Too Good to Be Safe: Tricking Lane Detection in Autonomous Driving with Crafted Perturbations

Pengfei Jing, *The Hong Kong Polytechnic University and Keen Security Lab, Tencent;* Qiyi Tang and Yuefeng Du, *Keen Security Lab, Tencent;* Lei Xue and Xiapu Luo, *The Hong Kong Polytechnic University;* Ting Wang, *Pennsylvania State University;* Sen Nie and Shi Wu, *Keen Security Lab, Tencent*

https://www.usenix.org/conference/usenixsecurity21/presentation/jing

This paper is included in the Proceedings of the 30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Too Good to Be Safe: Tricking Lane Detection in Autonomous Driving with Crafted Perturbations

Pengfei Jing[12], Qiyi Tang[2], Yuefeng Du[2], Lei Xue[1], Xiapu Luo[1*], Ting Wang[3], Sen Nie[2], Shi Wu[2]

[1]Department of Computing, The Hong Kong Polytechnic University
[2]Keen Security Lab, Tencent
[3]College of Information Sciences and Technology, Pennsylvania State University

Abstract

Autonomous driving is developing rapidly and has achieved promising performance by adopting machine learning algorithms to finish various tasks automatically. Lane detection is one of the major tasks because its result directly affects the steering decisions. Although recent studies have discovered some vulnerabilities in autonomous vehicles, to the best of our knowledge, none has investigated the security of lane detection module in real vehicles. In this paper, we conduct the *first* investigation on the lane detection module in a real vehicle, and reveal that the over-sensitivity of the target module can be exploited to launch attacks on the vehicle. More precisely, an over-sensitive lane detection module may regard small markings on the road surface, which are introduced by an adversary, as a valid lane and then drive the vehicle in the wrong direction. It is challenging to design such small road markings that should be perceived by the lane detection module but unnoticeable to the driver. Manual manipulation of the road markings to launch attacks on the lane detection module is very labor-intensive and error-prone. We propose a novel two-stage approach to automatically determine such road markings after tackling several technical challenges. Our approach first decides the optimal perturbations on the camera image and then maps them to road markings in physical world. We conduct extensive experiments on a Tesla Model S vehicle, and the experimental results show that the lane detection module can be deceived by very unobtrusive perturbations to create a lane, thus misleading the vehicle in auto-steer mode.

1 Introduction

Autonomous vehicles (AVs) are evolving rapidly in recent years, which rely on multiple sensors and machine learning algorithms to detect and reconstruct the surrounding environment for finishing various tasks automatically. Lane detection is one of the major tasks because its result directly affects the steering decisions.Therefore, misleading the lane detection module can lead to severe consequences. For example, if the lane detection module can be trapped into recognizing the small road markings added by an adversary as a valid lane, the vehicle will be misled by the fake lane and even be steered into the reverse traffic lane as shown in Fig.1.

Although a few recent studies demonstrated the feasibility of exploiting the camera-based perception in autonomous vehicles[37, 42, 49], they have the following limitations. First, some studies conducted white-box analysis that requires full knowledge of the target model[42, 49]. Unfortunately, it is very difficult to collect such information from real vehicles. Second, very few experiments were done on a real vehicle. To our best knowledge, only Nassi et al. recently demonstrated the feasibility of launching the phantom attack[37] on the camera-based perception of Tesla. However, the phantom attack only works in dark environments and can be easily noticed by the driver.

In this paper, we conduct the *first* investigation on the security of the lane detection module used in real vehicles. In particular, using Tesla Autopilot[13] as an example, we reveal that it is feasible to trick the lane detection module with crafted physical perturbations to mislead a Tesla vehicle in auto-steer mode and cause severe consequences, such as hitting the road curbs, driving into oncoming traffic, etc. Sur-

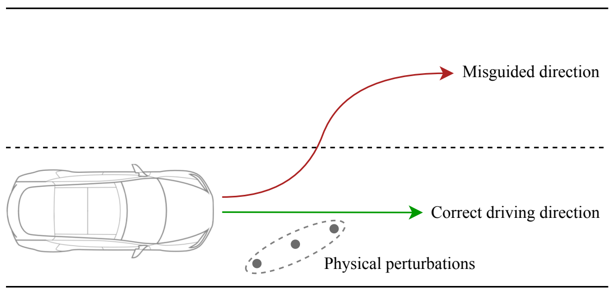
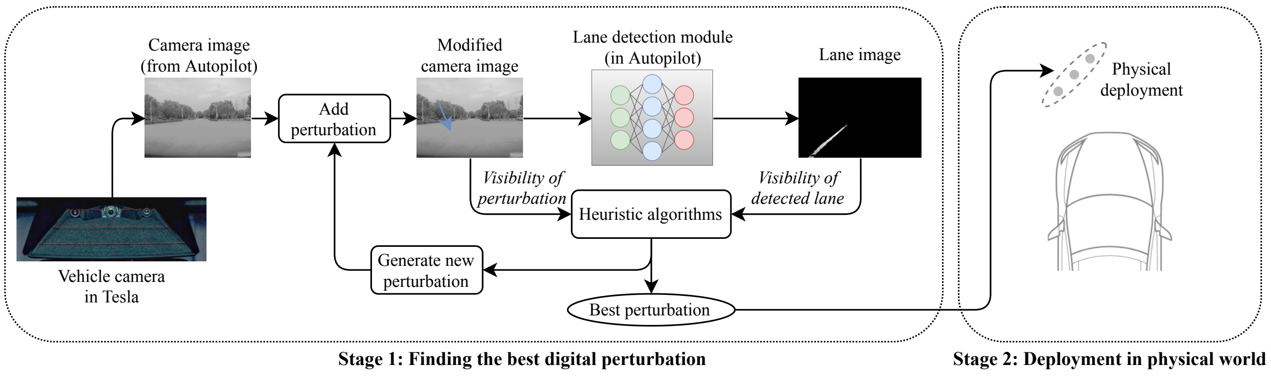


Figure 1: Tricking the autonomous vehicle to steer into the reverse traffic lane. If the physical perturbations added by an adversary are recognized as a lane, the vehicle is likely to follow the fake lane and swerve into the wrong direction.

[*]The corresponding author.



Camera image (from Autopilot)
Add perturbation
Modified camera image
Lane detection module (in Autopilot)
Lane image
Vehicle camera in Tesla
Visibility of perturbation
Heuristic algorithms
Visibility of detected lane
Generate new perturbation
Best perturbation
Physical deployment
Stage 1: Finding the best digital perturbation
Stage 2: Deployment in physical world

Figure 2: Overview of our two-stage approach. **In the first stage**, we add the perturbation, which is based on physical coordinate, to the camera image, and then feed the modified camera image to the lane detection module to generate the corresponding lane image. We formulate an optimization problem based on the visibility of perturbation and that of detected lane and adopt heuristic algorithms to find the best perturbation, which is unobtrusive to human but causes the lane detection module to output an obvious lane. **In the second stage**, we deploy the best perturbation in physical world according to the attributes of the best perturbation.

prisingly, we reveal that the vulnerability is not due to the incapability of its deep learning based lane detection algorithm. On the contrary, its algorithm is so sensitive that some unobtrusive stickers on the road surface will be regarded as a valid lane, and thus the vehicle will be misled.

It is challenging to inspect the lane detection module in a real vehicle. First, since the lane detection system is embedded in the vehicle without open source, it is difficult to access its binary and understand its computational logic. Specifically, it is challenging to extract and comprehend the deep learning algorithms executed in GPU. Second, it is non-trivial to determine the best perturbations for misguiding the vehicle, which should be perceived by the lane detection module but unnoticeable to the driver. Third, even if the perturbations imposed to the input of the lane detection module can mislead the vehicle, it is not easy to decide how to launch the attack in real world by adding unobtrusive road markings on the ground. An intuitive approach to find the best perturbation is to place stickers on the ground and then check whether the vehicle will be misguided manually. If not, the stickers should be changed or relocated. Unfortunately, such an approach is very labor-intensive and error-prone.

We propose a novel two-stage approach, as shown in Fig.2, to automatically determine the road markings for launching the attack on the lane detection module (in §4). More precisely, before the attack, we conduct reverse engineering on the firmware of Tesla Autopilot to determine the input (i.e., camera image) of its lane detection module and the corresponding output (i.e., lane image). This step is in §3. With such information, in the first stage, we conduct black-box attacks on the lane detection module by imposing the crafted perturbations to the camera image and capturing the corresponding lane image. We design metrics to quantify the visibility of the perturbation and the visibility of the corresponding detected lane, and formulate an optimization problem to find the best perturbation that can lead to a fake lane but is unnoticeable to human perception (in §4). We employ 5 heuristic algorithms to find the optimal solution, and find that Particle Swarm Optimization (PSO) is the best one (in §5).

In the second stage, we place markings on the ground according to the optimal perturbation and evaluate its effectiveness. It is worth noting that we use physical metrics in the parametric description of the digital perturbation (in §4.1), and therefore the optimal perturbation can be easily mapped to the markings in physical world. We conduct extensive experiments on a Tesla Model S vehicle, and the experimental results show that the lane detection module can be deceived by unobtrusive perturbations to create a fake lane, thus the vehicle in auto-steer mode can be misled.

In summary, we make the following major contributions:

- We conduct the first investigation on the security of the lane detection module in real vehicles and reveal that its sensitivity can be exploited by an adversary to generate fake lanes and consequently mislead the vehicle.
- We perform reverse engineering on the firmware of Tesla Autopilot to locate the input camera image and the output lane image. With this information, we propose a novel two-stage approach to generate the optimal perturbations against the lane detection module.
- We conduct extensive experiments on a Tesla vehicle (Tesla Model S)[15] to evaluate our approach. The experimental results show that the lane detection module in Tesla Autopilot is vulnerable to our attack and our approach can quickly generate effective perturbations.

2 Attack Overview

In this section, we first introduce the threat model and then give an overview of our two-stage attack approach.

2.1 Threat Model

We assume that an attacker has an autonomous vehicle, whose lane detection module is the same as that of other vehicles of the same model, but does not have any previous knowledge about the module (i.e., black-box setting). The attacker aims to add unobtrusive markings on the ground so that the lane detection module recognizes them as a valid lane and consequently the victim autonomous vehicle will be misled.

An intuitive attack approach is to place markings at the possible area of the road and check whether the vehicle will be misguided. If not, the attacker can change the position and the shape of the markings and repeat the try-and-error method until the attack succeeds. However, this approach is very labor-intensive and error-prone because of the unlimited number of possible ways to modify and place the markings. Our approach to be described in §2.2 tackles these limitations.

2.2 Our Approach

This section introduces the workflow of our approach, the challenges to be addressed, and the key ideas of our solutions.

2.2.1 Workflow

We first locate the input camera image to the lane detection module and the corresponding output lane image by conducting static and dynamic analysis on the firmware (in §3). Then, we carry out the two-stage attack as shown in Fig. 2.
Stage 1. Finding the best perturbation in digital world. We formulate an optimization problem based on the visibility of the perturbation and the visibility of the corresponding detected lane to find the best perturbation that can lead to a fake lane but is unnoticeable to human perception.
Stage 2. Deploying markings in physical world according to the best perturbation. According to the best perturbation in digital world, we deploy the markings in physical world and then evaluate the attacks on a real vehicle.

