

ADAPS: Autonomous Driving Via Principled Simulations

Weizi Li¹, David Wolinski¹, and Ming C. Lin^{1,2}

Abstract—Autonomous driving has gained significant advancements in recent years. However, obtaining a robust control policy for driving remains challenging as it requires training data from a variety of scenarios, including rare situations (e.g., accidents), an effective policy architecture, and an efficient learning mechanism. We propose ADAPS for producing robust control policies for autonomous vehicles. ADAPS consists of two simulation platforms in generating and analyzing accidents to automatically produce labeled training data, and a memory-enabled hierarchical control policy. Additionally, ADAPS offers a more efficient online learning mechanism that reduces the number of iterations required in learning compared to existing methods such as DAGGER [1]. We present both theoretical and experimental results. The latter are produced in simulated environments, where qualitative and quantitative results are generated to demonstrate the benefits of ADAPS.

I. INTRODUCTION

Autonomous driving consists of many complex sub-tasks that consider the dynamics of an environment and often lack accurate definitions of various driving behaviors. These characteristics lead to conventional control methods to suffer subpar performance on the task [2], [3]. However, driving and many other tasks can be easily demonstrated by human experts. This observation inspires *imitation learning*, which leverages expert demonstrations to synthesize a controller.

While there are many advantages of using imitation learning, it also has drawbacks. For autonomous driving, the most critical one is *covariate shift*, meaning the training and test distributions are different. This could lead autonomous vehicles (AVs) to accidents since a learned policy may fail to respond to unseen scenarios including those dangerous situations that do not occur often.

In order to mitigate this issue, the training dataset needs to be augmented with more expert demonstrations covering a wide spectrum of driving scenarios—especially ones of significant safety threats to the passengers—so that a policy can learn how to recover from its own mistakes. This is emphasized by Pomerleau [4], who synthesized a neural network based controller for AVs: “the network must not solely be shown examples of accurate driving, but also how to recover (i.e. return to the road center) once a mistake has been made.”

Although critical, obtaining recovery data from accidents in the physical world is impractical due to the high cost of a vehicle and potential injuries to both passengers and pedestrians. In addition, even one managed to collect accident data,

human experts are usually needed to label them, which is inefficient and may subject to judgmental errors [5].

These difficulties naturally lead us to the virtual world, where accidents can be simulated and analyzed [6]. We have developed ADAPS (Autonomous Driving Via Principled Simulations) to achieve this goal. ADAPS consists of two simulation platforms and a memory-enabled hierarchical control policy based on deep neural networks (DNNs). The first simulation platform, referred to as *SimLearner*, runs in a 3D environment and is used to test a learned policy, simulate accidents, and collect training data. The second simulation platform, referred to as *SimExpert*, acts in a 2D environment and serves as the “expert” to analyze and resolve an accident via *principled simulations* that can plan alternative safe trajectories for a vehicle by taking its physical, kinematic, and geometric constraints into account.

Furthermore, ADAPS represents a more efficient online learning mechanism than existing methods such as DAGGER [1]. This is useful consider learning to drive requires iterative testing and update of a control policy. Ideally, we want to obtain a robust policy using minimal iterations since one iteration corresponds to one incident. This would require the generation of training data at each iteration to be *accurate*, *efficient*, and *sufficient* so that a policy can gain a large improvement going into the next iteration. ADAPS can assist to achieve this goal.

The **main contributions** of this research are specifically: (1) The accidents generated in *SimLearner* will be analyzed by *SimExpert* to produce alternative safe trajectories. (2) These trajectories will be automatically processed to generate a large number of annotated and segmented training data. Because *SimExpert* is parameterized and has taken the physical, kinematic, and geometric constraints of a vehicle into account (i.e., principled), the resulting training examples are more heterogeneous than data collected via running a learned policy multiple times and are more effective than data collected through random sampling. (3) We present both theoretical and experimental results to demonstrate that ADAPS is an efficient online learning mechanism.

The Appendix, which contains supporting material, can be found at <http://gamma.cs.unc.edu/ADAPS/>.

II. RELATED WORK

We sample previous studies that are related to each aspect of our framework and discuss the differences within.

Autonomous Driving. Among various methods to plan and control an AV [7], we focus on end-to-end imitation learning as it can avoid manually designed features and lead to a more compact policy compared to conventional

¹W. Li, D. Wolinski, M. Lin are with the Department of Computer Science, University of North Carolina at Chapel Hill, NC, USA {weizili, dwolinsk, lin}@cs.unc.edu

²M. Lin is now with the Department of Computer Science, University of Maryland at College Park, MD, USA lin@cs.umd.edu

mediation perception approaches [8]. The early studies done by Pomerleau [4] and LeCun et al. [9] have shown that neural networks can be used for an AV to achieve lane-following and off-road obstacle avoidance. Due to the advancements of deep neural networks (DNNs), a number of studies have emerged [10], [11], [12], [13]. While significant improvements have been made, these results mainly inherit normal driving conditions and restrict a vehicle to the lane-following behavior [13]. Our policy, in contrast, learns from accidents and enables a vehicle to achieve *on-road* collision avoidance with both static and dynamic obstacles.

Hierarchical Control Policy. There have been many efforts in constructing a hierarchical policy to control an agent at different stages of a task [14]. Example studies include the options framework [15] and transferable motor skills [16]. When combined with DNNs, the hierarchical approach has been adopted for virtual characters to learn locomotion tasks [17]. In these studies, the goal is to discover a hierarchical relationship from complex sensorimotor behaviors. We apply a hierarchical and memory-enabled policy to autonomous driving based on multiple DNNs. Our policy enables an AV to continuously categorize the road condition as safe or dangerous, and execute corresponding control commands to achieve accident-free driving.

Generative Policy Learning. Using *principled simulations* to assist learning is essentially taking a *generative model* approach. Several studies have adopted the same philosophy to learn (near-)optimal policy, examples including function approximations [18], Sparse Sampling [19], and Fitted Value Iteration [20]. These studies leverage a generative model to *stochastically* generate training samples. The emphasize is to simulate the feedback from an environment instead of the dynamics of an agent assuming the *reward function is known*. Our system, on the other hand, does not assume any reward function of a driving behavior but models the physical, kinematic, and geometric constraints of a vehicle, and uses simulations to plan their trajectories w.r.t. environment characteristics. In essence, our method learns from expert demonstrations rather than self-exploration [21] as of the previous studies.

III. PRELIMINARIES

Autonomous driving is a *sequential prediction* and *controlled* (SPC) task, for which a system must predict a sequence of control commands based on inputs that depend on past predicted control commands. Because the control and prediction processes are intertwined, SPC tasks often encounter *covariate shift*, meaning the training and test distributions vary. In this section, we will first introduce notation and definitions to formulate an SPC task and then briefly discuss its existing solutions.

A. Notation and Definitions

The problem we consider is a T -step control task. Given the observation $\phi = \phi(s)$ of a state s at each step $t \in \llbracket 1, T \rrbracket$, the goal of a learner is to find a policy $\pi \in \Pi$ such that its produced action $a = \pi(\phi)$ will lead to the minimal cost:

$$\hat{\pi} = \arg \min_{\pi \in \Pi} \sum_{t=1}^T C(s_t, a_t), \quad (1)$$

where $C(s, a)$ is the expected immediate cost of performing a in s . For many tasks such as driving, we may not know the true value of C . So, we instead minimize the observed surrogate loss $l(\phi, \pi, a^*)$, which is assumed to upper bound C , based on the approximation of the learner's action $a = \pi(\phi)$ to the expert's action $a^* = \pi^*(\phi)$. We denote the distribution of observations at t as d_{π}^t , which is the result of executing π from 1 to $t-1$. Consequently, $d_{\pi} = \frac{1}{T} \sum_{t=1}^T d_{\pi}^t$ is the average distribution of observations by executing π for T steps. Our goal is to solve an SPC task by obtaining $\hat{\pi}$ that minimizes the observed surrogate loss under its own induced observations w.r.t. expert's actions in those observations:

$$\hat{\pi} = \arg \min_{\pi \in \Pi} \mathbb{E}_{\phi \sim d_{\pi}, a^* \sim \pi^*(\phi)} [l(\phi, \pi, a^*)]. \quad (2)$$

We further denote $\epsilon = \mathbb{E}_{\phi \sim d_{\pi^*}, a^* \sim \pi^*(\phi)} [l(\phi, \pi, a^*)]$ as the expected loss under the training distribution induced by the expert's policy π^* , and the cost-to-go over T steps of $\hat{\pi}$ as $J(\hat{\pi})$ and of π^* as $J(\pi^*)$. It has been shown that by simply treating expert demonstrations as i.i.d. samples the discrepancy between $J(\hat{\pi})$ and $J(\pi^*)$ is $\mathcal{O}(T^2\epsilon)$ [22], [1]. Given the error of a typical supervised learning is $\mathcal{O}(T\epsilon)$, this demonstrates the additional cost due to covariate shift when solving an SPC task via standard supervised learning¹.

