autonomous driving application, we suppose the action space $\mathbb{A}$ is a normed space with norm $\|\cdot\|$ and consider the natural metric induced by the chosen norm, i.e. $d(x,y) = \|x-y\|$.

3.5 Two Approaches to Imitation Learning

3.5.1 Online Imitation Learning We first present the objective function for the online learning approach to IL. We achieve this by upper bounding the performance difference using (3) as follows

$$\begin{aligned} &J(\pi) - J(\pi^\star) \\ &= \mathbb{E}_{s,t\sim d^\pi}\left[\mathbb{E}_{a\sim\pi_s}[Q_t^{\pi^\star}(s,a)] - \mathbb{E}_{a^\star\sim\pi_s^\star}[Q_t^{\pi^\star}(s,a^\star)]\right] \\ &\le C^{\pi^\star}\mathbb{E}_{s,t\sim d^\pi}\left[D_W(\pi,\pi^\star)\right] \\ &\le C^{\pi^\star}\mathbb{E}_{s,t\sim d^\pi}\mathbb{E}_{a\sim\pi_s}\mathbb{E}_{a^\star\sim\pi_s^\star}[\|a-a^\star\|], \end{aligned} \tag{7}$$

where the first equality is simply (3) but with the advantage function replaced with its definition $A_t^{\pi^\star}(s,a) = Q_t^{\pi^\star}(s,a) - \mathbb{E}_{a^\star\sim\pi_s^\star}[Q_t^{\pi^\star}(s,a^\star)]$, and the two inequalities are due to (5) and (6), respectively.

Define $\hat{c}(s,a) = \mathbb{E}_{a^\star\sim\pi_s^\star}[\|a-a^\star\|]$. Thus, to make π perform as well as $\pi^\star$, we can minimize the upper bound in (7), i.e.

$$\min_\pi \mathbb{E}_{\rho_\pi}\left[\sum_{t=0}^{T-1}\hat{c}(s_t,a_t)\right]. \tag{8}$$

This leads to a new surrogate RL problem in (8), which we call the *online* IL problem, as the trajectory distribution in (8) still depends on π. One can choose to solve (8) with pure RL techniques (e.g. policy gradient methods); however, while this new problem might be simpler than the original one in (1) (e.g. (8) has denser cost functions than (1)), tackling it directly could still be sample inefficient due to the necessity of back-propagating information through trajectories. To circumvent this difficulty, the online learning approach to IL (Ross et al. 2011; Ross and Bagnell 2014; Cheng et al. 2019c,b) leverages the structural property of cost function $\hat{c}$ in (8) and relies on a reduction from (8) to online learning problems (Shalev-Shwartz 2012) to optimize policies. As a result, back-propagating through trajectories is no longer necessary, and provable performance guarantees can be achieved (Cheng et al. 2018).

In this paper, we use the meta-algorithm DAgger (Ross et al. 2011), which reduces (8) to a sequence of supervised learning problems: Let $\mathcal{D}$ be the training data. DAgger initializes $\mathcal{D}$ with samples gathered by running $\pi^\star$. Then, in the ith iteration, it trains π_i by supervised learning,

$$\pi_i = \arg\min_\pi \mathbb{E}_{\mathcal{D}}[\hat{c}(s_t,a_t)], \tag{9}$$

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 (8), is returned. Suppose the policy is linearly parametrized. When the instantaneous cost $\hat{c}(s_t,\cdot)$ is strongly convex, running DAgger to solve (8) finds a policy π with performance $J(\pi) \le J(\pi^\star) + O(TC^{\pi^\star})$. Therefore, when the $\pi^\star$ is an admissible expert (i.e. $C^{\pi^\star} =$

[§] $d_t^\pi(s)$ is an unnormalized time-state distribution of the time-state distribution $\frac{1}{T}d_t^\pi(s)$. One can verify that $\sum_{t=0}^{T-1}\int_{s\in\mathcal{S}}\frac{1}{T}d_t^\pi(s) = 1$.

$O(1)$), DAgger would achieve our initial goal of linear error dependency.

We note here the instantaneous cost $\hat{c}(s_t, \cdot)$ can be selected to be any suitable norm according the problem's property since norms in finite dimensional space are equivalent. 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.

3.5.2 Batch Imitation Learning The batch approach to IL takes a different viewpoint of performance difference. Using the other equality in (4), we can derive another upper bound and use it to construct a different surrogate problem: define $\tilde{c}_\pi(s^\star, a^\star) = \mathbb{E}_{a\sim\pi_{s^\star}}[\|a - a^\star\|]$ and $C_t^\pi(s^\star) = \text{Lip}(Q_t^\pi(s^\star, \cdot))$, then we can write

$$\begin{aligned} &J(\pi) - J(\pi^\star) \\ &= \mathbb{E}_{s^\star,t\sim d_{\pi^\star}}\left[\mathbb{E}_{a\sim\pi_{s^\star}}[Q_t^\pi(s^\star, a)] - \mathbb{E}_{a^\star\sim\pi^\star_{s^\star}}[Q_t^\pi(s^\star, a^\star)]\right] \\ &\leq \mathbb{E}_{s^\star,t\sim d_{\pi^\star}}\mathbb{E}_{a^\star\sim\pi^\star_{s^\star}}\left[C_t^\pi(s^\star)\tilde{c}_\pi(s^\star, a^\star)\right]. \end{aligned} \tag{10}$$

where the derivation is similar to the one in (7) but the first equality is instead based on (4). The problem of minimizing the upper-bound (10) is called the *batch* IL problem (Rausch et al. 2017; Bojarski et al. 2017) and can be written equivalently as:

$$\min_\pi \mathbb{E}_{\rho_{\pi^\star}}\left[\sum_{t=0}^{T-1}\tilde{c}_\pi(s_t^\star, a_t^\star)\right], \tag{11}$$

In contrast to the surrogate problem in online IL (8), batch IL reduces to a supervised learning problem, because the expectation is defined by a fixed policy $\pi^\star$.

