Adversarial Objects Against LiDAR-Based Autonomous Driving Systems

Yulong Cao [*1] **Chaowei Xiao**[*1] **Dawei Yang**[*1] **Jing Fang** [2] **Ruigang Yang** [2]
Mingyan Liu[1] **Bo Li**[3]

[1]University of Michigan, Ann Arbor
[2]Baidu Research, Baidu Inc.
[3] University of Illinois at Urbana-Champaign

Abstract

Deep neural networks (DNNs) are found to be vulnerable against adversarial examples, which are carefully crafted inputs with a small magnitude of perturbation aiming to induce arbitrarily incorrect predictions. Recent studies show that adversarial examples can pose a threat to real-world security-critical applications: a "physically adversarial *Stop Sign*" can be synthesized such that the autonomous driving cars will misrecognize it as others (e.g., a speed limit sign). However, these image-based adversarial examples cannot easily alter 3D scans such as widely equipped LiDAR or radar on autonomous vehicles. In this paper, we reveal the potential vulnerabilities of LiDAR-based autonomous driving detection systems, by proposing an optimization based approach *LiDAR-Adv* to generate real-world adversarial objects that can evade the LiDAR-based detection systems under various conditions. We first explore the vulnerabilities of LiDAR using an evolution-based blackbox attack algorithm, and then propose a strong attack strategy, using our gradient-based approach *LiDAR-Adv*. We test the generated adversarial objects on the Baidu Apollo autonomous driving platform and show that such physical systems are indeed vulnerable to the proposed attacks. We 3D-print our adversarial objects and perform physical experiments with LiDAR equipped cars to illustrate the effectiveness of *LiDAR-Adv*. Please find more visualizations and physical experimental results on this website: `https://sites.google.com/view/lidar-adv`.

1 Introduction

Machine learning, especially deep neural networks (DNNs), have achieved great successes in various domains, [5, 6, 9, 17]. Several safety-critical applications such as autonomous vehicles (AV) have also adopted machine learning models and achieved promising performance. However, recent studies show that machine learning models are vulnerable to adversarial attacks [2, 8, 18, 20, 21, 23]. In these attacks, small perturbations are sufficient to cause various well-trained models to output "adversarial" prediction. In this paper we aim to explore similar vulnerabilities in today's autonomous driving systems.

Such adversarial attacks have been largely explored in the image domain. In addition, to demonstrate such attacks pose a threat in the real world, some studies propose to generate physical stickers or printable textures that can confuse a classifier to recognize a stop sign [1, 7]. However, an autonomous driving system is not merely an image-based classifier. For perception, most autonomous driving detection systems are equipped with LiDAR (Light Detection And Ranging) or RADAR (Radio Detection and Ranging) devices which are capable of directly probing the surrounding 3D environment with laser beams. This raises the doubt of whether texture perturbation in previous work will affect LiDAR-scanned point clouds. In addition, the LiDAR-based detection system consists of multiple non-differentiable steps, rather than a single end-to-end network, which largely limits the use of gradient-based end-to-end attacks. These two key obstacles not only invalidate previous image-based approaches, but also raise several new challenges when we want to construct an adversarial object: 1) LiDAR-based detection system projects 3D shape to a point cloud using physical LiDAR equipment. The point cloud is then fed into the machine learning detection system. Therefore, how

[*]The first three authors contributed equally.

shape perturbation affects the scanned point cloud is not clear. 2) The preprocessing of the LiDAR point clouds is non-differentiable, preventing the naive use of gradient-based optimizers. 3) The perturbation space is limited due to multiple aspects. First, we need to ensure the perturbed object can be reconstructed in the real world. Second, a valid LiDAR scan of an object is a constrained subset of point cloud, making the perturbation space much smaller compared to perturbing the point cloud without any constraint [19].

In this paper, we propose *LiDAR-Adv* to address the above issues and generate adversarial object against real-world LiDAR system as shown in Figure 1. We first simulate a differentiable LiDAR renderer to bridge the perturbations from 3D objects to LiDAR scans (or point cloud). Then we formulate 3D feature aggregation with a differentiable proxy function. Finally, we devise different losses to ensure the smoothness of the generated 3D adversarial objects. To better demonstrate the flexibility of the proposed attack approach, we evaluate our attacking approach under two different attacking scenarios: 1) *Hiding Object*: synthesizing an "adversarial object" that will not be detected by the detector. 2) *Changing Label*: synthesizing an "adversarial object" that is recognized as a specified adversarial target by the detector. We also compare *LiDAR-Adv* with the evolution algorithm in the blackbox setting.

To evaluate the real-world impact of *LiDAR-Adv*, we 3D print out the generated adversarial objects and test them on the Baidu Apollo autonomous driving platform, an industry-level system which is not only highly adopted for research purpose but also actively used in industries. We show that with 3D perception and a production-level multi-stage detector, we are able to mislead the autonomous driving system to achieve different adversarial targets.

To summarize, our contributions are as follows: (1) We propose *LiDAR-Adv*, an end-to-end approach to generate physically plausible adversarial objects against LiDAR-based autonomous driving detection systems. To the best of our knowledge, this is the first work to exploit adversarial objects for such systems. (2)We experiment on Apollo, an industry-level autonomous driving platform, to illustrate the effectiveness and robustness of the attacks in practice. We also compare the objects generated by *LiDAR-Adv* with evolution algorithm to show that *LiDAR-Adv* can provide smoother objects. (3) We conduct physical experiments by 3D-printing the optimized adversarial object and show that it can consistently mislead the LiDAR system equipped in a moving car.

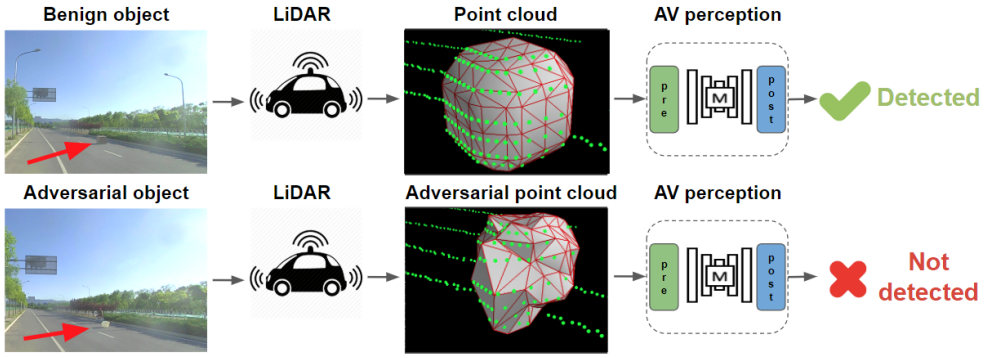


Figure 1: Overview of *LiDAR-Adv*. The first row shows that a normal box will be detected by the LiDAR-based detection system; while the generated adversarial object with similar size in row 2 cannot be detected.