B. Existing Techniques

Several approaches have been proposed to solve SPC tasks using supervised learning while keeping the error growing linearly instead of quadratically with T [22], [1], [23]. Essentially, these methods reduce an SPC task to online learning. By further leveraging interactions with experts and no-regret algorithms that have strong guarantees on convex loss functions [24], at each iteration, these methods train one or multiple policies using standard supervised learning and improve the trained policies as the iteration continues.

To illustrate, we denote the best policy at the i th iteration (trained using all observations from the previous $i-1$ iterations) as π_i and for any policy $\pi \in \Pi$ we have its expected loss under the observation distribution induced by π_i as $l_i(\pi) = \mathbb{E}_{\phi \sim d_{\pi_i}, a^* \sim \pi^*(\phi)} [l_i(\phi, \pi, a^*)]$, $l_i \in [0, l_{max}]^2$. In addition, we denote the minimal loss in hindsight after $N \geq i$ iterations as $\epsilon_{min} = \min_{\pi \in \Pi} \frac{1}{N} \sum_{i=1}^N l_i(\pi)$ (i.e., the training loss after using all observations from N iterations). Then, we can represent the average regret of this online learning program as $\epsilon_{regret} = \frac{1}{N} \sum_{i=1}^N l_i(\pi_i) - \epsilon_{min}$. Using DAGGER [1] as an example method, the accumulated error difference becomes the summation of three terms:

$$J(\hat{\pi}) \leq T\epsilon_{min} + T\epsilon_{regret} + \mathcal{O}\left(\frac{f(T, l_{max})}{N}\right), \quad (3)$$

¹The proofs regarding results $\mathcal{O}(T^2\epsilon)$ and $\mathcal{O}(T\epsilon)$ can be found in Appendix IX-A.

²In online learning, the surrogate loss l can be seen as chosen by some adversary which varies at each iteration.

where $f(\cdot)$ is the function of fixed T and l_{max} . As $N \rightarrow \infty$, the third term tends to 0 so as the second term if a no-regret algorithm such as the Follow-the-Leader [25] is used.

The aforementioned approach provides a practical way to solve SPC tasks. However, it may require many iterations for obtaining a good policy. In addition, usually human experts or pre-defined controllers are needed for labeling the generated training data, which could be inefficient or difficult to generalize. For autonomous driving, we want the iteration number to be minimal since it directly corresponds to the number of accidents. This requires the generation of training data being accurate, efficient, and sufficient.

IV. ADAPS

In the following, we present theoretical analysis of our framework and introduce our framework pipeline.

A. Theoretical Analysis

We have evaluated our approach against existing learning mechanisms such as DAGGER [1], with our method’s results proving to be more effective. Specifically, DAGGER [1] assumes that an underlying learning algorithm has access to a *reset model*. So, the training examples can be obtained only *online* by putting an agent to its initial state distribution and executing a learned policy, thus achieving “small changes” at each iteration [1], [23], [26], [27]. In comparison, our method allows a learning algorithm to access a *generative model* so that the training examples can be acquired *offline* by putting an agent to arbitrary states during the analysis of an accident and letting a *generative* model simulate its behavior. This approach results in massive training data, thus achieving “large changes” of a policy at one iteration.

Additionally, existing techniques such as DAGGER [1] usually incorporate the demonstrations of a few experts into training. Because of the *reset* model assumption and the lack of a diversity requirement on experts, these demonstrations can be homogeneous. In contrast, using our parameterized model to retrace and analyze each accident, the number of recovery actions obtained can be multiple orders of magnitude higher. Subsequently, we can treat the generated trajectories and the additional data generated based on them (described in Section VI-B) as running a learned policy to sample independent expert trajectories at different states, since 1) a policy that is learned using DNNs can achieve a small training error and 2) our model provides near-exhaustive coverage of the configuration space of a vehicle. With these assumptions, we derive the following theorem.

Theorem 1: If the surrogate loss l upper bounds the true cost C , by collecting K trajectories using ADAPS at each iteration, with probability at least $1 - \mu$, $\mu \in (0, 1)$, we have the following guarantee:

$$J(\hat{\pi}) \leq J(\bar{\pi}) \leq T\hat{\epsilon}_{min} + T\hat{\epsilon}_{regret} + \mathcal{O}\left(Tl_{max}\sqrt{\frac{\log\frac{1}{\mu}}{KN}}\right).$$

Proof: See Appendix IX-A.3. ■

Theorem 1 provides a bound for the expected cost-to-go of the best learned policy $\hat{\pi}$ based on the empirical error of the

best policy in Π (i.e., $\hat{\epsilon}_{min}$) and the empirical average regret of the learner (i.e., $\hat{\epsilon}_{regret}$). The second term can be eliminated if a no-regret algorithm such as Follow-the-Leader [25] is used and the third term suggests that we need the number of training examples KN to be $\mathcal{O}\left(T^2l_{max}^2\log\frac{1}{\mu}\right)$ in order to have a negligible generalization error, which is easily achievable using ADAPS. Summarizing these changes, we derive the following Corollary.

Corollary 1: If l is convex in π for any s and it upper bounds C , and Follow-the-Leader is used to select the learned policy, then for any $\epsilon > 0$, after collecting $\mathcal{O}\left(\frac{T^2l_{max}^2\log\frac{1}{\mu}}{\epsilon^2}\right)$ training examples, with probability at least $1 - \mu$, $\mu \in (0, 1)$, we have the following guarantee:

$$J(\hat{\pi}) \leq J(\bar{\pi}) \leq T\hat{\epsilon}_{min} + \mathcal{O}(\epsilon).$$

Proof: Following Theorem 1 and the aforementioned deduction. ■

Now we only need the best policy to have a small training error $\hat{\epsilon}_{min}$. This can be achieved using DNNs since they have rich representing capabilities.

B. Framework Pipeline

The pipeline of our framework is the following. First, in *SimLearner*, we test a learned policy by letting it control an AV. During the testing, an accident may occur, in which case the trajectory of the vehicle and the full specifications of the situation (e.g., positions of obstacles, road configuration, etc.) are known. Next, we switch to *SimExpert* and replicate the specifications of the accident so that we can “solve” the accident (i.e., find alternative safe trajectories and dangerous zones). After obtaining the solutions, we then use them to generate additional training data in *SimLearner*, which will be combined with previously generated data to update the policy. Finally, we test the updated policy again.

V. POLICY LEARNING

In this section, we will detail our control policy by first explaining our design rationale then formulating our problem and introducing the training data collection.

Driving is a hierarchical decision process. In its simplest form, a driver needs to constantly monitor the road condition, decide it is “safe” or “dangerous”, and make corresponding maneuvers. When designing a control policy for AVs, we need to consider this hierarchical aspect. In addition, driving is a temporal behavior. Drivers need reaction time to respond to various road situations [28], [29]. A Markovian-based control policy will not model this aspect and instead likely to give a vehicle jerky motions. Consider these factors, we propose a *hierarchical* and *memory-enabled* control policy.

The task we consider is autonomous driving via a single front-facing camera. Our control policy consists of three modules: *Detection*, *Following*, and *Avoidance*. The *Detection* module keeps monitoring road conditions and activates either *Following* or *Avoidance* to produce a steering command. All these modules are trained via end-to-end imitation learning and share a similar network specification which is detailed in Appendix IX-B.

