Learning Deep Neural Network Control Policies for Agile Off-Road Autonomous Driving

Yunpeng Pan[*]
JD.com, Inc.
Santa Clara, CA 95054

Ching-An Cheng[†] **Kamil Saigol**[†] **Keuntaek Lee**[‡]
Xinyan Yan[†] **Evangelos Theodorou**[§] **Byron Boots**[¶]
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

We present an end-to-end learning system for agile, off-road autonomous driving using only low-cost on-board sensors. By imitating an optimal controller, we train a deep neural network control policy to map raw, high-dimensional observations to continuous steering and throttle commands, the latter of which is essential to successfully drive on varied terrain at high speed. Compared with recent approaches to similar tasks, our method requires neither state estimation nor online planning to navigate the vehicle. Real-world experimental results demonstrate successful autonomous driving, matching the state-of-the-art performance.

1 Introduction

High-speed autonomous off-road driving is a challenging robotics problem [1, 2, 3]. 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 must reason about unstructured, stochastic natural environments and operate at high speed. Consequently, designing a control policy by following the traditional model-plan-then-act approach [1, 4] 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. [2, 3] 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.

We aim to relax these requirements by designing a reflexive driving policy that uses only *low-cost, on-board* sensors (e.g. camera, wheel speed sensors). Building on the success of deep reinforcement learning (RL) [5, 6], 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 [6], 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.

[*]JD-X Robotics Research Center. `yunpeng.pan@jd.com`
[†]Institute for Robotics and Intelligent Machines. `{cacheng, kamilsaigol, xyan43}@gatech.edu`
[‡]School of Electrical and Computer Engineering. `keuntaek.lee@gatech.edu`
[§]School of Aerospace Engineering. `evangelos.theodorou@gatech.edu`
[¶]Institute for Robotics and Intelligent Machines. `bboots@cc.gatech.edu`

Table 1: Comparison of our method to prior work on end-to-end learning for autonomous driving

Methods	Tasks	Observations	Action	Algorithm	Experiment
[7]	On-road low-speed	Single image	Steering	Batch	Real &simulated
[8]	On-road low-speed	Single image & laser	Steering	Batch	Real &simulated
[9]	On-road low-speed	Single image	Steering	Batch	Simulated
[10]	Off-road low-speed	Left & right images	Steering	Batch	Real
[11]	On-road unknown speed	Single image	Steering + break	Online	Simulated
Our Method	Off-road high-speed	Single image + wheel speeds	Steering + throttle	Batch & online	Real & simulated

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* [8] to optimize the control policy instead. A major benefit of using imitation learning 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. the MPC controller using a pre-trained dynamics model and a state estimator based on high-end sensors in [2]) usually requires expensive sensors and dedicated computational resources, with imitation learning we can train a lower-cost robot to behave similarly, without carrying the expert's hardware burdens over to the learner. Note that 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 [12] where the expert can be modified to accommodate to the learning progress.

In this work, we present an end-to-end learning system for real-world high-speed off-road driving tasks. By leveraging demonstrations from an algorithmic expert, our system can learn a control policy that achieves similar performance as the expert. The system was implemented on a 1/5-scale autonomous rally car. In real-world experiments, we show that our rally car—without any state estimator or online planning, but with a DNN policy that directly inputs sensor measurements from a low-cost monocular camera and wheel speed sensors—could learn to perform high-speed navigation at an average speed of $\sim$6 m/s and a top speed of $\sim$8 m/s, matching the state of the art performance [3].

Related Work: End-to-end learning for self-driving cars has been explored since late 1980's. Autonomous Land Vehicle in a Neural Network (ALVINN) [8] 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 [10, 7, 9] have recently started to employ a batch imitation learning approach: parameterizing control policies with DNNs, these systems require only human driver demonstrations during the training phase and on-board measurements during the testing phase. For example, Nvidia's PilotNet [7], a convolutional neural network that outputs steering angle given an image, was trained to mimic human drivers' reaction to visual input with demonstrations collected in real-world road tests. A Dataset Aggregation (DAgger) [13] related online imitation learning algorithm for autonomous driving was recently demonstrated in [11], but only considered simulated environments.