2.2.2 Challenges

Three challenges should be tackled to realize our approach.
C1. How to locate the input camera image and the corresponding output lane image in the vehicle? Our two-stage attack approach needs to access the input camera image and the output lane image. However, it is non-trivial to locate them since the lane detection module is in the closed-source firmware of Tesla Autopilot and the algorithms are executed in GPU using undocumented proprietary instruction sets.
C2. How to add perturbations to input camera image? An intuitive method is to add perturbations at the pixel level without considering the physical deployment. However, it may not be possible to implement such perturbations in physical world because it is not easy to accurately project the pixels to physical world, considering the distortion of the lens.
C3. How to find the best perturbations? The best perturbations should be as unobtrusive as possible so that drivers cannot notice them and meanwhile they can force the lane detection module to output a fake lane. It is challenging to find the best perturbations because the target model is in black-box setting so that the gradient-based optimization methods[41] cannot be applied.

2.2.3 Solutions

S1 (§3). We reverse engineer the firmware of Tesla Autopilot through static and dynamic analysis to locate the input camera image and output lane image. In particular, by exploiting the observation that Tesla Autopilot is powered by NVIDIA DRIVE technology [14] and its deep-learning computation follows the CUDA programming model [4] and is finished in GPU, we focus on locating and extracting the images in GPU memory. More precisely, after finding the binary responsible for lane detection, we conduct static analysis to find out when the images are available in GPU memory, and then instrument the binary and perform dynamic analysis to determine the memory addresses of the images. After that, we employ CUDA APIs to extract and modify the target images.
S2. We use a vector containing metrics from the physical world to represent the perturbations in digital world, and design the formula, which is based on the pinhole camera model and camera calibration (in Appendix B), to map the digital perturbation to the markings in physical world (in §4.1).
S3. We design two metrics to quantify the visibility of the perturbation and that of the corresponding detected lane, and formulate an optimization problem for the best perturbations (in §4.2.2). Then, we use five heuristic algorithms (in Appendix C.1) to find the best perturbation in digital world.

3 Accessing Data in Tesla Autopilot

This section details **S1** for locating the input camera image and the corresponding output lane image in the vehicle.

3.1 Overview

3.1.1 Firmware under examination

Our target vehicle is Tesla Model S 75, with the Autopilot hardware version of 2.5 and software version of 2018.6.1. It is worth noting that our methodology can be applied to other autonomous vehicles. The vehicle is running an AArch64 Linux operating system and uses NVIDIA GPU for deep learning computation. In the file system of Tesla Autopilot, there is a binary named *vision*. Through reverse engineering, we find that this binary is responsible for vision-related tasks including lane detection. It transmits the data of camera images into the GPU memory and finishes the vision-related computing tasks, in which lane detection is involved. The lane recognized by this binary will affect the steering decision when Autopilot is in auto-steer mode (demonstrated in §5). Since this *vision* binary can directly interact with the camera image and lane image in GPU memory, we carry out static and dynamic analysis on it to locate and access the target images.

3.1.2 CUDA

Tesla Autopilot uses NVIDIA GPU to execute its deep-learning algorithms, whose implementation follows the CUDA programming model [4]. We first introduce some necessary knowledge about CUDA programming because it is exploited by us to locate the target images.

CUDA programs usually involve two kinds of hardware: host (CPU) and device (GPU). If CPU needs to access data in GPU memory, it invokes a special kind of function named *kernel*s. A kernel is a function executed in the GPU as an array of threads in parallel [4]. These kernels will be launched and executed on GPU and manipulate data in GPU memory. In other words, kernels are the functions that run on GPU and launched by CPU. Since the lane detection is finished in GPU and the target images (camera image and lane image) are related to lane detection, the target images will be stored in GPU memory at certain time, and thus all we need to do is to determine "when" and "where".

CUDA provides memory management functions [3] to access and manipulate data in GPU memory.

• ***cudaMalloc**** [6]: Functions whose names begin with *cudaMalloc* are used to allocate memory in GPU (except *cudaMallocHost* that allocates memory on CPU). We denote such functions as *cudaMalloc**, each of which has two types of parameters. One is the pointer to the allocated memory and the other represents the data's size information. *cudaMalloc** will act as the instrumentation location for locating the lane image in GPU memory (in §3.3 and §3.4).

• ***cudaMemcpy**** [7]: Functions whose names begin with *cudaMemcpy* are used to copy data from one address to another. We denote these functions as *cudaMemcpy**, which take in four types of parameters including source address, destination address, size information, and the mode that represents the direction of the copying operation: host to GPU, GPU to host, host to host or GPU to GPU. *cudaMemcpy** will act as the instrumentation location for locating the camera image in GPU memory (in §3.3 and §3.4). We also employ these functions to dump the target images from GPU memory after we get their address and size information.

• ***cudaConfigurecall*** [5]: This function will be called before each kernel is invoked by the host to configure the launch on GPU. Hence, we can locate the kernels by locating the positions of *cudaConfigurecalls* in the binary for analysis. *cudaConfigurecall* will act as the instrumentation location for dumping lane image (in §3.3 and §3.4).

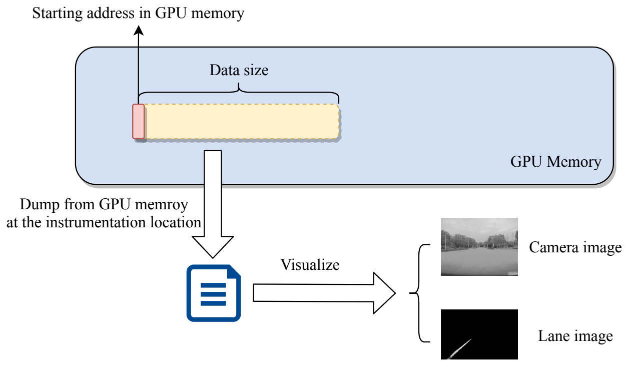


Figure 3: The process of dumping and visualize the target data

3.1.3 Factors required for dumping target images

We leverage the documented CUDA APIs to determine "where" and "when" to get the target images. In particular, we need to know the following three factors.

1. Instrumentation location. Since the lane detection is finished in GPU, the input camera images and the output lane images should be available in GPU memory after some specific functions are executed. We add instrumentation right after the invocation of such functions to get the target images.

2. Starting address of the images in GPU memory. It refers to the memory address where the image is stored in GPU memory. We need such addresses to locate the target images.

3. Data size. We need the size information to dump the images because they are stored in GPU memory as raw bytes. Moreover, to visualize the raw data (i.e., show the images), we need to know the image resolution (i.e., rows and columns) and the bit depth in each pixel. Fig. 3 shows how we dump and visualize the images from GPU memory. With the known starting address and data size, we instrument the binary to dump the image from the GPU memory in dynamic execution. The raw data in GPU memory are saved into a file and visualized according to the learnt resolution and bit depth.

We perform the following steps to determine these factors.

(1) Estimating data size (§3.2). We estimate the data size of camera images from the relevant document of the hardware camera [14]. For lane image, we conclude the data size from a file in Tesla Autopilot.

(2) Conducting static analysis to collect instrumentation location candidates (§3.3). We aim to dump the camera image right after *cudaMemcpy** is used to copy the image into GPU memory. Similarly, we dump the lane image right after the kernel for lane detection finishes its task. We conduct static analysis on the *vision* binary to find a list of candidates, including the invocations of *cudaMemcpy** (i.e., candidates for dumping the camera images) and the kernels (i.e., candidates for dumping the lane images).

(3) Performing dynamic analysis to determine instrumentation location and starting address in GPU memory (§3.4). Since the specific GPU memory address and the context can only be revealed during execution, we perform dynamic analysis to determine the correct instrumentation location and starting address. Specifically, for input camera image, we hook all *cudaMemcpy** calls and locate the one responsible for copying camera image by checking its parameters. Similarly, we first hook all *cudaMalloc** to find the starting address of the output lane image, and then determine the kernel by checking the visualized lane image after all possible kernels based on the data size and starting address.

3.2 Estimating Data Size

Size of camera image. We find the camera image's resolution (i.e., 1280×960 pixels) according to its hardware [14], however, the bit depth is still missing. Therefore, we compute 32 possible data sizes according to the possible bit depth, namely from 1-bit to 32-bit, to cover most of the possible bit depth used in digital images. For example, if an image is in 16-bit bit depth, the data size is 1280×960×16=19,660,800 bits (or 2,457,600 bytes). After this estimation, we get a list of the possible data size for camera image. The specific bit depth will be determined in dynamic analysis in §3.4 by hooking the *cudaMemcpy** calls.

Size of lane image. We find a file in the file system of Tesla Autopilot, which provides information about the architecture of the deep neural network used for object detection tasks (including lane detection), such as data size and pixel depth of the data matrix in each layer. This network has several outputs and the lane detection result is one of them, which is a 640×416 matrix with 32 bits float values. With this information, we can estimate the data size of the lane image output, which should be $640 \times 416 \times 32 = 8,519,680$ bits (or 1,064,960 bytes). The size of the lane image will be the key information for hooking the *cudaMalloc** in order to find the starting address of the lane image (in §3.3 and §3.4).

3.3 Conducting Static Analysis

Using IDA-Pro [9], we conduct static analysis on *vision* binary to determine the instrumentation locations and add instrumentation code. We detail the instrumentation locations for collecting camera images and lane images, respectively.

1.Instrumentation locations for collecting camera images. Since lane detection is finished in GPU, *cudaMemcpy* will be used to copy the input camera image into GPU memory before processing. Hence, we add instrumentation right after the invocation of *cudaMemcpy* for copying data into GPU memory. The instrumentation code will collect the parameters passed to the *cudaMemcpy*, including (1) source address, (2) destination address, (3) data size, and (4) mode of transfer, when being executed in dynamic analysis.

2. Instrumentation locations for collecting lane images. We are interested in two kinds of instrumentation locations:

• **Hooking *cudaMalloc** to determine the starting address**. Since *cudaMalloc** is responsible for allocating memory in GPU, the memory of the lane image will be allocated by *cudaMalloc**. In this case, we add instrumentation right after the invocation of each *cudaMalloc**, and collect the (1) memory address and (2) data size passed to *cudaMalloc**. By locating the *cudaMalloc** whose data size is equal to the estimated lane image size, we can determine the *cudaMalloc** that allocates the memory of the lane image, thus knowing the starting address of the lane image in GPU memory.

• **Hooking kernels to determine instrumentation location for dumping lane images**. Since kernel functions are responsible for the computation in GPU, we first enumerate all kernels according to the invocation of *cudaConfigureCall*. There are totally 75 calls of *cudaConfigureCall* by 22 different callers. Then, we add instrumentation right after the invocation of each kernel, because one of them will be responsible for lane detection and we can collect the lane image right after it finishes. The instrumentation code will dump the lane image in GPU memory according to the given starting address (found by hooking *cudaMalloc**) and data size.By checking whether the visualized image is the desired lane image, we identify the kernel function for lane detection.