A. End-to-end Imitation Learning

The objective of imitation learning is to train a model that behaves or makes decisions like an expert through demonstrations. The model could be a classifier or a regressor π parameterized by θ_π :

$$\hat{\theta} = \arg \min_{\theta_\pi} \sum_{t=1}^T \mathcal{F}(\pi(\phi_t; \theta_\pi), a_t^*), \quad (4)$$

where \mathcal{F} is a distance function.

The end-to-end aspect denotes the mapping from raw observations to decision/control commands. For our policy, we need one decision module $\pi_{Detection}$ and two control modules $\pi_{Following}$ and $\pi_{Avoidance}$. The input for $\pi_{Detection}$ is a sequence of annotated images while the outputs are binary labels indicating whether a road condition is dangerous or safe. The inputs for $\pi_{Following}$ and $\pi_{Avoidance}$ are sequences of annotated images while the outputs are steering angles. Together, these learned policies form a hierarchical control mechanism enabling an AV to drive safely on roads and avoid obstacles when needed.

B. Training Data Collection

For training *Following*, inspired by the technique used by Bojarski et al. [10], we collect images from three front-facing cameras behind the main windshield: one at the center, one at the left side, and one at the right side. The image from the center camera is labeled with the exact steering angle while the images from the other two cameras are labeled with adjusted steering angles. However, once *Following* is learned, it only needs images from the center camera to operate.

For training *Avoidance*, we rely on *SimExpert*, which can generate numerous intermediate collision-free trajectories between the first moment and the last moment of a potential accident (see Section VI-A). By positioning an AV on these trajectories, we collect images from the center front-facing camera along with corresponding steering angles. The training of *Detection* requires a more sophisticated mechanism and is the subject of the next section.

VI. LEARNING FROM ACCIDENTS

We explain how we analyze an accident in *SimExpert* and use the generated data to train the *Avoidance* and *Detection* modules of our policy. *SimExpert* is built based on the multi-agent simulator WarpDriver [30].

A. Solving Accidents

When an accident occurs, we know the trajectory of the tested vehicle for the latest K frames, which we note as a collection of states $\mathcal{S} = \bigcup_{k \in [1, K]} s_k$, where each state $s_k \in \mathbb{R}^4$ contains the 2-dimensional position and velocity vectors of the vehicle. Then, there are three notable states on this trajectory that we need to track. The first is the earliest state where the vehicle involved in an accident (is in a collision) s_{k_a} (at frame k_a). The second is the last state s_{k_l} (at frame k_l) where the expert algorithm can still avoid a collision. The final one is the first state s_{k_f} (at frame k_f) where the expert

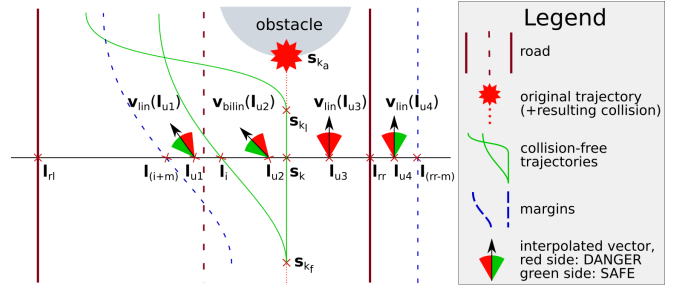


Fig. 1. Illustration of important points and DANGER/SAFE labels from Section VI for a vehicle traveling on the right lane of a straight road, with an obstacle in front. Labels are shown for four points $\{I_{u1}, I_{u2}, I_{u3}, I_{u4}\}$ illustrating the four possible cases.

algorithm perceives the interaction leading to the accident with the other involved agent, before that accident.

In order to compute these notable states, we briefly recall the high-level components of WarpDriver [30]. This collision-avoidance algorithm consists of two parts. The first is the function p , which given the current state of an agent s_k and any prediction point $\mathbf{x} \in \mathbb{R}^3$ in 2-dimensional space and time (in this agent's referential), gives the probability of that agent's colliding with any neighbor $p(s_k, \mathbf{x}) \in [0, 1]$. The second part is the solver, which based on this function, computes the agent's probability of colliding with neighbors along its future trajectory starting from a state s_k (i.e., computed for \mathbf{x} spanning the future predicted trajectory of the agent, we denote this probability $P(s_k)$), and then proposes a new velocity to lower this probability. Subsequently, we can initialize an agent in this algorithm to any state $s_k \in \mathcal{S}$ and compute a new trajectory consisting of \hat{K} new states $\hat{\mathcal{S}}_k = \bigcup_{\hat{k} \in [1, \hat{K}]} \hat{s}_{\hat{k}}$, where $\hat{s}_1 = s_k$.

Additionally, since $\mathbf{x} = (0, 0, 0)$ in space and time in an agent's referential represents the agent's position at the current time (we can use this point \mathbf{x} with function p to determine if the agent is currently colliding with anyone), we find s_{k_a} where $k_a = \min(k)$ subject to $k \in [1, K]$ and $p(s_k, (0, 0, 0)) > 0$. We note that a trajectory $\hat{\mathcal{S}}_k$ produced by the expert algorithm could contain collisions (accounting for vehicle dynamics) depending on the state s_k that it was initialized from. We can denote the set of colliding states along this trajectory as $coll(\hat{\mathcal{S}}_k) = \{\hat{s}_{\hat{k}} \in \hat{\mathcal{S}}_k \mid p(\hat{s}_{\hat{k}}, (0, 0, 0)) > 0\}$. Then, we can compute s_{k_l} where $k_l = \max(k)$ subject to $k \in [1, k_a]$ and $coll(\hat{\mathcal{S}}_k) = \emptyset$. Finally, we can compute s_{k_f} with $k_f = 1 + \max(k)$ subject to $k \in [1, k_l]$ and $P(s_k) = 0$.

Knowing these notable states, we can solve the accident situation by computing the set of collision-free trajectories $solve(\mathcal{S}) = \{\hat{\mathcal{S}}_k \mid k \in [k_f, k_l]\}$. An example can be found in Appendix IX-C. These trajectories can then be used to generate training examples in *SimLearner* in order to train the *Avoidance* module.

B. Additional Data Coverage

The previous step generated collision-free trajectories $solve(\mathcal{S})$ between s_{k_f} and s_{k_l} . It is possible to build on these trajectories if the tested steering algorithm has particular data/training requirements. Here we detail the data we derive in order to train the *Detection* module, where the task is to

determine if a situation is dangerous and tell *Avoidance* to address it.

To proceed, we essentially generate a number of trajectories parallel to $\{s_{k_f}, \dots, s_{k_a}\}$, and for each position on them, generate several images for various orientations of the vehicle. These images are then labeled based on understeering/oversteering as compared to the “ideal” trajectories in $solve(\mathcal{S})$. This way, we scan the region of the road before the accident locus, generating several images (different vehicle orientations) for each point in that region.

In summary (a thorough version can be found in Appendix IX-D), and as depicted in Figure 1, at each state s_k , we construct a line perpendicular to the original trajectory. Then on this line, we define three points and a margin $g = 0.5 m$. The first point l_i is the furthest (from s_k) intersection between this line and the collision-free trajectories $solve(\mathcal{S})$. The other two points $\{l_{rl}, l_{rr}\}$ are the intersections between the constructed line and the left and right road borders, respectively. From these points, a generated image at a position l_u along the constructed line and with a given direction vector has either a DANGER or SAFE label (red and green ranges in Figure 1) depending on the direction vector being on the “left” or “right” of the vector resulting from the interpolation of the velocity vectors of states belonging to nearby collision-free trajectories (bilinear interpolation if l_u is between two collision-free trajectories, linear otherwise).

If a point is on the same side of the original trajectory as the collision-free trajectories (l_{u1} and l_{u2} in Figure 1, l_{u1} is “outside” but within the margin g of the collision-free trajectories, l_{u2} is “inside” the collision-free trajectories), the label is SAFE on the exterior of the avoidance maneuver, and DANGER otherwise.

If a point is on the other side of the original trajectory as compared to the collision-free trajectories (l_{u3} and l_{u4} in Figure 1), inside the road (l_{u3}) the label is always DANGER, while outside but within the margin g of the road (l_{u4}), the label is DANGER when directed towards the road, and SAFE otherwise.

