Attacking Vision-based Perception in End-to-End Autonomous Driving Models

Adith Boloor*, Karthik Garimella*, Xin He†, Christopher Gill*, Yevgeniy Vorobeychik*, Xuan Zhang*
*Washington University in St. Louis
{adith, kvgarimella, cdgill, yvorobeychik, xuan.zhang}@wustl.edu
†University of Michigan, Ann Arbor
{xinhe}@umich.edu

arXiv:1910.01907v1 [cs.LG] 2 Oct 2019

Abstract*—Recent advances in machine learning, especially techniques such as deep neural networks, are enabling a range of emerging applications. One such example is autonomous driving, which often relies on deep learning for perception. However, deep learning-based perception has been shown to be vulnerable to a host of subtle adversarial manipulations of images. Nevertheless, the vast majority of such demonstrations focus on perception that is disembodied from end-to-end control. We present novel end-to-end attacks on autonomous driving in simulation, using simple physically realizable attacks: the painting of black lines on the road. These attacks target deep neural network models for end-to-end autonomous driving control. A systematic investigation shows that such attacks are easy to engineer, and we describe scenarios (e.g., right turns) in which they are highly effective. We define several objective functions that quantify the success of an attack and develop techniques based on Bayesian Optimization to efficiently traverse the search space of higher dimensional attacks. Additionally, we define a novel class of *hijacking* attacks, where painted lines on the road cause the driverless car to follow a target path. Through the use of network deconvolution, we provide insights into the successful attacks, which appear to work by mimicking activations of entirely different scenarios. Our code is available on *https://github.com/xz-group/AdverseDrive

***Index Terms*—machine learning, adversarial examples, autonomous driving, end-to-end learning, bayesian optimization**

I. Introduction

With billions of dollars being pumped into autonomous vehicle research to reach Level 5 Autonomy, where vehicles will not require human intervention, safety has become a critical issue [3]. Remarkable advances in deep learning, in turn, suggest such approaches as natural candidates for integration into autonomous control. One way to use deep learning in autonomous driving control is in an end-to-end (e2e) fashion, where learned models directly translate perceptual inputs into control decisions, such as the vehicle's steering angle. Indeed, recent work demonstrated such approaches to be remarkably successful, particularly when learned to imitate human drivers [4].

Despite the success of deep learning in enabling greater autonomy, a number of parallel efforts also have exhibited concerning fragility of deep learning approaches to small adversarial perturbations of inputs such as images [5], [6]. Moreover, such perturbations have been shown to effectively translate to physically realizable attacks on deep models, such as placing stickers on stop signs to cause miscategorization of these as speed limit signs [2]. Fig. 1(a) offers several canonical illustrations.

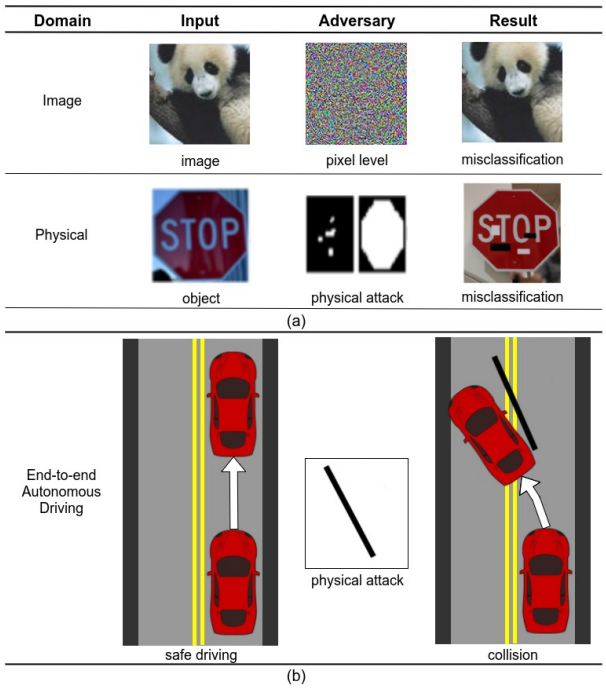


Fig. 1. (a) Existing attacks on machine learning models in the image [1] and the physical domain [2]; (b) conceptual illustration of potential physical attacks in the end-to-end driving domain studied in our work.

There is, however, a crucial missing aspect of most adversarial example attacks to date: manipulations of the physical environment that have a demonstrable *physical* impact (e.g., a crash). For example, typical attacks consider only prediction error as an outcome measure and focus either on a static image, or a fixed set of views, without consideration of the dynamics of closed-loop autonomous control. To bridge this gap, our aim is to study *end-to-end* adversarial examples. We require such adversarial examples to: 1) modify the physical environment, 2) be simple to implement, 3) appear unsuspicious, and 4) have a physical impact, such as causing an infraction (lane violation or collision). The existing attacks that introduce carefully

This research was partially supported by NSF grants CNS-1739643, IIS-1905558 and CNS-1640624, ARO grant W911NF1610069 and MURI grant W911NF1810208.

engineered manipulations fail the simplicity criterion [5], [7], whereas the simpler physical attacks, such as stickers on a stop sign, are evaluated solely on prediction accuracy [2].

The particular class of attacks we systematically study is the painting of black lines on the road, as shown in Fig. 1(b). These are unsuspicious since they are semantically inconsequential (few human drivers would be confused) and are similar to common imperfections observed in the real world, such as skid marks or construction markers. Furthermore, we demonstrate a systematic approach for designing such attacks so as to maximize a series of objective functions, and demonstrate actual physical impact (lane violations and crashes) over a variety of scenarios, in the context of end-to-end deep learning-based controllers in the CARLA autonomous driving simulator [8].

We consider scenarios where correct behavior involves turning right, left, and driving straight. Surprisingly, we find that right turns are by far the riskiest, meaning that the right scenario is the easiest to attack; on the other hand, as expected, going straight is comparatively robust to our class of attacks. We use network deconvolution to explore the reasons behind successful attacks. Here, our findings suggest that one of the causes of controller failure is partially mistaking painted lines on the road for a curb or barrier common during left-turn scenarios, thereby causing the vehicle to steer sharply left when it would otherwise turn right. By increasing the dimensionality of our attack space and using a more efficient Bayesian optimization strategy, we are able to find successful attacks even for cases where the driving agent needs to go straight. Our final contribution is a demonstration of novel *hijacking* attacks, where painting black lines on the road causes the car to follow a target path, even when it is quite different from the correct route (e.g., causing the car to turn left instead of right).

This paper is an extension our previous work [9], with the key additions of new objective functions, a new optimization strategy, Bayesian Optimization, and a new type of adversary in the form of hijacking self-driving models. In this paper, we first talk about relevant prior work on deep neural networks, adversarial machine learning in the context of autonomous vehicles, in Section II. Then in Section III we define the problem statement and present several objective functions that mathematically represent the problem statement. In Section IV, we introduce some optimization strategies. In Section V, we discuss our experimental setup including our adversary generation library and simulation pipeline. Section VI shows how we were able to successfully generate adversaries against e2e models, and presents a new form of attack, dubbed the hijacking attack where we control the route of the e2e model.

II. Related Work

A. Deep Neural Networks for Perception and Control

Neural Networks (NN) are machine learning models that consist of multiple layers of neurons, where each neuron implements a simple function (such as a sigmoid function), and where the output is a prediction. Deep Neural Networks (DNNs) are neural networks with more than two layers of neurons, and have come to be the state-of-the-art approach for a host of problems in perception, such as image classification and semantic segmentation [10]. Despite having complete autonomous driving stacks which include trained DNN models for perception, a series of real-world crashes involving autonomous vehicles demonstrate the stakes, and some of the existing limitations of the technology [11], [12].

B. End-to-end Deep Learning

End-to-end (e2e) learning models are comprised of DNNs that accept raw input parameters in one end and directly calculate the desired output at the other end. Rather than explicitly decomposing a complex problem into its constituent parts and solving them separately, e2e models directly generate the output from the inputs. It is achieved by applying gradient-based learning to the system as a whole. Recently, e2e models have been shown to have good performance in the domain of autonomous vehicles, where the forward facing camera input can be directly translated to control (steer, throttle and brake) commands [13]–[16].

