Adversarial Attacks of Automated Lane Centering Systems in Autonomous Vehicles

Ilies BENHABBOUR - Yassine GUENDOUZ

April 26, 2022

1 Introduction

Nowadays, more and more autonomous vehicles are using Automated Lane Centering (ALC) which consists in keeping a vehicle on the road by steering it so that it stays in a lane. However, important risks such as the deviation of the car in another lane or a collision with a barrier or a pedestrian make this system considered as sensitive due to its security issues.

This project aims at trying to find some vulnerabilities to make the car crash or have an accident in order to quantify the safety of ALC systems when attacking them through different methods. To do so, we worked on two different attacks, a billboard and a black lines attack and we mostly focused on the first one. We also used the simulator Carla[DRC+17] to make our experiments as well as Unreal Engine[Gam] to be able to add new features on the maps and OpenPilot[com] to try an ALC software.

2 DeepBillboard

2.1 Description

We saw during our research that studies were mainly focusing on providing physical attacks. Indeed, interacting with real environment is a way to create a reliable attack that may seem harmless to a driver but in fact is fooling the steering system using deep learning. In DeepBillboard[ZLK+20a], the authors try to reproduce this idea by creating a perturbation adapted to various situations that would make an automated vehicle goes into the wrong direction. This perturbation is generated by an algorithm depending on images with the coordinates of the billboard at each frame and then this perturbation is printed directly onto a billboard on roadside. In the paper, they justified the use of such a billboard because it can be rent by any person and in most of the cases it will just be ignored by drivers. Moreover, they imagined a case when just a part of the billboard contains the perturbation while the other part contains a dummy advertisement.

Figure 1: Picture of a roadside billboard

2.2 Goal of the attack

ALC systems use two types of deep learning networks. The first one is Convolutional Neural Network; this is the one we will focus on. It takes one image as the input and gives as an output the prediction of the steering angle. The second one is Recurrent Neural Network which uses feedback connections to predict the steering angle. In the paper, they imagined that the attack can be adapted to this architecture by modifying the calculation of the gradient when generating the billboard.

As previously told, the goal of Deepbillboard[ZLK+20b] is to create a perturbation which will try to modify the steering angle of the car. Let us define $X = \{x_0,\ x_1,\ ...\ ,\ x_n\}$ which is a set of image frames captured by a drive-by car. The goal here will be to maximize the distance between the ground truth steering angle and the prediction after having added the perturbation. For one picture it can be represented by the following equation:

$$max\ H(f(x+\delta), A_x)$$

Where H is a chosen distance function, $f(x)$ the prediction steering angle function, δ the perturbation which was added and A_x the ground truth steering angle. This equation can then be reformulated into the Lagrangian relaxed form:

$$arg\ \min_{\delta} -L(f(x+\delta), A_x)$$

Here L is the loss function that calculates the difference between the ground truth and the model's prediction steering angle. The problem with this equation is that we only take one image frame and try to create the highest deviation for only this image. The challenge here is that we want to introduce a perturbation that would affect all the frames that are captured by a drive-by vehicle. To do so, we need to adapt the optimization problem to all the images:

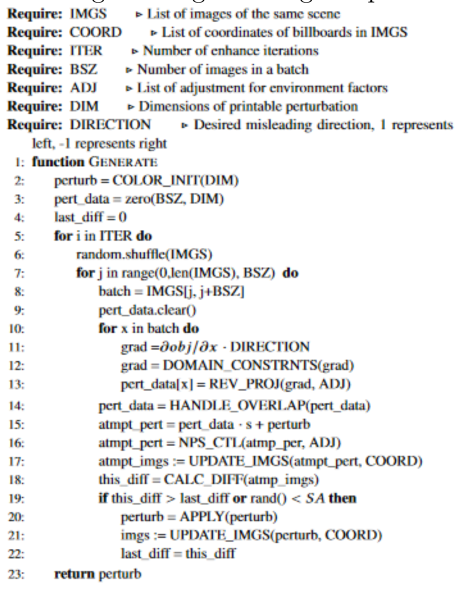
$$arg\ \min_{\Delta} \sum_{0<i<\|X\|} -L(f(x_i + p_i(\Delta)), A_{x_i})$$

$p_i(\Delta)$ denotes here the projection function of printable perturbation. This is needed because colors can be perceived differently according to the environment as for example the sunlight.

