Detecting Perception-based Attacks using Visual Odometry: Inconsistency Modeling and Checking on Robotic States

Yuan Xu[1], Gelei Deng[2], and Tianwei Zhang[3]

***Abstract*— Perception systems in robotic vehicles are crucial for safe and efficient operation, providing key state estimates necessary for planning and control. However, these systems are increasingly vulnerable to perception-based attacks, such as odometry spoofing, position spoofing, obstacle hiding, and object misclassification, which can lead to catastrophic failures. In this paper, we propose a novel approach to detect perception-based attacks by modeling inconsistencies between the physical and estimated states of the robot. Our approach offers a unified methodology for detecting different types of attacks with high accuracy and minimal computational overhead. We validate our method through extensive simulations and real-world scenarios, achieving a 99.5% success rate in detecting attacks, while maintaining a low latency (within 100ms).**

I. INTRODUCTION

Perception is a fundamental component in robots, as it provides the essential state estimates upon which planning and control decisions are made. Despite its importance, perception in robotic systems remains highly susceptible to various forms of attacks. Researchers have demonstrated that nearly all sensors used in perception can be deceived under certain conditions [1]. For instance, IMUs can be misled by acoustic resonance attacks, causing them to register completely incorrect angular velocities [2]–[4]. Similarly, perception functions based on deep learning are vulnerable to adversarial attacks, which can cause the system to fail in detecting an existing obstacle [5]–[8]. These erroneous state estimates could be propagated through the planning and control modules, potentially leading to catastrophic safety failures. Thus, ensuring the security of the perception module has become an urgent and critical task.

While these attacks can successfully deceive the perception module, they struggle to maintain consistency between the real-world physical state and estimated state within the robot. As illustrated in Figure 1, the physical space represents continuous states in the real world, while the estimated space reflects discrete states estimated by the perception module. Under normal scenarios, the perception module accurately estimates the robot's state x from the physical state p, ensuring consistency between the two (as indicated by the matching colors). However, under an attack scenario, the

*This work is supported by the National Research Foundation, Singapore and DSO National Laboratories under its AI Singapore Programme (AISG Award No: AISG2-GC-2023-008)

[1]Yuan Xu is with College of Computing and Data Science, Nanyang Technological University, Singapore xu.yaun@ntu.edu.sg

[2]Gelei Deng is with College of Computing and Data Science, Nanyang Technological University, Singapore gdeng003@e.ntu.edu.sg

[3]Tianwei Zhang is with the College of Computing and Data Science, Nanyang Technological University, Singapore tianwei.zhang@ntu.edu.sg

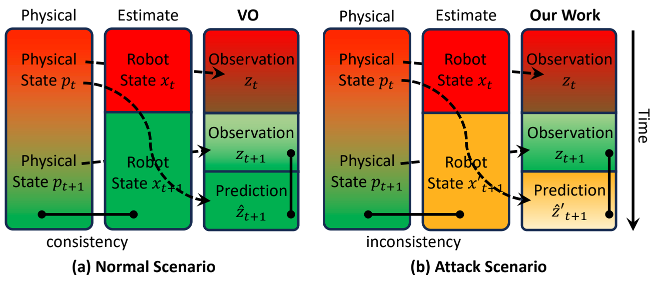


Fig. 1. Consistencies between physical and estimate spaces.

physical state p might be falsely estimated as a completely different robot state x'. Then, the consistency between the physical and estimated states is disrupted (indicated by the different colors). Despite this inconsistency, the robotic system may fail to detect the attack because both the transition of the robot state from x_t to x_{t+1}, and from x_t to x'_{t+1} are within the discrete state space, and appear valid to the system. This raises a critical challenge: *how can we quantify the inconsistency between the physical and robotic states to detect perception-level attacks?*