3.4 Performing Dynamic Analysis

We execute the instrumented *vision* binary to (1) get the parameters passed to the hooked *cudaMemcpy** for obtaining the starting address and data size of the camera image and determining the correct instrumentation location; (2) get the parameters passed to the hooked *cudaMalloc** for obtaining the starting address of the lane image, and (3) dump the lane image after each kernel candidate to determine the instrumentation location of the lane image. The processes for camera images and lane images are described as follows.

1. Camera image. Through dynamic analysis, we collect the following information relevant to the input camera image: (1) data size, (2) the call of *cudaMemcpy** which copies the camera image to GPU memory, (3) the starting address of camera image in GPU memory. As specified in static analysis, we add instrumentation after each *cudaMemcpy** and collect the parameters passed to *cudaMemcpy** in dynamic execution. From the experiment results, among the 32 different estimated sizes, only a data size of 2,457,600 bytes is found, meaning the bit depth of the input image is 16-bit. In the experiment, we find that there are 3 types of camera images, which match the three front cameras on the vehicle. However, we do not know which one is used in lane detection. To identify the camera image involved in lane detection, we design a correlation analysis method, which is detailed in Appendix.A.

2. Lane image. For lane image, we have determined the data size in §3.2, and list the 75 candidate kernels. Through dynamic analysis, we obtain the following information: (1) the starting address of the lane image, and (2) the kernel that is responsible for lane detection among the candidates. We first finish task (1) by hooking the *cudaMalloc**, and accomplish task (2) based on the found GPU address in task (1). Next, we describe how we determine the starting address (task (1)) and how we determine the instrumentation location (task (2)) of the lane image, respectively.

• **Determining starting address of the lane image.** As specified in static analysis, we select a list of instrumentation locations for *cudaMalloc** to find the starting address of the lane image. Using IDA-Pro, we find 77 calls of *cudaMalloc**. We add instrumentation to check the parameters passed to *cudaMalloc** every time it is called, and aim to find the *cudaMalloc** call whose data size is our estimated size. After dynamic execution, we find the specific call of *cudaMalloc** whose size is our estimated size (1,064,960 bytes), and locate the address of the lane images by this specific *cudaMalloc**.

• **Determining instrumentation location of the lane image.** As mentioned in static analysis, for lane image, we find 75 possible places in *vision* binary for instrumentation. Based on the found GPU memory address of the lane image, we add instrumentation to dump the images after all these kernel candidates. By visualizing the dumped data, we learn that the kernel in the function named *t_cuda_lane_detection::compute* is responsible for lane detection.

Remark. We summarize the factors for camera image and lane image. For camera image, the instrumentation location is right after the invocation of *cudaMemcpy**; the starting address is the destination address passed as a parameter to the specific *cudaMemcpy**; the data size is 2,457,600 bytes, with 1280×960 resolution and 16-bit bit depth. For lane image, the instrumentation location is right after the execution of function *t_cuda_lane_detection::compute*; the starting address is the address passed to the specific *cudaMalloc** which allocates the memory for the lane image; the data size is 1,064,960 bytes, with 640×416 resolution and 32-bit bit depth.

4 Two-Stage Attack

This section describes how we add digital perturbations based on the physical metrics and how to find the best perturbations, which are the solutions to **C2** and **C3**, respectively.

4.1 Adding Digital Perturbations

This subsection describes the solution to **C2**. The goal is to obtain the digital perturbation which is defined by physical-world attributes for easy physical deployment.

4.1.1 Projecting Physical World Markings

As shown in Fig.4, we use (X,Y,Z) to denote the coordinate of each pixel on the markings in real world, which is the coordinate relative to the vehicle camera, and utilize (u,v) to denote the coordinate of the corresponding pixel on the

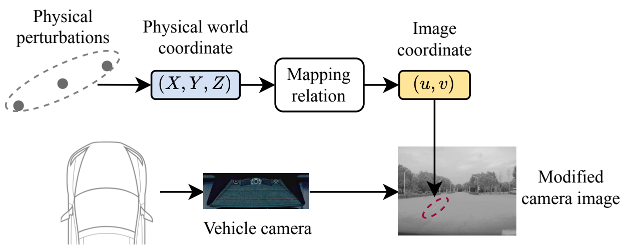


Figure 4: Mapping the coordinate of (X,Y,Z) on markings in physical world to the coordinate of (u,v) on perturbations in digital world.

perturbation added to the image. With the pinhole camera model, we project (X,Y,Z) to (u,v). We also undistort the image to eliminate errors due to lens distortion through camera calibration, thus making the projection more accurate. Appendix B details how we find the mapping relationship between these two coordinates. With this mapping relationship, we can map any physical world coordinate (X,Y,Z) to image coordinate (u,v). Hence, given a set of coordinates describing the position of the perturbations in physical world, we can project them to digital world and find their corresponding pixels in the camera image. Moreover, by modifying the grayscale value of the corresponding pixels, we can add the digital perturbations according to the physical perturbations. The reason is that in physical world the colors of the lane lines are mostly white and yellow, and they are brighter than the ground. Consequently, the lane line pixels in digital images are also brighter than the surrounding pixels on the ground. Therefore, raising the grayscale value (representing brightness) of the selected pixels in the captured digital image can result in the perturbations.

4.1.2 Parameterized Perturbations

For the ease of deployment, we use 8 parameters, which are listed in Table 1 and shown in Fig.5, to characterize the digital perturbations. len and wid determine the shape of the perturbations. D_1, D_2, and D_3 determine the position of the perturbations. ΔG is the increment of grayscale value of the pixels on the perturbation. n represents the number of perturbations (for example, $n = 2$ in Fig.5). Higher value of ΔG and more number of perturbations n make the added perturbation more obvious. θ is the rotation angle of the perturbations. The 8 parameters comprise a vector x:

$$x = (len, wid, D_1, D_2, D_3, \Delta G, \theta, n) \in X \qquad (1)$$

The measurement of len, wid, D_1, D_2, D_3 and θ is based on physical metrics. The unit of len, wid, D_1, D_2, D_3 is centimeter, and that of θ is degrees. ΔG is an 8-bit number ranging from 0 to 255 (we convert the 16-bit camera image into 8-bit for the ease of computing and visualization). Note that when $n = 1$, D_3 is invalid and has no influence on the added perturbation, because there is only one perturbation in view.

The range of x is denoted as X. len, wid, D_1, D_3, ΔG and n should be positive values. D_2 and θ can be positive or negative. Positive values of D_2 mean that the perturbation is on the left side of the vehicle, and negative means the right side. Positive value of θ represents that the perturbation is rotated towards the right direction of the vehicle, and negative means left.

Parameters	Explanation
len	Length of a single perturbation
wid	Width of a single perturbation
D_1	Longitudinal distance from the vehicle camera to the edge of the first perturbation
D_2	Lateral distance from the vehicle camera to the edge of the first perturbation
D_3	Distance between adjacent perturbations
ΔG	Increment of grayscale value of the perturbed pixels
θ	Rotation angle of the perturbation
n	Number of the perturbations

Table 1: Parameters determining the added perturbations

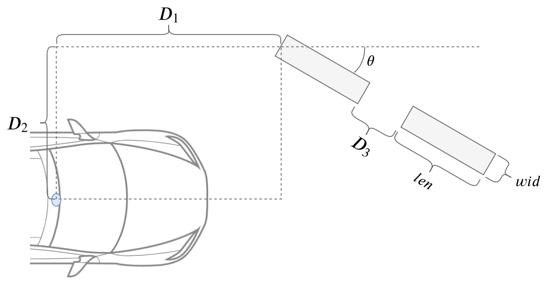


Figure 5: Illustration of the parameters of perturbations.

4.2 Finding the Best Perturbations

We design two metrics to quantify the quality of the perturbations in digital world, based on which we construct an optimization problem for finding the best perturbations.

4.2.1 Quality of Perturbations

Since a good perturbation should be unnoticeable to the driver but cause the lane detection module to generate a fake lane, we quantify its quality from the following two aspects:
Visibility of lane. The perturbations should lead to a strong and stable fake lane in the output lane image.
Visibility of perturbation. The perturbations should be as unobtrusive as possible.

We define two metrics: $V_{lane}(x) = \sum_{p \in lane_o(x)} G_p$ and $V_{perturb}(x) = \sum_{p \in perturb_i(x)} \Delta G, \Delta G \in x$ to quantify the visibility of lane and that of perturbation, respectively. We also define $S(x) = \frac{V_{lane}(x)}{V_{perturb}(x)}$ to be the overall score of perturbations. The explanations of the equations are listed in Table 2.

$V_{lane}(x)$ denotes the visibility of the fake lane in the output lane image. It is computed by summing up the grayscale values of each lane line pixel (each G_p represents the confidence of the current pixel). The higher value of $V_{lane}(x)$ represents higher visibility of the fake lane.

$V_{perturb}(x)$ is the visibility of the perturbation added to the input camera image. This score combines the number of added pixels and the increment of grayscale values of these pixels to represent visibility. The lower value of $V_{perturb}(x)$ means that the perturbations are more unobtrusive to human.

$S(x)$ is the overall score of the crafted perturbation. A high value of $S(x)$ means that the perturbation leads to a strong fake lane while being unobtrusive at the same time. If the perturbations fail to create a fake lane, $S(x)$ should be zero.

Parameters	Explanation
p	One single pixel in the image
$lane_o(x)$	Lane pixels in the output image
$perturb_i(x)$	Pixels on the added perturbations
G_p	Grayscale value of pixel p
$V_{lane}(x)$	Visibility of the fake lane created by x
$V_{perturb}(x)$	Visibility of the perturbations added by x
$S(x)$	Overall score of the parameter x

Table 2: Equation parameters explanations

4.2.2 Optimization problem

To achieve the best attack performance, we look for x^* that results in the highest overall score $S(x)$.

$$x^* = \max_{x \in X} S(x), \tag{2}$$

where x is a 8-dimension vector in range X, and the output score $S(x)$ is a real number. We use five heuristic algorithms to find x^*, namely beetle antennae search (BAS), particle swarm optimization (PSO), beetle swarm optimization (BSO), artificial bee colony (ABC) and simulated annealing (SA). To solve the optimization problem, these algorithms first initialize one or more random input vector(s), and iteratively improve the input vector(s) based on the output score.

These algorithms could be differentiated according to two aspects. First, is the algorithm greedy or not? "Greedy" means that the algorithm always updates the searching position to the direction where the target value is likely to be higher. BAS, PSO, and BSO are "Greedy", because they always encourage the searching position to move to coordinates where the value is higher, based on the hints found by the algorithms. ABC and SA are not "Greedy", because they essentially randomly update the position, and accept better solutions with higher possibilities. Second, do the searching individuals of an algorithm adopt a cooperative way to share information or not? "Cooperative" means that the searching individuals will share information with others, and update positions based on the group information. PSO, BSO, and ABC are "Cooperative", because each individual in the group shares his own information to help other individuals. By contrast, in BAS and SA, each individual works independently. The details of these algorithms are introduced in appendix C.1.