C. Attacks on Deep Learning for Perception and Control

Adversarial examples (also called attacks, and adversaries) [5], [17]–[19] are deliberately calculated perturbations to the input which result in an error in the output from a trained DNN model. The idea of using adversarial examples against static image classification models demonstrated that DNNs are highly susceptible to carefully designed pixel-level adversarial perturbations [5], [7], [20]. More recently, adversarial example attacks have been implemented in the physical domain [2], [6], [21], such as adding stickers to a stop sign that result in misclassification [2]. However, these attacks still focus on perception models disembodied from the target application, such as autonomous driving, and few efforts study such attacks deployed directly on dynamical systems, such as autonomous driving [22], [23].

III. Modeling Framework

In this paper, we focus on exploring the influence of a physical adversary that successfully subverts RGB camera-based e2e driving models. We define physical adversarial examples as attacks that are physically realizable in the real world. For example, deliberately painted shapes on the road or on stop signs would be classified as physically realizable. Fig. 1(b) displays the conceptual view of such an attack involving painting black lines. We define our adversarial examples as *patterns*. To create an adversarial example that forces the e2e model to crash the vehicle, we need to choose the parameters of *pattern*'s shape that maximize the objective functions that we present. This may cause the vehicle to veer into the wrong lane or go offroad, which we characterize as a successful attack. Conventional gradient-based attack techniques are not directly applicable, since we need to run simulations (using the CARLA autonomous driving simulator) both to implement

an attack pattern, and to evaluate the end-to-end autonomous driving agent's performance.

At the high level, our goal is to paint a pattern (such as a black line) somewhere on the road to cause a crash. We formalize such attacks in terms of optimizing an objective function that measures the success of the attack pattern at causing driving infractions. Since driving infractions themselves are difficult to optimize because of discontinuity in the objective (infraction either occurs, or not), one of our goals it to identify a high-quality proxy objective. Moreover, since the problem is dynamic, we must consider the impact of the object we paint on the road over a sequence of frames that capture the road, along with this pattern, as the vehicle moves towards and, eventually, over the modified road segment. Crucially, we modify the road itself, which is subsequently captured in vision, digitized, and used as input into the e2e model's controller.

To formalize, we now introduce some notation. Let δ refer to the pattern painted on the road, and let l denote the position on the road where we place the pattern. We use L to denote the set of feasible locations at which we can position the adversarial pattern δ, and S the set of possible patterns (along with associated modifications; in our case, we consider either a single black line, or a pair of black lines, with modifications involving, for example, the distance between the lines, and their rotation angles). Let a_l be the state of the road at position l, and $a_l + \delta$ then becomes the state of the road at this same position when the pattern δ is added to it. The state of the road at position l is captured by the vehicle's vision system when it comes into view; we denote the frame at which this location initially comes into view by F_l, and let Δ be the number of frames over which the road in position l is visible to the vehicle's vision system. Given the road state a_l at position l, the digital view of it in frame F is denoted by $I_F(a_l)$ or simply I_F. Finally, we let $\theta_F = g_{sa}(I_F)$ denote the predicted steering angle given observed digital image corresponding to frame F. With this formalism established, we introduce several candidates for a proxy objective function that would quantify the success of an attack.

A. Candidate Objective Functions

1) Steering Angle Summations: First, we denote the vector of predicted steering angles during an episode *with an attack* δ starting from frame F_l to frame $F_{l+\Delta}$ as:

$$\vec{\Theta}_\delta = [\theta_{F_l}, \theta_{F_{l+1}}, \cdots, \theta_{F_{l+\Delta}}] \tag{1}$$

We define two objective functions as:

$$\text{Collide Right}: \max_{l,\delta} \sum_{i=0}^{\Delta} \vec{\Theta}_{\delta_i} \tag{2a}$$

$$\text{Collide Left}: \min_{l,\delta} \sum_{i=0}^{\Delta} \vec{\Theta}_{\delta_i} \tag{2b}$$

$$\text{subject to}: \quad l \in L, \quad \delta \in S. \tag{2c}$$

Equation 2a says that to optimize an attack that causes the vehicle to veer off towards the right and collide, we need to maximize the sum of steering angles for that particular experiment for the frames in which the pattern is in view. And similarly in Equation 2b, we need to minimize the steering sum, to make the vehicle veer left. We convert Equation 2b to a maximization problem for consistency in our search procedures that we will describe. Using Equation 2 as the objective function allows us to have control over which direction we would like the car to crash. The following two metrics, the absolute steering angle difference and path deviation, lose this ability to distinguish direction-based attacks, since they are essentially L-1 and L-2 norms.

2) Absolute Steering Angle Differences: Again, let's denote the predicted steering angles *with an attack* δ over the frames F_l to $F_{l+\Delta}$ as $\vec{\Theta}_\delta$ as shown in Equation 1. Now, let's denote the predicted steering angles *without any attack* over the same frames as $\vec{\Theta}_{\text{baseline}}$. This represents an episode where no attack is added to the road (we refer to this as the baseline run) and the car travels the intended path with minimal infractions. We can now define our second candidate metric as:

$$\max_{l,\delta} ||\vec{\Theta}_\delta - \vec{\Theta}_{\text{baseline}}||_1 \tag{3a}$$

$$\text{subject to}: \quad l \in L, \quad \delta \in S. \tag{3b}$$

Equation 3a optimizes an attack over the frames Δ that cause the largest absolute deviation in predicted steering angles with respect to the predicted steering angles when no pattern has been added to the road.

3) Path Deviation: First denote the (x, y) position of the agent from frames F_l to $F_{l+\Delta}$ *with an attack* δ as:

$$\vec{\text{p}}_\delta = [(x_l, y_l), (x_{l+1}, y_{l+1}), \cdots, (x_{l+\Delta}, y_{l+\Delta})] \tag{4}$$

Define $\vec{\text{p}}_{\text{baseline}}$ as the position of the agent *with no attack* added to the road over the same frames (the baseline run). We can optimize the path deviation from the baseline path:

$$\max_{l,\delta} ||\vec{p}_\delta - \vec{p}_{\text{baseline}}||_2 \tag{5a}$$

$$\text{subject to}: \quad l \in L, \quad \delta \in S. \tag{5b}$$

Similar to Equation 3a, we can use this metric to optimize deviation from the baseline route, except we are now attacking the position of the vehicle which is directly influenced by the outputs of the e2e models.

IV. Approaches for Generating Adversaries

We now describe our approaches for computing adversarial patterns or, equivalently, optimizing the objective functions defined above.

A. Random and Grid Search

Each *pattern* we generate (labeled earlier as δ) can be described by a set of parameters such as length, width, and rotation angle with respect to the road. Two naive methods of finding successful attacks would be to generate a pattern through either a random or grid search (using a coarse grid) and evaluate this pattern using one of the above objective

functions. Algorithm 1 shows this setup. The function *RunScenario()* runs the simulation and returns data such as vehicle speed, predicted acceleration, GPS position, and steering angle. We use these results to calculate one of the objective functions (*CalculateObjectiveFunction()*). As our goal is to maximize this metric, we use *MetricsList* to store the results of the objective function at each iteration. Finally, we return the parameters that maximized our objective function.

Algorithm 1 Adversary Search Algorithm

Require: Strategy $\in$ Random, Grid
  $i \leftarrow 0$
  MetricsList $\leftarrow$ [ ]
  loop
    $\delta_i \leftarrow$ GenerateAttack(Strategy)
    results $\leftarrow$ RunScenario(δ_i)
    $y_i \leftarrow$ CalculateObjectiveFunction(results)
    MetricsList.append(y_i)
    $i \leftarrow i + 1$
  end loop
  return $\arg\max$ MetricsList

B. Bayesian Optimization Search Policy

Algorithm 1 works well when the number of parameters for δ are relatively small. For a larger pattern space, and to enable us to explore the space more finely, we turn to Bayesian Optimization, which is designed for optimizing an objective function that is expensive to query without requiring gradient information [24]. It has been shown that Bayesian Optimization (BayesOpt) can be useful for optimizing expensive functions in various domains such as hyper-parameter tuning, reinforcement learning, and sensor calibration [25]–[28]. In our case, since we use an autonomous driving simulator, it is *expensive* to run a simulation with a generated attack in order to find, for example, the sum of steering angles as shown in Equation (2). On average, one episode takes between 20 to 40 seconds depending upon the scenario; consequently, it is important for the optimization to be sample efficient.