We propose a novel methodology to detect perception-level attacks against robotic systems by quantifying the inconsistencies between the physical environment and robot's estimated states. Our solution is inspired by PhyScout [9], a framework for identifying sensor spoofing attacks against autonomous driving. Building upon it, we provide a theoretical foundation that models these inconsistencies, enabling our methodology to extend from autonomous driving to a wider range of robotic systems. Our approach is based on principles from Visual Odometry (VO), a widely used ego-motion estimation technique in robotics that extracts features and matches them between adjacent image frames to obtain observations z and corresponding predicted values $\hat{z}$. The observation z reflects the current physical state p of the real world, while the predicted value $\hat{z}_{t+1}$ is derived from the robot state estimation transformation (from x_t to x_{t+1}) through reprojection. The mean error between these matching pairs can indicate the consistency between the physical state and the robot's estimated state.

By quantifying these inconsistencies, we develop a robust methodology capable of detecting perception-level attacks in real time, thereby enhancing the security and reliability of robotic systems. It covers different threats in a unified manner, including odometry spoofing, position spoofing, obstacle hiding and object misclassification attacks. We conduct extensive experiments on datasets and simulators across different robot types. The results demonstrate that our inconsistency detection method effectively identifies all types of attacks with a success rate of up to 99.5%.

II. Background and Related Work

A. Perception-level Attack

In this paper, we focus on four mainstream perception-level attack goals.

Odometry Spoofing. The objective of odometry spoofing is to manipulate the angular velocity data, causing the robot to lose balance and potentially overturn. Robots rely on dynamic adjustments to maintain stability based on IMU sensor data. Adversaries can exploit this by injecting false IMU data through spoofing, destabilizing the robot. For instance, an attacker can trigger a crash by manipulating IMU data, leading the robot to incorrectly adjust its position [3], [4], [10], [11].

Position Spoofing. Position spoofing involves deceiving the robot's GPS to induce incorrect positioning and trigger unintended actions. By generating false GPS signals, the attacker misleads the robot's navigation system. A notable example is a drone being forced to land in a restricted no-fly zone due to GPS spoofing [12]–[14].

Obstacle Hiding. This attack tricks the robot into failing to detect an existing obstacle. Adversaries may use small adversarial patches or spoofed laser points to deceive perception systems relying on LiDAR or cameras. For example, drones positioned at strategic points can generate false laser data to interfere with the robot's perception, causing it to overlook obstacles [5], [8], [15]–[22].

Object Misclassification. The goal of object misclassification is to cause a robot to misidentify traffic signs or signals, leading to dangerous decisions. By projecting phantom images or attaching small stickers to signs, attackers can fool deep learning models into misclassifying objects. For example, research has demonstrated how stop signs can be misclassified as other objects, resulting in unsafe robot behavior [5], [21], [23]–[31].

B. Attack Detection Methods

Certified Defenses [32]–[38]: These approaches typically rely on *prediction consistency* within a strict perturbation scale to detect spoofing activities. The common strategy is to either smooth classification boundaries or relax the model's intermediate states to reduce sensitivity to minor input variations. During model prediction, multiple input replicas are often needed, posing concerns about their feasibility in real-time applications. Furthermore, adaptive attacks can still exploit certain gaps, as the defense does not guarantee a 100% success rate within its perturbation bounds.

Vision/Lidar consistency methods [8-14]: SCEME [39] and SCENE [40] utilize transformer models to verify the *contextual consistency* between objects. For instance, a scene containing both a toothbrush and a bus would be identified as inconsistent. However, these models also have several issues. For example, by misclassifying a traffic light into a traffic sign, they cannot identify such attack [40]. KEMLP [41] focuses on *semantic consistency*, employing a CNN to deduce object-specific rules, like recognizing a stop sign's octagonal shape. AdvIT [42] generates pseudo frames from historical data to assess *temporal consistency* between the pseudo and target frames. PercepGuard [43] merges the principles of KEMLP and AdvIT, using an RNN to check *spatio-temporal consistency* from both current and historical frames. LOP [44] and Zhang et al [45] are most aligned with ours. They use depth data from camera or lidar to produce a depth map for *spatial inconsistency* detection using CNN/GNN models. Despite their innovation, these defenses predominantly rely on DNN models, which pose two primary issues: *heavy reliance on GPU resources* and *vulnerabilities to adaptive attacks* [46]–[48].