Note that n and θ are not put into the algorithms due to two reasons. First, since perturbation number n is a discrete variable while other parameters are all continuous variables, the optimization problem will become a mixed discrete-continuous optimization problem if n is considered and it is hard to find the optimal result. We will investigate it in future works. Second, since rotation angle θ is determined by the intention of the attack, a value of θ found by the algorithms may not meet the demand of the attacker. Therefore, we fix n and θ to constants, and discuss their impact in §5.

5 Evaluation

We evaluate our attack on the lane detection module by answering six research questions (RQs).

RQ1: How efficient are the heuristic algorithms to find the best perturbation?

Motivation: We want to identify the most efficient heuristic algorithm for finding the best perturbations.

Approach: We carry out the experiment with five heuristic algorithms, namely BAS, PSO, BSO, ABC, and SA, where PSO, BSO and ABC require multiple inputs working together, because these inputs will share information with each other, whereas BAS and SA work with a single input. For fair comparison, we also let BAS and SA have multiple inputs.

When looking for the best x, we record both the highest score $S(x)$ of the perturbations in history (top-1 score) and the average score of the top 10 perturbations (top-10 averaged score) to rule out contingency (i.e., an algorithm *accidentally* finds the best solution). If one algorithm achieves high score in both top-1 score and top-10 averaged score, its efficiency is no coincidence and is reproducible.

Since the effect of parameter n and θ is evaluated in *RQ2*, in *RQ*1 we let $n = 1$ and $\theta = 0$. Moreover, we focus to generate perturbation only on the left-hand side in RQ1 and discuss the right-hand side in RQ2. We implement the five algorithms with Python, and evaluate their performance with different parameters. The parameter setting of these algorithms is shown in appendix C.2.

Results: Fig.6(a)-(f) show the experimental results. The X-axis is the number of search rounds, and the Y-axis is the best $S(x)$ (left figure) or the top-10 averaged $S(x)$ (right figure) of the current search round. For an efficient algorithm, it should (1) converge quickly, and (2) achieve high score in both top-1 $S(x)$ and top-10 averaged $S(x)$. Fig.6(a)-(e) represent the performance of BAS, PSO, BSO, ABC and SA, respectively, and Fig.6(f) compares the best results of the five algorithms. As shown in Fig.6(f), the five algorithms have different performance. The experimental results show that "Greedy" and "Cooperative" algorithms (e.g. PSO, BSO) converge faster and find higher score in both top-1 $S(x)$ and top-10 $S(x)$, than other algorithms. Moreover, according to Fig.6.(f), PSO finds the highest $S(x)$ (both top-1 and top-10) among all five algorithms. Only ABC converges faster than PSO, however, the top-1 and top-10 averaged $S(x)$ found by ABC are much lower than that of PSO.

Fig.7 shows one of the best perturbations. Given the original input camera image, the lane detection module does not output a lane. After an unobtrusive perturbation (pointed out by the arrow in the image) is added, a clear lane is detected and shown in the output lane image, although the perturbation is nearly invisible to human perception and is unlikely to be treated as a valid lane. The parameters of this perturbation is: $wid = 1cm$, $len = 92cm$, $D_1 = 1365cm$, $D_2 = 233cm$, $\Delta G = 12$ (n, θ and D_3 have no influence in the setting here).

Answer: All heuristic algorithms can find best perturbations. PSO is the most efficient one and thus we use it in other experiments.

RQ2: How do the perturbation number n and the rotation angle θ affect the best perturbation?

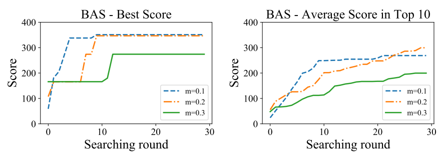
Motivation: As mentioned in §4.2.2, we do not put perturbation number n and rotation angle θ into the heuristic algorithms. In this RQ, we study how n and θ influence $S(x)$.

Approach: We adopt the same image used in RQ1 as input to generate the perturbations. The perturbation number is set from 1 to 5, and the absolute value of θ is 0 to 30 degrees with the interval of 5 degrees. In this case, we have 5 settings of n (from 1 to 5), and 14 settings of θ (from 0 to 30 degrees on both sides of the image). We consider all the possible settings for n and θ, thus getting totally $5 \times 14 = 70$ different settings of n and θ. Then, we search for the best perturbations on each setting, and record their $S(x)$s.

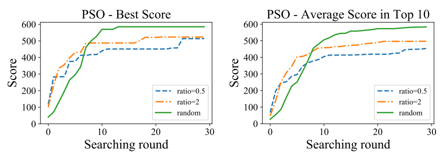
Results: Fig.8 shows the scores of the best perturbations found in different number n and rotation angle θ. The X-axis represents θ and Y-axis represents n. The intersection of two coordinates represents the best score $S(x)$ under the corresponding settings. The first row of the figure represents the average $S(x)$ under the specific θ, and the last column represents the average $S(x)$ under the specific n. The average $S(x)$ under each setting represents the overall effectiveness for this setting. For example, the third element on the first row represents the average $S(x)$ when $\theta = 10°$ and n is from 1 to 5, and this $S(x)$ represents the overall effectiveness of the perturbations when $\theta = 10°$.

By observing the average $S(x)$ in each θ (first row of Fig.8), we find that the average $S(x)$ decreases with θ, for both left and right lane. Specifically, when $\theta = 25°$ and $30°$, the average $S(x)$ is obviously lower than that in other settings of θ. Similarly, by observing the average $S(x)$ in each n (last column of Fig.8), for left lane, we find that the perturbations with $n \leq 3$ have the higher average $S(x)$ than $n = 4$ and $n = 5$, while for right lane, the average $S(x)$ is similar among all settings of n.

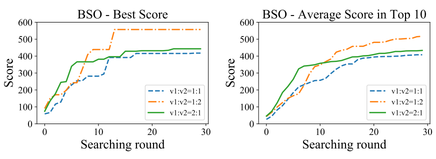
Answer: Perturbation number n does not have significant ef-



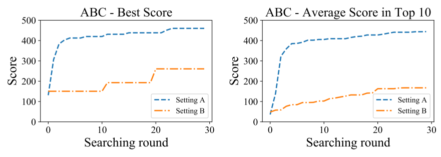
(a) Performance of BAS



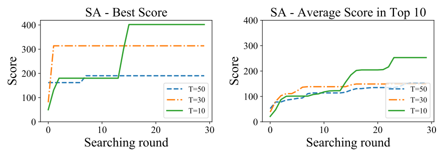
(b) Performance of PSO



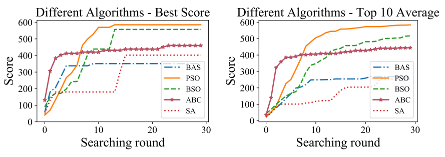
(c) Performance of BSO



(d) Performance of ABC



(e) Performance of SA



(f) Comparison between different algorithms

Figure 6: Results of the different algorithms. Overall, PSO has the best performance, and is the most suitable heuristic algorithm in our research.

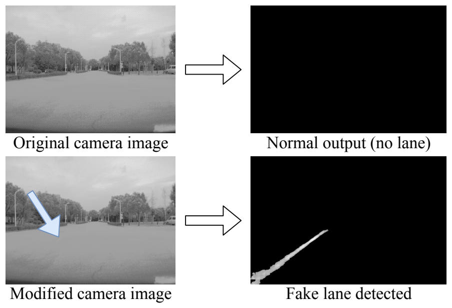
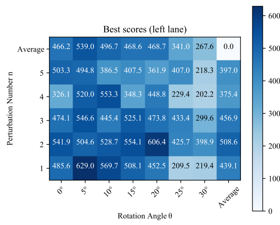
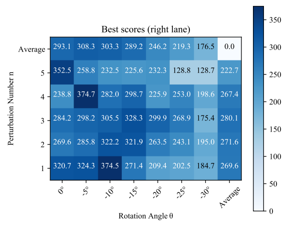


Figure 7: Effect of a best perturbation. The added perturbation is only 1cm wide in physical world, but it causes the lane detection module to generate a fake lane.



(a) Left lane



(b) Right lane

Figure 8: Best scores ($S(x)$) in different setting of n and θ in *RQ*2. The perturbations works well in different perturbation number n, and the score reduces with perturbation angle θ increasing.

fect on $S(x)$. Rotation angle θ reduces $S(x)$ when it increases.

RQ3: How is the performance of our approach given different input camera images?

Motivation: The experiments for answering RQ1 and RQ2 are based on the same input image shown in Fig.7. To answer RQ3, we generate perturbations on different input images to evaluate the effectiveness of our approach.

Approach: Besides the input image shown in Fig.7, we use four other images taken by the vehicle camera in different environments to carry out the experiment. They are shown in Fig.9 and their environmental features are listed in Table 3. NUM.1 is the input image used in *RQ1* and *RQ2*. NUM.2 and NUM.3 are in the same outdoor environment but under different light conditions. NUM.4 is taken in an underground garage, where the ground is clean and the light is dim. NUM.5 is a corner where the ground is dirty. The corresponding output lane images of these original input images do not have a lane on the expected side before we add any perturbation to them. Similar to the settings in **RQ1**, we let $n = 1$ and $\theta = 0$ and use $S(x)$ to evaluate the effectiveness of our attack.

Num	Environmental Features
1	Clean and bright ground, without other disturbing objects in view
2	Clean and bright ground, with disturbing objects in view
3	Clean and dark ground, with disturbing objects in view
4	Clean and dark ground, without other disturbing objects in view
5	Dirty and bright ground, with disturbing objects in view

Table 3: Environmental features of different input images

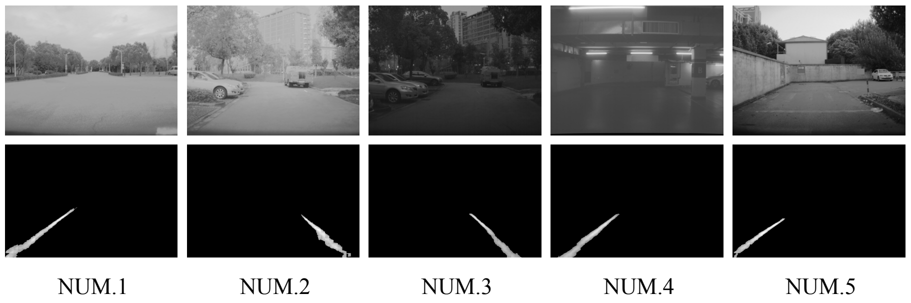
Results: The input images with the best perturbations and the corresponding lane images are shown in the upper row and the lower row of Fig.9.(a), respectively. The $S(x)$ of these examples are shown in Fig.9.(b). NUM.1 and NUM.4 lead to higher score than the others, because the grounds in both images are clean and a small perturbation can easily result in a fake lane in the output lane image. Although the scores of NUM.2/3/5 are relatively low, the perturbations are unnoticeable to human eyes and the fake lane is valid and strong.

Answer: Given different input images, our approach can successfully generate high-score perturbations that can mislead the lane detection module without being noticed by the driver.