-At the high level, our goal is to generate physical adversaries that successfully attack e2e autonomous driving models, where a successful attack can be quantified as trying to maximize some objective function $f(\delta)$. Our goal, therefore, is to find a physical attack, δ^*, such that:

$$\delta^* = \arg\max_{\delta} f(\delta), \tag{6}$$

where $\delta^* \in \mathcal{R}^d$ and d is the number of parameters of the physical attack. We first assume that the objective f can be represented by a Gaussian Process, which we denote by $GP(f, \mu(\delta), k(\delta, \delta'))$ with a mean function of $\mu(\delta)$ and a covariance function $k(\delta, \delta')$ [29]. We assume the prior mean function to be $\mu(\delta) = \mathbf{0}$ and the covariance function to be the Matérn $5/2$ kernel:

$$k(\delta, \delta') = \left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}r}{\ell}\right), \tag{7}$$

where r is the Euclidean distance between the two input points, $||\delta - \delta'||_2$, and ℓ is a scaling factor optimized during simulation run-time. Let's suppose that we have already generated several adversaries and evaluated our objective function f for each of these adversaries. We can denote this dataset as $\mathcal{D} = \{(\delta_1, y_1), \cdots, (\delta_{n-1}, y_{n-1})\}$. Therefore, if we would like to sample our function f at some point along the input space δ, we would obtain some posterior mean value $\mu_{f|\mathcal{D}}(\delta)$ along with a posterior confidence or standard deviation value of $\sigma_{f|\mathcal{D}}(\delta)$. As noted earlier, our objective function f is expensive to query. When we use Bayesian optimization to find the parameters that define our next adversary δ_n, we instead maximize a proxy function known as the acquisition function, $u(\delta)$. Compared to the objective function, it is trivial to maximize the acquisition function using an optimizer such as the L-BFGS-B algorithm with a number of restarts to avoid local minima. In our case, we utilize the Expected Improvement (EI) acquisition function. Given our dataset, $\mathcal{D}$, we first let y_{max} be the highest objective function value we have seen so far. The EI can be evaluated at some point δ as:

$$u(\delta) = \mathrm{E}[\max(0, f(\delta) - y_{\text{max}})]. \tag{8}$$

Given the properties of a Gaussian Process, this can be written in closed form as follows:

$$z = \frac{\mu_{f|\mathcal{D}}(\delta) - y_{\text{max}}}{\sigma_{f|\mathcal{D}}(\delta)}; \tag{9}$$

$$u(\delta) = (\mu_{f|\mathcal{D}}(\delta) - y_{\text{max}})\Phi(z) + \sigma_{f|\mathcal{D}}(\delta)\phi(z), \tag{10}$$

where Φ and ϕ are the cumulative and probability distribution functions of the Gaussian distribution. Effectively, the first term in the above acquisition function leads to exploiting information from previously generated adversaries to generate parameters for δ_n while the second term prefers exploring the input space of the adversary parameters. Given this setup, Algorithm 2 presents a Bayesian Optimization approach for generating and searching for adversarial patterns. In this algorithm, the Gaussian process is updated in each iteration, and the acquisition function reflects those changes. An initial warm-up phase where the adversary parameters are chosen at random and the simulation is queried for the objective function is used for hyper parameter tuning.

Algorithm 2 Bayesian Adversary Search Algorithm

  $i \leftarrow 0$
  MetricsList $\leftarrow$ [ ]
  loop
    $\delta_i \leftarrow \arg\max u(\delta)$
    results $\leftarrow$ RunScenario(δ_i)
    $y_i \leftarrow$ CalculateObjectiveFunction(results)
    MetricsList.append(y_i)
    Update Gaussian Process and $\mathcal{D}$ with (δ_i, y_i)
    $i \leftarrow i + 1$
  end loop
  return $\arg\max$ MetricsList

V. Experimental Methodology

This section introduces the various building blocks that we use to perform our experiments. Fig. 2 shows the overall architecture of our experimentation method, including the CARLA simulator block, the python client block, and how they communicate with each other to generate and test the attack patterns on the simulator.

A. Autonomous Vehicle Simulator

Autonomous driving simulators are often used to test autonomous vehicles for the sake of efficiency and safety [30]–[33]. After testing popular autonomous simulators [34]–[37], we chose to run our experiments on the CARLA [8] (CAR Learning to Act) autonomous vehicle simulator, due to its feature-set and ease of source code modification . With Unreal Engine 4 [38] as its backend, CARLA has sufficient flexibility to create realistic simulated environments, with a robust physics engine, lifelike lighting, 3D objects including roads, buildings, traffic signs, vehicles and pedestrians. Fig. 2 shows how the simulator looks in the third person view. It allows us to acquire sensor data like the camera image for each frame (camera view), vehicle measurements (speed, throttle, steering angle and brake) and other environmental metrics like how the vehicle interacts with the environment in the form of infractions and collisions. Since we use e2e models that use only the RGB camera, we disable the LiDAR, semantic segmentation, and depth cameras. Steering angle, throttle and brake parameters are the primary control parameters to drive the vehicle in the simulation. CARLA (v0.8.2) comes with two maps: a large training map and a smaller testing map which are used for training and testing the e2e models respectively. CARLA also allows the user to run experiments under various weather conditions like sunset, overcast and rain, which are determined by the client input. To keep consistent frame rate and execution time, we run CARLA using a fixed time-step.

B. End-to-end Driving Models

The CARLA simulator comes with two trained end-to-end models: Conditional Imitation Learning (IL) [39] and Reinforcement Learning (RL) [8]. Their commonality ends at using the camera image as the input to produce output controls that include steering angle, acceleration, and brake. The IL model uses a trained network which consists of demonstrations of human driving on the simulator. In other words, the IL model tries to mimic the actions of the expert from whom it was trained [40]. The IL model's structure comprises of a conditional, branched neural architecture model where the *conditional* part is a high-level command given by the CARLA simulator at each frame. This high-level command can be *left, right or straight at an intersection, and lane follow when not at an intersection*.

At each frame, the image, current speed, and high-level command are used as inputs to the branched IL network to directly output the controls of the vehicle. Therefore, each branch is allocated a sub-task within the driving problem (making a decision to cross an intersection or following the current lane). The RL model uses a trained deep network based on a rewards system, provided by the environment based on the corresponding actions, without the aid of human drivers. More specifically, for RL, the asynchronous advantage actor-critic (A3C) algorithm was used. It is worth mentioning that the IL model performed better than the RL model in untrained (test) scenarios [8]. Because of this, we focus our research primarily on attacking the IL model.

TABLE I
Different types of attacks and their respective parameters and constraints. var - variable, const - constant, NA - Not Applicable

	Attack Type			
params	**Single Line**	**Double Line**	**Two Lines**	**N-Lines**
# lines	1	2	2	N
position	var	var	var	var
rotation	var	var	var	var
length	const	const	const	var
width	const	const	const	var
gap	NA	var	NA	NA
color	const	const	const	var
opacity	const	const	const	var
dimensions	2	3	4	N * 6

C. Physical Adversary Generation

1) Unreal Engine: To generate physically realizable adversaries in a systematic manner, we modify CARLA's source code. The CARLA simulator (v0.8.2) does not allow spawning of objects into the scene which do not already exist in the CARLA blueprint library (which includes models of vehicles, pedestrians, and props). With the Unreal Engine 4 (UE4), we create a new *Adversarial Plane Blueprint*, which is a 200×200 pixel plane or canvas with a dynamic UE4 material, which we can overlay on desired portions of the road. The key attribute of this blueprint is that it reads a generated attack image (a .png file) and places it within CARLA in real time. Hence this blueprint has the ability to continuously read an image via an HTTP server. The canvas allows the use of images with an alpha channel which allows attacks which are partly transparent, like the one shown in Fig. 3. Then, we clone the two maps that are provided by CARLA and choose regions of interest within each of them where attacks spawn. Some interesting regions are at the turns and intersections. We place the *Adversarial Plane Blueprint* canvas in each of these locations. When CARLA runs, an image found on the HTTP server gets overlaid on each canvas. Finally, we compile and package this modified version of CARLA. Hence we are able to place physical attacks within the CARLA simulator.