In contrast, our approach introduces a novel detection methods using inconsistency modeling. This not only offers *a holistic defense against spoofing attacks with a unified methodology* but also *significantly increase the attack difficulty*, which now need to manipulate the majority of distributed keypoints. Meanwhile, such the non-DNN-model-based methodology can achieve *low detection latency ((<100ms))* with *low performance overhead (CPU friendly)*.

C. Visual Odometry

Visual Odometry (VO) has been a cornerstone technique in robotics for estimating ego-motion by analyzing sequential images captured by a robot's camera. The fundamental idea behind VO is to track the movement of features in the environment across successive frames, which allows the estimation of the robot's trajectory relative to its initial position. Early works in VO, such as those by Nistér et al. [49] and Scaramuzza et al. [50], laid the foundation by focusing on geometric methods to solve the motion estimation problem using point correspondences between images.

Over time, VO has evolved significantly, incorporating advancements in feature extraction, matching, and optimization techniques. For instance, modern VO systems like ORB-SLAM [51] utilize robust feature descriptors and sophisticated optimization frameworks to achieve high accuracy and robustness in diverse environments. Furthermore, research has expanded VO's applications beyond pure ego-motion estimation, integrating it with simultaneous localization and mapping (SLAM) to build dense maps of the environment while tracking motion [52]. Unlike traditional VO researches, our work proposes extending VO's framework to detect inconsistencies introduced by perception-level attacks, thereby enhancing the resilience of robots against such threats.

III. Inconsistency Modeling

In this section, we present a model for detecting perception-level attacks by quantifying inconsistencies between the physical and robotic states. Given the diverse targets of various perception-level attacks, we divide our modeling approach into two categories: global inconsistency modeling and local inconsistency modeling. As illustrated in Figure 2, global inconsistency is modeled based on each keyframe received by the camera, where all the feature points (green dots) within that keyframe are considered for analysis.

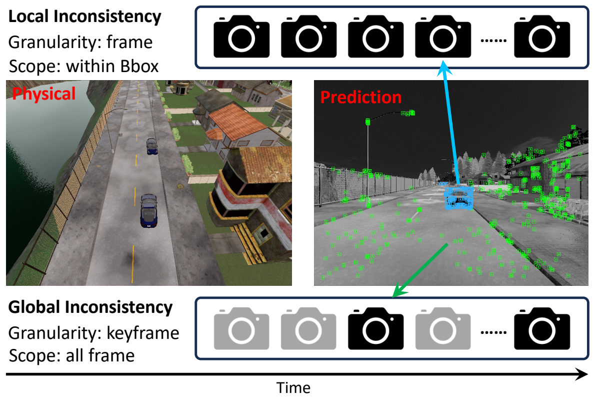


Fig. 2. Global and local consistency modeling.

On the other hand, local inconsistency is modeled on a per-frame basis, where the model processes only the feature points (blue dots) within the detected bounding boxes (blue rectangles) of each frame.

A. Inconsistency Analysis

Inconsistency arises from the mismatch between the observation z_t at the current time step and the predicted value $\hat{z}_t$ based on an incorrect state transition $(x_{t-1} \to x_t)$, as discussed in Section I, . Thus, we can analyze this inconsistency using the following equation:

$$e_t = \Theta(\hat{z}_t, z_t)$$
$$\text{w.r.t.} \begin{cases} \hat{z}_t = \Phi(x_{t-1}, x_t, z_{t-1}) \\ z_t = \rho(p_t) \end{cases} \tag{1}$$

Here, e_t represents the error used to quantify the inconsistency. The associated functions, Θ, Φ, ρ, denote the inconsistency estimation, feature prediction, and feature extraction processes, respectively. In the following subsections, we will detail how these functions are modeled in both global and local inconsistency scenarios.