Our problem differs substantially from these previous 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 a high speed, and prominent visual features such as lane markers are absent. Compared with paved roads, the surface of our dirt tracks are constantly evolving and highly stochastic. As a result, to successfully perform high-speed driving in our task, high-frequency decision and execution of both steering and throttle commands are required. Previous work only focuses on steering commands [10, 7, 9]. A comparison of different imitation learning approaches to autonomous driving is presented in Table 1.

Our task is most similar to the task considered by Williams et al. [2, 3] and Drews et al. [14]. 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 and the learning component is not end-to-end. In [2, 3], 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 [14], 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 low cost on-board sensors and its control policy can run reactively in high frequency, without on-the-fly planning and optimization. Additionally, our experimental results match that in [2, 3] and are faster and more data efficient than that in [14].

2 From Reinforcement Learning to Imitation Learning

To design a policy for off-road autonomous driving, we formulate policy optimization as a reinforcement learning problem and then show how a policy can be learned via imitation learning.

2.1 Problem Definition

To mathematically formulate the autonomous driving task, it is natural to 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 deterministic policy $\pi : \mathbb{O} \mapsto \mathbb{A}$ (e.g. a DNN policy) such that π achieves low accumulated cost over a finite horizon of length T

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

in which ρ_π is the distribution of $s_t \in \mathbb{S}$, $o_t \in \mathbb{O}$, and $a_t \in \mathbb{A}$ under policy $a_t = \pi(o_t)$, for $t = 1 \dots T$. Here $c(s_t, a_t)$ is the instantaneous cost, which, for example, encourages maximal speed driving while staying on the track. For notations, we denote $Q_\pi^t(s, a)$ as the Q-function at time t under policy π and $V_\pi^t = \mathbb{E}_{a \sim \pi}[Q_\pi^t(s, a)]$ as its associated value function.

2.2 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 the policy π by imitation learning. We assume the access to an oracle policy or *expert* π^* to generate demonstrations during the training phase, which relies on resources that are unavailable in the testing phase, e.g. additional sensors and computation. For example, the expert can be a computationally intensive optimal controller that relies on exteroceptive sensors not available at test time (e.g. GPS for state estimation), or an experienced human driver.

Online imitation learning [13] is a recent approach to learn policies that can perform as well as the expert with an error that has at most linear dependency on T. Its objective function for continuous action spaces (as required in the autonomous driving task) is as follows:

$$\min_{\pi} \mathbb{E}_{\rho_\pi} \left[\sum_{t=1}^{T} \hat{c}(s_t, a_t) \right]. \tag{2}$$

in which π^* is an expert to (1), and $\hat{c}(s_t, a_t) = \mathbb{E}_{a_t^* \sim \pi^*}[||a_t - a_t^*||]$. Solving this objective function can be shown to minimize an upper bound of $J(\pi) - J(\pi^*)$. The details of the derivation is given in Appendix, which simplifies the derivation in [13] and extends it to continuous action spaces as required in the autonomous driving task.

This surrogate problem is comparatively more structured than the original RL problem (1), so we can adopt algorithms with provable performance guarantees. In this paper, we use the meta-learning algorithm DAgger [13], which reduces (11) to a sequence of supervised learning problems: Let $\mathcal{D}$