2) Pattern Generator Library: We built a pattern generator that creates different kinds of shapes as shown in Fig. 3 using the pattern parameters (Table I). For the pattern generator, we explore parameters like the position, width, and rotation of the line(s). We sweep the position and rotation from 0-200 pixels and 0-180 degrees respectively to generate variations of attacks. Similarly, we create a more advanced pattern which involves two parallel black lines called the *double-line* pattern as described in Table I. It comprises of the previous

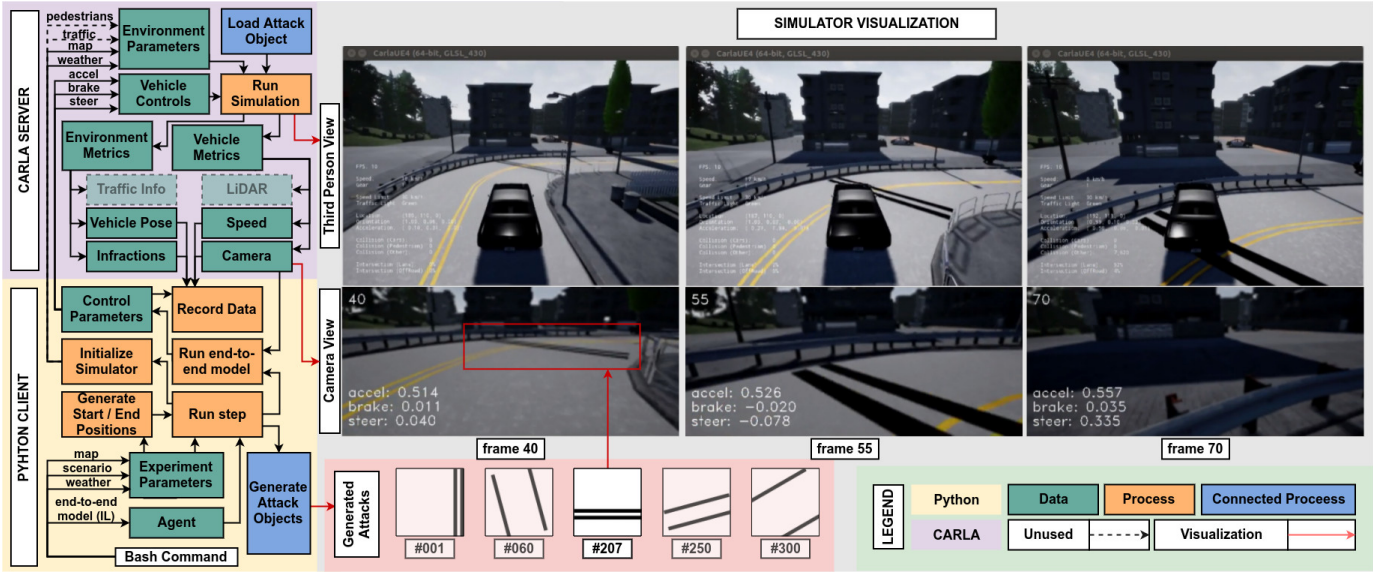
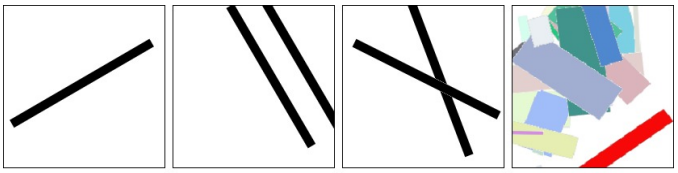


Fig. 2. Architecture overview of our simulation infrastructure including the interfaces between the CARLA simulator and the pattern generator scripts. Visualization of the camera and the third person views from one attack episode are also shown.



(a) Single Line (b) Double Line (c) Two Lines (d) N-Lines

Fig. 3. Attack Generator Capabilities. (a) shows the most basic attack which is a single line. (b) and (c) show attacks using two lines, but (b) has a constraint that the lines always need to be parallel, (d) shows the ability of the generator to generate N number of lines with various shapes and color.

parameters, namely, position, rotation, and width, with the addition of a new gap parameter which is the distance between the two parallel lines. Lastly, we remove the parallel constraint on double lines to increase the search space of the attacks while preserving simplicity. Fig. 2 shows some examples of the generated double line patterns which can be seen overlaid on the road in frames 55 and 70.

Additionally, our library has the ability to read a dictionary object containing the number of lines, the parameters (position, rotation, width, length and color) for each line, and produce a corresponding attack pattern as shown in Fig. 3 (d). Once the pattern is generated, it is read via the HTTP server and is placed within the Carla simulator.

3) OpenAI-Gym Environment for Carla: Since CARLA runs nearly in real-time, experiments take a long time to run. In order to efficiently run simulations with our desired parameters, we convert the CARLA setup to an OpenAI-Gym environment [41]. While the OpenAI-Gym framework is primarily used for training reinforcement learning models, we find the format helpful as we are able to easily run the simulator with a set of initial parameters like the task (straight, right, left), the map, the scene, the end-to-end model and the desired output metric (eg. average infraction percent for that episode). With this set up, we are able to use an optimizer to generate an attack with a set of defined constraints, run an episode and get the resulting output metric.

D. Experiment Setup and Parallelism

To ensure a broad scope to test the effectiveness of the different attacks in various settings, we conduct experiments by changing various environment parameters like the maps (training map and testing map), scenes, weather (clear sky, rain, and sunset), driving scenarios (straight road, right turn, and left turn), e2e models (IL and RL) and the entire search space for the patterns. Here, we describe the six available driving scenarios for CARLA:

1) *Right Turn*: the agent follows a lane that smoothly turns 90 degrees towards the right.
2) *Left Turn*: the agent follows a lane that smoothly turns 90 degrees towards the left.
3) *Straight Road*: the agent follows a straight path.
4) *Right Intersection*: the agent takes a right turn at an intersection.
5) *Left Intersection*: the agent takes a left turn at an intersection.
6) *Straight Intersection*: the agent navigates straight through intersection.

We choose the baseline scenarios (no attack) where the e2e models drive the vehicle with minimal infractions. We run the experiments at 10 frames per second (fps) and collect the following data for each camera frame (a typical experiment takes between 60 to 100 frames to run): camera image from the mounted RGB camera, vehicle speed, predicted acceleration, steering and brake, percentage of vehicle in the wrong lane, percentage of vehicle on the sidewalk (offroad), GPS position of the vehicle, and collision intensity. Fig. 2 shows this

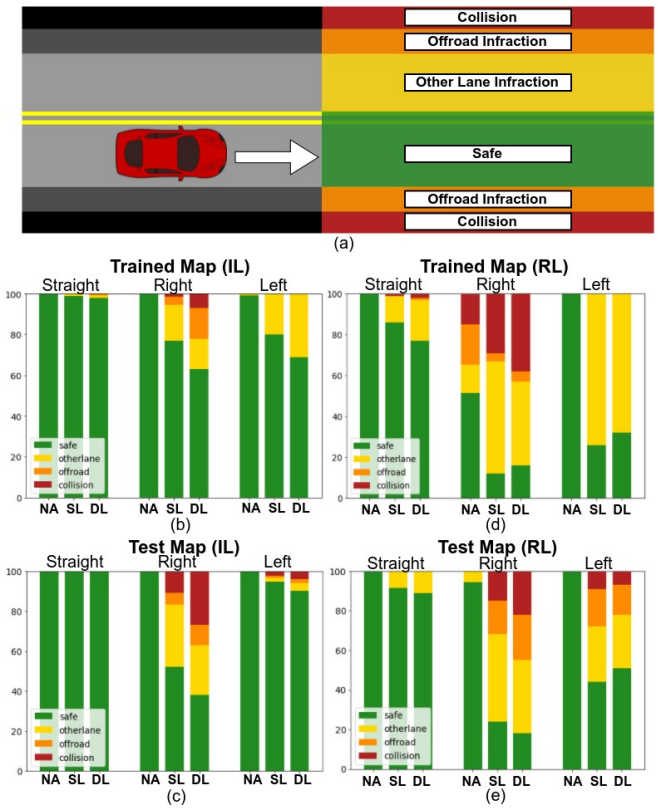


Fig. 4. Comparison of the infractions caused by different patterns. (a) Driving Infraction regions; (b)(c) Infraction percentages for IL; (d)(e) Infraction percentages for RL; NA - No Attack, SL - Single Line pattern, DL - Double Lines pattern; Straight - Straight Road Driving, Right - Right Turn Driving, Left - Left Turn Driving

dataflow which is sufficient to assess the ramifications of a particular attack in an experiment.