3.5.3 Comparison Comparing (7) and (10), we observe that in batch IL the Lipschitz constant $C_t^\pi(s^\star)$, without π being an admissible 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 IL (i.e. (10) is minimized to an error in $O(T)$), the difference between $J(\pi)$ and $J(\pi^\star)$ 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 (Ross et al. 2011), the quadratic bound is tight.

Therefore, if we have access to an admissible expert policy $\pi^\star$ 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. However, on the contrary, when the expert is not admissible (i.e. $C^{\pi^\star} \geq \Omega(1)$), online IL would also lead to a superlinear error dependency. This could happen when human demonstrators are adopted within online IL to perform off-road driving tasks. Because the human drivers depend heavily on instant feedback from the car to overcome stochastic disturbances, the frame-by-frame labeling approach Ross et al. (2013), for example, can lead to a very counter-intuitive, inefficient data collection process and effectively result in an inadmissible expert policy. Overall, when using human demonstrations, online IL can be as bad as batch IL (Laskey et al. 2016), simply due to inconsistencies introduced by human nature.

4 Design of Algorithmic Expert

We showed that if an admissible expert is available then online IL (e.g. DAgger) provides a learning framework that can achieve the desirable linear error dependency. Due to the dynamic nature of our high-speed driving task, we consider *algorithmic* experts, because human experts might not be able to provide stable and consistent high-frequency feedbacks while the learner is driving the car.

More precisely, we recall that online IL requires action demonstrations on the trajectories generated by running the learner's policy. For high-speed driving, this means that human experts need to provide action demonstrations (desired steering and throttle commands here) when the vehicle is being autonomously controlled by the (suboptimal) learner's policy. Deprived of the usual sensory-motor feedback, human drivers often provide poor feedback: for example, we have observed that human drivers tend to overcompensate when providing steering when faced with unexpected vehicle dynamics (under the control of the learner's policy). This inconsistency can introduce bias into the demonstrated actions, in the worst case, effectively creating an inadmissble expert policy for IL (see Section 3.5.3).

By contrast, a natural candidate with such stability property would be the optimal policy of (1). Specifically, suppose the dynamics of (1) is known, an expert policy can be obtained by solving problem (1) via Dynamic Programming, and its value function is the solution to the Bellman equation

$$V_t^{\pi\star}(s_t) = \min_{a_t\in\mathbb{A}} c(s_t, a_t) + \mathbb{E}_{s_{t+1}\sim p_{s_{t+1}|s_t,a_t}}\left[V_{\pi\star}^{t+1}(s_{t+1})\right] \tag{12}$$

where $p_{s'|s,a}$ denotes the distribution of vehicle dynamics (i.e. the state transition).

However, in practice, the above idealistic approach faces two main challenges: 1) the transition probability $p_{s'|s,a}$ is hard to obtain due to the complexity of vehicle dynamics at high-speed in off-road conditions. 2) Solving (12) for all $s \in \mathbb{S}$ is computationally intractable due to the curse of dimensionality of Dynamic Programming. In this work, we address these two challenges using a probabilistic dynamics model and trajectory optimization. We describe these techniques in the following and they will be used as the foundation to design the algorithmic expert in our IL system, as later described in Section 5.1.

4.1 Probabilistic Dynamics Model

Under normal driving conditions, a planar single-track vehicle model derived from Newtonian physics (Kong et al. 2015) and an empirical tire model (Rajamani 2011) are widely used and usually sufficient for control design. In contrast, controlling a race car in aggressive maneuvers, e.g., cornering at the limit of tire-road friction, requires more sophisticated techniques to estimate the tire-road friction coefficient (Laurense et al. 2017). In our case, the friction changes rapidly due to the uneven dirt surface, which makes it more challenging to estimate the coefficient. In practice, physics-based models do not capture the aforementioned dynamics effects well, and neural networks (NNs) have been used for vehicle dynamics model identification (Rutherford

and Cole 2010; Williams et al. 2017). However, NNs typically do not adapt to rapidly changing road conditions in real-time. In addition, NNs do not provide estimates of model uncertainty given limited amount of training data, which can hamper accurate long-range prediction. Motivated by these challenges, we consider learning a probabilistic model, Sparse Spectrum Gaussian Processes (Lázaro-Gredilla et al. 2010) (SSGPs) from data to approximate the vehicle dynamics, and Bayesian inference to predict the vehicle's future states.

4.1.1 SSGP Regression Gaussian process regression (GPR) (Williams and Rasmussen 2006) is a principled way to perform Bayesian inference in function space. Consider the task of learning function $f : \mathbb{R}^d \to \mathbb{R}$ (the vehicle dynamics model in our case, and we treat each output dimension independently) given a dataset $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$ that are sampled according to

$$y_n = f(x_n) + \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(0, \sigma^2), \tag{13}$$

where ϵ is an independent additive zero-mean Gaussian noise with covariance σ^2. Data collection details will be provided in section 6.3. GPR reasons about potential candidates of the latent function, under the assumption that f has a prior GP distribution $f \sim \mathcal{GP}(\bar{m}, k)$, with mean function $\bar{m} : \mathbb{R}^d \to \mathbb{R}$ and covariance function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. That is, for any $x, x' \in \mathbb{R}^d$, $\mathbb{E}[f(x)] = \bar{m}(x)$, $\mathbb{C}[f(x), f(x')] = k(x, x')$, and for any finite subset $\{x_n \in \mathbb{R}^d\}_{n=1}^K$, $\{f(x_n)\}_{n=1}^K$ is Gaussian distributed. Using the dataset $\mathcal{D}$, the inference problem of GRP computes the posterior distribution of the latent function f. Without loss of generality, we assume $\bar{m}(x) = 0$ *a priori*.

While theoretically sound, the exact inference problem of GPR is challenging for large datasets due to its $O(N^3)$ time and $O(N^2)$ space complexities (Williams and Rasmussen 2006), which is a direct consequence of storing and inverting an $N \times N$ Gram matrix. Many approximate techniques have been proposed to tackle this challenge, including using random Fourier features (Lázaro-Gredilla et al. 2010), degenerate priors (Snelson and Ghahramani 2006), variational posteriors (Titsias 2009; Hensman et al. 2013; Cheng and Boots 2017; Salimbeni et al. 2018). In this work, we adopt the approach by Lázaro-Gredilla et al. (2010) for its implementation simplicity.