2 Related work

Image-space adversarial attacks Adversarial examples have been heavily explored in 2D image domains [3, 8, 13, 14, 21]. Various works [1, 7, 11] start to study robust physical adversarial examples. Evtimov et al. [7] has created printable 2D stickers to attach to a stop sign and cause a detector to predict wrong labels. Following this line, there are also works [12, 22] aiming to optimize the 3D shapes to show that even the surface geometry itself can produce adversarial behaviors.

In this work, we exploit the object surfaces to generate adversarial objects, and one fundamental challenge that differentiates our work from the previous ones is: the sensor in a LiDAR-based system directly probes the 3D environment as the input, bypassing surface textures of the adversarial objects. This means we may only rely on shape geometry to perform any attacks. On the other hand, compared to prior works that have shown successes on attacking single models, it is worth noting that the victim model which we experiment on

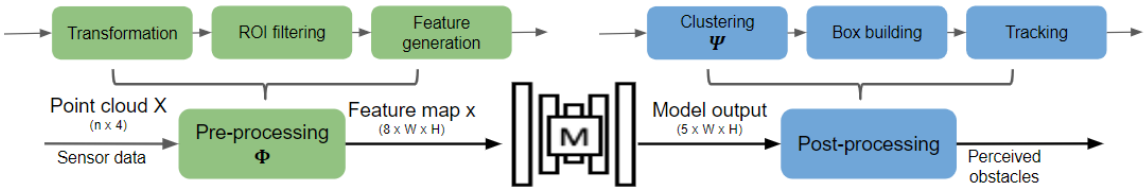


Figure 2: Overview of LiDAR-based detection on AV.

(Apollo), is not merely an end-to-end deep learning model but an industry-level autonomous driving platform that consists of multiple non-differentiable parts.

Adversarial point clouds Xiang et al. [19] show a proof of concept, that models taking raw 3D point clouds as input [15] can be vulnerable to adversarial point clouds. However, this approach is only evaluated with a single digital model. It is not clear that the generated point clouds can form plausible 3D shape surfaces, or it can be reconstructed through LiDAR scans. While in our approach, though the victim model takes point clouds as input similarly, these point clouds have to satisfy extra constraints such as: all points have to be the intersections of the laser beams and the object surfaces. We address this challenge by proposing a differentiable renderer which simulates the reconstructed laser beams projecting onto object surfaces. As we will show later, when the object moves, the point cloud changes in accordance with the laser hits, and how to enforce the robustness against such LiDAR scans is non-trivial.

3 LiDAR-based Detection

In this section, we provide the details of the LiDAR-based detection system that are directly related to our proposed adversarial attacks. Refer to the online repository[2] for more details.

An overview of the system is shown in Fig. 2. First, a LiDAR sensor scans the 3D environment and obtains a raw point cloud of the scene. Next, the point cloud goes through preprocessing, and is fed to a detection model. Finally, post-processing is applied to the detection output to obtain the detection predictions.

LiDAR. A LiDAR sensor scans the surrounding environment and generates a point cloud of $X \in \mathbb{R}^{n\times 4}$ with 3D coordinates (u^X, v^X, w^X) and intensity int^X. First, a sensor fires off an array of laser beams consecutive in horizontal and vertical directions. It then captures the light intensity reflecting back, and calculates the time that photons have traveled along each beam (Time of Flight). The distance and the coordinate of the surface points along the beam can be computed. These points then form a raw point cloud of the object surfaces in the environment. LiDAR sensors are supposedly robust to object surface textures, as the Time of Flight is not easily affected by texture change. Though it also detects the intensity of reflected lights, it is unclear how adversarial algorithms designed for natural lighting in image space can be adapted to invisible laser beams used as light sources. Therefore, image-based adversarial attacks may have limited effects on such LiDAR-based detection system.

Preprocessing phase. The previous raw point cloud X goes through a preprocessing phase to form a feature map of $x \in \mathbb{R}^{H\times W\times 8}$ (see Sec. B). The raw point cloud X is first transformed and filtered based on a High Definition Map (HDMap) to attain a ROI point cloud X_{roi}. This point cloud X_{roi} is then sliced into $H \times W$ vertical cells at $(\lfloor u^{X_{roi}} \rfloor, \lfloor v^{X_{roi}} \rfloor)$. This "hard" assignment of points into cells will introduce piecewise zero gradients for **counting** and **max** w.r.t. the input. After slicing, in each cell, the information of the points are aggregated to generate a feature of size 8 for this cell, including heights, intensity, point counts *etc*. (detailed in Sec. B). This $H \times W \times 8$ feature map x will then be fed into a machine learning model.

In this procedure, many operations (*e.g.* max height, count) introduce zero gradients due to the "hard" assignment, so the end-to-end optimization-based attack algorithms are not directly applicable.

Machine learning model. Deep Neural Networks (DNNs) are used to process the $H \times W \times 8$ feature map, and then output the metrics for each one of the $H \times W$ cells. The metrics are listed in Sec. B.

Post-processing phase. The post-processing phase aggregates previous outputs from the machine learning model and recognizes the detected objects. The Post-processing can be roughly divided into 3 major sequential components: *clustering*, *box building* and *tracking*. The clustering process composes obstacle candidates

[2]https://github.com/ApolloAuto/apollo/tree/r2.0.0/docs

using both the model output metrics and ROI point cloud X_{roi} generated from the preprocessing phase. In the clustering process, cells with higher *objectness* confidence (greater than 0.5 by default) are used for constructing clusters by building a connected graph using *center offset*. The obstacle candidates are produced by selecting the clusters with two constraints: (1) the average *confidence* of cells in the cluster needs to be greater than 0.1 (2) the number of points in the ROI point cloud that are assigned to the cluster is greater than 3. The class probabilities of the obstacle candidate are calculated by summing up class probabilities of all cells in the cluster. The box builder then reconstructs the bounding boxes including the height, width, length of the obstacle candidates from the point cloud assigned to the candidate. Finally, the tracker integrates multiple frames of processed results to generate tracked objects as the output of the LiDAR-based detection, together with additional information such as object id, speed etc.