VII. EXPERIMENTS

We test our framework in three scenarios: a straight road representing a linear geometry, a curved road representing a non-linear geometry, and an open ground. The first two scenarios demonstrate on-road situations with a static obstacle while the last one demonstrates an off-road situation with a dynamic obstacle. The specifications of our experiments are detailed in Appendix IX-E.

For evaluation, we compare our policy to the “flat policy” that essentially consists of a single DNN [8], [11], [31], [13]. Usually, this type of policy contains a few convolutional layers followed by a few dense layers. Although the specifications may vary, without human intervention, they are mainly limited to single-lane following [13]. In this work, we select Bojarski et al. [10] as an example network, as it is one of the most tested control policies. In the following, we will first demonstrate the effectiveness of our policy and then qualitatively illustrate the efficiency of our framework.

A. Control Policy

1) *On-road*: We derive our training datasets from *straight road with or without an obstacle* and *curved road with or without an obstacle*. This separation allows us to train multiple policies and test the effect of *learning from accidents* using our policy compared to Bojarski et al. [10]. By progressively increasing the training datasets, we obtain six policies for evaluation:

- Our own policy: trained with only lane-following data O_{follow} ; O_{follow} additionally trained after analyzing one accident on the straight road $O_{straight}$; and $O_{straight}$ additionally trained after producing one accident on the curved road O_{full} .
- Similarly, for the policy from Bojarski et al. [10]: B_{follow} , $B_{straight}$, and B_{full} .

We first evaluate B_{follow} and O_{follow} using both the straight and curved roads by counting how many laps (out of 50) the AV can finish. As a result, both policies managed to finish all laps while keeping the vehicle in the lane. We then test these two policies on the straight road with a static obstacle added. Both policies result in the vehicle collides into the obstacle, which is expected since no accident data were used during the training.

Having the occurred accident, we can now use *SimExpert* to generate additional training data to obtain $B_{straight}^3$ and $O_{straight}$. As a result, $B_{straight}$ continues to cause collision while $O_{straight}$ avoids the obstacle. Nevertheless, when testing $O_{straight}$ on the curved road with an obstacle, accident still occurs because of the corresponding accident data are not yet included in training.

By further including the accident data from the curved road into training, we obtain B_{full} and O_{full} . O_{full} manages to perform both lane-following and collision avoidance in all runs. B_{full} , on the other hand, leads the vehicle to drift away from the road.

For the studies involved an obstacle, we uniformly sampled 50 obstacle positions on a 3 m line segment that is perpendicular to the direction of a road and in the same lane as the vehicle. We compute the success rate as how many times a policy can avoid the obstacle (while stay in the lane) and resume lane-following afterwards. The results are shown in Table II and example trajectories are shown in Figure 2 LEFT and CENTER.

2) *Off-road*: We further test our method on an open ground which involves a dynamic obstacle. The AV is trained heading towards a green sphere while an adversary vehicle is scripted to collide with the AV on its default course. The result showing our policy can steer the AV away from the adversary vehicle and resume its direction to the sphere target. This can be seen in Figure 2 RIGHT.

B. Algorithm Efficiency

The key to rapid policy improvement is to generate training data accurately, efficiently, and sufficiently. Using

³The accident data are only used to perform a regression task as the policy by Bojarski et al. [10] does not have a classification module.

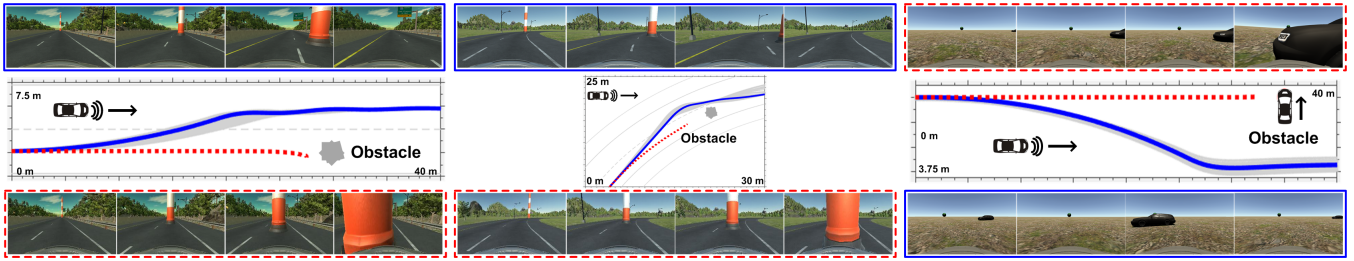


Fig. 2. LEFT and CENTER: the comparisons between our policy O_{full} (TOP) and Bojarski et al. [10], B_{full} (BOTTOM). O_{full} can steer the AV away from the obstacle while B_{full} causes collision. RIGHT: the accident analysis results on the open ground. We show the accident caused by an adversary vehicle (TOP); then we show after additional training the AV can now avoid the adversary vehicle (BOTTOM).

Scenarios	Training Module (Data)			Total	Data Augmentation	Other Specs		
	Following (#Images)	Avoidance (#Images)	Detection (#Images)			#Safe Trajectories	Road Type	Obstacle
Straight road	33 642	34 516	32 538	97 854	212x	74	on-road	static
Curved road	31 419	33 624	71 859	136 855	98x	40	on-road	static
Open ground	30 000	33 741	67 102	130 843	178x	46	off-road	dynamic

TABLE I

TRAINING DATA SUMMARY: OUR METHOD CAN ACHIEVE OVER **200** TIMES MORE TRAINING EXAMPLES THAN DAGGER [1] AT ONE ITERATION LEADING TO LARGE IMPROVEMENTS OF A POLICY.

Scenario	Test Policy and Success Rate (out of 50 runs)					
	B_{follow}	O_{follow}	$B_{straight}$	$O_{straight}$	B_{full}	O_{full}
Straight rd. / Curved rd.	100%	100%	100%	100%	100%	100%
Straight rd. + Obst.	0%	0%	0%	100%	0%	100%
Curved rd. + Obst.	0%	0%	0%	0%	0%	100%

TABLE II

TEST RESULTS OF ON-ROAD SCENARIOS: OUR POLICIES $O_{straight}$ & O_{full} CAN LEAD TO ROBUST COLLISION AVOIDANCE AND LANE-FOLLOWING BEHAVIORS.

principled simulations covers the first two criteria, now we demonstrate the third. Compared to the average number of training data collected by DAGGER [1] at one iteration, our method can achieve over 200 times more training examples for one iteration⁴. This is shown in Table I.

In Figure 3, we show the visualization results of images collected using our method and DAGGER [1] within one iteration via progressively increasing the number of sampled trajectories. Our method generates much more heterogeneous training data, which when produced in a large quantity can greatly facilitate the update of a control policy.

VIII. CONCLUSION

In this work, we have proposed ADAPS, a framework that consists of two simulation platforms and a control policy. Using ADAPS, one can easily simulate accidents. Then, ADAPS will automatically retrace each accident, analyze it, and plan alternative safe trajectories. With the additional training data generation technique, our method can produce a large number of heterogeneous training examples compared to existing methods such as DAGGER [1], thus representing a more efficient learning mechanism. Our hierarchical and

⁴The result is computed via dividing the total number of training images via our method by the average number of training data collected using the safe trajectories in each scenario.

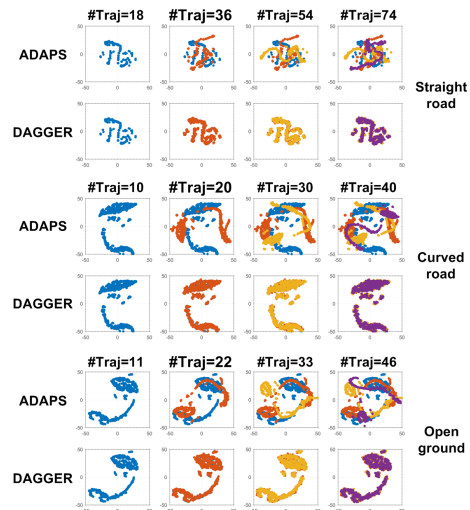


Fig. 3. The visualization results of collected images using t-SNE [32]. Our method can generate more heterogeneous training data compared to DAGGER [1] at one iteration as the sampled trajectories progress.

memory-enabled policy offers robust collision avoidance behaviors that previous policies fail to achieve. We have evaluated our method using multiple simulated scenarios, in which our method shows a variety of benefits.