To reduce the complexity, Lázaro-Gredilla et al. (2010) use approximate GP models, SSGPs, which are a class of GPs with prior covariance function in the form:

$$k(x, x') = \phi(x)^\top \phi(x') + \sigma^2 \delta(x - x'), \ \phi(x) = \begin{bmatrix} \phi^c(x) \\ \phi^s(x) \end{bmatrix},$$
$$\phi^c(x) = \begin{bmatrix} \phi_1^c(x) & \dots & \phi_m^c(x) \end{bmatrix}^\top, \phi^s(x) = \begin{bmatrix} \phi_1^s(x) & \dots & \phi_m^s(x) \end{bmatrix}^\top$$
$$\phi_i^c(x) = \eta \cos(\omega_i^\top x), \ \phi_i^s(x) = \eta \sin(\omega_i^\top x), \ \omega_i \sim p(\omega),$$

where function $\phi : \mathbb{R}^d \to \mathbb{R}^{2m}$ is an explicit finite-dimensional feature map[¶], η is a scalar scaling coefficient, δ is the Kronecker delta function, and vector ω_i is sampled according to some spectral density $p(\omega)$ function. Based on Bochner's theorem, it can be shown that SSGPs can unbiasedly approximate any continuous shift-invariant kernels if $p(\omega)$ is constructed properly with respect to the original covariance function (Lázaro-Gredilla et al. 2010).

Because of the explicit finite-dimensional feature map ϕ, each SSGP is equivalent to a Gaussian distribution over the weights of the features $w \in \mathbb{R}^{2m}$ and has a prior distribution of weights w as $\mathcal{N}(0, I)$ (Lázaro-Gredilla et al. 2010). Given a fixed feature map, the posterior distribution of w conditioned on the data $\mathcal{D} = \{x_n, y_n\}_{n=1}^N$ is

$$w \sim \mathcal{N}(\alpha, \ \sigma^2 A^{-1}), \tag{14}$$
$$\alpha = A^{-1} \Phi Y, \quad A = \Phi \Phi^\top + \sigma^2 I, \tag{15}$$

which can be derived through Bayesian linear regression. In (15), the column vector Y and the matrix Φ are specified by the data $\mathcal{D}$, in which $Y = \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}^\top$ and $\Phi = \begin{bmatrix} \phi(x_1) & \dots & \phi(x_n) \end{bmatrix}$. Consequently, the posterior distribution over the output y in (13) at a test point x is *exactly Gaussian*

$$p(y|x) = \mathcal{N}(\alpha^\top \phi(x), \ \sigma^2 + \sigma^2 \|\phi(x)\|^2_{A^{-1}}). \tag{16}$$

in which the posterior variance explicitly captures the model uncertainty in predicting $f(x)$.

We use this SSGP framework to model the state transition of the unknown vehicle dynamics (i.e. the latent function $f : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$ that determines the change of state $\Delta s_t := s_{t+1} - s_t$ given a concatenated state-action pair (s_t, a_t) as input). We assume each output dimension is conditionally independent[‖] and use a SSGP model for each output dimension. The hyper-parameters σ, η are optimized via maximizing the GP marginal likelihood (Williams and Rasmussen 2006).

4.1.2 Incremental Update In order to cope with rapidly changing dynamics (e.g, caused by stochastic road conditions), when a new vehicle state is available, we incorporate it to incrementally *update* the posterior distribution over weightz w in (14) of the SSGP dynamics model. We note this can be done rather efficiently without storing and inverting the A matrix explicitly. Instead we keep track of its Cholesky factor R where $A = R^\top R$ and perform rank-1 update given a new sample (Gijsberts and Metta 2013). The computation requires $O(m^2)$ time and can be performed in real-time if m is moderate. To cope with time-varying dynamics, we employ a forgetting factor $\lambda \in (0, 1)$ such that the previous samples' impact decays exponentially in time (Ljung 1998). Details are omitted.

4.1.3 Multi-step Prediction We need to be able to perform multi-step prediction in order to use the SSGP dynamics model inside a trajectory optimization algorithm. We provide the details of information propagation across SSGP dynamics models in the following. At time t, suppose the distribution at the current state s_t is distributed according to $p(s_t) = \mathcal{N}(\mu_t, \Sigma_t)$ and the current action is a_t. We wish to compute the state distribution $p(s_{t+1})$ at time $t + 1$, which is related to the current state s_t through

$$s_{t+1} = s_t + f(s_t, a_t) + w_t, \quad w_t \sim \mathcal{N}(0, \sigma^2), \tag{17}$$

[¶] ϕ is obtained by concatenating m random features into a vector form.

[‖] We assume that, for outputs in different dimensions y_a and y_b, $p(y_a, y_b|x) = p(y_a|x)p(y_b|x)$.

where f is a random function given by the SSGP model. As in general $p(s_{t+1})$ can be quite complicated, in this work, we approximate the predictive distribution with a Gaussian distribution $p(s_{t+1}) \approx \mathcal{N}(\mu_{t+1}, \Sigma_{t+1})$ through linearizing the predictive mean function of the SSGP model. The moments of this approximate Gaussian predictive distribution can be represented as follows (Pan et al. 2017):

$$\begin{aligned} \mu_{t+1} &= \mu_t + \mathbb{E}[\Delta s_t] \\ \Sigma_{t+1} &= \Sigma_t + \mathrm{Var}[\Delta s_t] + \mathrm{Cov}(s_t, \Delta s_t) + \mathrm{Cov}(\Delta s_t, s_t). \end{aligned} \tag{18}$$

Equivalently, we can write the propagation of statistics in (18) in terms of the belief of state s_t. Define the belief as $b_t = [\mu_t \ \mathrm{vec}(\Sigma_t)]^\top$, where $\mathrm{vec}(\Sigma_t)$ is the vectorization of Σ_t, and denote the space of all beliefs by $\mathbb{B} \subset \mathbb{R}$. We can write (18) in a compact form as

$$b_{t+1} = \mathcal{F}(b_t, a_t), \tag{19}$$

where $\mathcal{F} : \mathbb{B} \times \mathbb{A} \to \mathbb{B}$ is the effective map by (18). This new equation corresponds to the belief-space representation of the dynamics; below we introduce a trajectory optimization method to obtain optimal actions based on the belief-space dynamics model in (19).