2.3 Generation of the billboard

To generate the perturbation, they had to train data with their own algorithm. They needed to capture a set of images driving on a road with a monochromatic billboard on the way. After having done this, they sent the inputs to the following algorithm.

Figure 2: Algorithm generating the perturbation



```
Require: IMGS        ▹ List of images of the same scene
Require: COORD       ▹ List of coordinates of billboards in IMGS
Require: ITER        ▹ Number of enhance iterations
Require: BSZ         ▹ Number of images in a batch
Require: ADJ         ▹ List of adjustment for environment factors
Require: DIM         ▹ Dimensions of printable perturbation
Require: DIRECTION   ▹ Desired misleading direction, 1 represents
    left, -1 represents right
function GENERATE
    perturb = COLOR_INIT(DIM)
    pert_data = zero(BSZ, DIM)
    last_diff = 0
    for i in ITER do
        random.shuffle(IMGS)
        for j in range(0,len(IMGS), BSZ) do
            batch = IMGS[j, j+BSZ]
            pert_data.clear()
            for x in batch do
                grad = $\partial obj/\partial x$ · DIRECTION
                grad = DOMAIN_CONSTRNTS(grad)
                pert_data[x] = REV_PROJ(grad, ADJ)
            pert_data = HANDLE_OVERLAP(pert_data)
            atmpt_pert = pert_data · s + perturb
            atmpt_pert = NPS_CTL(atmp_per, ADJ)
            atmpt_imgs := UPDATE_IMGS(atmpt_pert, COORD)
            this_diff = CALC_DIFF(atmp_imgs)
            if this_diff > last_diff or rand() < $SA$ then
                perturb = APPLY(perturb)
                imgs := UPDATE_IMGS(perturb, COORD)
                last_diff = this_diff
    return perturb
```

This algorithm is iteration based and it takes as input the images recorded before with the four coordinates of the billboard corners associated to each frame. It also needs to select the dimension of the perturbation, the direction (either right or left) and a batch size from which perturbation will be created. First, we need to create the perturbation which is monochromatic (line 2), in the paper it showed that the best options were gold, blue or green. After it is done, we need to create a perturbation matrix which is empty (line 3) and initiate a value (line 4) which will verify that at each step the perturbation increases the deviation compared to the previous iteration. Then the algorithm enters the loop and will process the images by random batches to avoid convergence to a non-optimal point at early video frames (line 6). The gradient is calculated and verified (line 11-12) and after it is done, it creates the perturbation with the help of the projection function. The pixels overlap need to be handled (line 14), in the paper they proposed 3 methods: the first one consists in taking the maximum gradient among proposed perturbation, the second is to update the overlapped pixels with the sum of the gradients and the last is to update the pixels with the gradient that has the most impact on the objective function. Finally, when the pixels overlapped are handled the algorithm needs to see whether the perturbation increases the deviation, if so, we update the perturbation (line 18-22).

2.4 Set up of a simulation environment

In order to reproduce and test the attacks we needed to use a simulator that could allow us to implement the ALC system as if we were doing physical attacks. As a consequence, we used Carla which is an open-source simulator created to support development, training, and validation of autonomous driving systems. We built the version 0.9.11 available on GitHub associated with Unreal Engine 4.24.3. It was important to build Carla directly with UE24 and not run the executable directly because otherwise we would not be able to create and add our billboards to the different maps already implemented in Carla. After having set up the Unreal Engine environment adapted to Carla we needed to add customized objects to the map. Therefore, we used Blender, a modeling software, to create two billboards: one big and one small. It was important to create billboards that can be modified easily since it can provide a real advantage over physical attacks. Indeed, with such environment, we can simulate the real world with accuracy and switch parameters easily. For example, it would be much harder to replace the billboard and run again and again the experiments in the real world. It is important to notice that in order to successfully use Carla in further works, a good setup is needed (an NVIDIA GPU is mandatory). The Carla GitHub provides a recommended system that would make the simulator run smoothly.