Note that in this paper, we only consider a single frame for the adversarial attacks as a demonstration of feasibility. For the case of multiple frames, it can be treated as enhancing robustness against object motions, and such robustness against different locations is shown in later experiments (§ 5.4).

4 Generating Adversarial Object Against LiDAR-based Detection

In this section, we will formulate the problem first and describe the adversarial goals and challenges. We then describe our whitebox method *LiDAR-Adv* which aims to tackle the challenges and fulfill diverse adversarial goals. Finally, we propose an evolution-based attack method for blackbox settings.

4.1 Methodology overview

Given a 3D object S in a scene, as stated in the background, after the scene is scanned by a LiDAR sensor, a point cloud X is then generated based on S so that $X = \mathrm{render}(S, \mathrm{background})$ For preprocessing, this point cloud X is sliced and aggregated to generate x, which is a $H \times W \times 8$ feature vector, and we call this aggregation process as Φ: $x = \Phi(X)$. Then a machine learning model M maps this 2D feature $x \in R^{H \times W \times 8}$ to $O = M(x)$, where $O \in R^{H \times W \times 7}$ (see Sec. B for concrete output meanings). O is then post-processed by a clustering process Ψ to generate the confidence y_{conf} and label y_{label} of detected obstacles so that $(y_{conf}, y_{label}) = \Psi(O)$ An adversarial attacker aims to manipulate the object S to achieve the adversarial goals. Here we define two types of adversarial goals: 1) *Hiding object*: Hide an existing object S by manipulating S; 2) *Changing label*: Change the label y of the detected object S to a specified target y'.

To achieve the above adversarial goals in LiDAR-based detection is non-trivial, and there are the following challenges: 1) **Multiple pre/post-processing stages.** Unlike the adversarial attacks on traditional image-space against machine learning tasks such as classification and object detection, the LiDAR-based detection here is not a single end-to-end learning model, It consists of the differentiable learning model M and several non-differentiable parts including preprocessing and post-processing. Thus, the direct gradient based attacks are not directly applicable. 2) **Manipulation constraints.** Instead of directly manipulating the point cloud X as in [19], we manipulate the 3D shape of S given the limitation of LiDAR. The points in X are the intersections of laser beams and object surfaces and cannot move freely, so the perturbations on each point may affect each other. Keeping the shape plausible and smooth adds additional constraints [22]. 3) **Limited Manipulation Space.** Consider the practical size of the object versus the size of the scene that is processed by LiDAR, the 3D manipulation space is rather small ($< 2\%$ in our experiments), as shown in Fig. 1.

Given the above challenges, we design an end-to-end attacking pipeline. In order to facilitate gradient-based algorithms, we implement an approximated differentiable renderer R , which simulates the functionality of LiDAR, to intersect a set of predefined rays with the 3D object surface (S) consisting of vertices V and faces W. The points at the intersections form the raw point cloud X. After preprocessing, the point cloud is then fed to a preprocessing function Φ to generate the feature map $x = \Phi(X)$. The feature map x is then taken as input for a machine learning model M to obtain the output metrics $O = M(x)$.

The whole progress can be symbolized as $F(S) = M(\Phi(R(S)))$. Note that by differentiating the renderer R, the whole process $F(*) = M(\Phi(R(*)))$ is differentiable w.r.t. S. In this way, we can manipulate S to generate adversarial S_{adv} via our designed objective function operating on the final output $F(S)$.

4.2 Approximate differentiable renderer

LiDAR simulation The renderer R simulates the physics of a LiDAR sensor that probes the objects in the scene by casting laser beams. The renderer first takes a mesh S as input, and compute the intersections of a

set of predefined ray directions to the meshes in the scene to generate point cloud X. After depth testing, the distance along each beam is then captured, representing the surface point of a mesh that it first encounters, as if a LiDAR system receives a reflection from an object surface. Knowing ray directions of the beams, the exact positions of the intersection points can be inferred from the distance, in the form of point clouds X.

Real background from a road scene We render our synthetic object onto a realistically captured point cloud. First, we obtain the 3D scan of a road scene, using the LiDAR sensor Riegl VMX-1HA mounted on a car. Then, we obtain the laser beam directions by computing the normalized vectors from the origin (LiDAR) pointing to the scanned points. This fixed set of ray directions are then used for rendering our synthetic objects throughout the paper. Note that we can also manually set ray directions given sensor specifications, but it will be less real, because it may not model the noises and fluctuations that occur in a real LiDAR sensor.

Hybrid rendering of synthetic objects onto a realistic background Given the ray directions reconstructed from the background point cloud, a subset will intersect with the object, forming the point cloud for the object of interest. The corresponding background points are then removed since these background points are occluded by the foreground object. In this way, we obtain a semi-real synthetic point cloud scene: the background points come from the captured real data; the foreground points are physically accurate simulated based on the collected real data.

4.3 Differentiable proxy function for feature aggregation

As in Section 3, in the preprocessing of Apollo, it aggregates the point cloud into hardcoded 2D features, including **count**, **max height**, **mean height**, **intensity** and **non-empty**. These operations are non-differentiable. In order to apply end-to-end optimizers to for our synthetic object S, we need to flow the gradient through the feature aggregation step, with the help of our proxy functions.

Given a point cloud X with coordinate (u^X, v^X, w^X), and we hope to count the number of points falling into the cells of a 3D grid $G \in \mathbb{R}^{H\times W\times P}$. For a point X_i with location$(u^{X_i}, v^{X_i}, w^{X_i})$, we increase the count of 8 cells: the centers of these 8 cells form a cube, and the point X_i is inside this cube. Specifically, we increase the count of 8 cells using trilinear weights:

$$G(u_i, v_i, w_i) = \sum_p (1 - d(u_p, u^{X_i})) \cdot (1 - d(v_p, w^{X_i})) \cdot (1 - d(w_p, w^{X_i})), \tag{1}$$

where $p \in \mathcal{N}(u^{X_i}, v^{X_i}, w^{X_i})$ are the indices of the 8-pixel neighbors at location $(u^{X_i}, v^{X_i}, w^{X_i})$ and $d(\cdot, \cdot)$ represents the L_1 distance. The **count** feature $x_{\mathbf{count}}$ is the value $G_p = G(u_i, v_i, w_i)$ computed for each grid i. Note that this feature is no longer an integer and can have non-zero gradients w.r.t. the point coordinates.