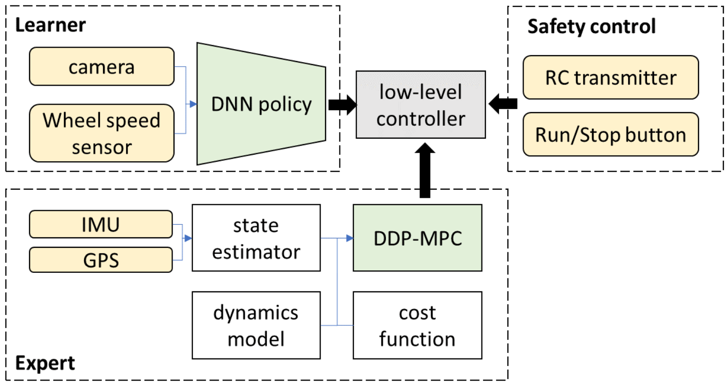


Figure 1: System diagram

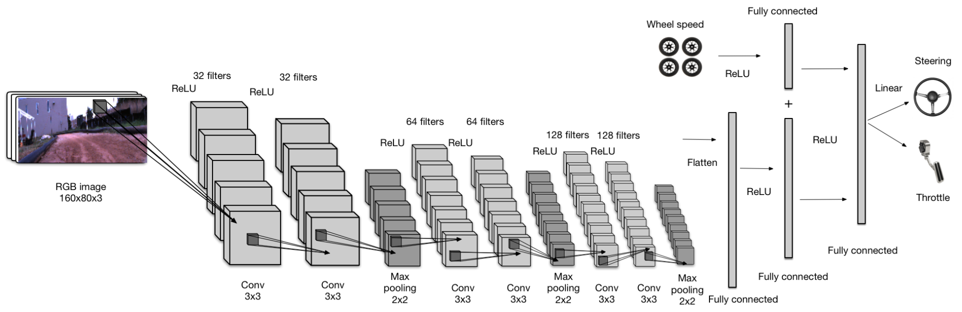


Figure 2: The DNN control policy

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,

$$\pi_i = \arg\min_{\pi} \mathbb{E}_{\mathcal{D}}[\hat{c}(s_t, a_t)], \tag{3}$$

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 (11), is returned. Suppose the policy is linearly parametrized and $o_t = x_t$. Since our instantaneous cost $\hat{c}(s_t, \cdot)$ is strongly convex, the theoretical analysis of DAgger applies. Therefore, running DAgger to solve (11) finds a policy π with performance $J(\pi) \leq J(\pi^*) + O(T)$, achieving our initial goal.

In contrast to the surrogate problem in online imitation learning (11), *batch* imitation learning [8, 10, 7, 9] is a standard supervised learning problem

$$\min_{\pi} \mathbb{E}_{\rho_{\pi^*}} \left[\sum_{t=1}^{T} \tilde{c}_\pi(s_t^*, a_t^*) \right], \tag{4}$$

where the expectation is defined by a fixed policy π^*. Further analysis of online and batch learning approaches to imitation learning can be found in Appendix.

3 The Autonomous Driving System

Building on the analyses in the previous section, 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. 1. 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.

We assume the expert is algorithmic and has access to expensive sensors (GPS and IMU) for accurate global state estimates[6] 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

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

of the task, and a trajectory optimization algorithm for planning (see Section 3.1). 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 3.2). In this setting, the sensors that the learner uses can be significantly cheaper than that of the expert; specifically on our experimental platform, the AutoRally car (see Section 3.3), the IMU and the GPS sensors required by the expert in Section 3.1 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 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 [15] was built using the same ROS interface but without the safety control module; the simulator was used to evaluate the performance of the system before real track tests.

3.1 Algorithmic Expert with Model-Predictive Control

We use an MPC expert [16] based on an incremental Sparse Spectrum Gaussian Process (SSGP) dynamics model (which was learned from 30 minute-long driving data) and an iSAM2 state estimator [17]. To generate actions, the MPC expert solves a finite horizon optimal control problem for every sampling time: at time t, the expert policy $\pi^*(a_t|s_t)$ is a locally optimal policy such that

$$\pi^*(a_t|s_t) \approx \arg\min_{\pi} \mathbb{E}_{\rho_\pi}\left[\sum_{\tau=t}^{t+T_h} c(s_\tau, a_\tau)|s_t\right] \tag{5}$$

where T_h is the length of horizon it previews.

The computation is realized by the trajectory optimization algorithm, Differential Dynamic Programming (DDP) [18]: 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.

3.2 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. 2, 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[7], 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 convolutonal 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 imitation learning, the action space $\mathbb{A}$ is equipped with $\|\cdot\|_1$ for filtering outliers, and the optimization problem defined in (2.2) is solved using ADAM [19], which is a stochastic gradient descent algorithm with an adaptive learning rate. 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 raw images.