When everything related to the setup was over regarding Carla, we needed to create vehicles and run our experiments. Our choice was to use OpenPilot which is a well-known open-source ALC system developed by Comma.ai. Unfortunately, the driving neural network is closed-source for economic reason. Carla and OpenPilot will communicate with each other using a client-server connection. Here Carla is the server listening on port 2000 whereas OpenPilot uses the Carla Python API to send and receive data through that port. In order to avoid building OpenPilot, we used a lighter Docker version that is directly available on their GitHub. Hence, when we run Carla before OpenPilot, it creates a Tesla vehicle in a random position that can steer by itself, and we can start or stop it using the keyboard. After this, we can move the car to spawn it to a road where we can launch our experiments.

2.5 Launching the attacks

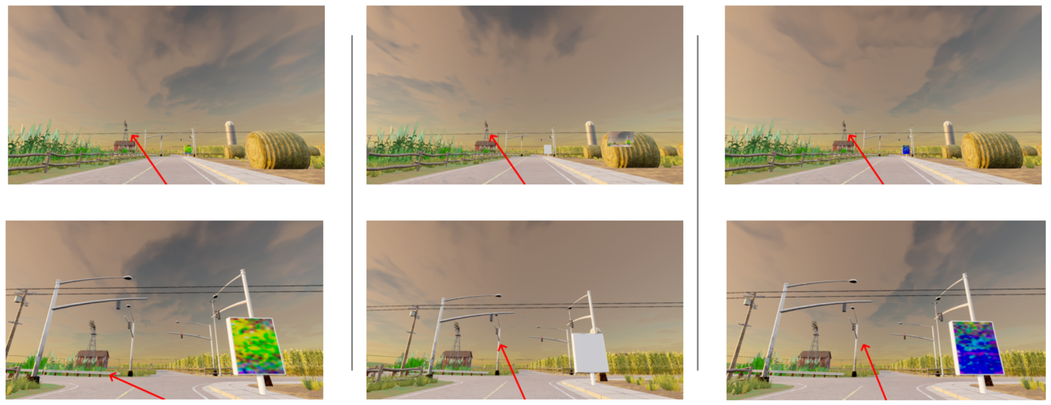
As we have described before, DeepBillboard is designed for Convolutional Neural Network and in order to see whether the attack works we need to use a driving neural network that is known and reliable. In their paper, they used four models and we tried to reproduce their results using Dave v1.

First, we placed our crafted billboard alongside the road. We then generated perturbations using the algorithm provided using their dataset for both directions (left and right). When we are done, we stick the correct perturbation to the material associated to the billboard in Carla (either it can be white when there is no perturbation, green for left and blue for right). We then launched OpenPilot to steer the car in the correct way. While the simulation is running, pictures are taken from the driver point of view and when the algorithm is stopped we can see whether the DAVE v1 algorithm steers differently according to the perturbation introduced.

2.6 Results

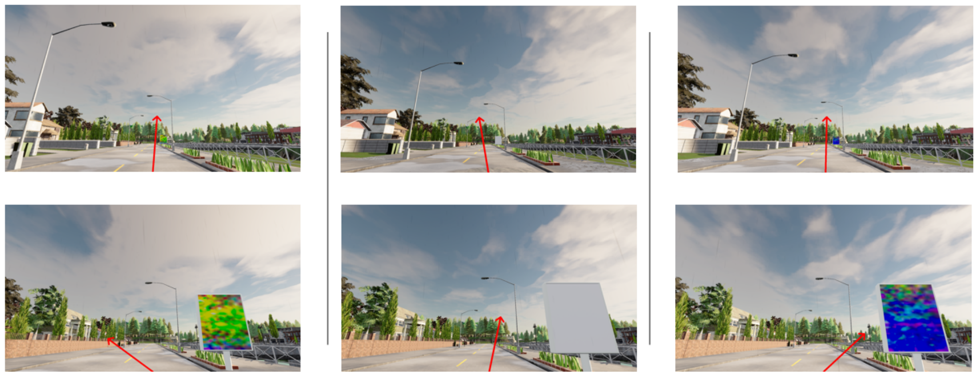
We did our experiments based with different parameters which could influence the prediction of the steering angle: the sunlight, the town, the direction and the dimension of the billboard. First, we tried with the map named Town 7 in Carla with a small billboard on a crossroad.