We then use this "soft count" feature to further compute "mean height" and "max height" features. For simplicity, we first define a constant height matrix $T \in \mathcal{R}^{H,W,P}$, where $T(.,.,p) = p, p \in \{1...P\}$. This matrix stores the height of each cell. Next, we can formulate the **mean height** $x_{\mathbf{mean-height}}$ and **max height** $x_{\mathbf{max-height}}$ using soft count G:

$$x_{\mathbf{mean-height}} = \cdot \frac{\sum_{p\in P} G_p \circ T_p}{\sum_{p\in P} G_p + \epsilon} \qquad \text{and} \qquad x_{\mathbf{max-height}} = \max_p \operatorname{sign}(G(.,.,p)) \circ T(.,.,p) \tag{2}$$

where $\epsilon = 1e^{-7}$ to prevent zero denominators. Note that the sign function is non-differentiable, so we approximate the gradient using $\operatorname{sign}(G) = G$ during back propagation. The feature **intensity** has the similar formulation of **height** so we omit them here. The feature **non-empty** is formulated as $x_{\mathbf{non-empty}} = \operatorname{sign}(G)$.

We denote the above trilinear approximator as Φ', in constrast to the original non-differentiable preprocessing step Φ. A visualization of output of our $\Phi'(X)_{\mathbf{count}}$ compared to the original $\Phi(X)_{\mathbf{count}}$ is shown in Sec. C. Since our approximation introduces differences in counting, $\Phi'(X)$ is not strictly equal to $\Phi(X)$, resulting in different obj values of the final model prediction. We observe that this difference will raise new challenges to transfer the adversarial object generated based on Φ' to Φ. To solve this problem, we reduce the difference between Φ' and Φ, by replacing the L1 distance d in Eq. 1 with $d(u_1, u_2) = 0.5 + 0.5 \cdot \tanh(\mu \cdot (u_1 - u_2 - 1))$ where $\mu = 20$. We name this approximation "tanh approximator" and denote it Φ''. We observe that the input difference between Φ'' and the original Φ is largely reduced compared to Φ', allowing for smaller errors of the model predictions and better transferability. To extend our approximator and further reduce the gap betweer Φ'' and Φ, we interpolate the distance: $d(u_1, u_2) = \alpha \cdot (0.5 + 0.5 \cdot \tanh(5\mu \cdot (u_1 - u_2 - 1))) + (1 - \alpha) \cdot (u_1 - \lfloor u_2 \rfloor)$, where α is a hyper-parameter balancing the accuracy of approximation and the availability of gradients.

4.4 Objective functions

Our objective is to generate a synthetic adversarial object S^{adv} from an original object S by perturbing its vertices, such that the LiDAR-based detection model will make incorrect predictions. We first optimize S^{adv} against the semi-real simulator detection model M.

$$\mathcal{L}(S^{\text{adv}}) = \mathcal{L}_{\text{adv}}(S^{\text{adv}}, M) + \lambda \mathcal{L}_{\mathbf{r}}(S^{\text{adv}}; S) \tag{3}$$

The objective function $\mathcal{L}$ consists of two losses. $\mathcal{L}_{\text{adv}}$ is the adversarial loss to achieve the target goals while the $\mathcal{L}_{\mathbf{r}}$ is the distance loss to keep the properties of the "realistic" adversarial 3D object S^{adv}. We optimize the objective function by manipulating the vertices. The distance loss is defined as follows:

$$\mathcal{L}_{\mathbf{r}} = \sum_{\boldsymbol{v}_i \in V} \sum_{q \in \mathcal{N}(\boldsymbol{v}_i)} \|\Delta \boldsymbol{v}_i - \Delta \boldsymbol{v}_q\|_2^2 + \beta \sum_{\boldsymbol{v}_i \in V} \|\Delta \boldsymbol{v}_i\|_2^2, \tag{4}$$

where $\Delta \boldsymbol{v}_i = \boldsymbol{v}_i^{\text{adv}} - \boldsymbol{v}_i$ represents the displacement between the adversarial vertex $\boldsymbol{v}_i^{\text{adv}}$ and pristine vertex $\boldsymbol{v}_i$. β is the hyperparameter balancing these two losses. The first losses [22] is a Laplacian loss preserving the smoothness of the perterbation applied on the adversarial object S^{adv}. The second part is the $L2$ distance loss to limit the magnitude of perturbation.

Objective: hide the inserted adversarial object As introduced in the background section, the existence of the object highly depends on the "positiveness" metric. $H(*, M, S)$ denotes a function extracting $*$ metric from the model M given an object S. $\mathcal{A}$ is the mask of the target object's bounding box. Our adversarial loss is represented as follows:

$$\mathcal{L}_{\text{adv}} = H(\text{pos}, M, S) * \mathcal{A} \tag{5}$$

Objective: changing label In order to change the predicted label of the object, it needs to increase the logits of the target label and decrease the logits of the ground-truth label. Moreover, it also needs to preserve the high positiveness. Based on this, our adversarial loss is written as

$$\mathcal{L}_{\text{adv}} = (-H(\text{c}_{y'}, M, S) + H((c)_y, M, S) \cdot \mathcal{A} * H(\text{pos}, M, S) \tag{6}$$

In order to ensure that adversarial behaviors still exist when the settings are slightly different, we create robust adversarial objects that can perform successful attacks within a range of settings, such as different object orientations, different positions to the LiDAR sensor *etc*. To achieve such goal, we sample a set of physical transformations to optimize the loss expectation. In reality, we create a victim set D by rendering the object S at different positions and orientations. Instead of optimizing an adversarial object S by attacking single position and orientation, we generate an universal adversarial object S to attack all positions and orientations in the victim set D.

4.5 Blackbox Attack

In reality, it is possible that the attackers do not have complete access to the internal model parameters, *i.e*. the model is a black box. Therefore, in this subsection, we also develop an evolution-based approach to perform blackbox attack.

In evolution, a set of individuals represent the solutions in the search space, and the fitness score defines how good the individuals are. In our case, the individuals are mesh vertices of our adversarial object, and the fitness score is $-\mathcal{L}(S^{\text{adv}})$. We initialize m mesh vertices using the benign object S. For each iteration, new population of n mesh vertices are generated by adding random perturbations, drawn from a Gaussian distribution $\mathcal{N}(0, \sigma)$, to each mesh vertices in the old population. m mesh vertices with top fitness scores will remain for the next iteration, while the others will be replaced. We iterate the process until we find a valid solution or reach a maximum number of steps.