There are many future directions. First of all, we would like to combine long-range vision into ADAPS so that an AV can plan ahead in time. Secondly, the generation of accidents can be parameterized using knowledge from traffic engineering studies. Lastly, we would like to combine more sensors and fuse their inputs so that an AV can navigate in more complicated traffic scenarios [33].

ACKNOWLEDGMENT

The authors would like to thank US Army Research Office and UNC Arts & Science Foundation, and Dr. Feng “Bill” Shi for insightful discussions.

REFERENCES

- [1] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.
- [2] N. Ratliff, "Learning to search: structured prediction techniques for imitation learning," Ph.D. dissertation, Carnegie Mellon University, 2009.
- [3] D. Silver, "Learning preference models for autonomous mobile robots in complex domains," Ph.D. dissertation, 2010.
- [4] D. Pomerleau, "ALVINN: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, 1989, pp. 305–313.
- [5] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," in *Robotics and Automation, 2013 IEEE International Conference on*. IEEE, 2013, pp. 1765–1772.
- [6] Q. Chao, H. Bi, W. Li, T. Mao, Z. Wang, M. C. Lin, and Z. Deng, "A survey on visual traffic simulation: Models, evaluations, and applications in autonomous driving," *Computer Graphics Fourm*, 2019.
- [7] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, 2018.
- [8] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Computer Vision, 2015 IEEE International Conference on*, 2015, pp. 2722–2730.
- [9] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp, "Off-road obstacle avoidance through end-to-end learning," in *Advances in neural information processing systems*, 2005, pp. 739–746.
- [10] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.
- [11] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 3530–3538.
- [12] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile off-road autonomous driving using end-to-end deep imitation learning," in *Robotics: Science and Systems*, 2018.
- [13] F. Codevilla, M. Müller, A. Dosovitskiy, A. López, and V. Koltun, "End-to-end driving via conditional imitation learning," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 746–753.
- [14] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.
- [15] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.
- [16] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Robot learning from demonstration by constructing skill trees," *The International Journal of Robotics Research*, vol. 31, no. 3, pp. 360–375, 2012.
- [17] S. Levine and V. Koltun, "Guided policy search," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013, pp. 1–9.
- [18] G. J. Gordon, "Stable function approximation in dynamic programming," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 261–268.
- [19] M. Kearns, Y. Mansour, and A. Y. Ng, "A sparse sampling algorithm for near-optimal planning in large markov decision processes," *Machine learning*, vol. 49, no. 2-3, pp. 193–208, 2002.
- [20] C. Szepesvári and R. Munos, "Finite time bounds for sampling based fitted value iteration," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 880–887.
- [21] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.
- [22] U. Syed and R. E. Schapire, "A reduction from apprenticeship learning to classification," in *Advances in Neural Information Processing Systems*, 2010, pp. 2253–2261.
- [23] H. Daumé, J. Langford, and D. Marcu, "Search-based structured prediction," *Machine learning*, vol. 75, no. 3, pp. 297–325, 2009.
- [24] S. M. Kakade and A. Tewari, "On the generalization ability of online strongly convex programming algorithms," in *Advances in Neural Information Processing Systems*, 2009, pp. 801–808.
- [25] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Machine Learning*, vol. 69, no. 2-3, pp. 169–192, 2007.
- [26] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, vol. 2, 2002, pp. 267–274.
- [27] J. A. Bagnell, S. M. Kakade, J. G. Schneider, and A. Y. Ng, "Policy search by dynamic programming," in *Advances in neural information processing systems*, 2004, pp. 831–838.
- [28] G. Johansson and K. Rumar, "Drivers' brake reaction times," *Human factors*, vol. 13, no. 1, pp. 23–27, 1971.
- [29] D. V. McGehee, E. N. Mazzae, and G. S. Baldwin, "Driver reaction time in crash avoidance research: validation of a driving simulator study on a test track," in *Proceedings of the human factors and ergonomics society annual meeting*, vol. 44, no. 20, 2000.
- [30] D. Wolinski, M. Lin, and J. Pettré, "Warpdriver: context-aware probabilistic motion prediction for crowd simulation," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, 2016.
- [31] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end simulated driving," in *AAAI*, 2017, pp. 2891–2897.
- [32] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.
- [33] W. Li, D. Wolinski, and M. C. Lin, "City-scale traffic animation using statistical learning and metamodel-based optimization," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 200:1–200:12, Nov. 2017.
- [34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [35] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.
- [36] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ICLR*, 2015.

IX. APPENDIX

A. Solving An SPC Task

We show the proofs of solving an SPC task using standard supervised learning, DAGGER [1], and ADAPS, respectively. We use “state” and “observation” interchangeably here as for these proofs we can always find a deterministic function to map the two.

1) *Supervised Learning*: The following proof is adapted and simplified from Ross et al. [1]. We include it here for completeness.

Theorem 2: Consider a T -step control task. Let $\epsilon = \mathbb{E}_{\phi \sim d_{\pi^*}, a^* \sim \pi^*(\phi)} [l(\phi, \pi, a^*)]$ be the observed surrogate loss under the training distribution induced by the expert’s policy π^* . We assume $C \in [0, C_{max}]$ and l upper bounds the 0-1 loss. $J(\pi)$ and $J(\pi^*)$ denote the cost-to-go over T steps of executing π and π^* , respectively. Then, we have the following result:

$$J(\pi) \leq J(\pi^*) + C_{max} T^2 \epsilon.$$

Proof: In order to prove this theorem, we introduce the following notation and definitions:

- $d_{t,c}^\pi$: the state distribution at t as a result of the following event: π is executed and has been choosing the same actions as π^* from time 1 to $t-1$.
- $p_{t-1} \in [0, 1]$: the probability that the above-mentioned event holds true.
- $d_{t,e}^\pi$: the state distribution at t as a result of the following event: π is executed and has chosen at least one different action than π^* from time 1 to $t-1$.
- $(1 - p_{t-1}) \in [0, 1]$: the probability that the above-mentioned event holds true.
- $d_t^\pi = p_{t-1} d_{t,c}^\pi + (1 - p_{t-1}) d_{t,e}^\pi$: the state distribution at t .
- $\epsilon_{t,c}$: the probability that π chooses a different action than π^* in $d_{t,c}^\pi$.
- $\epsilon_{t,e}$: the probability that π chooses a different action than π^* in $d_{t,e}^\pi$.
- $\epsilon_t = p_{t-1} \epsilon_{t,c} + (1 - p_{t-1}) \epsilon_{t,e}$: the probability that π chooses a different action than π^* in d_t^π .
- $C_{t,c}$: the expected immediate cost of executing π in $d_{t,c}^\pi$.
- $C_{t,e}$: the expected immediate cost of executing π in $d_{t,e}^\pi$.
- $C_t = p_{t-1} C_{t,c} + (1 - p_{t-1}) C_{t,e}$: the expected immediate cost of executing π in d_t^π .
- $C_{t,c}^*$: the expected immediate cost of executing π^* in $d_{t,c}^\pi$.
- C_{max} : the upper bound of an expected immediate cost.
- $J(\pi) = \sum_{t=1}^T C_t$: the cost-to-go of executing π for T steps.
- $J(\pi^*) = \sum_{t=1}^T C_{t,c}^*$: the cost-to-go of executing π^* for T steps.