Figure 3: Prediction of the steering angle depending on the perturbation, here we use Town 7 with no sun. In the upper row we are far from the billboard, in the lower one we are near the billboard. On the left, we have the perturbation which misleads on the left, on the right, we have the right perturbation and in the middle, we have a billboard without attack



We can clearly see that a deviation is introduced when the camera is near the billboard, in other words in the late video frames for the left direction. The problem here is that the right direction perturbation has no real impact when we compare it to the white billboard. It might be due to the fact that the pretrained perturbation is not adapted for the environment (here the town) in our experiment. Therefore, we decided to reproduce the exact same experiment with Town 1 from Carla.

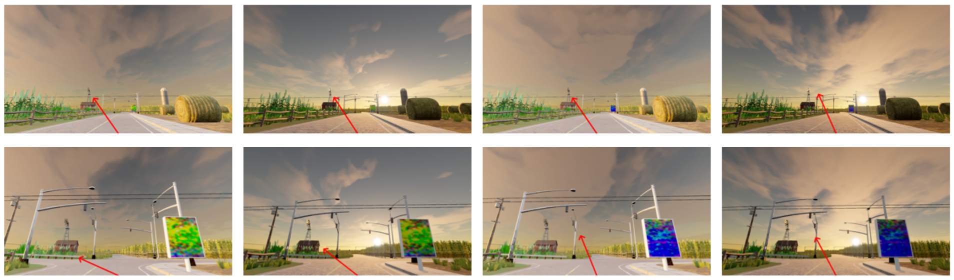
Figure 4: Steering angles in Town 1 with a small billboard



From these pictures, we can clearly see that the attacks work better than in Town 7. When the billboard is close, the perturbations make the vehicle steers on the left or on the right depending on which is used. However, when we pass the billboard the steering angle is approximately the same for all the pictures, we consider it as a normal state where there is no attack. We can imagine that it is a huge security problem since a car could have come from the opposite direction for the left perturbation or a pedestrian could have been walking on the right for the opposite perturbation. The issue in such a case is that the driver may not react on time because the angles are bigger and bigger and thus making the car goes on the left without them noticing.

As it was discussed previously, we wanted to see if the sun could have an impact on the perturbation or if the size of the billboard could improve our results. To do so, we used both Town 1 and Town 7.

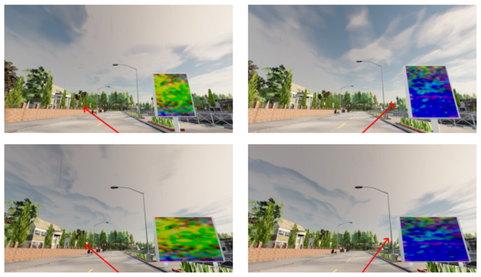
Figure 5: Comparison of the perturbation impact depending on the sunlight in Town 7



Here in the two first columns, we have the left perturbation, whereas we have the right one in the two last columns. We can say regarding the results that the light has an impact and it is more or less the result of what was defined in the projection function. However, the angles are not that impacted. Here, our cameras are not at the same distance from the billboard in all the cases and we assume this difference in the distance can be the parameter that influenced the small variations.

After having used a small billboard for our experiments, we crafted another one which would allow us to see if its dimensions have an impact or not on the predicted steering angle in order to improve our results for the first town.

Figure 6: Impact of the dimension of the billboard on the steering angle



We can see that the dimensions of the billboard had no real impact on the steering angle, it goes on the same direction with the same angle. One interesting thing could be to change the dimensions of the generated billboard when trained to match the new billboard size to see whether it has an impact.

3 Black lines attack

3.1 Description

We also focused on another type of attack that can make the car steer in another direction than the normal behaviour. The goal of this attack[BGH+19a] is to paint black lines on the road that will then be considered as lines that differentiate the lanes. This attack can be very effective because drivers may not take into consideration the risk that some black lines will deviate the vehicle because these lines can be considered as some imperfections on the road such as skid marks, constructions markers as well as shadows.

With these black lines, they defined two different types of attacks. The first one is the collision attack which makes the car go off its lane and collides either with another car that is coming in front, a barrier, a road sign, or a pedestrian. The second one is the hijacking attack which makes the car take another direction that the one that was expected. For example, at an intersection, we will make the car go left instead of turning right. Comparing both attacks, the first one is clearly the most dangerous because it creates an accident and can lead to more serious one whereas the second attack will only change the direction of the car without trying to make an accident.