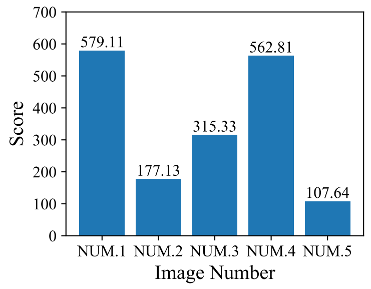
RQ4: What are the common characteristics of the best perturbations?

Motivation: We want to summarize the common characteristics of the best perturbations obtained in different scenarios and discuss their implication.

Method: We analyze the parameters x of the best perturbations obtained in the five different scenarios for answering *RQ*3 and summarize the common characteristics. x is a 8-



(a) Perturbations in different input images and the corresponding outputs



(b) Scores of the perturbations

Figure 9: *RQ*3 : The output lane and corresponding scores based on different input images. In all five different settings, we manage to find the unobtrusive perturbations which fool the lane detection module.

Num \ *x*	*wid*	*len*	D_1	D_2	ΔG
NUM.1	1cm	117cm	15.30m	2.23m	12
NUM.2	5cm	59cm	13.37m	2.27m	28
NUM.3	3cm	72cm	12.53m	1.51m	12
NUM.4	1cm	133cm	11.68m	1.79m	7
NUM.5	1cm	83cm	10.14m	2.38m	25
Average	2cm	93cm	12.60m	2.04m	17

Table 4: Parameter values of the best perturbations generated for five different input camera images.

dimension vector but we focus on five dimensions in x, including *wid* and *len* that denote the shape of the perturbation, D_1 and D_2 that indicate the relative position, and ΔG represents the increment of grayscale value of the perturbations. We do not study n and θ because they are fixed in the experiments. Moreover, D_3 is meaningless when $n = 1$.

Results: Table 4 lists the values of these five dimensions of the best perturbations to different images. We summarize the characteristics from the following three aspects.

• **Shape** In all scenarios, *wid* is much smaller than *len*, meaning that the 'narrow but long' perturbations are more effective than the 'wide but short' perturbations.

• **Position** For the position of the perturbation, D_1 ranges from 10.14m to 15.30m, and D_2 ranges from 1.51m to 2.23m.

• **Increment of grayscale value:** The value of ΔG varies in different input images. For clean ground (NUM.1 and NUM.4) or dark grounds (NUM.3 and NUM.4), a small increment can make the lane in the output image very obvious, whereas 'dirty' grounds (NUM.2 and NUM.5) require larger value of ΔG to generate a fake lane.

Answer: 'Narrow but long' perturbations are more likely to create a fake lane. The required increment of grayscale value (ΔG) depends on the brightness and cleanliness of the ground.

RQ5: How effective is the attack in physical world?

Motivation: As RQ1-4 study the attacks in digital world, for RQ5, we evaluate the attacks in physical world by deploying markings on road surface according to the best perturbations.

Approach: We first let the vehicle generate the input camera

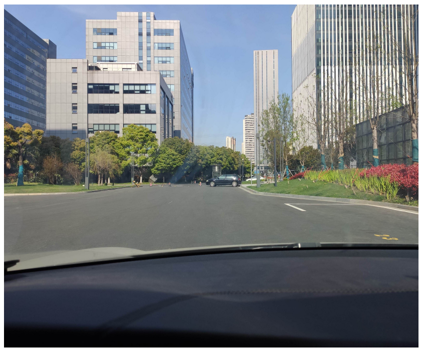


Figure 10: The road with the crafted markings from the driver's view. The sticker on the left side of the road is very unobtrusive and can hardly be noticed by human.

image in an area for conducting this experiment, and then user our approach to find the best perturbations. After that, according to the information of the best perturbation, we deploy the markings (i.e., stickers) on road surface and evaluate the visibility of the fake lane in the lane image. We adopt the following settings for this experiment.

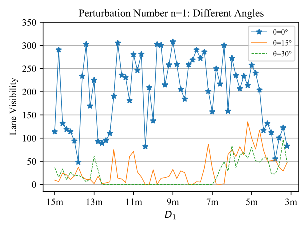
• **Perturbation number** n**.** Since the answer to RQ2 shows that n has little effect on $S(x)$ of the perturbations, we choose $n = 1$ and $n = 2$ for the ease of deployment.

• **Rotation angle** θ**.** Since the answer to RQ2 shows that θ will reduce the value of $S(x)$, to evaluate whether the visibility of the lane will also be affected in physical world, we set different values to θ (0, 15° and 30°) in the experiment.

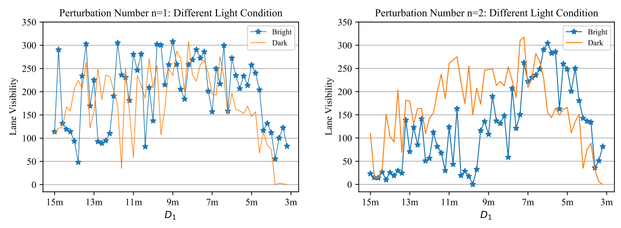
• **Light condition.** We conduct the experiment in both light and dark environments to evaluate the effect.

• **Longitudinal Distance** D_1**.** After deploying the stickers, we drive the vehicle from far to close to them, and record the visibility of the lane image ($V_{lane}(x)$) to evaluate the effectiveness of the attack with different D_1. Specifically, we drive from $D_1 = 15m$ to $D_1 = 3m$, and record 60 frames of the lane images during the process.

• ΔG**.** It is difficult to implement ΔG precisely in physical world because it will be affected by some uncontrollable fac-



(a) Lane visibility of each frame in different n and θ



(b) Lane visibility of each frame in different n and light conditions

Figure 11: The visibility of lane changes with D_1. Straight perturbations ($\theta = 0$) have higher lane visibility. Perturbation number n and light condition have little effect on the lane visibility. Interested readers are referred to our demo video[8].

tors, such as the environment's light condition of and the texture of the physical perturbations. In this experiment, we use white stickers, which offers high value of ΔG, to construct the perturbations in physical world.

We use the algorithm (PSO) to find the best digital perturbation for the scenarios of $n = 1$ and $n = 2$, respectively. When $n = 1$, its length len is 1.5m, and its width wid is 1cm. when $n = 2$, the length of each perturbation is 0.4m, the width is 1cm, and the adjacent distance is (D_3) 0.7m.

Fig.10 shows the driver's view of the road with the crafted markings (single perturbation). The stickers are placed on the left side of the vehicle, and can hardly be noticed by human.

Results: The lane visibility in this experiment is represented in Fig.11. The X-axis is the longitudinal distance (D_1) of each frame, and the Y-axis is the lane visibility $V_{lane}(x)$. Larger value of $V_{lane}(x)$ means that the attack is more effective. We also have the following observations.

• **Perturbation number** n**.** Compared with the setting of $n = 2$, the lane visibility is higher in the setting of $n = 1$ when $D_1 \geq 9m$. Therefore, the fake lane can be detected with different perturbation numbers. Even a single perturbation can work in physical world.

• **Rotation angle** θ**.** Fig.11(a) shows the influence of θ, when $n = 1$ and $n = 2$, respectively. In both scenarios, the lane visibility with $\theta = 15°$ and $\theta = 30°$ is obviously lower than the lane visibility with $\theta = 0$. Therefore, straight perturbations ($\theta = 0$) are more likely to be detected.

• **Light condition.** Fig.11(b) shows the influence of light condition, when $n = 1$ and $n = 2$, respectively. When $n = 1$, the lane visibility under bright and dark condition is similar in all values of D_1. When $n = 2$, the lane visibility under dark condition is higher than that in the bright condition, when $D_1 \geq 7m$. Hence, the perturbations work in both bright and dark environments. Darker environments even makes the lane visibility higher (see $n = 2$ in Fig.11.(b)).

• **Longitudinal Distance** D_1**.** When $n = 1$, the lane visibility is higher when $5m \leq D_1 \leq 12m$. When $n = 2$, the lane visibility is higher when $5m \leq D_1 \leq 7m$. Therefore, the fake lane can be detected in a large range of D_1 (from 15m to 3m) if the perturbations are properly implemented (like $n = 1$, $\theta = 0$ in Fig.11.(a)). Closer distances ($D_1 \leq 9m$) makes the perturbation easier to be detected.

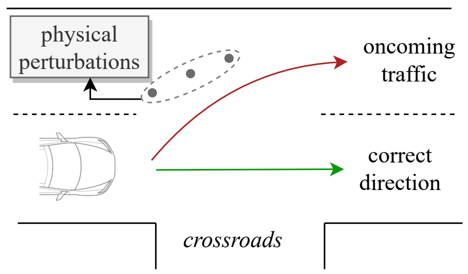


Figure 12: RQ6: Misguide the vehicle into the oncoming traffic in the crossroads scenario.

Answer: The crafted perturbations can be detected as fake lanes while staying imperceptible to humans. A demo video for physical attacks can be found at [8].

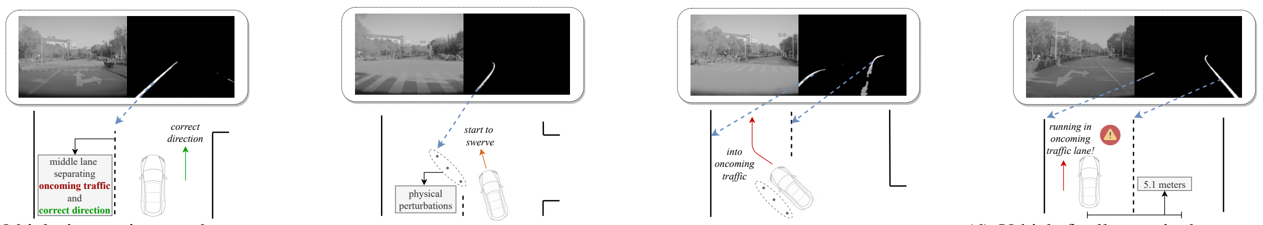
RQ6: Can we misguide the vehicle in physical world?

Motivation: The over-sensitivity of the target lane detection module has been demonstrated in both digital world and physical world through the answers to the previous RQs (i.e., RQ1, 2, 3, 4 for digital world and RQ5 for physical world). This RQ aims to investigate whether the control policy of the Autopilot will be affected by the crafted markings. Specifically, if Autopilot reacts to the fake lane, our attacks can impose a severe threat to the security and safety of the victim vehicle.

Approach: We find that in a commonly-seen crossroads scenario (i.e., the straight lanes disappear in front of the vehicle), the perturbations can mislead the vehicle to the oncoming traffic lane (illustrated in Fig.12). Specifically, in a crossroads scenario, we generate the perturbations that can trick the lane detection module to output an obvious lane. After physical deployment, we switch the vehicle to auto-steer mode and let it pass the crossroads where the markers have been added.

Results: We record the video showing the camera images and lane images when the vehicle is passing the crossroads. The result shows that the perturbations can lead to a fake lane which makes the vehicle swerve. Moreover, the vehicle was deviated by 5.1 meters (more than 2.5 times the width of the vehicle), and followed the fake lane to the oncoming traffic, demonstrating a severe and threat in real world.