To search the design space thoroughly, we build a CARLA docker which allows us to run as many as 16 CARLA instances simultaneously, spread over 8 RTX GPUs [42].

VI. Experimental Results

Through experimentation, we demonstrate the existence of conspicuous physical adversaries that successfully break the e2e driving models. These adversaries do not need to be subtle or sophisticated modifications to the scene. Although they can be distinguished and ignored by humans drivers with ease, they effectively cause serious traffic infractions against the e2e driving models we evaluate.

A. Simple Physical Adversarial Examples

1) Effectiveness of Attacks: To begin, we generated two types of adversarial patterns: single line (with varying positions and rotation angles), and double lines (with varying positions, rotation angles, and distance between the lines). In Fig. 4(a), we define different safety regions of the road in ascending order of risk. We start with the vehicle's own lane (safe region), the opposite lane (unsafe), offroad/sidewalk (dangerous) and regions of collisions (very dangerous) past the offroad region. Fig.4(b)(c)(d)(e) shows that by sweeping through the three scenarios (straight road driving, right turn driving, left turn driving) with the single and double line patterns, for both the training map and testing maps, we see that some patterns cause infractions. Here we use a naive grid search approach to traverse the search space with the *Steering Sum optimization metric* defined in Equation 2a. First, we observe the transfer-ability of adversaries since some of our generated adversarial examples cause both IL (Fig.4(b)) and RL (Fig.4(d)) models to produce infractions. Second, attacks are more successful against the RL model than the IL model. Additionally, we notice that the double line adversarial examples cause more severe infractions than their single line counterparts. Lastly, we observe that *Straight Road Driving* and *Left Turn Driving* are more resilient to attacks that cause stronger infractions.

2) Analysis of Attack Objectives: To find the optimal adversary which produce infractions like collisions for the case of *Right Turn Driving* scenario, the optimizer has to find a pattern that maximizes the first candidate objective function: the sum of steering angles as hypothesized in Equation 2. A positive steering angle denotes steering towards the right and a negative steering angle implies steering towards the left. Fig. 5(a)(b) show the sum of steering angles and the sum of infractions respectively, for each of the 375 combinations of double line patterns. The infractions are normalized because collision data is recorded in SI units of intensity $[kg \times m/s]$, whereas the lane infractions are in percentages of the vehicle area in the respective regions. Fig. 5 also shows the three lowest points (minima) for the steering sum and the three highest points (maxima) for the collisions plot. In Fig. 5(c), we use the *argmin* and *argmax* on the set of attacks to observe the shapes of the corresponding adversarial examples for both the steering sum and infraction results. We observe that the *patterns that minimize the sum of steering angle and correspondingly maximize the collision intensity are very similar.* Thus, the objective based on maximizing or minimizing steering angles is clearly yielding valuable information for the underlying optimization problem. However, this does not mean that it's the best objective, among the three choices we considered above. We explore this issue in greater depth in the next subsection, as we move towards studying more complex attacks using Bayesian optimization.

B. Exploration of Large Design Spaces

In Fig. 4, we observe that when we switch from a *Single Line* attack (with 2 dimensions) to a *Double Line* attack (with 3 dimensions), in most cases, there is a significant increase in the number of successful attacks. It is reasonable to assume that as we increase the number of degrees of freedom in the attack, it should be possible to also increase the success rate. We lend further support to this intuition by considering an attack called the *Two Line* attack, shown in Fig. 3(c), with 4 dimensions by removing the constraint that the two lines must be parallel. As shown in Fig. 4, attack success rates increase considerably compared to the more restricted attack.

TABLE II
Comparison of Candidate Objective Functions as listed in Section III (in %). $\sum$ st. angles - sum of steering angles, abs. st. diff. - absolute steering difference

	Left				Straight				Right			
Metric	safe	collision	offroad	opp. lane	safe	collision	offroad	opp. lane	safe	collision	offroad	opp. lane
$\sum$ st. angles	18.2	0	0	81.8	99	0	0	1	72.2	9.5	13.8	24.5
path deviation	64.6	0	0	35.4	23.8	2.5	2.8	76.2	57.2	24.0	28.3	40.2
abs. st. diff.	**0.2**	0	0	**99.8**	**22.7**	**7.5**	**9.3**	**77.3**	**0**	**95.2**	**99.2**	**100**

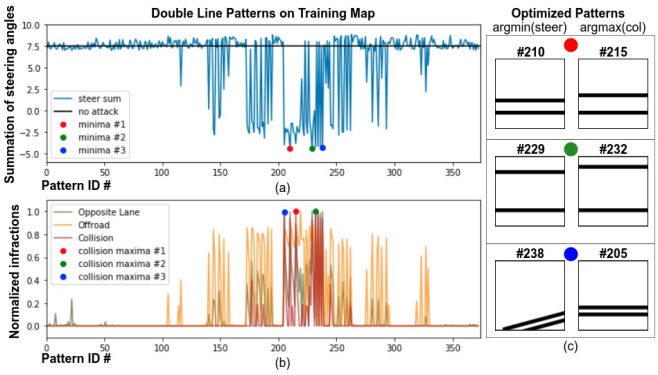


Fig. 5. Adversary against "Right Turn Driving". (a) Adversarial examples significantly changes the steering control. (b) Some patterns cause minor infractions whereas others cause level 3 infractions. (c) The patterns that cause the minimum steering sum and maximum collisions look similar.

However, increasing the dimensionality of the attack search space makes grid search impractical. For example, the Single Line attack with grid search requires around 375 iterations to sweep the search space at a 20 pixel resolution. Preserving the same parameter resolution (or precision) would require 1440 iterations for Double Lines, and 12,960 iterations for the Two Line attack. Naive search would require around 45 days to sweep through the search space for a *single scenario* on a modern GPU. Additionally, using a sparser resolution for the attack parameters means that we would not find potential attacks which can only be found at a higher resolution.

We address this issue by adopting the Bayesian Optimization framework (BayesOpt) for identifying attack patterns (introduced in Section IV-B). This requires a change in our search procedure as shown in Algorithm 2. In short, it uses the prior history of the probed search space to suggest the next probing point.

Fig. 6 shows the comparison between the 3 optimization techniques we employ for the straight, left-turn, and right-turn scenarios. We see that for all three cases, BayesOpt outperforms the naive grid search and the random search methods. In Fig. 6, BayesOpt uses 400 initial random points to sample the search space and subsequently samples 600 optimizing points. Hence, we observe that for the first 400 iterations, BayesOpt follows closely with random search, and after probing those initial random points, we observe a significant increase in the number of successful attacks.

Because we observe many more successful attacks against the Left and Right Turn scenarios as compared to the Straight Scenario, Fig. 6 further supports our notion that driving straight is harder to attack as compared to the right and left turn scenarios.

Equipped with BayesOpt, we now systematically evaluate the relative effectiveness of the different objective functions. Table II shows the infractions caused by each of the objective functions (path deviation, sum of steering angles, and absolute difference in steering angles with the baseline). For Left Turn, Straight Road, and Right Turn Driving, we list the percentage out of 600 simulation runs using BayesOpt that were safe, incurred collisions, off road infractions, or opposite lane infractions. We observe that the absolute difference in steering angles with respect to the baseline run is the strongest metric when coupled with BayesOpt to discover unique, successful attacks. While the most natural metric would seem to be *steering sum*, it is in practice considerably less effective than maximizing absolute difference in the steering angle. The *path deviation* objective function performs well in *right turn* and *straight scenarios*, but fails to find optimal attacks in the *left turn driving* scenario. Overall it still under-performs when compared to the absolute steering difference objective function.