4.2 Trajectory Optimization

Solving (12) globally is notoriously difficult, requiring discretization of the state space $\mathbb{S}$ and incurs exponentially large complexity. In order to solve the optimal control problem efficiently, we use a trajectory optimization method, Differential Dynamic Programming (DDP) (Jacobson and Mayne 1970), which approximates the solution to (12) locally around a trajectory. To control the vehicle under model uncertainty, we use the belief-space dynamics model in (19). The instantaneous cost function $l : \mathbb{B} \times \mathbb{A} \to \mathbb{R}$ defined as $l(b, a) = \mathbb{E}_s[c(s, a)|b]$ where the cost $c(s, a)$ is designed to 1) keep the car close to the middle of the track, 2) travel at a target speed, 3) stabilize the car from slipping, and 4) minimize throttle, brake and steering efforts. The details will be described in Section 6.1.

DDP is an iterative method. At each iteration, it creates a local model along a nominal trajectory in the belief space including: 1) a linear approximation of the dynamics model; 2) a second-order approximation of the value function. Denote the belief and control nominal trajectory as $(\bar{b}_{1:T}, \bar{a}_{1:T})$ and deviations from this trajectory as $\delta b_t = b_t - \bar{b}_t$, $\delta a_t = a_t - \bar{a}_t$. The linear approximation of the belief dynamics along the nominal trajectory is

$$\delta b_{t+1} = \mathcal{F}_t^b \delta b_t + \mathcal{F}_t^a \delta a_t. \tag{20}$$

where $\mathcal{F}_t^b, \mathcal{F}_t^a$ are Jacobian matrices and the superscripts denote the variables involved in the partial derivatives**. Given the closed-form expression of $\mathcal{F}$, these derivatives can be evaluated efficiently without using numerical differentiation techniques. To propagate the value function, we construct a quadratic approximation of the instantaneous cost function l along the nominal belief and control trajectory, i.e.,

$$\begin{aligned} l(b_t, a_t) \approx\ & l_t^0 + (l_t^b)^\top \delta b_t + (l_t^a)^\top \delta a_t \\ & + \frac{1}{2} \begin{bmatrix} \delta b_t \\ \delta a_t \end{bmatrix}^\top \begin{bmatrix} l_t^{bb} & l_t^{bu} \\ l_t^{ub} & l_t^{uu} \end{bmatrix} \begin{bmatrix} \delta b_t \\ \delta a_t \end{bmatrix}, \end{aligned} \tag{21}$$

where $l_t^0 = l(\bar{b}_t, \bar{a}_t)$. Given the above local approximations of dynamics (20) and cost function (21), DDP creates a quadratic model of the value function

$$V_t^{\pi^\star}(b) = \min_{a_t \in \mathbb{A}} \left(l(b_t, a_t) + V_t^{\pi^\star}\left(\mathcal{F}(b_t, a_t)\right)\right) \tag{22}$$

$$\approx V_t^0 + (V_t^b)^\top \delta b_t + \frac{1}{2} \delta b_t^\top V_t^{bb} \delta b_t. \tag{23}$$

In order to compute V_t^0, V_t^b, V_t^{bb}, we define the term inside the min operator in (22) as the Q-function

$$Q_t^{\pi^\star}(b_t, a_t) = l(b_t, a_t) + V_t^{\pi^\star}\left(\mathcal{F}(b_t, a_t)\right).$$

Then we expand it up to the second order in δb and δa along $\bar{b}_t, \bar{a}_t$.

$$\begin{aligned} Q_t^{\pi^\star}(\bar{b}_t + \delta b_t, \bar{a}_t + \delta a_t) \approx\ & Q_t^0 + Q_t^b \delta b_t + Q_t^a \delta a_t \\ & + \frac{1}{2} \begin{bmatrix} \delta b_t \\ \delta a_t \end{bmatrix}^\top \begin{bmatrix} Q_t^{bb} & Q_t^{ba} \\ Q_t^{ab} & Q_t^{aa} \end{bmatrix} \begin{bmatrix} \delta b_t \\ \delta a_t \end{bmatrix}, \end{aligned} \tag{24}$$

and find an optimized control law by minimizing (24), i.e.,

$$\begin{aligned} \delta \hat{a}_t &= \arg\min_{\delta a_t} \left[Q(\bar{b}_t + \delta b_t, \bar{a}_t + \delta a_t) \right] \\ &= -(Q_t^{aa})^{-1}(Q_t^a + Q_t^{ab} \delta b_t). \end{aligned} \tag{25}$$

The quadratic model (22) of the value function V_t^0, V_t^b, V_t^{bb} can be computed in a backward pass by inserting the optimized control law $\hat{a}_t = \bar{a}_t + \delta \hat{a}_t$ into (24), see (Jacobson and Mayne 1970; Tassa et al. 2008) for details. Once the optimized control law along the entire nominal trajectory is computed through the backward pass, it is applied to the dynamics (19) to generate a new nominal trajectory in a forward pass. This backward-forward scheme is repeated for multiple iterations until convergence.

Unlike Quadratic programming (QP)-based approaches (Borrelli et al. 2005), our DDP-based approach is self-contained and does not rely on an external optimization solver. Compared to sampling-based method (Williams et al. 2017; Wagener et al. 2019) that uses massive forward simulations, our approach is more efficient as it exploits of the structure of the dynamics model (19).

5 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, 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 3.5.3, we assume the expert is algorithmic and has access to expensive sensors (GPS and IMU) for accurate global state estimates†† and

**We will use this superscript rule for dynamics and cost-related terms.

††Global position, heading and roll angles, linear velocities, and yaw rate.