5 Experiments

In this section, we first expose the vulnerability of the LiDAR-based detection system via the evolution-based blackbox algorithm by achieving the goal of "hiding object", because missing obstacles can cause accidents in real life. We then show the qualitative results and quantitative results of *LiDAR-Adv* under whitebox settings. In addition, we also show that *LiDAR-Adv* can achieve other adversarial goals such as "changing label". Moreover, the point clouds are continuously captured in real life, so attacks in a single static frame may not have much effect in real-world cases. Therefore, in our experiments, we generate a universal robust adversarial object against a victim dataset which consists of different orientations and positions. We 3D-print such universal adversarial object and conduct the real-world drive-by experiments, to show that they indeed can pose a threat on road.

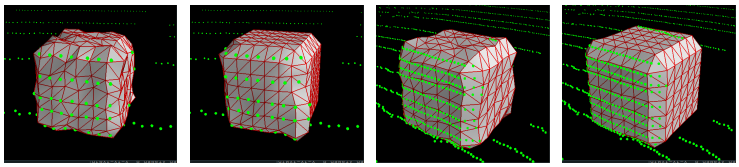
5.1 Experimental setup

We conduct the evaluation on the perception module of Baidu Apollo Autonomous Driving platform (V2.0). We initialize the adversarial object as a resampled 3D cube-shaped CAD model using MeshLab [4]. For rendering, we implement a fully differential LiDAR simulator with predefined laser beam ray directions extracted from a real scene captured by the Velodyne HDL-64E sensor, as stated in § 4.2. It has around 2000 angles in the azimuth angle and around 60 angles in the elevation angle. We use Adam optimizers [10], and choose λ as 0.003 in Eq. 3 using binary search. For the evolution-based blackbox algorihtm , we choose $\sigma = 0.1$, $n = 500$ and $m = 5$.

5.2 Vulnerability analysis

Here, we first show the existence of the vulnerability using our evolution-based blackbox attacks, with the goal of "hiding object". We generate adversarial objects in different size (50cm and 75cm in edge length). For each object, we select 45 different position and orientation pairs for evaluation, and the results are shown in Table 1. The results indicate that the LiDAR-based detection system is vulnerable. The visualization of the adversarial object is shown in Figure 3(a) and (c).

5.3 *LiDAR-Adv* with different adversarial goals



(a) 50cm: Evo. (b) *LiDAR-Adv* (c) 75cm: Evo. (d) *LiDAR-Adv*

Figure 3: Adversarial meshes of different sizes can fool the detectors even with more LiDAR hits. We generate the object with *LiDAR-Adv* and evolution-based method (Evo.).

After showing the vulnerability of the LiDAR-based detection system, here we focus on whitebox settings to explore what a powerful adversary can do, since "the design of a system should not require secrecy" [16]. Therefore, we evaluate the effectiveness of our whitebox attack *LiDAR-Adv* with the goal of "hidding object". We also evaluate the feasibility of *LiDAR-Adv* to achieve another goal of "changing label".

Hiding object We follow the same settings as in the above sections, and Table 1 shows the results. We find that *LiDAR-Adv* can achieve 71% attack success rate with size 50cm. The attack success rate is consistently higher than the evolution-based blackbox attacks. Figure 3 (b) and (c) show the visualizations of the adversarial objects. We visually observe that the adversarial objects generated by *LiDAR-Adv* are smoother than that of evolution.

Table 1: Attack success rate of *LiDAR-Adv* and evolution based method under different settings.

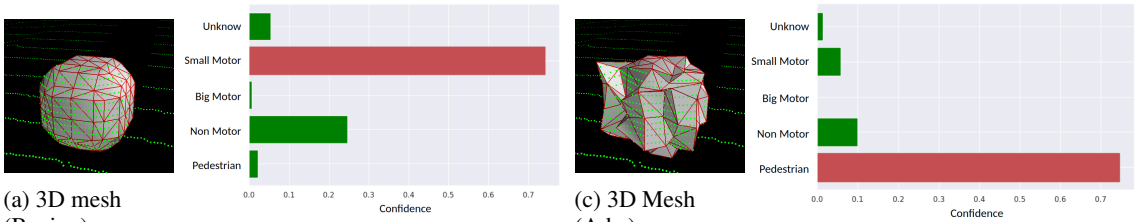
Attacks	Object size	
	50cm	75cm
LiDAR-Adv	32/45 (71%)	23/45 (51%)
Evolution-based	28/45 (62%)	16/45 (36%)

Changing label The result shown in Figure 4 indicates that we can successfully change the label of the object. We also experiment with different initial shapes and target labels. More details can be found in Sec. D.

5.4 *LiDAR-Adv* on generating robust physical adversarial objects

To ensure the generated *LiDAR-Adv* preserves adversarial behaviors under various physical conditions, we optimize the object by sampling a set of physical transformations such as possible positions and orientations. We show that the generated robust adversarial object is able to achieve the attack goal of hiding object with a high success rate in Table 2. An interesting phenomenon is that some attack performance under the unseen settings is even better than that within the controlled environment. This implies that our adversarial objects are robust enough to generalize to unseen settings.

Furthermore, we evaluate the generated robust adversarial object in the physical world by 3D printing the generated object. We collect the point cloud data using a Velodyne HDL-64E sensor with a real car driving by



(a) 3D mesh (Benign)

(b) Predictions (Benign)

(c) 3D Mesh (Adv.)

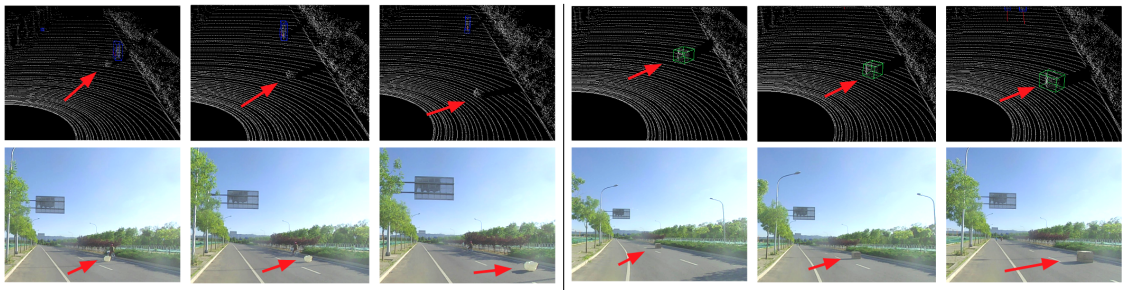
(d) Predictions (Adv.)