C. Importance of Selecting a Reliable Objective Function

In Section VI-B, we evaluated three different objective functions: *path deviation*, *sum of steering angles* and *absolute steering difference*. We observed that the choice of the right objective function is crucial for success, and this choice is not necessarily obvious.

Most surprisingly, perhaps, we found that the objective that uses the steering angles to guide adversarial example construction is not the best choice, even though it is perhaps the first that comes to mind, and one used in prior work [32]. We now investigate why this choice of the objective can fail.

Fig. 7 shows three driving scenarios (left turn, driving straight, and right turn) respectively. Fig. 7(a) shows the paths taken by the vehicle for 3 cases: a *baseline* case where there is no attack, an *unsuccessful attack* case where an attack pattern does not cause the e2e model to deviate significantly from the *baseline* path, and a *successful attack* case where an attack causes a large deviation resulting in an infraction. Fig. 7(b) shows the sum of steering angles for each of the corresponding cases in Fig. 7(a). Note that for *Left Turn Driving*, we try to maximize Eq. (2a), which is to collide to the right, and for *Straight Driving* and *Right Turn Driving*, we maximize Eq. (2b), which is to collide to the left. For the *right turn driving* scenario, we observe that there is indeed a large difference between the steering sums for a strong attack and a weak attack, but in the other two scenarios, we notice

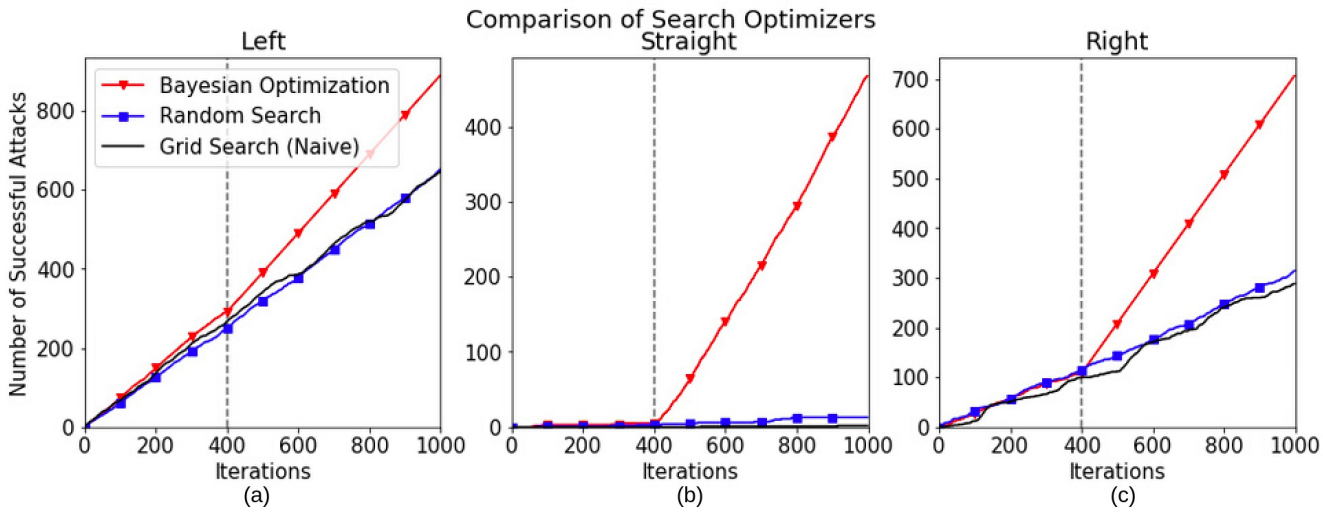


Fig. 6. A comparison of different search algorithms for generating successful attacks. In each driving scenario: Left Turn (a) , Straight Road (b), and Right Turn (c) Driving, the Bayesian approach not only finds more unique, successful adversaries in the same number of iterations but also finds these attacks at a faster rate. BayesOpt randomly samples the adversary search space for the first 400 iterations (shown before the dashed line) to tune the hyper-parameters of the kernel function. After these randomly sampled points, BayesOpt utilizes an acquisition function to sample the search space. While a dense grid search would eventually find at the least the same number of attacks as BayesOpt, we constrain our experiments to 1000 iterations given our computational resources.

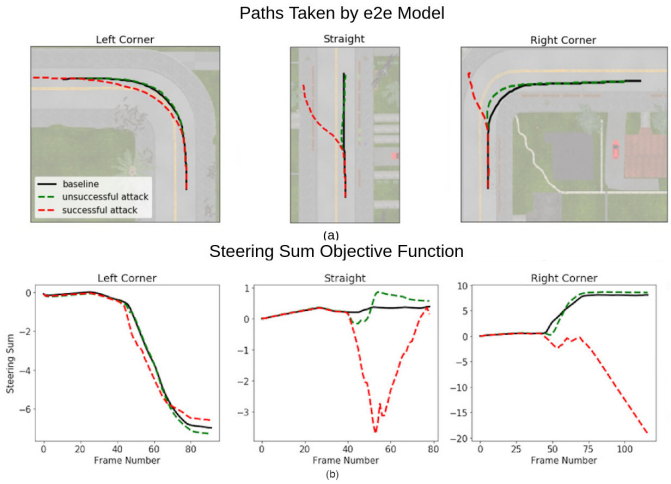


Fig. 7. Paths taken by e2e model in Left Turn, Straight Road, and Right Turn Driving with no attack (baseline), an unsuccessful attack, and a successful attack (a). Cumulative sum of steering angles for each scenario (b). While the successful attack is able to cause the e2e agent to incur an infraction or collision in each scenario, the steering sum metric is unable to capture distinguish between the successful and unsuccessful attack in two of the three scenarios.

that the baseline, unsuccessful attack and successful attack have very similar steering sums. Hence, the optimizer has a difficult time distinguishing between an unsuccessful and successful attack. In *straight driving scenario*, we see that the steering sum for a successful attack begins increasing and then sharply decreases, even though the vehicle has deviated significantly from the baseline path. This is due to the ability of the IL e2e model to recover in this case, resulting from data augmentation at training time where initial position of the car was randomly perturbed. The *sum of steering angles*

TABLE III
Success rate of Hijacking Attacks for six scenarios.

Hijack Success Rates	% Successful	% Unsuccessful
Straight → Right	14.8	85.2
Straight → Left	0.0	100.0
Left → Straight	23.7	76.3
Left → Right	14.3	85.7
Right → Left	1.4	98.6
Right → Straight	25.9	74.1

objective function is unable to capture this behavior. For the case of *left turn driving*, we discover that the successful attack not only causes a change in steering angle, but also a change in throttle, resulting in the vehicle speeding up and reaching a position further along the baseline path, which opens up new possibilities for generating attacks as well as causing new types of infractions.

The *absolute steering difference* mitigates the above issues by summing up the absolute steering differences between the baseline and attack cases. This allows the objective function to counteract the recovery ability of the e2e models. However, we do lose the ability to directly control the direction towards which we desire the vehicle to crash.

D. Vehicle Hijacking Attacks

Thus far, our exploration of adversarial examples against autonomous driving models focused on causing the car to crash, or cause other infractions. We now explore a different kind of attack: vehicle hijacking. In this attack, the primary purpose is to stealthily lead the car along a target path of the adversary's choice.