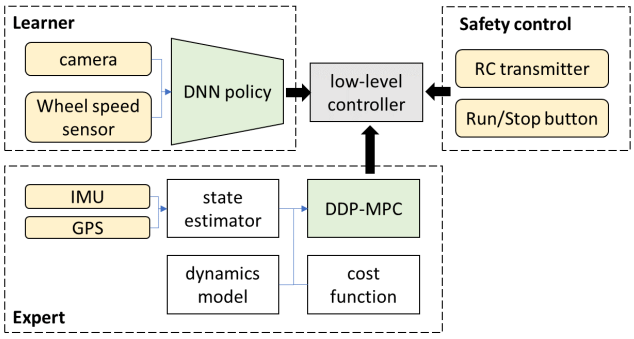


Figure 2. System diagram.

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 5.1). 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 5.2). 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 5.3), the IMU and the GPS sensors required by the expert in Section 5.1 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 to 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 Koenig and Howard (2004) was built (see Fig 4) using the same ROS interface but without the safety control module. Due to the limitation of the Gazebo simulator in terms of graphics quality and physics engine, we do not use simulated data to pre-train our control policy. The simulator was used to evaluate the performance of the software before real track tests.

5.1 Algorithmic Expert: Model-Predictive Control

We have introduced the algorithmic expert in section 4. This algorithm is able to approximate the solution to the original problem (1). However, in practice the task time horizon T may be very long (e.g., 1 minute) therefore solving the optimal problem (1) in a single pass is computationally inefficient and not robust to long-term prediction errors (given the fact that we never have enough data). Therefore we apply the algorithmic expert in a Model Predictive Control (MPC) fashion, which solves a shorter time horizon optimal problem at *every* sampling time: at time t, the expert policy $\pi^\star$ is a locally optimal policy such that

$$\pi^\star \approx \arg\min_\pi \mathbb{E}_{\rho_\pi}\left[\sum_{\tau=t}^{t+T_h} c(s_\tau, a_\tau)|s_t\right] \qquad (26)$$

where T_h is the length of horizon it previews. The computation is realized by the algorithm introduced in section 4, and iSAM2 Kaess et al. (2012) is used to estimate the vehicle states. Upon convergence, the algorithm 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. $a_t^* = \hat{a}_t$). When a new vehicle state is available from the state estimator, the state-control pair is incorporated to perform incremental regression as described in section 4.1.2. In order to ensure fast convergence, we use a 'warm-start' approach that initializes the nominal trajectory with the optimal control sequence obtained at the previous step. Empirically the trajectory optimization algorithm converges within 3 iterations with warm-start.

In view of the analysis in Section 3.2, we can assume that the MPC expert satisfies Definition 1, because it updates the approximate solution to the original RL problem (1) at a high-frequency using global state information. However, because MPC requires replanning for every time step, running the expert policy (26) on-the-fly consumes significantly more computational power than what is required by the learner.

5.2 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,[*] 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, (9) or (11), is solved using ADAM Kingma and Ba (2014), which is a stochastic gradient descent algorithm with an adaptive learning rate. Note while s_t or $s_t^\star$ is used in (9) or (11), 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. [†] The only pre-processing was scaling and cropping of raw images.

[‡‡] It might be possible to build a model that maps raw observations to state estimates, though, in general, a model may need to consider also the temporal dependency that the current state depends on the previous state. Learning such a mapping faces several challenges, however. We did not choose to do so because we wanted to completely remove the use of the expert policy, which requires real-time trajectory optimization.

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

[†] Data collection or augmentation techniques such as Geist et al. (2017); Bojarski et al. (2016) can be used in conjunction with our method.

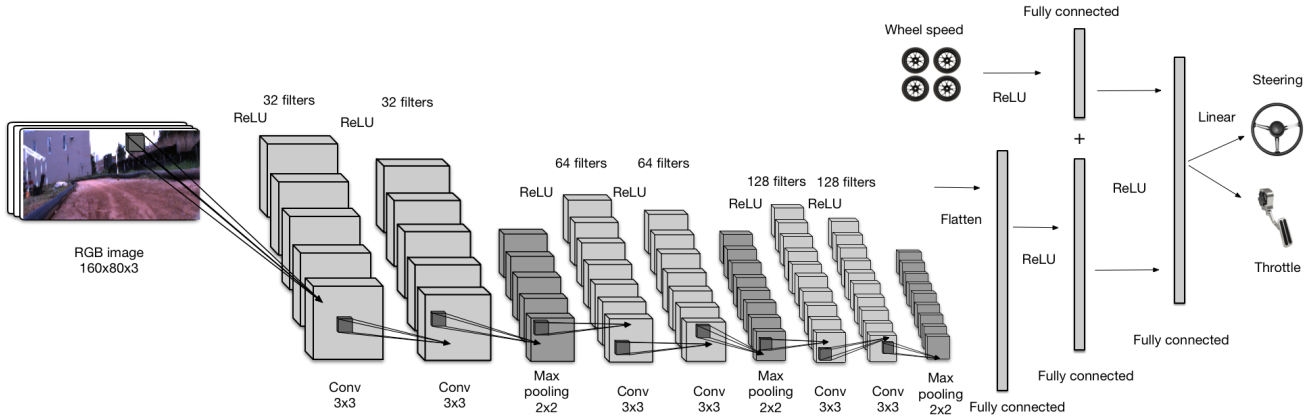


Figure 3. The DNN control policy.

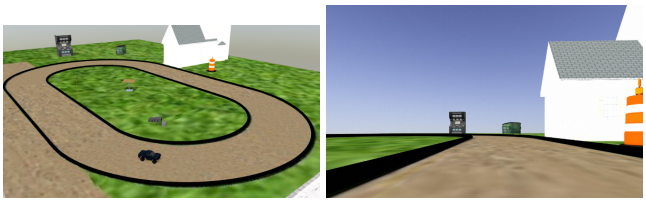


Figure 4. The Gazebo-based simulation enviorment (left) and a snapshot from the on-board camera (right).