3.3 The Autonomous Driving Platform

To validate our imitation learning 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. 3. The car was equipped with an ASUS mini-ITX

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

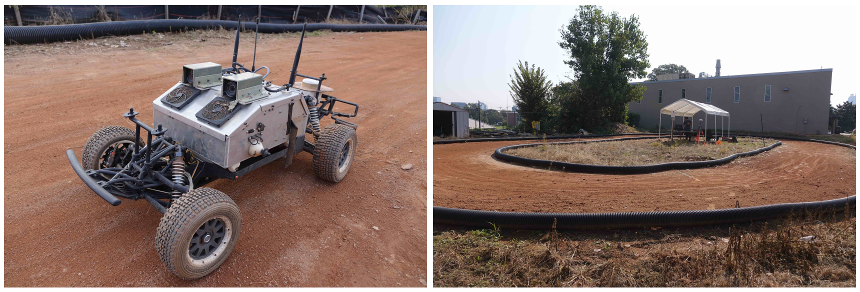


Figure 3: The AutoRally car and the test track.

motherboard, an Intel quad-core i7 CPU, 16GB RAM, a Nvidia GTX 750ti GPU, and a 11000mAh battery. For sensors, two forward facing machine vision cameras[8], 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 [7] (10 Hz), [9] (12 Hz), and [10] (15 Hz).

4 Experimental Setup

High-speed Navigation Task: We tested the performance of the proposed imitation learning system in Section 3 in a high-speed navigation task with a desired speed of 7.5 m/s. The performance index of the task was formulated as the cost function in the finite-horizon RL problem (1) with

$$c(s_t, a_t) = \alpha_1 \text{cost}_{\textbf{pos}}(s_t) + \alpha_2 \text{cost}_{\textbf{spd}}(s_t) \quad + \alpha_3 \text{cost}_{\textbf{slip}}(s_t) + \alpha_3 \text{cost}_{\textbf{act}}(a_t), \tag{6}$$

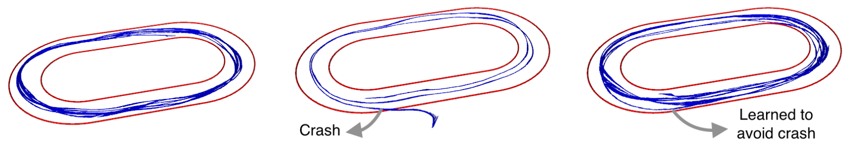
in which $cost_{\textbf{pos}}$ favors the vehicle to stay in the middle of the track, $cost_{\textbf{spd}}$ drives the vehicle to reach the desired speed, $cost_{\textbf{slip}}$ stabilizes the car from slipping, and $cost_{\textbf{act}}$ inhibits large control commands (see Appendix for details).

The goal of the high-speed navigation 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 $T = 3,000$. The cost information (6) was given to the MPC expert in Fig. 1 to perform online trajectory optimization with a two-second preview horizon (i.e. $T_h = 100$). In the experiments, the weighting in (6) were set as $\alpha_1 = 2.5$, $\alpha_2 = 1$, $\alpha_3 = 100$ and $\alpha_4 = 60$, so that the MPC expert in Section 3.1 could perform reasonably well. The learner's policy was tuned by online/batch imitation learning in attempts to match the expert's performance.

Test Track: All the experiments were performed on an elliptical dirt track, shown in the bottom figure of Fig. 3, with the AutoRally car described in Section 3.3. 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.

Data Collection: Training data was collected in two ways. In batch imitation learning, the MPC expert was executed, and the camera images, wheel speed readings, and the corresponding steering and throttle commands were recorded. In online imitation learning, 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 [13] 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

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



(a) MPC expert (b) Batch learning (c) Online learning

Figure 4: Examples of vehicle trajectories. (a) Demonstration of the MPC expert. (b) Crashing case when using batch imitation learning. (c) The vehicle avoids crashing when using online imitation learning. Subfigures (b) and (c) depict test runs of the policies after training on 9,000 samples

Table 2: Test statistics. Total loss denotes the imitation loss in (11), which is the average of the steering and the throttle losses. Completion ratio 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	NA	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

on an outdoor track, and consisted on 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.

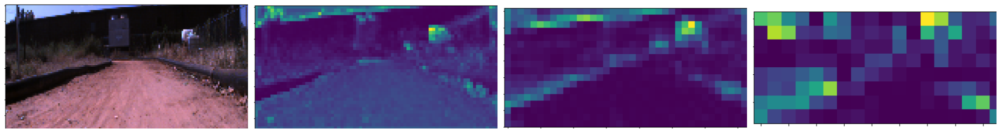
Policy Learning: In online imitation learning, three iterations of DAgger were performed. At each iteration, the robot executed one rollout using the mixed policy described above (the probabilities of executing the expert policy were 60%, 36%, and 21%, respectively). For a fair comparison, the amount of training data collected in batch imitation learning was the same as all of the data collected over the three iterations of online imitation learning. At each training phase, the optimization problem 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 3.2 for details. Finally, after the entire learning session of a policy, three rollouts were performed using the learned policy for performance evaluation.