When attacking the IL model, previous experiments have only targeted the *Lane Follow* branch of this model. Now, we focus our attacks on three different branches of the IL Model:

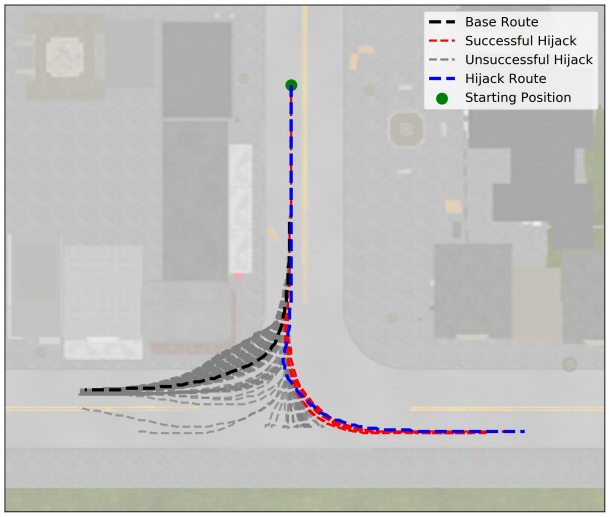


Fig. 8. Illustration of a *hijack attack* where we use an attack to trick the vehicle to deviate from its normal path (base route) to a target hijack route. It demonstrates a successful hijack where we make a vehicle otherwise turning right at an intersection, to turn left.

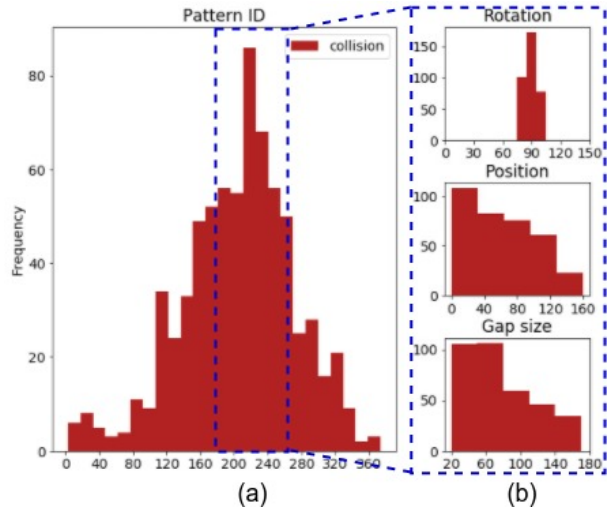


Fig. 9. (a) Histogram showing strong adversaries. (b) Depiction of range of rotation, position and gap parameters for the most robust adversaries.

Right Intersection, Left Intersection, and *Straight Intersection*. Here, we define a successful attack to be an adversary that 1) causes no infractions or collisions and 2) causes the agent to make a turn chosen by the attacker rather than the ground truth at a particular intersection (e.g. the attacker creates an adversary to make the agent turn left instead of go straight through an intersection). With this definition, an attack that causes the agent to incur an infraction is not considered to be a successful attack. In order to produce such attacks, we modify our experimental setup. After choosing a particular intersection, we run the simulation with no attack to record the baseline steering angles over the course of the episode. The high-level command provided by CARLA directs the agent to take a particular action at that intersection (for example, go straight). We then modify the CARLA high-level command to the direction desired by the attacker (for example, take a right turn). After running the simulation, we store these target steering angles over the entire episode. Finally, we revert the CARLA high-level command to the original command provided to the agent during the baseline simulation run and begin generating attacks at the intersection. We modify our optimization problem to minimize the difference in the steering angles recorded during an episode with an attack ($\vec{\Theta}_\delta$ as defined in III-A) and the steering angles of the target run (defined as $\vec{\Theta}_{\text{target}}$):

$$\min_{l,\delta} ||\vec{\Theta}_\delta - \vec{\Theta}_{\text{target}}||_1 \tag{11a}$$

$$\text{subject to}: \quad l \in L, \quad \delta \in S. \tag{11b}$$

CARLA (v0.8.2) did not include a four-way intersection in their provided maps, which constrain our experiments to a three-way junction as shown in Fig. 8. Of the six possible hijacking configurations, we were able to generate adversaries that successfully hijacked the car to take a desired route rather than the baseline route for five configurations. For example, Fig. 8 shows the car being hijacked to take a right turn instead of going straight. While we were able to produce attacks that incurred an infraction in each scenario shown in Fig. 8 (the gray paths), these episodes did not count as successful hijacks as the car did not take the target route. Table III shows the rate of successful attacks for the six available hijacking scenarios in CARLA v0.8.2. To conclude, we were able to modify our optimization problem and generate adversaries at intersections which caused the agent to take a hijacking route, rather than the intended route.

E. Interpretation of Attacks using DeConvNet

In this section, our goal is to better understand what makes the attacks effective. We begin by quantitatively analyzing the range of parameters of attacks that will generate the most robust attacks in the context of right turns. For simplicity, we analyze the Double Line attack whose parameters include rotation angle, position, and gap size. Fig. 9 shows a histogram of the collision incidence rates versus the pattern IDs, and its corresponding parameters for an experiment with 375 iterations. Fig. 9(b), in particular, shows that some parameters play a stronger role than others in generating a successful attack. For example, in this particular setting Double Line attacks, successful adversaries have a narrow range of rotation angles (90 - 115 degrees). Fig. 9(b) also shows that smaller gap sizes perform slightly better than larger ones.

To better understand the working mechanisms of the successful attack to the underlying imitation learning algorithm, we use network deconvolution, using a state-of-the-art technique, DeConvNet [43]. Specifically, we attach each CONV block (a convolution layer with ReLU and a batch normalizer) to a DeConv counterpart, since the backbone of the imitation learning algorithm is a convolutional neural network which consists of eight CONV blocks for feature extraction and two fully connected (FC) blocks for regression. Each

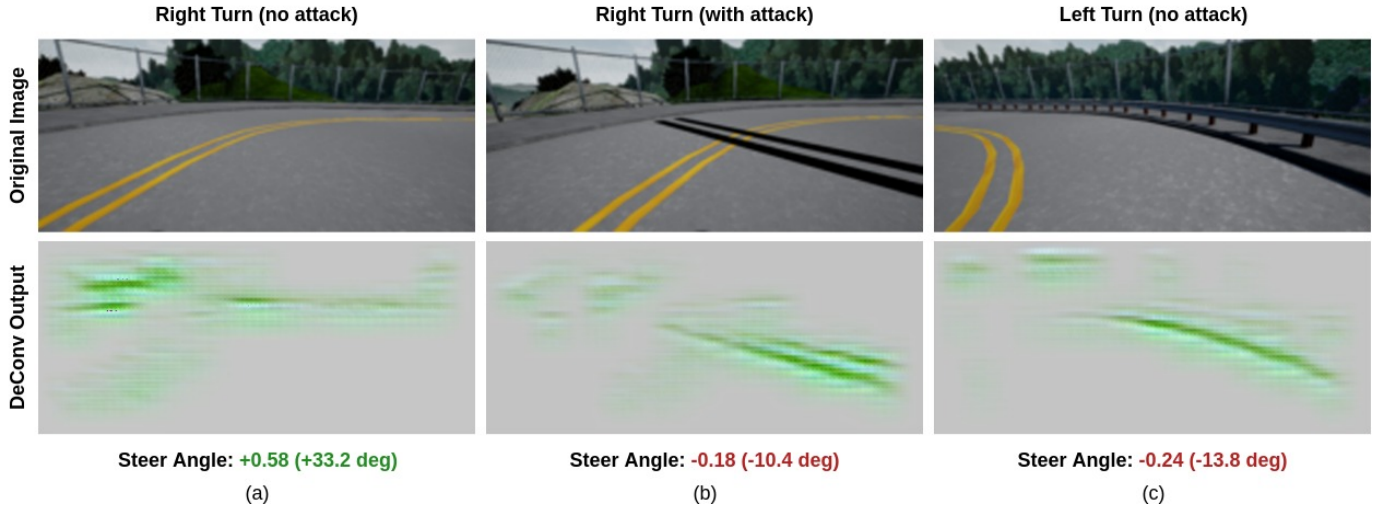


Fig. 10. Attacks against Right Turn Driving: The top row shows the camera input while the bottom deconvolution images show that the reconstructed inputs from the strongest activations determine the steering angle. (a) Right Turn Driving without attack, (b) Right Turn Driving with attack and (c) Left Turn Driving without attack for comparison

DeConv block uses the same filters, batch norm parameters, and activation functions as the CONV block, except that the operations are reversed. In this paper, DeConvNet is used merely as a probe to the already trained imitation learning network: it provides a continuous path to map high-level feature maps down to the input image. To interpret the network, the imitation learning network first processes the input image and computes the feature maps throughout the network layers. To view selected activations in the feature maps of a layer, other activations are set to zero, and the feature maps backtrack through the rectification, reverse-batch norm, and transpose layers. Then, activations that contribute to the chosen activations in the lower layer are reconstructed. The process is repeated until the input pixel space is reached. Finally, the input pixels which give rise to the activations are visualized. In this experiment, we chose the *top-200* strongest/largest activations in the *fifth* convolution layer and mapped these activations down to the input pixel space for visualization. The reasons behind this choice are twofold: 1) The strongest activations stand out and dominate the decision-making in NNs and the *top-200* activations are sufficient to cover the important activations, and 2) activations of the fifth CONV layer are more representative than other layers, since going deeper would mean that the amount of non-zero activations reduces significantly, which invalidates the deconvolution operations, while shallow layers fail to fully capture the relation between different extracted features.