Figure 5. The AutoRally car and the test track.

5.3 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×0.6m×0.4m), shown in the top figure in Fig. 5. 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,[‡] 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 Bojarski et al. (2017) (10 Hz), Rausch et al. (2017) (12 Hz), and Muller et al. (2006) (15 Hz).

6 Experimental Setup

6.1 High-speed Driving Task

We tested the performance of the proposed IL system in Section 5 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 cost function in the finite-horizon RL problem (1) with

$$c(s_t, a_t) = \alpha_1 c_{\text{pos}} + \alpha_2 c_{\text{spd}} + \alpha_3 c_{\text{slip}} + \alpha_3 c_{\text{act}}, \qquad (27)$$

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.

The position cost c_{pos} for the high-speed navigation task is a 16-term cubic function of the vehicle's global position (x, y):

$$\begin{aligned} c_{\text{pos}} = c_0 + c_1 y + c_2 y^2 + c_3 y^3 + c_4 x + c_5 xy \\ + c_6 xy^2 + c_7 xy^3 + c_8 x^2 + c_9 x^2 y + c_{10} x^2 y^2 + c_{11} x^2 y^3 \\ + c_{12} x^3 + c_{13} x^3 y + c_{14} x^3 y^2 + c_{15} x^3 y^3. \end{aligned}$$

The coefficients $c_0, .., c_{15}$ in this cost function were identified by performing a regression to fit the track's boundary: First, a thorough GPS survey of the track was taken. Points along the inner and the outer boundaries were assigned values of -1 and $+1$, respectively, resulting in a zero-cost path along the center of the track. The coefficient values c_i were then determined by a least-squares regression of the polynomials in c_{pos} to fit the boundary data. The speed cost $c_{\text{spd}} = \|v_x - v_{\text{desired}}\|^2$ is a quadratic function which penalizes the difference between the desired speed v_{desired} and the longitudinal velocity v_x in the body frame. The side slip angle cost is defined as $c_{\text{slip}} = -\arctan(\frac{v_y}{\|v_x\|})$, where v_y is the lateral velocity in the body frame. The action cost is a quadratic function defined as $c_{\text{act}} = \gamma_1 a_1^2 + \gamma_2 a_2^2$, where a_1 and a_2 correspond to the steering and the throttle commands, respectively. In the experiments, $\gamma_1 = 1$ and $\gamma_2 = 1$ were selected.

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

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

information (27) 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 (27) were set as $\alpha_1 = 2.5$, $\alpha_2 = 1$, $\alpha_3 = 100$ and $\alpha_4 = 60$, so that the MPC expert in Section 5.1 could perform reasonably well. The learner's policy was tuned by online/batch IL in attempts to match the expert's performance.

6.2 Test Track

All the experiments were performed on an elliptical dirt track, shown in the bottom figure of Fig. 5, with the AutoRally car described in Section 5.3. The test track was ∼3m wide and ∼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.

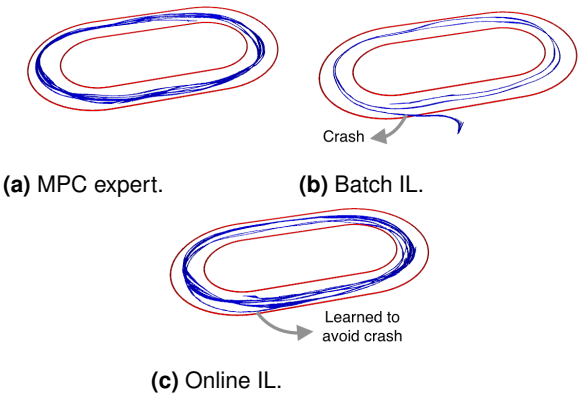
6.3 Data Collection

For dynamics model learning, data was collected in two phases: first, a human driver drove the vehicle at various speeds (3 - 6 m/s) for 10 minutes, during which the state and action data were recorded, processed with spline smoothing, and re-sampled. A SSGP model of the vehicle dynamics was learned offline using the dataset (see Section 4.1 for details). Next, when the MPC expert (based on the learned dynamics model) was executed on the vehicle, we continued to collect new state-action data of transition dynamics and incorporated them to perform incremental updates, as described in Section 4.1.2.

For IL, 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 $\hat{\pi}_i = \beta^i \pi^\star + (1 - \beta^i)\pi_{i-1}$, 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 Ross et al. (2011); Cheng and Boots (2018) 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.

6.4 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.



(a) MPC expert. **(b)** Batch IL. **(c)** Online IL.

Figure 6. 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.

At each training phase, the optimization problem (9) or (11) 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 5.2 for details. Finally, after the entire learning session of a policy, three rollouts were performed using the learned policy for performance evaluation.

7 Experimental Results

7.1 Algorithmic Expert vs Human Expert

First, we study the performance of the algorithmic expert. For comparison the same task was demonstrated by a human driver using a remote controller. We use speed (the faster the better) as the metric for both MPC and human driver. Other metrics such as cross-track error may not be intuitive because MPC and human driver have different objectives. For example, MPC does path following while the human driver does lane keeping, e.g., it is more desirable to not follow the center of the lane during cornering. Statistics for both MPC and human expert are shown in Table 2. Results were averaged over 3 independent trials. The MPC expert outperforms the human expert significantly in terms of speed (our target speed is 7.5 m/s).

While a professional race car driver could be better at utilizing the tire force potential than a hand-crafted controller (Laurense et al. 2017), the MPC expert performs better, because, in our case, the human driver controls the vehicle via a transmitter in a third-person view, which results in a delayed response and differences in terms of sensing and handling capabilities. In addition, human drivers can be problematic for online imitation learning tasks due to the lack of instantaneous feedback from the vehicle caused by his or her own actions (as we discussed in Section 3.5.3 and at the beginning of Section 4). Labeling the actions frame-by-frame offline (Ross et al. 2013) is not possible because of the continuous throttle and steering commands. In the

following experiments we focus on comparing batch and online imitation learning with the MPC expert.