B. Global Inconsistency Modeling and Implementation

We begin by formulating the concept of global inconsistency. Following the principles of visual odometry (VO) and the SLAM framework, our model incorporates the following key components:

Feature Extraction (Θ). The goal of this process is to identify 2D features corresponding to the 3D physical world, mathematically expressed as: $\mathbb{R}^{n*3} \to \mathbb{R}^{n*2}$. Features consist of a set of stable 2D keypoints, typically located in image regions such as corners, edges, and blocks. Let F_t represent the frame captured by the camera at time t (where $t \in [0, T]$), and let k_t denote the set of 2D keypoints extracted from this frame. The feature extraction process can thus be described by the following equations:

$$k_t = \Theta(F_t), t \in [0, T]$$
$$k = (k_1, k_2, \ldots, k_n), k_i = (k_i^x, k_i^y), \tag{2}$$

There are several methods to implement the function Θ for feature extraction from images. One of the most commonly used algorithms is the Features from Accelerated Segment Test (FAST) algorithm [53]. In FAST, a pixel p is selected as the center, and 16 surrounding pixels on a circular pattern, with a radius of 3 pixels, are analyzed. If the intensity difference between p and any of these surrounding pixels exceeds a predefined threshold (e.g., 20%) for N consecutive pixels, the pixel p is identified as a keypoint and is selected as a feature.

Feature Prediction (Φ). This process aims to predict the current state value $\hat{z}_t$ based on the past observation $z_{0\ldots t-1}$ and the pose transition from x_{t-1} to x_t. Since the extracted 2D feature points k do not contain depth information, the first step is to reproject these 2D points into 3D map points M, based on both the current and previous observations.

Inspired by local mapping module in ORB-SLAM, we use a short temporal window to triangulate the 2D keypoints from multiple past consecutive keyframes and compute their 3D coordinates. Unlike the global map maintained in ORB-SLAM, our approach focuses on constructing and updating a localized map over a limited time period, reducing the computational overhead while still capturing relevant spatial information. The reprojection function Γ is used to transform the 2D keypoints and camera poses into 3D map points. This process is formalized as:

$$M_t = \Gamma(x_{0\ldots t-1}, k_{0\ldots t-1}, k_t), t \in [0, T]$$
$$M = (M_1, M_2, \ldots, M_n), M_i = (M_i^x, M_i^y, M_i^z) \tag{3}$$

Here, M_t represents the set of 3D map points computed at time t by reprojection. The function Γ takes as input the camera poses $x_{0\ldots t-1}$ and 2D keypoints $k_{0\ldots t-1}$ from previous keyframes, as well as the current set of keypoints k_t, to generate the 3D coordinates of the map points.

The 3D coordinates of these map points are then transformed to the next time step using the following equation:

$$[M_{t+1}, 1]' = T_t \cdot [M_t, 1]'$$
$$\text{w.r.t.} \quad T = \begin{bmatrix} R & r \\ 0' & 1 \end{bmatrix} \tag{4}$$

Here, T_t is a 4×4 transformation matrix that represents the transformation from M_t to M_{t+1}. Specifically, the transformation matrix T_t consists of a 3×3 rotation matrix R and a 3×1 translation matrix r. The transformation matrix T_t can be approximated using a constant velocity motion model based on T_{t-1}, which can be obtained from the robot's current odometry system.