Fig.13 illustrates the whole process. In each subfigure, the


correct direction
middle lane separating oncoming traffic and correct direction
start to swerve
physical perturbations
into oncoming traffic
running in oncoming traffic lane!
5.1 meters

(a) Vehicle is running on the correct direction. (b) Fake lane is detected and vehicle starts to swerve. (c) Vehicle follows the fake lane into oncoming traffic. (d) Vehicle finally runs in the oncoming traffic lane!

Figure 13: RQ6: The vehicle in auto-steer mode is misled into the oncoming traffic.

upper row includes the camera image and lane image, and the lower row shows what was happening when the frame was recorded. The interpretation of each frame is as below:
Fig.13 (a). Before approaching the crossroads, the vehicle runs on the right-hand side (correct direction) of the road, and the middle lane, which separates the two directions, is correctly recognized as the left-hand side lane (shown in the lane image).
Fig.13 (b). Right before the vehicle runs into the crossroads, the perturbations are detected and recognized as the fake lane, and therefore the vehicle starts to swerve along with the detected lane.
Fig.13 (c). The vehicle follows the fake lane and swerves to the left-hand side of the road (oncoming traffic lane). During this process, the middle lane (right lane in lane image) was recognized as the right-hand side lane. Based on this detection result, the vehicle runs into the oncoming traffic.
Fig.13 (d). Finally, the vehicle is deviated by 5.1 meters (more than 2.5 times the width of the vehicle), and is misled into the oncoming traffic lane, and further keeps running on this wrong direction.

Note that there is no human operation in the above process. The vehicle is in auto-steer mode, and its average speed is above 40km/h, which is already very dangerous in real world. Interested readers are referred to our demo video [8].
Answer: The experimental result shows that the fake lane resulted from the unobtrusive perturbations can successfully fool the vehicle in auto-steer mode to swerve, and even misguide the vehicle into oncoming traffic (might hit other cars in the oncoming traffic lane), thus demonstrating the potential severe threats in real world.

6 Defense

In this section, we propose two kinds of mechanisms to defend against this attack.
Enhancing the lane detection module. The lane detection module can be improved to distinguish crafted perturbations by two ways: *(1) Detecting abnormal lane lines by features.* Since the attackers want to make the perturbations unobtrusive, the size of the perturbations for generating the fake lane should be much smaller than the normal lanes. Moreover, as the attackers want to mislead the vehicle to cause safety and/or security consequences, the detected fake lane will be inconsistent with the real lanes (e.g., generating sharp turns [37]). As a result, the lane detection module can leverage these features to reject the abnormal lanes in advance. *(2) Including adversarial examples in training data.* As suggested by Goodfellow et al. [25], adding adversarial examples in the training data can make the model more robust to adversarial attacks. Hence, images with perturbations can be included in the training data to help the lane detection module distinguish between crafted perturbations and real lane lines.
Enhancing the control policy. To make the control policy more robust is another option for defense: *(1) Taking into consideration other visual elements.* The vehicle is vulnerable to our attacks if the steering control policy just relies on the lane detection result. Hence, it can be enhanced by involving other visual elements (i.e., coming traffic, pedestrian) to assist the steering control. *(2) Multi-Sensor fusion.* In Tesla Autopilot, the lane detection module relies on visual data. A possible defense method is to adopt multi-sensor fusion. That is, the control policy should also take into account the information from sensors like LiDAR, Radar, sonar and GPS. For example, the data from GPS and Radar can be used to detect whether the vehicle is deviated or running in the oncoming traffic lane. *(3) Advanced warning.* As the security of autonomous driving may not be fully guaranteed, the vehicle should warn the driver in advance when any abnormal lane line is detected (e.g., the size of the lane is too small or the angle of the lane is too sharp, etc.). Moreover, to ensure safety, the vehicle should demand the driver for manual control and quit auto-steer mode.

7 Responsible Disclosure

We have informed Tesla of our findings by providing the details of our attack method and the demo videos. Tesla has confirmed that this attack can change the target car's behavior when the vehicle is in auto-steer mode, and meanwhile emphasized that the driver should still pay full attention while auto-steer mode is on. Tesla did not mention any plan on fixing this vulnerability in the response. We will check whether Autopilot will adopt any countermeasure in future work.

8 Limitations and Discussion

Limitations. Since our attacks exploit the over-sensitivity of the lane detection module to mislead the vehicle, the crafted perturbations need to be detected by the lane detection module and thus they cannot be completely invisible. Hence, the driver may notice them if she knows the attack and pays full attention to the ground. However, our attack still poses severe threats to current autonomous driving because of the following reasons. *First, drivers are likely to pay less attention in auto-steer mode.* Without being informed of our attack, the driver may simply ignore the perturbations, not to mention that the vehicle is in auto-steer mode. According to the statistics given by the surveys [12][17], distracted driving is the top-1 reason for car crashing. In auto-steer mode, drivers are likely to pay less attention so that they may not notice the small perturbations which are quite different from the real lane. *Second, there is not enough time for reaction.* Even if the driver notices the perturbations when the vehicle is going to the place where the crafted perturbations have been deployed, there may not be enough time for the driver to react. For instance, in the experiment for answering RQ6, the speed of the vehicle is around 40km/h, and thus it takes only 0.918 seconds to deviate the vehicle for 5.1m. M. Green [26] shows that the driver's reaction times for unexpected events are between 1.20s and 1.35s ($> 0.918s$ in our experiment). Therefore, there is not sufficient time for the driver to take action against our attack, and severe consequences might have already been caused.

Future works. We will extend our work from two aspects. First, we will assess the vulnerability of the lane detection modules in other autonomous driving systems, including Apollo [1] and Openpilot [11]. Second, we will explore the feasibility of launching attacks on the lane detection modules by adding perturbations on real lanes, such as using dark markings to cover part of real lanes or adding markings to change the shape of real lanes.

9 Related Work

Adversarial Attacks. Deep neural network (DNN) have achieved great performance in many areas. However, researchers found that these models show their vulnerability when faced with crafted inputs [35, 39, 44, 47]. These malicious inputs can guide the models to make wrong decisions while staying imperceptible to humans. Besides, the models may also be subject to other attacks[24, 28, 40], such as poisoning, backdoor, etc.

Lane Detection. Lane detection is an important task in environmental perception of autonomous vehicles, because it provides the position information and further keeps the vehicles within the lane lines. Traditional lane detection methods rely on the selected features to identify lane markings [21, 30, 45], and thus their performance highly depends on the features. Recently, DNN has been widely used in lane detection for its great power in feature extraction [27, 34, 36, 38].

Autonomous Driving Security. To date, many autonomous driving systems adopt DNN to process data, especially vision data [10, 13, 19, 20]. These vision-based models take the camera data as the input, and the corresponding steering angle as the output. Although these models perform well in most cases, they can still make wrong decisions in some cases, which can lead to severe consequences [16, 18]. Eykholt et al. used a physical adversarial example to make DNN model misclassify the stop sign [23]. Although the method in [23] and ours are both "two-stage", they have different meanings. The "two-stage" method in [23] is for evaluating the attack (after the attack is already deployed), whereas our "two-stage" is for implementing the attack. Zhou et al. proposed a method called DeepBillboard to generate physical adversarial examples to make the DNN-based autonomous driving system steer to the wrong direction [49]. Shen et al. [43] misguided the vehicle to the wrong direction by GPS spoofing. Ben Nassi et al. [37] utilize projection to make the vehicle believe that the projection is the real object (phantom attack), and they also tested the lane detection module of Tesla Autopilot. However, the phantom attack only works at night, and it can be easily noticed. In contrast, our attack can be launched during the day and is more stealthy.

10 Conclusion

We conduct the first investigation on the lane detection module in a real vehicle, and reveal that its sensitivity can be exploited to launch attacks on the vehicle. Specifically, we propose a novel two-stage approach to automatically determine the best perturbations in digital world and then project them back to the markings in physical world after addressing several technical challenges. We conduct extensive experiments on a Tesla Model S vehicle. The experimental results show that the lane detection module can be deceived by crafted perturbations and mislead the vehicle in auto-steer mode.

11 Acknowledgment

We thank our shepherd Yongdae Kim and the anonymous reviewers for their constructive comments. We thank Prof. Yubin Xia for helping us in conducting experiments. This work is partly supported by Hong Kong RGC Projects (No. 152239/18E), Hong Kong ITF Project (No. ITS/197/17FP), HKPolyU Research Grant (ZVQ8), Start-up Fund (ZVU7), NSFC for Young Scientists of China (No. 62002306), and CCF-Tencent Open Research Fund. Ting Wang was partly supported by the National Science Foundation under Grant No. 1953893, 1953813, and 1951729.

References

[1] Apollo autonomous driving. https://github.com/ApolloAuto/apollo.

[2] Camera Calibration and 3D Reconstruction - OpenCV. https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html.

[3] CUDA memory management APIs. https://bit.ly/3dlFozE.

[4] CUDA Toolkit Documentation. https://docs.nvidia.com/cuda/cuda-c-programming-guide/.

[5] cudaConfigureCall. https://bit.ly/2ZucaX1.

[6] cudaMalloc. https://bit.ly/2M2Qnmb.

[7] cudaMemcpy. https://bit.ly/3aulsIV.

[8] Demonstration video: misguiding the vehicle in real world. https://youtu.be/a__Se2MrjVs.

[9] IDA Pro. https://www.hex-rays.com/products/ida/.

[10] Nvidia, Drive AP2X. https://www.nvidia.com/en-us/self-driving-cars/drive-platform.

[11] Openpilot autonomous driving. https://github.com/commaai/openpilot.

[12] Past statistics on texting & cell phone use while driving. https://www.edgarsnyder.com/car-accident/cause-of-accident/cell-phone/past-cell-phone-statistics.html.

[13] Tesla Autopilot System. https://www.tesla.com/autopilot.

[14] Tesla Hardware Information. https://teslatap.com/undocumented/.

[15] Tesla Model S. https://www.tesla.com/models.

[16] Tesla Model S crash. https://www.wired.com/story/tesla-autopilot-why-crash-radar.

[17] Texting and driving accident statistics. https://www.edgarsnyder.com/car-accident/cause-of-accident/cell-phone/cell-phone-statistics.html.

[18] Uber's Self-Driving Cars Were Struggling Before Arizona Crash. https://www.nytimes.com/2018/03/23/technology/uber-self-driving-cars-arizona.html.

[19] Udacity Self-driving Car. https://github.com/udacity/self-driving-car.

[20] 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:1604.07316*, 2016.

[21] A. Borkar, M. Hayes, and M. T. Smith. A novel lane detection system with efficient ground truth generation. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):365–374, 2011.

[22] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[23] 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 visual classification. In *Proc. CVPR*, 2018.

[24] M. Goldblum, D. Tsipras, C. Xie, X. Chen, A. Schwarzschild, D. Song, A. Madry, B. Li, and T. Goldstein. Dataset security for machine learning: Data poisoning,backdoor attacks, and defenses. *arXiv:2012.10544*, 2020.

[25] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv:1412.6572*, 2014.

[26] M. Green. " how long does it take to stop?" methodological analysis of driver perception-brake times. *Transportation human factors*, 2(3), 2000.

[27] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. *arXiv:1504.01716*, 2015.