Figure 4: The adversarial mesh generated by *LiDAR-Adv* is mis-detected as a "Pedestrian".

Table 2: Attack success rates of *LiDAR-Adv* at different positions and orientations under both controlled and unseen settings.

Controlled Setting		Unseen Setting			
		Distance (cm)		Orientation (°)	
Distance (cm) & Orientation (°)	Attack	0-50	50-100	0-5	0-10
$\{0, \pm 50\} \times \{0, \pm 2.5, \pm 5\}$	41/45	96/100	91/100	10/10	9/10
$\{0, \pm 50\} \times \{0, \pm 2.5, \pm 5\}$	43/45	96/100	90/100	8/10	10/10

and evaluate the collected traces on the LiDAR perceptual module of Baidu Apollo. As shown in Figure 5a, we find that the adversarial object is not detected around the target position among all 36 different frames. To compare, the box object (in Figure 5b) is detected in 12 frames among all 18 frames. The number of total frames is different due to the different vehicle speed. More details can be found in Sec. D.



(a) Adversarial object

(b) Benign object

Figure 5: Results of physical attack. Our 3D-printed robust adversarial object by *LiDAR-Adv* is not detected by the LiDAR-based detection system in a moving car. Row 1 shows the point cloud data collected by LiDAR sensor, and Row 2 presents the corresponding images captured by a dash camera.

6 Conclusion

We show that LiDAR-based detection systems for autonomous driving are vulnerable against adversarial attacks. By integrating our proxy differentiable approximator, we are able to generate robust physical adversarial objects. We show that the adversarial objects are able to attack the Baidu Apollo system at different positions with various orientations. We also show *LiDAR-Adv* can generate much smoother object than evolution based attack algorithm. Our findings raise great concerns about the security of LiDAR systems in AV, and we hope this work will shed light on potential defense methods.

References

[1] A. Athalye and I. Sutskever. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

[2] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy, 2017*, 2017.

[3] N. Carlini and D. A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 39–57, 2017. doi: 10.1109/SP.2017.49. URL `https://doi.org/10.1109/SP.2017.49`.

[4] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference*, volume 2008, pages 129–136, 2008.

[5] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[6] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. L. Seltzer, G. Zweig, X. He, J. D. Williams, et al. Recent advances in deep learning for speech research at microsoft. In *ICASSP*, volume 26, page 64, 2013.

[7] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song. Robust physical-world attacks on deep learning models. *arXiv preprint arXiv:1707.08945*, 1, 2017.

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

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[12] H.-T. D. Liu, M. Tao, C.-L. Li, D. Nowrouzezahrai, and A. Jacobson. Adversarial geometry and lighting using a differentiable renderer. *CoRR*, abs/1808.02651, 2018.

[13] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

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

[15] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.

[16] C. E. Shannon. Communication theory of secrecy systems. *Bell Labs Technical Journal*, 28(4):656–715, 1949.

[17] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[18] M. Sun, J. Tang, H. Li, B. Li, C. Xiao, Y. Chen, and D. Song. Data poisoning attack against unsupervised node embedding methods. *arXiv preprint arXiv:1810.12881*, 2018.

[19] C. Xiang, C. R. Qi, and B. Li. Generating 3d adversarial point clouds. *arXiv preprint arXiv:1809.07016*, 2018.

[20] C. Xiao, R. Deng, B. Li, F. Yu, D. Song, et al. Characterizing adversarial examples based on spatial consistency information for semantic segmentation. In *Proceedings of the (ECCV)*, pages 217–234, 2018.

[21] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song. Generating adversarial examples with adversarial networks. *arXiv preprint arXiv:1801.02610*, 2018.

[22] C. Xiao, D. Yang, B. Li, J. Deng, and M. Liu. Meshadv: Adversarial meshes for visual recognition. In *CVPR*, 2018.

[23] C. Xiao, J.-Y. Zhu, B. Li, W. He, M. Liu, and D. Song. Spatially transformed adversarial examples. *arXiv preprint arXiv:1801.02612*, 2018.

A Differential Renderer

LiDAR Simulation The renderer simulates the physics of a LiDAR sensor that probes the objects in the scene by casting laser N_{ray} rays: $R = \{\mathbf{r}_i \in \mathbb{R}^3, \|\mathbf{r}_i\| = 1, i = 1, 2, \cdots, N_{\text{ray}}\}$, with $\mathbf{r}_i$ representing the direction of the i-th ray. Given a shape S with the surface ∂S as input, the renderer computes the intersections of rays R to the mesh faces in the scene. For each ray $\mathbf{r}_i$, the intersection coordinate P_i are computed through depth testing (assuming the center of the rays is at origin, *i.e.* the reference frame of LiDAR):

$$\mathbf{p}_i = \arg\min_{\mathbf{p}} \{\|\mathbf{p}\| \mid \exists t > 0, \mathbf{p} = t \cdot \mathbf{r}_i, \mathbf{p} \in \partial S\}, \quad i = 1, 2, \cdots, N_{\text{ray}} \tag{7}$$

Object insertion Notice that we have a predefined set of rays R. To obtain these rays, one can refer to the specifications of a LiDAR device. In our paper, we directly compute the directions from the captured background point cloud P', so that the rays are close to real world cases:

$$\mathbf{r_i} = \frac{\mathbf{p}'_i}{\|\mathbf{p}'_i\|} \tag{8}$$

With this, Eq. (7) becomes:

$$\mathbf{p}_i = \arg\min_{\mathbf{p}} \{\|\mathbf{p}\| \mid \mathbf{p} = \mathbf{p}'_i \vee \mathbf{p} = t \cdot \mathbf{r}_i, t > 0, \mathbf{p} \in \partial S\}, \quad i = 1, 2, \cdots, N_{\text{ray}} \tag{9}$$

This means when rays intersect with an object, the corresponding background points blocked by the above-ground parts of the object are removed during depth testing; if the object is below the ground, the intersections leave those corresponding background points intact also due to depth testing. In this way, we obtain a semi-real synthetic point cloud scene: the background points come from the captured real data; the foreground points are physically accurate simulations based on the captured real data.