7.2 Empirical Performance

Next we study the performance of training a control policy with online and batch IL algorithms. Fig. 6 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. 7 shows the performance in terms of distance traveled without crashing (we used the safe control module shown in Fig. 2 to manually terminate the rollout when the car crashed into the soft boundary) and Table 2 shows the statistics of the experimental results. Overall, DNN policies trained with both 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. (2016). The discrepancy can be explained with a recent theoretical analysis by Cheng and Boots (2018); Lee et al. (2018); Cheng et al. (2019a), 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 2). 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.

7.3 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) (Maaten and Hinton 2008), as shown in Fig. 8 and Fig. 9. 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. For the online setting, the train data include the data in all DAgger iterations; for the batch

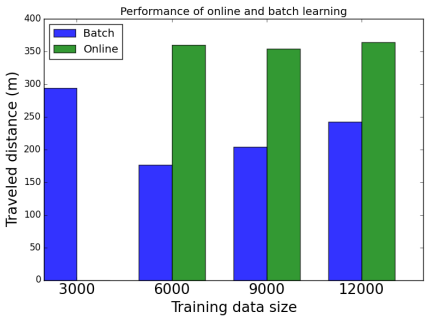


Figure 7. Performance of online and batch IL in the distance (meters) traveled without crashing. The policy trained with a batch of 3,000 samples was used to initialize online IL.

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.

We first observe in Fig. 8 that, while the wheel speed data have similar training and testing distributions, the raw image 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 (Shimodaira 2000), which can significantly complicate the learning process.

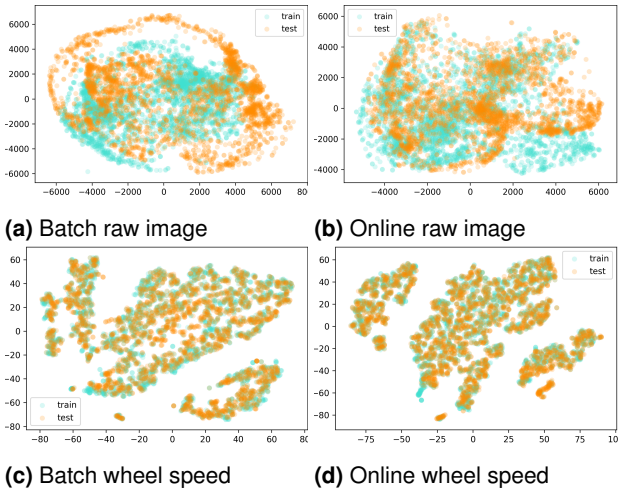
The policy trained with online IL yet still demonstrated great performance in the experiments. To further understand how it could generalize across different image distributions, we embed its feature distribution in Fig. 9 (a) and (b). The feature here are the last hidden layer of the neural network; namely, the output layer is a linear function of the features. In comparison with the raw images, these abstract features (e.g. lane boundary, building shown in Fig. 10) extracted by the encoder CNN ideally can be more invariant across different situations.

Interestingly, despite the difference in the raw image distributions in Fig. 8 (a) and (b), the DNN policy trained with online IL are able to map the train and test data to similar feature distributions, as shown in Fig. 9 (a) and (b). An insight to this is that the online IL algorithm (e.g. DAgger) forces the DNN to learn a set of features such that a linear combination (the last layer) of those features is sufficient to represent a good policy for a range of distributions generated during the interactive training process. In other words, the online IL paradigm effectively makes the DNN face a multi-task learning situation: it must find an invariant feature embedding that is admissible to the use of linear policies. This explains the coherency between Fig. 9 (a) and (b), compared with Fig. 8 (a) and (b).

On the contrary, the DNN policy trained with batch IL fails to learn a coherent feature embedding, as shown in Fig. 9 (c) and (d). (They are still better than Fig. 8 (a) and (b), but worse than Fig. 9 (a) and (b).) Based on the discussion above, this is because the DNN only needs to work well on the distribution visited by the expert policy, which is comparably simpler

Table 2. Test statistics. Total loss denotes the imitation loss in (8), 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.

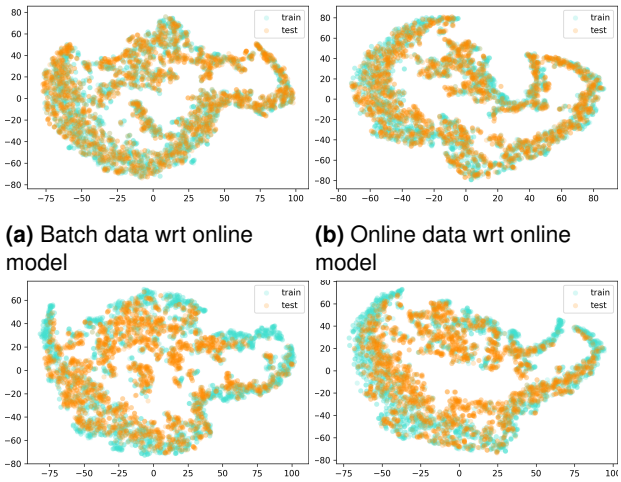
Policy	Avg. speed	Top speed	Training data	Completion ratio	Total loss	Steering/Throttle loss
Expert	6.05 m/s	8.14 m/s	N/A	100 %	0	0
Expert (human)	5.09 m/s	6.1 m/s	N/A	100 %	0	0
Batch	4.97 m/s	5.51 m/s	3000	100 %	0.108	0.092/0.124
Batch	6.02 m/s	8.18 m/s	6000	51 %	0108	0.162/0.055
Batch	5.79 m/s	7.78 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



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

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

Figure 8. The distributions (t-SNE) of the raw images and wheel speed used as DNN policy's inputs (details in Section 7.3).