5 Experimental Results

Online vs Batch Learning: We first study the performance of training a control policy with online and batch imitation learning algorithms. Fig. 4 illustrates the vehicle trajectories of different policies. Due to accumulating errors, the policy trained with batch imitation learning crashed into the lower-left boundary, an area of the state space-action rarely explored in the expert's demonstrations. On the contrary, online imitation learning let the policy learn to successfully cope with corner cases as the learned policy occasionally ventured into new areas of the state-action space.

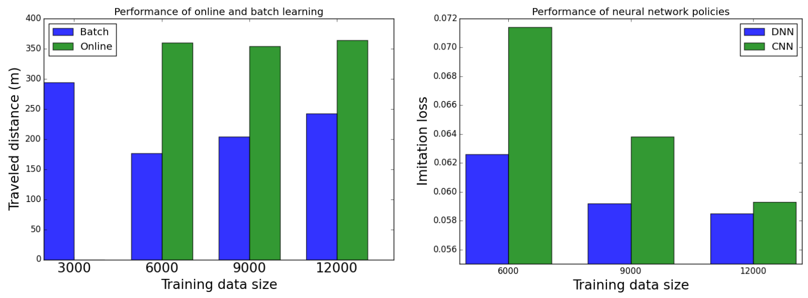
Fig. 6a shows the performance in terms of distance traveled without crashing[9] and Table 2 shows the statistics of the experimental results. Overall, DNN policies trained with both online and batch imitation learning algorithms were able to achieve a similar speed as the MPC expert. However, with the same amount of training data, the policies trained with online imitation learning in general outperformed those trained with batch imitation learning. In particular, the policies trained using online imitation learning achieved better performance in terms of both completion ratio and imitation loss. It is worth noting that the traveled distance of the policy learned with a batch of 3,000 samples was longer than that of other batch learning policies. As shown in Table 2, this is mainly because this policy achieved better steering performance than throttle performance. As a result, although the vehicle was able to navigate without crashing, it actually traveled at a much slower speed. By

[9]We used the safe control module shown in Fig. 1 to manually terminate the rollout when the car crashed into the soft boundary.



(a) raw image (b) max-pooling1 (c) max-pooling2 (d) max-pooling3

Figure 5: From left to right: input RGB image, averaged feature maps for each max-pooling layer.



(a) (b)

Figure 6: (a) Performance of online and batch imitation learning in the distance (meters) traveled without crashing. The policy trained with a batch of 3,000 samples was used to initialize online imitation learning. (b) Performance comparison between our DNN policy and its CNN sub-network in terms of batch imitation learning loss, where the horizontal axis is the size of data used to train the neural network policies.

contrast, the batch learning policies that used more data had better throttle performance and worse steering performance, resulting in faster speeds but higher chances of crashing.

Deep Neural Network Policy: One main feature 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. 5 the averaged feature map at each max-pooling layer (see Fig. 2), where each pixel represents the averaged unit activation across different filter outputs. We can observe that at a deeper level, the detected salient objects are boundaries of the track and parts of a building. Grass and dirt contribute little to the DNN's output.

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 imitation learning. The data was collected in accordance with Section 4. Fig. 6b shows the batch imitation learning loss of different network architectures. The full DNN policy in Fig. 2 achieved better performance consistently. While images contain position information, it is insufficient to infer velocities. Therefore, we conjecture state-of-the-art CNNs (e.g. [7]) cannot be directly used in this task. 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.

6 Conclusion

We introduce an end-to-end learning system to learn deep neural network driving policies that map raw on-board observations to steering and throttle commands by mimicking an expert's behavior. In real-world experiments, our system was able to perform fast off-road navigation autonomously at an average speed of $\sim$6 m/s and a top speed of $\sim$8 m/s, while only using low-cost monocular camera and wheel speeds sensors. Our current and future work include developing more complex policy representations, such as recurrent neural networks, and to improve robustness to visual distractions.