B Background

B.1 LiDAR perception system

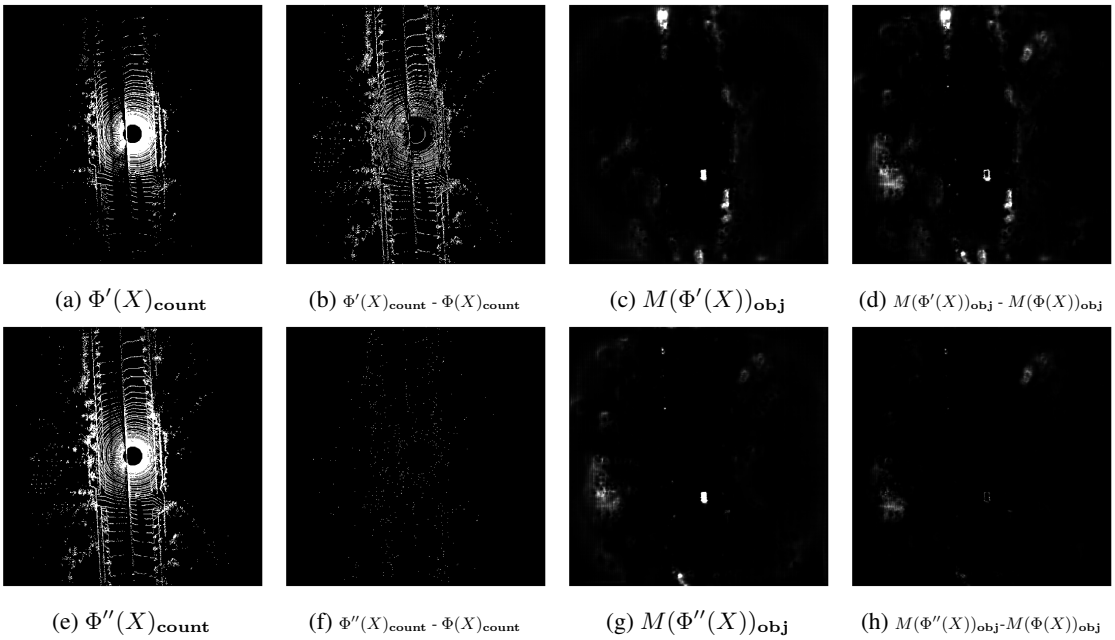
Detailed machine learning model input features and machine learning model output metrics are shown in Table C and Table D.

Table C: Machine learning model input features extracted in the preprocessing phase.

Feature	Description
Max height	Maximum height of points in the cell.
Max intensity	Intensity of the highest point in the cell.
Mean height	Mean height of points in the cell.
Mean intensity	Mean intensity of points in the cell.
Count	Number of points in the cell.
Direction	Angle of the cell's center with respect to the origin.
Distance	Distance between the cell's center and the origin.
Non-empty	Binary value indicating whether the cell is empty or occupied.

Table D: Output metrics of the segmentation model.

Metric	Description
Center offset (off)	Offset to predicted center of the cluster the cell belongs to.
Objectness (obj)	The probability of a cell belonging to an obstacle.
Positiveness (pos)	The confidence score of the detection.
Object height (hei)	The predicted object height.
i**th Class Probability** ($\mathrm{cls_i}$)	The probability of the cell being from class i (vehicle, pedestrian, etc.).



(a) $\Phi'(X)_{\mathbf{count}}$ (b) $\Phi'(X)_{\mathbf{count}}$ - $\Phi(X)_{\mathbf{count}}$ (c) $M(\Phi'(X))_{\mathbf{obj}}$ (d) $M(\Phi'(X))_{\mathbf{obj}}$ - $M(\Phi(X))_{\mathbf{obj}}$

(e) $\Phi''(X)_{\mathbf{count}}$ (f) $\Phi''(X)_{\mathbf{count}}$ - $\Phi(X)_{\mathbf{count}}$ (g) $M(\Phi''(X))_{\mathbf{obj}}$ (h) $M(\Phi''(X))_{\mathbf{obj}}$-$M(\Phi(X))_{\mathbf{obj}}$

Figure F: The performance of trilinear approximator and tanh approximator. The format "$\phi(\cdot)''_{\mathbf{count}}$ represents the 2D count feature calculated by trilinear approximator Φ'; $M(\Phi'(X))_{\mathbf{obj}}$ represents visualization of the "objectiveness " metric in the output of model M using trilinear approximator with; $\Phi'(X)_{\mathbf{count}}$ - $\Phi(X)_{\mathbf{count}}$ represents the approximator 's error of Φ'. The same notation for tanh approximiator Φ''

C Generating Adversarial Object Against LiDAR Perception

C.1 Gradient of proxy functions

Figure F visualizes the improvement of our tanh approximator Φ'' compared to the trilinear approximator Φ' in terms of the **count** feature and the **objectness** metric. Given object S, $\Phi'(X)_{\mathbf{a}}$ represents the aggregated feature a of the point cloud X. $M(\Phi'(X))_{\mathbf{a}}$ represents the model output with respect to metric a. We observe that our approximator Φ' will introduce errors due to our approximation, which will finally leads to model output difference. However, the error of the approximator has been largely decreased by using a more accurate approximator Φ''. This reduces the error in model output, as can be seen in Figure F.

D Additional results

D.1 Changing label

We conduct experiments with 3 pristine meshes (cuba, sphere, tetrahedron) and set the target label to the other 4 labels except for the original label. The results are shown in Table E, showing that our *LiDAR-Adv* has a high chance to trick the detector to output target labels, regardless of different pristine meshes that it starts from.

Table E: The attack success rate of the adversarial objects generating using *LiDAR-Adv*, starting from different types of pristine meshes. The target labels are the other four labels different from the original predictions.