[28] Y. Ji, X. Zhang, S. Ji, X. Luo, and T. Wang. Model-reuse attacks on deep learning systems. In *Proc. CCS*, 2018.

[29] X. Jiang and S. Li. Bas: beetle antennae search algorithm for optimization problems. *arXiv:1710.10724*, 2017.

[30] H. Jung, J. Min, and J. Kim. An efficient lane detection algorithm for lane departure detection. In *Proc. IEEE Intelligent Vehicles Symposium*, 2013.

[31] D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39(3):459–471, 2007.

[32] J. Kennedy. Particle swarm optimization. *Encyclopedia of machine learning*, pages 760–766, 2010.

[33] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[34] S. Lee, J. Kim, J. Shin Yoon, S. Shin, O. Bailo, N. Kim, T.-H. Lee, H. Seok Hong, S.-H. Han, and I. So Kweon. Vpgnet: Vanishing point guided network for lane and road marking detection and recognition. In *Proc. ICCV*, 2017.

[35] H. Li, S. Zhou, W. Yuan, X. Luo, C. Gao, and S. Chen. Robust android malware detection against adversarial example attacks. In *Proc. WWW*, 2021.

[36] J. Li, X. Mei, D. Prokhorov, and D. Tao. Deep neural network for structural prediction and lane detection in traffic scene. *IEEE transactions on neural networks and learning systems*, 28(3):690–703, 2016.

[37] B. Nassi, D. Nassi, R. Ben-Netanel, Y. Mirsky, O. Drokin, and Y. Elovici. Phantom of the adas: Phantom attacks on driver-assistance systems. In *Proc. CCS*, 2020.

[38] D. Neven, B. De Brabandere, S. Georgoulis, M. Proesmans, and L. Van Gool. Towards end-to-end lane detection: an instance segmentation approach. In *Proc. IEEE Intelligent Vehicles Symposium*, 2018.

[39] R. Pang, H. Shen, X. Zhang, S. Ji, Y. Vorobeychik, X. Luo, A. X. Liu, and T. Wang. A tale of evil twins: Adversarial inputs versus poisoned models. In *Proc. CCS*, 2020.

[40] R. Pang, X. Zhang, S. Ji, X. Luo, and T. Wang. Advmind: Inferring adversary intent of black-box attacks. In *Proc. KDD*, 2020.

[41] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv:1609.04747*, 2016.

[42] T. Sato, J. Shen, N. Wang, Y. J. Jia, X. Lin, and Q. A. Chen. Security of deep learning based lane keeping system under physical-world adversarial attack. *arXiv:2003.01782*, 2020.

[43] J. Shen, J. Y. Won, Z. Chen, and Q. A. Chen. Drift with devil: Security of multi-sensor fusion based localization in high-level autonomous driving under GPS spoofing. In *Proc. USENIX Security Symposium*, 2020.

[44] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv:1312.6199*, 2013.

[45] H. Tan, Y. Zhou, Y. Zhu, D. Yao, and K. Li. A novel curve lane detection based on improved river flow and ransa. In *Proc. IEEE Conference on Intelligent Transportation Systems*, 2014.

[46] T. Wang, L. Yang, and Q. Liu. Beetle swarm optimization algorithm: Theory and application. *arXiv:1808.00206*, 2018.

[47] X. Zhang, N. Wang, H. Shen, S. Ji, X. Luo, and T. Wang. Interpretable deep learning under fire. In *Proc. USENIX Security Symposium*, 2020.

[48] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22, 2000.

[49] H. Zhou, W. Li, Y. Zhu, Y. Zhang, B. Yu, L. Zhang, and C. Liu. Deepbillboard: Systematic physical-world testing of autonomous driving systems. *arXiv:1812.10812*, 2018.

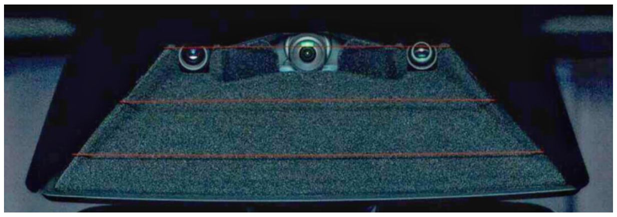


Figure 14: Three front cameras on the vehicle

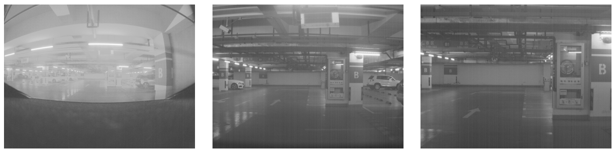


Figure 15: Three kinds of camera images dumped from GPU memory. Left one is the *fisheye* image; middle one is the *main* image and right one is the *narrow* image. We found that the lane detection is based on the *main* image.

A Correlation Analysis on Camera Images

As mentioned in §3.4, we find that there are 3 types of camera images, which may come from the three front cameras on the vehicle (shown in Fig.14). We dump the data based on the data size and GPU address, and visualize them based on the known resolution and color depth. The visualized results are shown in Fig.15. We can see that there are three kinds of camera images in different focal lengths, and the three kinds of images exactly match the three front cameras on the vehicle. However, we do not know which image(s) is used for lane detection.

To find the relationship, we physically cover the cameras and see how this change affects the lane image. In Fig.15, from left to right, we name the cameras with different focal length as *fisheye*, *main* and *narrow*. We stop the vehicle in front of an obvious lane line to make sure that the lane detection module will output a solid lane. Then, we set the following three scenarios: (1) One camera is on: lane pixels are observed only when *main* camera is not covered; (2) Two cameras are on: the result is the same as (1), and the lane image is identical to the one observed in (1); (3) Three cameras are on: lane image is identical to the ones observed in (1). This result indicates that only the *main* camera image is used for lane detection.

B Camera Model and Coordinate Mapping

The goal of coordinate mapping is to deploy physical perturbations based on digital perturbations. Therefore, for every possible coordinate on the ground in physical world, we need to know the corresponding 2D coordinate in digital image. We build the coordinate mapping relationship based on the pinhole camera model. It is based on the physical structure

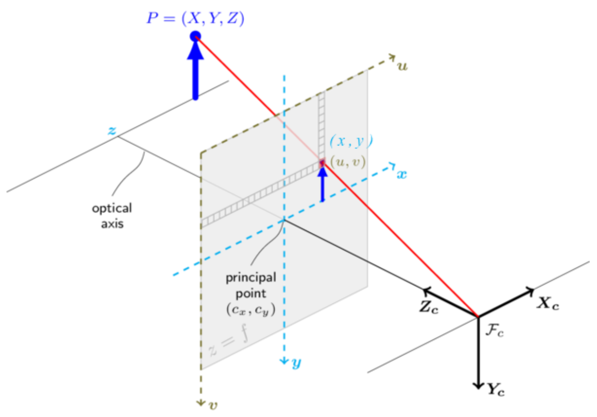


Figure 16: Pinhole camera model illustration (from OpenCV library [2])

of monocular cameras, which describes the mathematical relationship between 3D coordinate in physical world and 2D coordinate in digital world in the ideal case [22].

Fig.16 shows the pinhole camera model. Our purpose is to transform the physical world coordinate $P = (X,Y,Z)$ to (u,v) in digital world. F_c is the position of the camera, and from this position, three perpendicular axes are expanded. The camera projects the physical world coordinate $P = (X,Y,Z)$ to the focal lane $z = f$, where f is the intrinsic focal length of the camera. With the following mathematical relation, the model is able to transform 3D physical world coordinate (X,Y,Z) to 2D coordinate (x,y) in the focal plane:

$$\frac{X}{x} = \frac{Y}{y} = \frac{Z}{f} \tag{3}$$

Based on this equation, given the physical coordinate (X,Y,Z) and focal length f, we can project any 3D point in physical world to the 2D focal plane. However, since this coordinate (x,y) is still a continuous variable in physical unit, we need to transform it into discrete value (u,v). This is finished by the following equation:

$$\begin{cases} sx = u - c_x \\ sy = c_y - v \end{cases}, \tag{4}$$

where s is the scale factor between the physical coordinate and digital coordinate. (c_x, c_y) is the coordinate of the center point in the digital image, which can be accessed by checking the resolution of the image. By combining equation (3) and (4), we get the mapping relationship between the 3D and 2D coordinates as follows:

$$\begin{cases} u = sf\frac{X}{Z} + c_x \\ v = c_y - sf\frac{Y}{Z} \end{cases} \tag{5}$$

In practice, the scale factor s in the projection of two axes may be slightly different. Therefore, we use s_x and s_y to denote the scale factor in horizontal and vertical directions, respectively. Since f is bind to s_x and s_y in equation (5), we set $F_x = fs_x$ and $F_y = fs_y$ to represent them, respectively, and the mapping relationship can be written as:

$$\begin{cases} u = F_x\frac{X}{Z} + c_x \\ v = c_y - F_y\frac{Y}{Z} \end{cases} \tag{6}$$

In equation (6), (X,Y,Z) is the 3D coordinate of a point in physical world and (u,v) is the 2D coordinate of the point in digital world. c_x and c_y are learned from the resolution of the image. F_x and F_y are learned from the calibration of the camera. With these four variables (c_x, c_y, F_x and F_y), given any physical world coordinate (X,Y,Z), we can transform it into digital coordinate (u,v).

However, in most cases, the captured image is distorted due to the intrinsic flaws of the camera, and these distortions can affect the mapping accuracy [22] [48]. To make the mapping more accurate, based on the camera calibration theory [48], we eliminate the inaccuracy caused by lens distortion.

C Adopted Heuristic Algorithms

C.1 Introduction of the Algorithms

Beetle Antennae Search (BAS). Beetle Antennae Search [29] was inspired by the searching behavior of longhorn beetles. The position of the beetles represents the input parameter (x), and each of the beetles will search the area based on the information received from the antennae. BAS has three major steps:

(1) Randomly generating antennae direction. The direction of the antennae is generated by the following equation:

$$\vec{b} = \frac{random(k)}{\|random(k)\|}, \tag{7}$$

where $random(k)$ donates a random function to create a k-dimension vector. We fixe k to 8 as x has 8 dimensions.

(2) Updating antennae positions. BAS assumes that the two antennae are always at the opposite position. Based on the current position and generated direction, the position of the antennae could be computed as follows:

$$x_l = x^t + d^t\vec{b} \tag{8}$$
$$x_r = x^t - d^t\vec{b} \tag{9}$$
$$d^t = \eta d^{t-1} \tag{10}$$

d is the searching step (current antennae length) and it should decrease with t. η is the decreasing rate set to make d decrease with searching process ($0 < \eta < 1$).

(3) Updating parameters. The updating strategy is defined as follows:

$$x^t = x^{t-1} + \vec{b}D^t sign(S(x_l) - S(x_r)) \tag{11}$$
$$D^t = \eta D^t \tag{12}$$

sign denotes a sign function. D^t is the moving step of the current round, which should also decrease with the searching process. Generally, $D_t > d_t$.

Particle Swarm Optimization(PSO). PSO [32] is one of the most classic heuristic algorithms. It is performed in parallel and the particles can share information with each other.