3.2 Definition of the parameters

To do these attacks[BGH+19b], some black lines are painted on the road. Several parameters can be changed in order to make the attack more successful such as the thickness, the position on the road, the angle compared to the road, the number drawn (it could be either one or two) and the gap size between the black lines if there are more than one. In the paper, they made some experiments to have an approximate size of the different parameters. For example, if we want the attack to have a high probability of success, we should first paint on the road double lines, that is better than having a single line. Furthermore, they noticed that the rotation angle should be between 90 and 115 degrees and that small gap size between black lines performs better results than large ones. To determine if an attack is successful, we look at the steering angle of the car. For example, we know that if the steering angle increases, this means that the car is turning right. So, if we want to instead make the car turn left, our goal is to minimize the steering angle.

3.3 How the attack is done?

Both attacks rely on the same theory. They used the Bayesian Optimization and more specifically a Gaussian Process regression that is a particular approach for modelling functions. Bayesian optimization is a sequential design strategy (which means that the sample size is not fixed in advance) that is mostly used to optimize expensive-to-evaluate function. First, we must load the imitation learning network[csa] and execute the simulation in which no adversary has been placed on the road. We defined this scenario as the baseline run. The imitation learning network tries to learn a behaviour by mimicking a human. It first processes the input image that was taken from our dataset and then computes the feature maps throughout the network layers. Now that we have load the imitation learning and define our baseline scenario, we must define an objective function and try to maximise it because this is the goal of any optimization problem. They first assume that the objective function f can be represented by a Gaussian Process, which we denote by:

$$GP(f, \mu(\delta), k(\delta, \delta'))$$

Where μ is the mean function and k is the covariance function. They also assume the prior mean to be equal to 0 and the covariance function to be the Matérn 5/2 kernel:

$$k(\delta, \delta') = (1 + \frac{\sqrt{5}r}{l} + \frac{5r^2}{3l^2})exp(-\frac{\sqrt{5}r}{l})$$

Where r is the Euclidean distance between the two input points, and l is a scaling factor optimized during simulation run-time. Here the objective function will be defined as follow:

$$||\vec{\theta}_\delta - \vec{\theta}_{baseline}||$$

Where $\vec{\theta}_\delta$ is the steering angles collected over the simulation with an adversary placed on the road and $\vec{\theta}_{baseline}$ is the steering angles collected in the baseline scenario above with no adversary placed on the road.

Now that the objective function is defined, we are able to run a set of simulation in order to determine the objective function for each simulation. The Bayesian Optimization will choose the parameters of the attacks for each simulation. Finally to get the most successful attack, they defined an algorithm that will run different scenarios and calculate the objective function value of each scenario. This will be stored into a dataset D:

$$D = \{(\delta_1, y_1), ..., (\delta_{n-1}, y_{n-1})\}$$

Where δ is the pattern painted on the road (i.e. the black lines) and y is the objective function value corresponding to the scenario with the pattern δ.

The objective function f is expensive to query. When we use Bayesian Optimization to find the parameters that define our next adversary δ_n, they instead maximise an acquisition function, $u(\delta)$. They used the Expected Improvement (EI) acquisition function:

$$u(\delta) = E[max(0, f(\delta) - y_{max})]$$

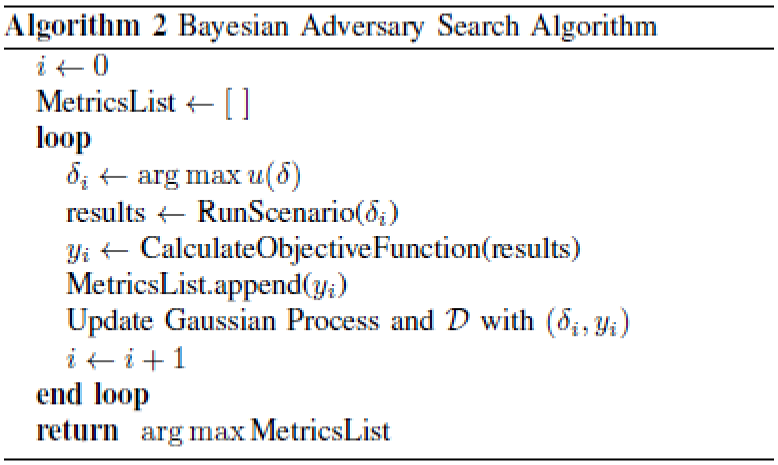
Where y_{max} is the highest objective function of the previous dataset. Furthermore, given the properties of a Gaussian Process, they were able to write the acquisition function as follow:

$$z = \frac{\mu_{f|D}(\delta) - y_{max}}{\sigma_{f|D}(\delta)}$$

$$u(\delta) = (\mu_{f|D}(\delta) - y_{max})\Phi(z) + \sigma_{f|D}(\delta)\Phi(z)$$

At the end, the algorithm will determine the best scenario that will make the car change its behaviour on the road and so create a collision or a change of direction.