The probability that the learner chooses at least one different action than the expert in the first t steps is:

$$(1 - p_t) = (1 - p_{t-1}) + p_{t-1} \epsilon_{t,c}.$$

This gives us $(1 - p_t) \leq (1 - p_{t-1}) + \epsilon_t$ since $p_{t-1} \in [0, 1]$. Solving this recurrence we arrive at:

$$1 - p_t \leq \sum_{i=1}^t \epsilon_i.$$

Now consider in state distribution $d_{t,c}^\pi$, if π chooses a different action than π^* with probability $\epsilon_{t,c}$, then π will incur a cost at most C_{max} more than π^* . This can be represented as:

$$C_{t,c} \leq C_{t,c}^* + \epsilon_{t,c} C_{max}.$$

Thus, we have:

$$\begin{aligned} C_t &= p_{t-1} C_{t,c} + (1 - p_{t-1}) C_{t,e} \\ &\leq p_{t-1} C_{t,c}^* + p_{t-1} \epsilon_{t,c} C_{max} + (1 - p_{t-1}) C_{max} \\ &= p_{t-1} C_{t,c}^* + (1 - p_t) C_{max} \\ &\leq C_{t,c}^* + (1 - p_t) C_{max} \\ &\leq C_{t,c}^* + C_{max} \sum_{i=1}^t \epsilon_i. \end{aligned}$$

We sum the above result over T steps and use the fact $\frac{1}{T} \sum_{t=1}^T \epsilon_t \leq \epsilon$:

$$\begin{aligned} J(\pi) &\leq J(\pi^*) + C_{max} \sum_{t=1}^T \sum_{i=1}^t \epsilon_i \\ &= J(\pi^*) + C_{max} \sum_{t=1}^T (T + 1 - t) \epsilon_t \\ &\leq J(\pi^*) + C_{max} T \sum_{t=1}^T \epsilon_t \\ &\leq J(\pi^*) + C_{max} T^2 \epsilon. \end{aligned}$$

2) *DAGGER*: The following proof is adapted from Ross et al. [1]. We include it here for completeness. Note that for Theorem 3, we have arrived at the different third term as of Ross et al. [1].

Lemma 1: [1] Let P and Q be any two distributions over elements $x \in \mathcal{X}$ and $f: \mathcal{X} \rightarrow \mathbb{R}$, any bounded function such that $f(x) \in [a, b]$ for all $x \in \mathcal{X}$. Let the range $r = b - a$. Then $|\mathbb{E}_{x \sim P} [f(x)] - \mathbb{E}_{x \sim Q} [f(x)]| \leq \frac{r}{2} \|P - Q\|_1$.

Proof:

$$\begin{aligned} &|\mathbb{E}_{x \sim P} [f(x)] - \mathbb{E}_{x \sim Q} [f(x)]| \\ &= \left| \int_x P(x) f(x) dx - \int_x Q(x) f(x) dx \right| \\ &= \left| \int_x f(x) (P(x) - Q(x)) dx \right| \\ &= \left| \int_x (f(x) - c) (P(x) - Q(x)) dx \right|, \forall c \in \mathbb{R} \\ &\leq \int_x |f(x) - c| |P(x) - Q(x)| dx \\ &\leq \max_x |f(x) - c| \int_x |P(x) - Q(x)| dx \\ &= \max_x |f(x) - c| \|P - Q\|_1. \end{aligned}$$

Taking $c = a + \frac{\tau}{2}$ leads to $\max_x |f(x) - c| \leq \frac{\tau}{2}$ and proves the lemma. ■

Lemma 2: [1] Let $\hat{\pi}_i$ be the learned policy, π^* be the expert's policy, and π_i be the policy used to collect training data with probability β_i executing π^* and probability $1 - \beta_i$ executing $\hat{\pi}_i$ over T steps. Then, we have $\|d_{\pi_i} - d_{\hat{\pi}_i}\|_1 \leq 2 \min(1, T\beta_i)$.

Proof: In contrast to $d_{\hat{\pi}_i}$ which is the state distribution as the result of solely executing $\hat{\pi}_i$, we denote d as the state distribution as the result of π_i executing π^* at least once over T steps. This gives us $d_{\pi_i} = (1 - \beta_i)^T d_{\hat{\pi}_i} + (1 - (1 - \beta_i)^T) d$. We also have the facts that for any two distributions P and Q , $\|P - Q\|_1 \leq 2$ and $(1 - \beta)^T \geq 1 - \beta T, \forall \beta \in [0, 1]$. Then, we have $\|d_{\pi_i} - d_{\hat{\pi}_i}\|_1 \leq 2$ and can further show:

$$\begin{aligned} \|d_{\pi_i} - d_{\hat{\pi}_i}\|_1 &= (1 - (1 - \beta_i)^T) \|d - d_{\hat{\pi}_i}\|_1 \\ &\leq 2(1 - (1 - \beta_i)^T) \\ &\leq 2T\beta_i. \end{aligned}$$

Theorem 3: [1] If the surrogate loss $l \in [0, l_{max}]$ is the same as the cost function C or upper bounds it, then after N iterations of DAGGER:

$$J(\hat{\pi}) \leq J(\bar{\pi}) \leq T\epsilon_{min} + T\epsilon_{regret} + \mathcal{O}\left(\frac{f(T, l_{max})}{N}\right). \quad (5)$$

Proof: Let $l_i(\pi) = \mathbb{E}_{\phi \sim d_{\pi_i}, a^* \sim \pi^*(\phi)} [l(\phi, \pi, a^*)]$ be the expected loss of any policy $\pi \in \Pi$ under the state distribution induced by the learned policy π_i at the i th iteration and $\epsilon_{min} = \min_{\pi \in \Pi} \frac{1}{N} \sum_{i=1}^N l_i(\pi)$ be the minimal loss in hindsight after $N \geq i$ iterations. Then, $\epsilon_{regret} = \frac{1}{N} \sum_{i=1}^N l_i(\pi_i) - \epsilon_{min}$ is the average regret of this online learning program. In addition, we denote the expected loss of any policy $\pi \in \Pi$ under its own induced state distribution as $L(\pi) = \mathbb{E}_{\phi \sim d_{\pi}, a^* \sim \pi^*(\phi)} [l(\phi, \pi, a^*)]$ and consider $\bar{\pi}$ as the mixed policy that samples the policies $\{\hat{\pi}_i\}_{i=1}^N$ uniformly at the beginning of each trajectory. Using Lemma 1 and Lemma 2, we can show:

$$\begin{aligned} L(\hat{\pi}_i) &= \mathbb{E}_{\phi \sim d_{\hat{\pi}_i}, a^* \sim \pi^*(\phi)} [l(\phi, \hat{\pi}_i, a^*)] \\ &\leq \mathbb{E}_{\phi \sim d_{\pi_i}, a^* \sim \pi^*(\phi)} [l(\phi, \hat{\pi}_i, a^*)] + \frac{l_{max}}{2} \|d_{\pi_i} - d_{\hat{\pi}_i}\|_1 \\ &\leq \mathbb{E}_{\phi \sim d_{\pi_i}, a^* \sim \pi^*(\phi)} [l(\phi, \hat{\pi}_i, a^*)] + l_{max} \min(1, T\beta_i) \\ &= l_i(\hat{\pi}_i) + l_{max} \min(1, T\beta_i). \end{aligned}$$

By further assuming β_i is monotonically decreasing and $n_\beta = \arg \max_n (\beta_n > \frac{1}{T}), n \leq N$, we have the following:

$$\begin{aligned} \min_{i \in 1:N} L(\hat{\pi}_i) &\leq L(\bar{\pi}) \\ &= \frac{1}{N} \sum_{i=1}^N L(\hat{\pi}_i) \\ &\leq \frac{1}{N} \sum_{i=1}^N l_i(\hat{\pi}_i) + \frac{l_{max}}{N} \sum_{i=1}^N \min(1, T\beta_i) \\ &= \epsilon_{min} + \epsilon_{regret} + \frac{l_{max}}{N} \left[n_\beta + T \sum_{i=n_\beta+1}^N \beta_i \right]. \end{aligned}$$

Summing over T gives us:

$$J(\bar{\pi}) \leq T\epsilon_{min} + T\epsilon_{regret} + \frac{Tl_{max}}{N} \left[n_\beta + T \sum_{i=n_\beta+1}^N \beta_i \right].$$

Define $\beta_i = (1 - \alpha)^{i-1}$, in order to have $\beta_i \leq \frac{1}{T}$, we need $(1 - \alpha)^{i-1} \leq \frac{1}{T}$ which gives us $i \leq 1 + \frac{\log \frac{1}{T}}{\log(1 - \alpha)}$. In addition, note now $i = n_\beta$ and $\sum_{i=n_\beta+1}^N \beta_i = \frac{(1 - \alpha)^{n_\beta} - (1 - \alpha)^N}{\alpha} \leq \frac{1}{T\alpha}$, continuing the above derivation, we have:

$$J(\bar{\pi}) \leq T\epsilon_{min} + T\epsilon_{regret} + \frac{Tl_{max}}{N} \left(1 + \frac{\log \frac{1}{T}}{\log(1 - \alpha)} + \frac{1}{\alpha} \right).$$

Given the fact $J(\hat{\pi}) = \min_{i \in 1:N} J(\hat{\pi}_i) \leq J(\bar{\pi})$ and representing the third term as $\mathcal{O}\left(\frac{f(T, l_{max})}{N}\right)$, we have proved the theorem. ■

3) *ADAPS:* With the assumption that we can treat the generated trajectories from our model and the additional data generated based on them as running a learned policy to sample independent expert trajectories at different states while performing policy roll-out, we have the following guarantee of ADAPS. To better understand the following theorem and proof, we recommend interested readers to read the proofs of Theorem 2 and 3 first.