References

[1] 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.

[2] 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.

[3] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James Rehg, Byron Boots, and Evangelos Theodorou. Information theoretic mpc for model-based reinforcement learning. In *IEEE Conference on Robotics and Automation*, 2017.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] Nathan Koenig and Andrew Howard Design. Use paradigms for gazebo, an open-source multi-robot simulator ieee. In *IEEE International Conference on Intelligent Robots and Systems*, 2004.

[16] 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.

[17] 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.

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

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

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

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

[22] 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.

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

[24] 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.

A 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 the policy π by imitation learning. We assume the access to an oracle policy or *expert* π^* to generate demonstrations during the training phase, which relies on resources that are unavailable in the testing phase, e.g. additional sensors and computation. For example, the expert can be a computationally intensive optimal controller that relies on exteroceptive sensors not available at test time (e.g. GPS for state estimation), or an experienced human driver.

The goal of imitation learning 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 [20] and define what we mean by an *expert*.

Lemma 1. *Define $d_\pi(s) = \sum_{t=0}^{T-1} d_\pi^t(s)$ as an unnormalized stationary state distribution, where d_π^t is the distribution of state at time t when running policy π. Let π and π' be two policies. Then*

$$J(\pi) = J(\pi') + \mathbb{E}_{s\sim d_\pi}\mathbb{E}_{a\sim\pi}[A_{\pi'}^t(s,a)] \tag{7}$$

where $A_{\pi'}^t(s,a) = Q_{\pi'}^t(s,a) - V_{\pi'}^t(s)$ is the advantage function at time t with respect to running π'.

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

The idea behind Definition 1 is that an expert policy π^* should perform stably under arbitrary action perturbation with respect to the cost function $c(\cdot,\cdot)$, regardless of where it starts.[10] As we will see in Section A.3, this requirement provides guidance for whether to choose batch learning vs. online learning to train a policy by imitation.

A.1 Online Imitation Learning

We now present the objective function for the online learning [21] approach to imitation learning, which simplifies the derivation in [13] and extends it to continuous action spaces as required in the autonomous driving task. Although our goal here is not to introduce a new algorithm, but rather to give a concise introduction to online imitation learning, we found that a connection between online imitation learning and DAgger-like algorithms [13] in continuous domains has not been formally introduced. DAgger has only been used heuristically in these domains as in [22, 11].

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 [23]: for two probability distributions p and q defined on a metric space $\mathcal{M}$ with metric d,

$$\begin{aligned} D_W(p,q) &:= \sup_{f:\mathrm{Lip}(f(\cdot))\le 1} \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)] & (8)\\ &= \inf_{\gamma\in\Gamma(p,q)} \int_{\mathcal{M}\times\mathcal{M}} d(x,y)d\gamma(x,y), & (9)\end{aligned}$$

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 [23]. 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:

$$\begin{aligned} &J(\pi) - J(\pi^*)\\ &= \mathbb{E}_{s_t\sim d_\pi}\left[\mathbb{E}_{a_t\sim\pi}[Q_{\pi^*}^t(s_t,a_t)] - \mathbb{E}_{a_t^*\sim\pi^*}[Q_{\pi^*}^t(s_t,a_t^*)]\right]\\ &\le C_{\pi^*}\mathbb{E}_{s_t\sim d_\pi}\left[D_W(\pi,\pi^*)\right]\\ &\le C_{\pi^*}\mathbb{E}_{s_t\sim d_\pi}\mathbb{E}_{a_t\sim\pi}\mathbb{E}_{a_t^*\sim\pi^*}[\|a_t - a_t^*\|], \end{aligned} \tag{10}$$

where we invoke the definition of advantage function $A_{\pi^*}^t(s_t,a_t) = Q_{\pi^*}^t(s_t,a_t) - \mathbb{E}_{a_t^*\sim\pi^*}[Q_{\pi^*}^t(s_t,a_t^*)]$, the first and the second inequalities is due to (8) and (9), respectively.