Figure 7: Bayesian Optimization algorithm for generating and searching for adversarial patterns



```
Algorithm 2 Bayesian Adversary Search Algorithm
  $i \leftarrow 0$
  MetricsList $\leftarrow$ [ ]
  loop
    $\delta_i \leftarrow \arg\max u(\delta)$
    results $\leftarrow$ RunScenario($\delta_i$)
    $y_i \leftarrow$ CalculateObjectiveFunction(results)
    MetricsList.append($y_i$)
    Update Gaussian Process and $\mathcal{D}$ with $(\delta_i, y_i)$
    $i \leftarrow i + 1$
  end loop
  return $\arg\max$ MetricsList
```

4 Conclusion and future work

During this project, we mainly focused on the DeepBillboard attack and obtained some results using Dave v1 algorithm. However, it could be interesting to try the same experiments using Dave v2 or Dave v3 in order to compare the influence of each algorithm and to select the best one that will make security and safety issues on the car. Furthermore, we trained our model using the datasets of the DeepBillboard project. It could be interesting to try it with our own dataset to have more accurate results.

Moreover, we think that OpenPilot uses Recurrent Neural Network. For our experiments related to the DeepBillboard attack, we chose to use the attack based on the design for Convolutional Neural Network. As a result we could not experiment this attack directly on OpenPilot. A next goal could be to try to modify the calculation of the gradient as it is proposed in the paper in order to attack RNNs or OpenPilot. This attack can be interesting since OpenPilot is making reliable devices that can be used by many car models in real-life.

Unfortunately, we did not have enough time to implement the black lines attack on Carla. After having understood its principles, it would have been interesting to test the collision and hijacking attack. It would have also been great to determine by ourselves the best combination or the approximate size of the different parameters in order to have the most successful one. We can also determine the best scenario where the attack is the most successful. For example, which combination has the best results between a car that should have gone right but were hijacked to turn left or a car that should have gone straight but were hijacked to turn right, etc... Furthermore, they mainly used the imitation learning method for their experiments, but another point would have been to test the same experiments with the reinforcement learning method[csb]. Reinforcement learning is a machine learning training method based on rewarding desired behaviors and punishing undesired ones. As it is another way of training, we should obtain other results compared to imitation learning method. The comparison of both methods would be a good work for the future as well.

References

[BGH$^+$19a] Adith Boloor, Karthik Garimella, Xin He, Christopher Gill, Yevgeniy Vorobeychik, and Xuan Zhang. Attacking vision-based perception in end-to-end autonomous driving models, 2019.

[BGH+19b] Adith Boloor, Karthik Garimella, Xin He, Christopher Gill, Yevgeniy Vorobeychik, and Xuan Zhang. Attacking vision-based perception in end-to-end autonomous driving models. https://github.com/xz-group/AdverseDrive, 2019.

[com] commaai. Openpilot. https://github.com/commaai/openpilot.

[csa] carla simulator. Imitation learning. https://github.com/carla-simulator/imitation-learning.

[csb] carla simulator. Reinforcement learning. https://github.com/carla-simulator/reinforcement-learning.

[DRC+17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[Gam] Epic Games. Unreal engine. https://github.com/EpicGames/UnrealEngine/tree/4.24.

[ZLK+20a] H. Zhou, W. Li, Z. Kong, J. Guo, Y. Zhang, B. Yu, L. Zhang, and C. Liu. Deepbillboard: Systematic physical-world testing of autonomous driving systems. 2020.

[ZLK+20b] H. Zhou, W. Li, Z. Kong, J. Guo, Y. Zhang, B. Yu, L. Zhang, and C. Liu. Deepbillboard: Systematic physical-world testing of autonomous driving systems. https://github.com/deepbillboard/DeepBillboard, 2020.