Once the 3D map points M_{t+1} have been computed, the corresponding 2D coordinates in the current camera frame can be estimated using the following projection equation:

$$\hat{z}_{t+1} = (m_i^x, m_i^y, 1)' = I_C \cdot (\frac{M_i^x}{M_i^z}, \frac{M_i^y}{M_i^z}, 1)'$$
$$\text{w.r.t.} \quad I_C = \begin{bmatrix} f_x & 0 & c_x \\ 0' & f_y & c_y \\ 0' & 0 & 1 \end{bmatrix}, \quad \begin{matrix} m = (m_1, m_2, \ldots, m_n) \\ m_i = (m_i^x, m_i^y), \end{matrix} \tag{5}$$

In this equation, (m_i^x, m_i^y) are the 2D coordinates of the projected map point $\hat{z}_{t+1}$ in the current frame, and I_C is the camera intrinsic matrix. The matrix contains the focal lengths f_x and f_y along the x- and y-axes, respectively, as well as the optical center offsets c_x and c_y. The 3D map points M_i are projected into the image plane to obtain their 2D coordinates.

Inconsistency Estimation (ρ). This process quantifies the inconsistency between the observed extracted 2D feature points z and predicted 2D map points $\hat{z}$. At time $t+1$, let the Feature Extraction process Θ extract m feature points from the current frame, denoted as z_{t+1}. Simultaneously, through the Feature Prediction process Φ, we obtain n projected 2D map points $\hat{z}_{t+1}$. Since the number of predicted map points n is typically greater than the number of observed feature points m, we need to match each projected map point $\hat{z}_i$ to the nearest observed feature point z_i. This matching is performed by the following function Ψ. Once the matching pairs are established, we calculate the average inconsistency between them.

$$\begin{aligned}&(z_1, z_2, \ldots, z_n)_{t+1} = \Psi((\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_n)_{t+1}, (z_1, z_2, \ldots, z_m)_{t+1})\\ &e^G_{t+1} = \Theta((\hat{z}_1, z_1), (\hat{z}_2, z_2), \ldots, (\hat{z}_n, z_n))_{t+1}\end{aligned} \tag{6}$$

In this context, the function Ψ matches each predicted 2D map point $\hat{z}_i$ with its closest corresponding feature point z_i with similar multi-scale pyramids level in the observation, based on their proximity. The matching process ensures that the numbers of $\hat{z}_{t+1}$ and z_{t+1} are same. The function Θ, inspired by ORB-SLAM, utilizes a nonlinear optimization approach to further refine the robot's pose by minimizing the overall reprojection error. Specifically, for each pair of matched points $(\hat{z}_i, z_i)$, Θ applies the Levenberg-Marquardt optimization algorithm, using a least squares method to adjust the robot's pose and minimize the reprojection error between the predicted and observed 2D points. This optimization ultimately yields an adjusted pose with minimized average error across all point pairs. To implement Θ, we use the graph-based nonlinear optimizer g2o [54], which applies the Levenberg-Marquardt algorithm for optimization. This tool allows us to iteratively adjust the robot's pose until the reprojection error is minimized, thereby providing an optimal estimation of the robot's state at time $t+1$.

Global Inconsistency Checking. The primary objective of global inconsistency modeling is to detect odometry spoofing and position spoofing attacks. Specifically, odometry spoofing attack can cause the robot to miscalculate its odometry, leading to errors in the transformation matrix $T`_t$ in (4). These errors propagate through the system, resulting in inaccuracies in the predicted 3D map points M'_{t+1} and their projected 2D coordinates $\hat{z'}_{t+1}$ in (5). When we optimize a set of incorrect matching pairs $(\hat{z'}_i, z_i)$, it typically results in an abnormally high mean error e^G_{t+1}. Thus, we define a threshold T_1, where e^G_{t+1} exceeding this value indicates the presence of an attack.

On the other hand, position spoofing attack primarily affects the robot's position estimation, causing large deviations in its estimated position. However, it has no direct impact on the odometry information, meaning that the error e^G_{t+1} derived from the matching process should remain within the normal range. Thus, the robot's mean error in global consistency checks may appear normal even during a position spoofing attack. To detect such attacks, we instead monitor the abnormality in the squared mean differences in the robot's 3D positional coordinates over consecutive time steps. Large deviations in the position, while maintaining normal odometry error, could indicate position spoofing. Thus, we use two conditions: e^G_{t+1} must remain below threshold T_2, while the positional deviation P_{t+1} must exceed threshold T_3.