Theorem 4: If the surrogate loss l upper bounds the true cost C , by collecting K trajectories using ADAPS at each iteration, with probability at least $1 - \mu, \mu \in (0, 1)$, we have the following guarantee:

$$J(\hat{\pi}) \leq J(\bar{\pi}) \leq T\hat{\epsilon}_{min} + T\hat{\epsilon}_{regret} + \mathcal{O}\left(Tl_{max} \sqrt{\frac{\log \frac{1}{\mu}}{KN}}\right).$$

Proof: Assuming at the i th iteration, our model generates K trajectories. These trajectories are independent from each other since they are generated using different parameters and at different states during the analysis of an accident. For the k th trajectory, $k \in [1, K]$, we can construct an estimate $\hat{l}_{ik}(\hat{\pi}_i) = \frac{1}{T} \sum_{t=1}^T l_i(\phi_{ikt}, \hat{\pi}_i, a_{ikt}^*)$, where $\hat{\pi}_i$ is the learned policy from data gathered in previous $i - 1$ iterations. Then, the approximated expected loss \hat{l}_i is the average of these K estimates: $\hat{l}_i(\hat{\pi}_i) = \frac{1}{K} \sum_{k=1}^K \hat{l}_{ik}(\hat{\pi}_i)$. We denote $\hat{\epsilon}_{min} = \min_{\pi \in \Pi} \frac{1}{N} \sum_{i=1}^N \hat{l}_i(\pi)$ as the approximated

minimal loss in hindsight after N iterations, then $\hat{\epsilon}_{regret} = \frac{1}{N} \sum_{i=1}^N \hat{l}_i(\hat{\pi}_i) - \hat{\epsilon}_{min}$ is the approximated average regret.

Let $Y_{i,k} = l_i(\hat{\pi}_i) - \hat{l}_{ik}(\hat{\pi}_i)$ and define random variables $X_{nK+m} = \sum_{i=1}^n \sum_{k=1}^K Y_{i,k} + \sum_{k=1}^m Y_{n+1,k}$, for $n \in [0, N-1]$ and $m \in [1, K]$. Consequently, $\{X_i\}_{i=1}^{NK}$ form a martingale and $|X_{i+1} - X_i| \leq l_{max}$. By Azuma-Hoeffding's inequality, with probability at least $1 - \mu$, we have $\frac{1}{KN} X_{KN} \leq l_{max} \sqrt{\frac{2 \log \frac{1}{\mu}}{KN}}$.

Next, we denote the expected loss of any policy $\pi \in \Pi$ under its own induced state distribution as $L(\pi) = \mathbb{E}_{\phi \sim d_{\pi}, a^* \sim \pi^*(\phi)} [l(\phi, \pi, a^*)]$ and consider $\bar{\pi}$ as the mixed policy that samples the policies $\{\hat{\pi}_i\}_{i=1}^N$ uniformly at the beginning of each trajectory. At each iteration, during the data collection, we only execute the learned policy instead of mix it with the expert's policy, which leads to $L(\hat{\pi}_i) = l(\hat{\pi}_i)$. Finally, we can show:

$$\begin{aligned}
\min_{i \in 1:N} L(\hat{\pi}_i) &\leq L(\bar{\pi}) \\
&= \frac{1}{N} \sum_{i=1}^N L(\hat{\pi}_i) \\
&= \frac{1}{N} \sum_{i=1}^N l_i(\hat{\pi}_i) \\
&= \frac{1}{KN} \sum_{i=1}^N \sum_{k=1}^K (\hat{l}_{ik}(\hat{\pi}_i) + Y_{i,k}) \\
&= \frac{1}{KN} \sum_{i=1}^N \sum_{k=1}^K \hat{l}_{ik}(\hat{\pi}_i) + \frac{1}{KN} X_{KN} \\
&= \frac{1}{N} \sum_{i=1}^N \hat{l}_i(\hat{\pi}_i) + \frac{1}{KN} X_{KN} \\
&\leq \frac{1}{N} \sum_{i=1}^N \hat{l}_i(\hat{\pi}_i) + l_{max} \sqrt{\frac{2 \log \frac{1}{\mu}}{KN}} \\
&= \hat{\epsilon}_{min} + \hat{\epsilon}_{regret} + l_{max} \sqrt{\frac{2 \log \frac{1}{\mu}}{KN}}.
\end{aligned}$$

Summing over T proves the theorem. ■

B. Network Specification

All modules within our control mechanism share a similar network architecture that combines Long Short-Term Memory (LSTM) [34] and Convolutional Neural Networks (CNN) [35]. Each image will first go through a CNN and then be combined with other images to form a training sample to go through a LSTM. The number of images of a training sample is empirically set to 5. We use the many-to-many mode of LSTM and set the number of hidden units of the LSTM to 100. The output is the average value of the output sequence.

The CNN consists of eight layers. The first five are convolutional layers and the last three are dense layers. The kernel size is 5×5 in the first three convolutional layers and 3×3 in the other two convolutional layers. The first three

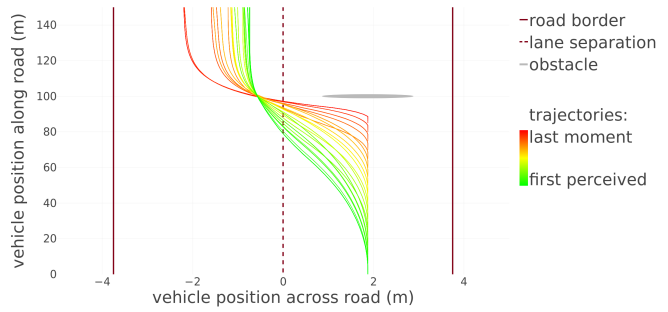


Fig. 4. Plotted collision-free trajectories generated by the expert algorithm for a vehicle traveling on the right lane of a straight road, with an obstacle in front. Spans 74 trajectories from the first moment the vehicle perceives the obstacle (green, progressive avoidance) to the last moment the collision can be avoided (red, sharp avoidance).

convolutional layers have a stride of two while the last two convolutional layers are non-strided. The filters for the five convolutional layers are 24, 36, 48, 64, 64, respectively. All convolutional layers use VALID padding. The three dense layers have 100, 50, and 10 units, respectively. We use ELU as the activation function and $\mathcal{L}2$ as the kernel regularizer set to 0.001 for all layers.

We train our model using Adam [36] with initial learning rate set to 0.0001. The batch size is 128 and the number of epochs is 500. For training *Detection* (a classification task), we use Softmax for generating the output and categorical cross entropy as the loss function. For training *Following* and *Avoidance* (regression tasks), we use mean squared error (MSE) as the loss function. We have also adopted cross-validation with 90/10 split. The input image data have 220×66 resolution in RGB channels.

C. Example Expert Trajectories

Figure 4 shows a set of generated trajectories for a situation where the vehicle had collided with a static obstacle in front of it after driving on a straight road. As expected, the trajectories feature sharper turns (red trajectories) as the starting state tends towards the last moment that the vehicle can still avoid the obstacle.