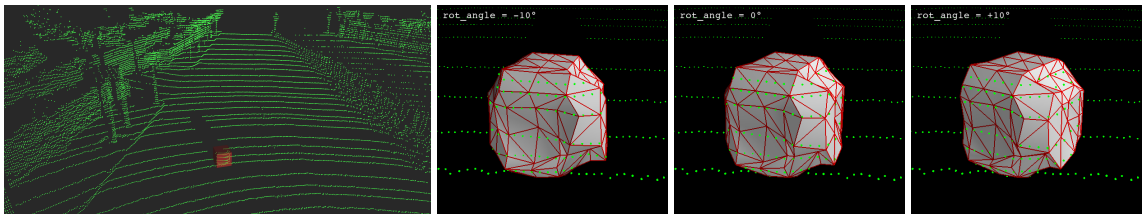
	Cube	Sphere	Tetrahedron	Cylinder	Overall
Attack Success Rate	75%	100%	75%	50%	75%

Table F: Robust Adversarial Object against different angles. The original confidence is x. Our success rate is 100%. (✓represents no object detected)

Angle		-10°	-5°	0°	5°	10°
Objectness (Confid.)	Model	✓	✓	✓	✓	✓
	Apollo	✓	✓	✓	✓	✓

D.1.1 *LiDAR-Adv* on generating robust physical adversarial objects

In this subsection, we add additional results to evaluate the robustness of the generated objects against different positions and different angles. By doing so, it can provide insight on the performance of our adversarial object in real-world settings, before we 3D-print the object.



(a) Benign frame (b) Adv (-10°) (c) Adv (0°) (d) Adv (10°)

Figure G: The visualization of adversarial object with different angles. In the benign frame (a), the system is able to detect the cube. When we replace the cube with our adversarial object, the system fails to detect the object at all three angles. We visualize the mesh along with the point clouds in a close-up view in (b), (c) and (d).

***LiDAR-Adv* against different angles** We generate the adversarial objects by attacking for 9 angles simultaneously and evaluate the attack success rate among these angles. Our approach achieves 100% attack success rate (Table F) both on our approximate model and the Apollo system. This indicates that our designed differentiable proxy functions are accurate enough to transfer the adversarial behavior to Apollo. Figure G shows qualitative results of the adversarial object from different close-up views. We can observe that the adversarial example is smooth and can be easily reconstructed in the real-world.

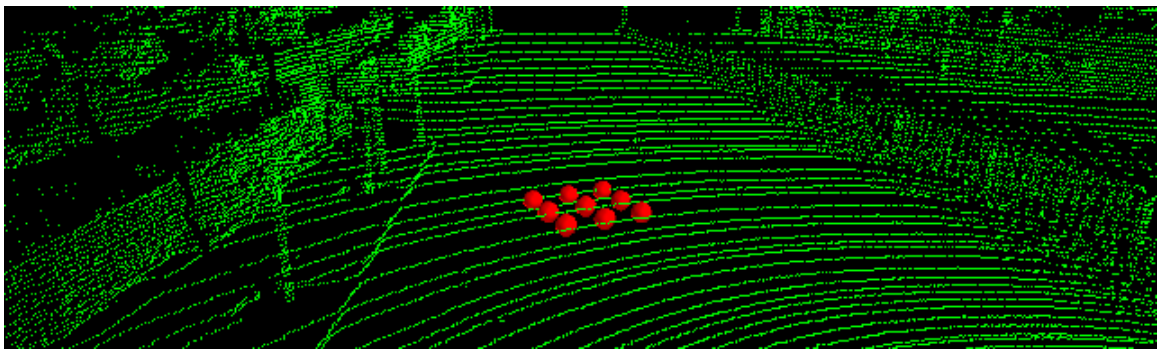


Figure H: Our adversarial object can successfully attack the detection system, while placed at different positions. The red spheres mark the locations we place the adversarial object.

***LiDAR-Adv* against different positions** Similarly, we generate a single robust adversarial object against different positions simultaneously, as is shown in Figure H. We select 9 positions and use our algorithm to generate a universal robust adversarial example against different positions. Figure I shows 7 views of the generated object from different angles, compared to the original object. This adversarial example is smooth from all views. It shows that our approach is able to achieve the goal while keeping the shape plausible, so we can easily print the obect to perform physical attack. Table G show the detailed results of our adversarial object against these 9 positions: it can successfully attack the system among these 9 positions.

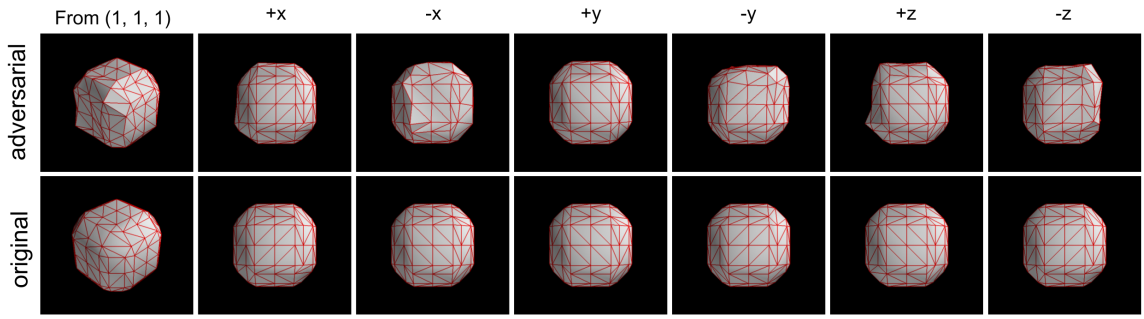


Figure I: The optimized robust adversarial objects from 6 principal views and a particular view, compared with the original pristine object.

Table G: Robust Adversarial Object against different positions. The original object can be detected by Apollo. Our success rate is 100%. (✓represents no object detected)

Position	Objectness (Confid.) Ours	Apollo	Position	Objectness (Confid.) Ours	Apollo	Position	Objectness (Confid.) Ours	Apollo
(-50, -50)	✓	✓	(0, -50)	✓	✓	(50, -50)	✓	✓
(-50, 0)	✓	✓	(0, 0)	✓	✓	(50, -50)	✓	✓
(-50, 50)	✓	✓	(0, 50)	✓	✓	(50, 50)	✓	✓

D.2 Physical experiments

We 3D-print our robust adversarial object at 1:1, and drive a real car mounted with LiDAR and dashcams. The adversarial object is put on the road, and a car drives by, collecting scanned point clouds and the reference dashcam videos. For comparison, we also put the benign object, which is a box of same size at the same location and follow the same protocol when collecting the point clouds.

(a) the road where we perform the physical experiment

(b) the benign object for comparison

(c) the car used to collect the dashcam videos and the point clouds

(d) our adversarial object

Figure J: Our physical experiment setting. We 3D-print the generated adversarial object at 1:1, and drive a car mounted with LiDAR and dashcams to collect the scanned point clouds and the reference videos.