We conduct a case study to understand why an attack works. Specifically, we take a deeper look inside the imitation network when adversaries are attacking the autonomous driving model for the right turn driving scenario. The baseline case without any attack is depicted in Fig. 10(a) while the one with a successful double-line attack is shown in Fig. 10(b). In the first row of Fig.10, the input images from the front camera mounted on the vehicle are displayed, which are fed to the imitation learning network. In Fig. 10(a), the imitation learning network guides the vehicle to turn right at the corner, as the steering angle output is set to a positive value (steering +0.58). The highlighted green regions in the reconstructed inputs in the corresponding second row show the imitation network makes this steering decision mainly following the curve of the double yellow line. However, when deliberate attack patterns are painted on the road as shown in Fig. 10(b), the imitation network fails to perceive the painted lines which could be easily ignored by a human; instead, the network regards the lines as physical barriers and guides the vehicle to steer left (steering -0.18) to avoid a fictitious collision, leading to an actual collision. The reconstructed image below confirms that the most outstanding features are the painted adversaries instead of the central double yellow lines. We speculate that the vehicle recognizes the adversaries as the road curb. And Fig. 10(c) confirms our speculations. In this case, the vehicle is turning left and the corresponding reconstructed image shows the curb would contribute the strongest activations in the network which will make the steering angle a negative value (steering -0.24) to turn left. The similarity of the reconstructed inputs between cases (b) and (c) suggests that the painted attacks are misrecognized as a curb which leads to an unwise driving decision. To summarize, the deliberate adversaries that mimic important road features are very likely to be able to successfully attack the imitation learning algorithm. This also emphasizes the importance of taking more diverse training samples into consideration when designing autonomous driving techniques. Note that since the imitation learning network makes driving decisions solely based on current camera input, using one frame per case for visualization is enough to unravel the root causes of an attack's success.

VII. Conclusion

In this paper, we develop a versatile modeling framework and simulation infrastructure to study adversarial examples on e2e autonomous driving models. Our model and simulation framework can be applied beyond the scope of this paper, providing useful tools for future research to expose latent flaws in current models with the ultimate goal of improving them. Through comprehensive experiment results, we demonstrate that simple physical adversarial examples that are easily realizable, such as mono-colored single-line and multi-line patterns, not only exist, but can be quite effective under certain driving scenarios, even for models that perform robustly without any attacks. We demonstrate that Bayesian Optimization coupled with a strong objective function is an effective approach to generating devastating adversarial examples. We also show that by modifying the objective function, we are able to hijack a vehicle where we cause the driverless car to deviate from its original route to a route chosen by an attacker. Finally, our analysis using the DeConvNet method offers critical insights to further explore attack generation and defense mechanisms. Our code repository is available at:

https://github.com/xz-group/AdverseDrive.

VIII. Acknowledgements

We would like to thank Dr. Ayan Chakrabarti for his advice on matters related to computer vision with this research and Dr. Roman Garnett for his suggestions regarding Bayesian Optimization. We would also like to thank the CARLA team for their technical support regarding the CARLA simulator. This research was partially supported by NSF grants CNS-1739643, IIS-1905558 and CNS-1640624, ARO grant W911NF1610069 and MURI grant W911NF1810208.

References

[1] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *CoRR*, vol. abs/1412.6572, 2015.

[2] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning models," 2017.

[3] R. Fan, J. Jiao, H. Ye, Y. Yu, I. Pitas, and M. Liu, "Key ingredients of self-driving cars," 2019.

[4] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. D. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *CoRR*, vol. abs/1704.07911, 2017.

[5] Y. Vorobeychik and M. Kantarcioglu, *Adversarial Machine Learning*. Morgan and Claypool, 2018.

[6] T. Dreossi, S. Jha, and S. A. Seshia, "Semantic adversarial deep learning," in *CAV*, 2018.

[7] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 372–387, 2016.

[8] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "Carla: An open urban driving simulator," in *CoRL*, 2017.

[9] A. Boloor, X. He, C. Gill, Y. Vorobeychik, and X. Zhang, "Simple physical adversarial examples against end-to-end autonomous driving models," in *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, pp. 1–7, June 2019.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, pp. 84–90, 2012.

[11] S. Alvarez, "Research group demos why tesla autopilot could crash into a stationary vehicle." https://www.teslarati.com/tesla-research-group-autopilot-crash-demo/, June 2018.

[12] T.S., "Why ubers self-driving car killed a pedestrian." https://www.economist.com/the-economist-explains/2018/05/29/why-ubers-self-driving-car-killed-a-pedestrian, May 2018.

[13] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. P. Monfort, U. Muller, J. Zhang, X. Zhang, J. J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016.

[14] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end autonomous driving," *CoRR*, vol. abs/1605.06450, 2016.

[15] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1856–1860, June 2017.

[16] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[17] A. Chernikova, A. Oprea, C. Nita-Rotaru, and B. Kim, "Are self-driving cars secure? evasion attacks against deep neural networks for steering angle prediction," 2019.

[18] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," 2018.

[19] D. Lowd and C. Meek, "Adversarial learning," in *KDD*, 2005.

[20] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014.

[21] J. Lu, H. Sibai, E. Fabry, and D. A. Forsyth, "No need to worry about adversarial examples in object detection in autonomous vehicles," *CoRR*, vol. abs/1707.03501, 2017.

[22] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, "Simulation-based adversarial test generation for autonomous vehicles with machine learning components," *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1555–1562, 2018.

[23] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?," *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182 – 193, 2016.

[24] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," 2010.

[25] P. I. Frazier, "A tutorial on bayesian optimization," 2018.

[26] M. O. R. Garnett and S. Roberts., "Bayesian optimization for sensor set selection," 2010.

[27] J. C. Barsce, J. A. Palombarini, and E. C. Martnez, "Towards autonomous reinforcement learning: Automatic setting of hyper-parameters using bayesian optimization," 2018.

[28] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, Prabhat, and R. P. Adams, "Scalable bayesian optimization using deep neural networks," 2015.

[29] C. E. Rasmussen, "Gaussian processes for machine learning," MIT Press, 2006.

[30] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *FSR*, 2017.

[31] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo em motion planner," *CoRR*, vol. abs/1807.08048, 2018.

[32] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 303–314, 2018.

[33] C. E. Tuncali, G. Fainekos, D. Prokhorov, H. Ito, and J. Kapinski, "Requirements-driven test generation for autonomous vehicles with machine learning components," 2019.

[34] C. Quiter and M. Ernst, "deepdrive/deepdrive: 2.0." https://doi.org/10.5281/zenodo.1248998, Mar. 2018.

[35] Baidu, "Apollo." http://apollo.auto/.

[36] NVIDIA, "Driveworks." https://developer.nvidia.com/drive/drive-software.

[37] Microsoft, "Microsoft airsim." https://github.com/microsoft/AirSim, 2018.

[38] Epic Games Inc., "What is unreal engine?." https://www.unrealengine.com, 2019.

[39] F. Codevilla, M. Miiller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–9, 2018.
[40] A. Attia and S. Dayan, "Global overview of imitation learning," 2018.
[41] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.
[42] NVIDIA Corporation, "What is geforce rtx?." https://www.nvidia.com/en-us/geforce/20-series/rtx/, 2019.
[43] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *ECCV*, 2014.