The velocity used to update the position of the particles could be described as follows:

$$v_s^{t+1} = wv_s^t + c_1 r_1 (p_s^t - x_s^t) + c_2 r_2 (g_s^t - x_s^t), \tag{13}$$

where s denotes the dimension of v or x, and t denotes the current round of iteration. w, c_1 and c_2 are positive constants, and r_1 and r_2 are two random numbers in the range [0,1]. p_s^t is the best historical position of the current particle, and g_s^t is the best historical position of all particles.

Beetle Swarm Optimization (BSO). BSO [46] is the combination of BAS and PSO. BAS has better ability to find the optimized direction near a single particle (beetle), and PSO allows the particles to share information with each other. Therefore, the updating formula of BSO is as follows:

$$x_s^{t+1} = x_s^t + mv_s^{t+1} + (1-m)\xi_s^{t+1}, \tag{14}$$

where m is a constant in the range [0,1] that represents the ratio of the two kinds of velocity. v_s^{t+1} is the speed concluded in the t_{th} iteration and ξ_s^{t+1} is the BAS speed in s_{th} dimension.

Artificial Bee Colony (ABC). ABC [31] is another representative swarm intelligent optimizing algorithm, which is inspired by the behaviors of bees. It has three kinds of bees (particles): employed bees, onlooker bees and scouts. Positions of the bees represent the input parameter (x) and the bees aim to find the location which has the most nectar ($S(x)$ as the fitness function). At the initialization stage, the employed bees are sent to random locations to look for nectar, and will share the information of the nectar amount ($S(x)$) with the onlooker bees. Then, the onlooker bees choose a food source depending on the probability value associated with the corresponding nectar amount. The probability of an onlooker bee choosing the position of the employed bee x_i is calculated by the following expression:

$$p_i = \frac{S(x_i)}{\sum_{n=1}^{Size} S(x_n)}, \tag{15}$$

where *Size* is the number of employed bees. Generally, positions where $S(x)$ has greater value are more likely to be chosen. After this step, the onlooker bees randomly choose a position near the employed bee and return the new fitness function value. If this new position has more food (higher S(x) value) than the position of the employed bee, this onlooker bee will become the employed bee and the old position will be abandoned. If the onlooker bees fail to find a better position around the employed bee for several rounds, this position will be abandoned and this employed bee will become the scout to randomly look for a new food source.

Simulated Annealing (SA). SA [33] is another classic heuristic algorithm aiming to find the global optimum of a certain function. In SA, the current searching individual randomly looks for a solution close to the current position, and then compares the function value ($S(x)$ in our situation) in the two positions. We denote the current position as x^t and the tested position as x_p. If $S(x_p) > S(x_t)$, this position will be accepted ($x^{t+1} = x^t$). Otherwise, this position will be accepted with a given possibility:

$$P = e^{\frac{\Delta E}{T}}, \tag{16}$$

where $\Delta E = S(x_p) - S(x_t)$, and T decreases in the searching process. Decreasing T means that worse solutions are more likely to be accepted at first, and as searching continues, SA should converge to the global optimum and is less likely to accept these worse solutions.

C.2 Parameter Setting of Algorithms in RQ1

The number of the input is set as 30. We set the number of searching rounds as 30 because we found all five algorithms converge within 30 rounds in the experiment.

BAS: We denote the norm of the maximum vector in X as $||x_{max}||$, then set the antennae length and moving step based on this value. Specifically, we set the antennae length $d^1 = \frac{||x_{max}||}{30}$, decreasing parameter $\eta = 0.95$. The step length is assigned three different values. We set $D^1 = m||x_{max}||$ and let $m = 0.1$, $m = 0.2$ and $m = 0.3$. m affects the convergence speed.

PSO: We fix $w = 0.3$ and $c_1 = c_2 = 1$. Then, we vary the ratio between r_1 and r_2, because they represent the updating speed derived from the individual information and from the whole group information, respectively. We denote $ratio = \frac{r_1}{r_2}$, and implement three settings: (1) $ratio = 0.5$; (2) $ratio = 2$; (3) random value. In (1), r_1 is a random number in range [0,1] and r_2 values double. In (2), r_2 is random in the same range, and r_1 is double r_2. In (3), $r1$ and $r2$ are independent random values in the range [0,1].

BSO: As BSO is the combination of BAS and PSO, we give different proportions to the speed derived from BAS and PSO. Let v_1 be the speed from BAS and v_2 be the speed from PSO. Then, we adjust the ratio between v_1 and v_2. The ratio is set to three values: 1, 0.5 and 2. For BAS, $m = 0.1$. For PSO, r_1 and r_2 are independent random values (setting (3)).

ABC: We implement two settings: (1) onlooker bees choose searching positions based on equation 15; (2) onlooker bees only choose the best $S(x_i)$ as the position to investigate. Setting (1) is the classic setting of ABC algorithm. For setting (2), since only positions with the best scores are accepted, the algorithm will converge more quickly, but meanwhile is more likely to fall into the local optimum (because setting (2) can only explore the currently best position, missing the chance to explore more positions).

SA: As the temperature T determines the probability that one solution will be accepted, it affects the convergence speed. We set T to three values (10, 30, 50).

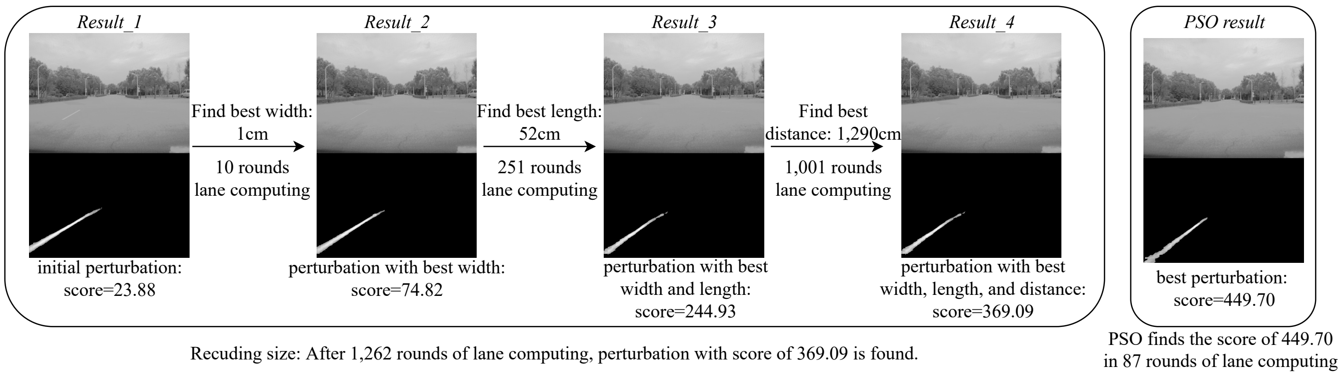

Result_1
Result_2
Result_3
Result_4
PSO result
Find best width: 1cm
10 rounds lane computing
Find best length: 52cm
251 rounds lane computing
Find best distance: 1,290cm
1,001 rounds lane computing
initial perturbation: score=23.88
perturbation with best width: score=74.82
perturbation with best width and length: score=244.93
perturbation with best width, length, and distance: score=369.09
best perturbation: score=449.70
Recuding size: After 1,262 rounds of lane computing, perturbation with score of 369.09 is found.
PSO finds the score of 449.70 in 87 rounds of lane computing

Figure 17: Comparison of the *RS* method and PSO.

Results / *Paras*	step	*Result*_1	*Result*_2	*Result*_3	*Result*_4
wid	1cm	20cm	**1cm**	1cm	1cm
len	1cm	300cm	300cm	**52cm**	52cm
D_1	1cm	16m	16m	16m	**12.90m**
score	\	23.88	74.82	244.93	369.09

Table 5: Finding the perturbation by *RS* method. The first column (*step*) lists the step used for searched parameter; The columns *Result_1* lists the initial values of *wid*, *len*, and D_1 and the corresponding score; The columns *Result_2 to Result_4* shows the values of *wid*, *len*, and D_1 after each major steps and the corresponding scores. The corresponding images are shown in Fig.17.

D Why not simply reducing the size of perturbations?

As introduced in §4, we formulate an optimization problem to find the best digital perturbations.Another possible way to find the best perturbation is *reducing the size* of perturbations until a minimum size that still registers on the lane detection model is found. We call it as *RS* method. Although the *RS* method is simple, the experimental results show that it is less effective and efficient than our heuristic algorithms, because it cannot jointly optimize the parameters to find the best perturbation. Below we explain how we carry out the experiments for comparison and the results are shown in Fig.17.

For the convenience of comparison, we fix four parameters and search the best value for the other three parameters. Specifically, the fixed parameters are: $n = 1$, $\theta = 0$, $D_2 = 2.50$m and $\Delta G = 20$. Then, the *RS* method looks for the best *wid*, *len* and D_1 through the following steps.

Step 1. Initialization. The basic idea of the *RS* method is to progressively keep reducing the perturbation size until the best score is found. To achieve this, we need to first set the initial perturbation that can generate the fake lane, and then adjust the parameter of this perturbation. The initial values of the parameters are set to $x = (n = 1, \theta = 0, D_2 = 2.50m, \Delta G = 20, wid = 20cm, len = 300cm, D_1 = 16m)$. As shown in *Result*_1 in Fig.17 and Tab.5, the score $S(x)$ of the initial perturbation is 23.88.

Step 2. Find the best width (wid). We let the width *wid* of the perturbation be from 10cm to 1cm (i.e., decreasing it by 1cm each round), and record the score of each perturbation. We finally find that the best width is 1cm, and the corresponding best score is 74.82 (shown in *Result*_2 in Fig.17 and Tab.5). Since *wid* is decreased by 1cm each round, the searching of this step takes 10 rounds of lane computing.

Step 3. Find the best length (len). We let the length *len* of the perturbation be from 300cm to 50cm (i.e., decreasing it by 1cm each round). Eventually, we find the best length of 52cm, and the corresponding best score is 244.93 (shown in *Result*_3 in Fig.17 and Tab.5). Since *len* is decreased by 1cm each round, the searching of this step takes 251 rounds of lane computing.

Step 4. Find best longitudinal distance (D_1). We let the longitudinal distance D_1 of the perturbation be from 2,000cm to 1,000cm (i.e., decreasing it by 1cm each round). We finally find that the best D_1 is 1,290cm, and the corresponding best score is 369.09 (shown in *Result*_4 in Fig.17 and Tab.5). Since D_1 is decreased by 1cm each round, the searching of this step takes 1,001 rounds of lane computing.

• **Heuristic algorithm:** Since the answer to RQ1 indicates that PSO is the best one for solving our optimization problem, we choose PSO for comparison. The experimental setup is the same as that in RQ1. As shown in *PSO result* in Fig.17, PSO finds the perturbation with a score of 449.70 by taking only 87 rounds of lane computing.

• **Conclusion.** PSO takes only 87 rounds of lane computing to find a better result ($S(x) = 449.70$) than the *RS* method ($S(x) = 369.09$) that takes 1,262 rounds of lane computing. Therefore, PSO is more effective and efficient (about 15 times faster) than the *RS* method.