D. Learning From Accidents

For the following paragraph, we abusively note $s_k.x, s_k.y$ the position coordinates at state s_k , and $s_k.vx, s_k.vy$ the velocity vector coordinates at state s_k . Then, for any state $s_k \in \{s_{k_f}, \dots, s_{k_a}\}$ we can define a line $L(s_k) = \{l_u = (s_k.x, s_k.y) + u \times (-s_k.vy, s_k.vx) \mid u \in \mathbb{R}\}$. On this line, we note l_i the furthest point on $L(s_k)$ from $(s_k.x, s_k.y)$ which is at an intersection between $L(s_k)$ and a collision-free trajectory from $solve(\mathcal{S})$. This point determines how far the vehicle can be expected to stray from the original trajectory \mathcal{S} before the accident, if it followed an arbitrary trajectory from $solve(\mathcal{S})$. We also note l_{rl} and l_{rr} the two intersections between $L(s_k)$ and the road edges (l_{rl} is on the "left" with $rl > 0$, and l_{rr} is on the "right" with $rr < 0$). These two points delimit how far from the original trajectory

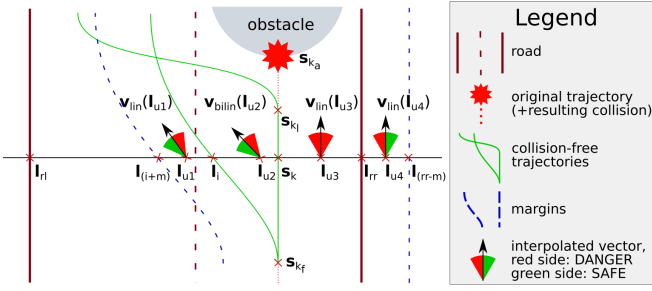


Fig. 5. (This figure is copied from the main text to here for completeness.) Illustration of important points and DANGER/SAFE labels from Section VI for a vehicle traveling on the right lane of a straight road, with an obstacle in front. Labels are shown for four points $\{l_{u1}, l_{u2}, l_{u3}, l_{u4}\}$ illustrating the four possible cases.

the vehicle could be. Finally, we define a user-set margin g as outlined below (we set $g = 0.5 m$).

Altogether, these points and margin are the limits of the region along the original trajectory wherein we generate images for training: a point $l_u \in L(s_k)$ is inside the region if it is between the original trajectory and the furthest collision-free trajectory plus a margin g (if l_u and l_i are on the same side, i.e. $sign(u) = sign(i)$), or if it is between the original trajectory and either road boundary plus a margin g (if l_u and l_i are not on the same side, i.e. $sign(u) \neq sign(i)$).

In addition, if a point $l_u \in L(s_k)$ is positioned between two collision-free trajectories $\hat{S}_{k1}, \hat{S}_{k2} \in solve(S)$, we consider the two closest states on \hat{S}_{k1} , and the two closest states from \hat{S}_{k2} , and bi-linearly interpolate these four states' velocity vectors, resulting in an approximate velocity vector $v_{bilin}(l_u)$ at l_u . Similarly, if a point $l_u \in L(s_k)$ is not positioned between two collision-free trajectories, we consider the two closest states on the single closest collision-free trajectory $\hat{S}_{k1} \in solve(S)$, and linearly interpolate their velocity vectors, resulting in an approximate velocity vector $v_{lin}(l_u)$ at l_u .

From here, we can construct images at various points l_u along $L(s_k)$ (increasing u by steps of $0.1 m$), with various orientation vectors (noted v_u and within 2.5 degrees of $(s_k.vx, s_k.vy)$), and label them using the following scheme (also illustrated in Figure 5). If the expert algorithm made the vehicle avoid obstacles by steering left (l_i with $i > 0$), there are four cases to consider when building a point l_u :

- $u < i + g$ and $u > i$: l_u is outside of the computed collision-free trajectories $solve(S)$, on the outside of the steering computed by the expert algorithm. The label is SAFE if $det(v_{lin}(l_u), v_u) \geq 0$, and DANGER otherwise.
- $u < i$ and $u > 0$: l_u is inside the computed collision-free trajectories $solve(S)$. The label is SAFE if $det(v_{bilin}(l_u), v_u) \geq 0$ (over-steering), and DANGER otherwise (under-steering).
- $u < 0$ and $u > rr$: l_u is outside the computed collision-free trajectories $solve(S)$ on the inside of the steering computed by the expert algorithm. The label is always DANGER.

- $u < rr$ and $u > rr - g$: l_u is in an unattainable region, but we include it to prevent false reactions to similar (but safe) future situations. The label is DANGER if $det(v_{lin}(l_u), v_u) > 0$, SAFE otherwise.

Here, the function $det(\cdot, \cdot)$ computes the determinant of two vectors from \mathbb{R}^2 .

Conversely, if the expert algorithm made the vehicle avoid obstacles by steering right (l_i with $i < 0$), there are four cases to consider when building a point l_u :

- $u > i - g$ and $u < i$: the label is SAFE if $det(v_{lin}(l_u), v_u) \leq 0$, and DANGER otherwise.
- $u > i$ and $u < 0$: the label is SAFE if $det(v_{bilin}(l_u), v_u) \leq 0$, and DANGER otherwise.
- $u > 0$ and $u < rl$: the label is always DANGER.
- $u > rl$ and $u < rl + g$: the label is DANGER if $det(v_{lin}(l_u), v_u) < 0$, SAFE otherwise.

We then generate images from these (position, orientation, label) triplets which are used to further train the *Detection* module of our policy.

E. Experiment Setup

1) *Scenarios*: We have tested our method in three scenarios. The first is a straight road which represents a linear geometry, the second is a curved road which represents a non-linear geometry, and the third is an open ground. The first two represent on-road situations while the last represents an off-road situation.

Both the straight and curved roads consist of two lanes. The width of each lane is $3.75 m$ and there is a $3 m$ shoulder on each side of the road. The curved road is half circular with radius at $50 m$ and is attached to two straight roads at each end. The open scenario is a $1000 m \times 1000 m$ ground, which has a green sphere treated as the target for the *Following* module to steer the AV.

2) *Vehicle Specs*: The vehicle's speed is set to $20 m/s$, which value is used to compute the throttle value in the simulator. Due to factors such as the rendering complexity and the delay of the communication module, the actual running speed is in the range of $20 \pm 1 m/s$. The length and width of the vehicle are $4.5 m$ and $2.5 m$, respectively. The distance between the rear axis and the rear of the vehicle is $0.75 m$. The front wheels can turn up to 25 degrees in either direction. We have three front-facing cameras set behind the main windshield, which are at $1.2 m$ height and $1 m$ front to the center of the vehicle. The two side cameras (one at left and one at right) are set to be $0.8 m$ away from the vehicle's center axis. These two cameras are only used to capture data for training *Following*. During runtime, our control policy only requires images from the center camera to operate.

3) *Obstacles*: For the on-road scenarios, we use a scaled version of a virtual traffic cone as the obstacle on both the straight and curved roads. This scaling operation is meant to preserve the obstacle's visibility, since at distances greater than $30 m$ a normal-sized obstacle is quickly reduced to just a few pixels. This is an intrinsic limitation of the single-camera setup (and its resolution), but in reality we can emulate this "scaling" using the camera's zoom function for instance.

For the off-road scenario, we use a vehicle with the same specifications as of the AV as the dynamic obstacle. This vehicle is scripted to collide into the AV on its default course when no avoidance behavior is applied by the AV.

4) *Training Data*: In order to train *Following*, we have built a waypoint system on the straight road and curved road for the AV to follow, respectively. By running the vehicle for roughly equal distances on both roads, we have gathered in total 65 061 images (33 642 images for the straight road and 31 419 images for the curved road). On the open ground, we have sampled 30 000 positions and computed the angle difference between the direction towards the sphere target and the forward direction. This gives us 30 000 training examples.

In order to train *Avoidance*, on the straight road, we rewind the accident by 74 frames starting from the frame that the accident takes place, which gives us 74 safe trajectories. On the curved road, we rewind the accident by 40 frames resulting in 40 safe trajectories. On the open ground, we rewind the accident by 46 frames resulting in 46 safe trajectories. By positioning the vehicle on these trajectories and capturing the image from the front-facing camera, we have collected 34 516 images for the straight road, 33 624 images for the curved road, and 33 741 images for the open ground.

For the training of *Detection*, using the mechanism explained in Subsection VI-B, we have collected 32 538 images for the straight road, 71 859 for the curved road, and 67 102 images for the open ground.