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

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

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

The optimization problem (11) is called the *online* imitation learning problem. This surrogate problem is comparatively more structured than the original RL problem (1), so we can adopt algorithms with provable performance guarantees. In this paper, we use the meta-learning algorithm DAgger [13], which reduces (11) 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,

$$\pi_i = \arg\min_{\pi} \mathbb{E}_{\mathcal{D}}[\hat{c}(s_t, a_t)], \tag{12}$$

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 (11), is returned. Suppose the policy is linearly parametrized and $o_t = x_t$. 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 (11) finds a policy π with performance $J(\pi) \leq J(\pi^*) + O(T)$, 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 $l1$-norm is preferable (e.g. over $l2$-norm) for its ability to filter outliers in a highly stochastic environment.

A.2 Batch Imitation Learning

By swapping the order of π and π^* in the above derivation in (10), we can derive another upper bound and use it to construct another surrogate problem: define $\tilde{c}_\pi(s_t^*, a_t^*) = \mathbb{E}_{a_t \sim \pi}[\|a_t - a_t^*\|]$ and $C_\pi^t(s_t^*) = \text{Lip}(Q_\pi^t(s_t^*, \cdot))$, then

$$\begin{aligned}
&J(\pi) - J(\pi^*) \\
&= \mathbb{E}_{s_t^* \sim d_{\pi^*}} \left[\mathbb{E}_{a_t \sim \pi}[Q_\pi^t(s_t^*, a_t)] - \mathbb{E}_{a_t^* \sim \pi^*}[Q_\pi^t(s_t^*, a_t^*)] \right] \\
&\leq \mathbb{E}_{s_t^* \sim d_{\pi^*}} \mathbb{E}_{a_t^* \sim \pi^*} \left[C_\pi^t(s_t^*) \tilde{c}_\pi(s_t^*, a_t^*) \right].
\end{aligned} \tag{13}$$

where we use again Lemma 1 for the equality and the property of Wasserstein distance for inequality. The minimization of the upper-bound (13) is called *batch* imitation learning problem [7, 9]:

$$\min_{\pi} \mathbb{E}_{\rho_{\pi^*}} \left[\sum_{t=1}^{T} \tilde{c}_\pi(s_t^*, a_t^*) \right], \tag{14}$$

In contrast to the surrogate problem in online imitation learning (11), batch imitation learning reduces to a supervised learning problem, because the expectation is defined by a fixed policy π^*.

A.3 Comparison of Imitation Learning Algorithms

Comparing (10) and (13), we observe that in batch imitation learning the Lipschitz constant $C_\pi^t(s_t^*)$, without π being an expert, can be on the order of $T - t$ in the worst case. Therefore, if we take a uniform bound and define $C_\pi = \sup_{t \in [0, T-1], s \in \mathbb{S}} C_\pi^t(s)$, we see $C_\pi \in O(T)$. In other words, under the same assumption in online imitation, i.e. (13) can be minimized to an error that depends linearly on T, the difference between $J(\pi)$ and $J(\pi^*)$ in batch imitation learning can actually grow quadratically in T due to error compounding. Therefore, in order to achieve the same level of performance as online imitation learning, batch imitation learning requires a more expressive policy class or more demonstration samples. As shown in [13], 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 imitation learning 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 imitation learning becomes hard to realize in real-road off-road driving tasks. Because humans rely on real-time sensory feedback (as in sampling from ρ_{π^*} but not from ρ_π) to generate ideal expert actions, the action samples collected in the online learning approach using ρ_π are often biased and inconsistent [24]. 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 [22], 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 imitation learning can be as bad as batch imitation learning [24], just due to inconsistencies introduced by human nature.

B Cost Function

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

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

This coefficients 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 $\text{cost}_{\textbf{pos}}(s)$ to fit the boundary data.

The speed cost $cost_{\textbf{spd}} = \|v_x - v_{\textbf{desired}}\|^2$ is a quadratic function which penalizes the difference between the desired speed $v_{\textbf{desired}}$ and the longitudinal velocity v_x in the body frame. The side slip angle cost is defined as $\text{cost}_{\textbf{slip}}(s) = -\arctan^2(\frac{v_y}{\|v_x\|})$, where v_y is the lateral velocity in the body frame. The action cost is a quadratic function defined as $\text{cost}_{\textbf{act}}(a) = \gamma_1 a_1 + \gamma_2 a_2$, 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.