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

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

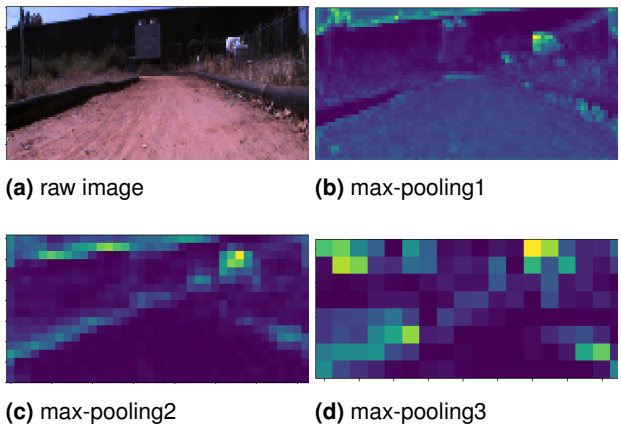
Figure 9. The distributions (t-SNE) of the learned DNN feature in the last fully-connected layer (details are in Section 7.3).

(an analogy to single-task learning). This could explain the inferior performance of batch IL, and its inability to deal with the corner case in Fig. 6 (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.

7.4 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. 10 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 6.3. Fig. 11 shows the batch IL loss in (11) of different network



(a) raw image **(b)** max-pooling1

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

Figure 10. The input RGB image and the averaged feature maps for each max-pooling layer.

architectures. The full DNN policy in Fig. 3 achieved better performance 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. Bojarski et al. (2017)) cannot be directly used to perform both lateral and longitudinal controls, as we do here. By contrast, while

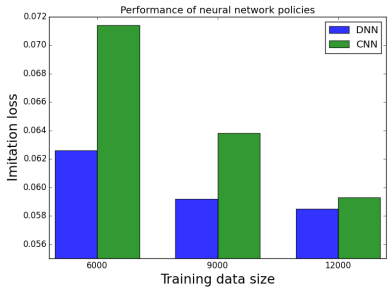


Figure 11. Performance comparison between our DNN policy and its CNN sub-network in terms of batch IL loss, where the horizontal axis is the size of data used to train the neural network policies.

without a recurrent architecture, our DNN policy learned to combine wheel speeds in conjunction with CNN to infer hidden state and achieve better performance. Recent work has shown that recurrent neural networks (such as LSTM) can be used to predict speed (Yang et al. 2018) or acceleration commands (Yu et al. 2017). However, a lower-level control module is required to compute the throttle commands, therefore the learned policy is not end-to-end. Incorporating recurrent structures into our imitation learning framework could be a interesting extension of this work and is left to future research.

8 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

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

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.

Francesco Borrelli, Paolo Falcone, Tamas Keviczky, Jahan Asgari, and Davor Hrovat. Mpc-based approach to active steering for autonomous vehicle systems. *International Journal of Vehicle Autonomous Systems*, 3(2):265–291, 2005.

Ching-An Cheng and Byron Boots. Variational inference for Gaussian process models with linear complexity. In *Advances in Neural Information Processing Systems*, 2017.

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.

Ching-An Cheng, Xinyan Yan, Nolan Wagener, and Byron Boots. Fast policy learning through imitation and reinforcement. In *Conference on Uncertainty in Artificial Intelligence*, 2018.

Ching-An Cheng, Jonathan Lee, Ken Goldberg, and Byron Boots. Online learning with continuous variations: Dynamic regret and reductions. *arXiv preprint arXiv:1902.07286*, 2019a.

Ching-An Cheng, Xinyan Yan, Nathan Ratliff, and Byron Boots. Predictor-corrector policy optimization. In *International Conference on Machine Learning*, 2019b.

Ching-An Cheng, Xinyan Yan, Evangelos A Theodorou, and Byron Boots. Accelerating imitation learning with predictive models. In *International Conference on Artificial Intelligence and Statistics*, 2019c.

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.

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.

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

Arjan Gijsberts and Giorgio Metta. Real-time model learning using incremental sparse spectrum gaussian process regression. *Neural Networks*, 41:59–69, 2013.

James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013.

David H Jacobson and David Q Mayne. Differential dynamic programming. 1970.

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.

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.

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

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

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.

Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099. IEEE, 2015.

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.

Vincent A Laurense, Jonathan Y Goh, and J Christian Gerdes. Path-tracking for autonomous vehicles at the limit of friction. In *2017 American Control Conference (ACC)*, pages 5586–5591, 2017.

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.

Jonathan Lee, Michael Laskey, Ajay Kumar Tanwani, Anil Aswani, and Ken Goldberg. A dynamic regret analysis and adaptive regularization algorithm for on-policy robot imitation learning. *arXiv preprint arXiv:1811.02184*, 2018.

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.

Lennart Ljung. System identification. In *Signal analysis and prediction*, pages 163–173. Springer, 1998.

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

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.

Igor Mordatch and Emo Todorov. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems*, pages 5–32, 2014.

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.

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.

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.

Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntak Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile autonomous driving via end-to-end deep imitation learning. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

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

Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

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.

Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

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.

Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya 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.

Simon J Rutherford and David J Cole. Modelling nonlinear vehicle dynamics with neural networks. *International journal of vehicle design*, 53(4):260–287, 2010.

Hugh Salimbeni, Ching-An Cheng, Byron Boots, and Marc Deisenroth. Orthogonally decoupled variational gaussian processes. *arXiv preprint arXiv:1809.08820*, 2018.

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

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.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, 2006.

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

Michalis Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, 2009.

Mnih Volodymyr, Kavukcuoglu Koray, Silver David, A Rusu Andrei, and Veness Joel. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Nolan Wagener, Ching-An Cheng, Jacob Sacks, and Byron Boots. An online learning approach to model predictive control. In *Robotics: Science and Systems*, 2019.

C.K.I Williams and C.E. Rasmussen. *Gaussian processes for machine learning*. MIT Press, 2006.

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.

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*, pages 1714–1721, 2017.

Zhengyuan Yang, Yixuan Zhang, Jerry Yu, Junjie Cai, and Jiebo Luo. End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perception. *arXiv preprint arXiv:1801.06734*, 2018.

Hao Yu, Shu Yang, Weihao Gu, and Shaoyu Zhang. Baidu driving dataset and end-to-end reactive control model. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 341–346, 2017.

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