Odometry Spoofing: $e^G_{t+1} > T_1$
Position Spoofing: $e^G_{t+1} < T_2$ and $P_{t+1} < T_3$

C. Local Inconsistency Modeling

Local inconsistency modeling primarily addresses state estimation errors caused by object detection modules. These errors may result from attacks such as obstacle appearance, obstacle hiding, or object misclassification. Such attacks affect the robot's perception at the local level, often leading to incorrect decisions related to navigation and obstacle avoidance.

Feature Extraction (Θ). The local feature extraction process builds upon (2) by introducing bounding box constraints. Specifically, we focus on the feature points that fall within the detected bounding boxes for objects in the scene. Let k^L_t represent the set of local feature points at time t, which includes all feature points within the bounding box of a detected object. This can be expressed as:

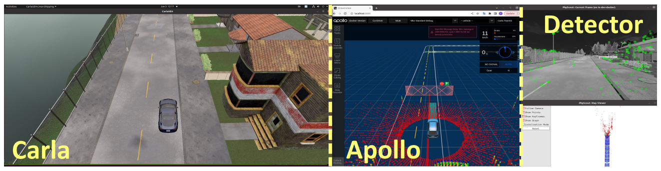
$$k^L_t = \{k_i \,|\, k_i \in k \text{ and } (k^x_i, k^y_i) \in \text{BoundingBox}\} \tag{7}$$

Here, k^L_t represents the subset of feature points k that lie within the bounding box of a detected object. The bounding box serves as a spatial constraint that defines the region of interest for local feature extraction. These local features are critical for detecting inconsistencies at the object level, where perception-level attacks are likely to manifest.

Feature Prediction (Φ). The local feature prediction process is similar to the global feature prediction process, but there are two key differences. First, the reprojection function Γ in (2) is not based on past consecutive keyframes, but rather on frames. This adjustment is necessary because the number of local feature points k^L_t is significantly smaller than the number of global feature points k_t. Thus, the reprojected 3D map point M^L_t using keyframes are limited, increasing the likelihood of errors. To mitigate this issue, we reproject the local feature points k^L_t from two consecutive frames into 3D map points M^L_t.

Second, the transformation matrix T^L_t in (4) for local map points must account for the relative motion between the detected object and the robot itself. This is in contrast to global modeling, where the robot's motion alone is considered. The relative position changes between the moving object and the robot can be captured by the robot's perception module, which integrates data from sensors such as LiDAR, cameras, or radar.

Inconsistency Estimation (ρ). The local inconsistency estimation process follows the same approach as the global inconsistency estimation process. We use (6) to obtain a set of matching pairs $(\hat{z}^L_i, z^L_i)$ within the bounding box. These matching pairs represent the predicted 2D points $\hat{z}^L_i$ from the



Attack Goal	Target Channel	Malicious Data
Odometry Spoofing	/localization/pose	(1, y, z, qx, qy, qz, qw)
Position Spoofing	/localization/pose	(x, y+0.2, z, qx, qy, qz, qw)
Obstacle Hiding	/apollo/perception/obstacles	Yolo4BBoxes(999, 999, z, ...)
Object Misclassification	/apollo/perception/obstacles	Yolo4BBoxes(object class: bird, ...)

Fig. 3. Experiment setup of simulation.

reprojected 3D map points $\hat{z}_{t+1}^L$ and the observed 2D points z_i^L extracted from the current frame. After establishing the matching pairs, we compute the corresponding local mean error e_{t+1}^L, which quantifies the local inconsistency at time $t+1$.

Local Inconsistency Checking. The primary goal of local inconsistency modeling is to detect obstacle hiding and object misclassification attacks. These attacks specifically affect the robot's perception of objects within its environment, leading to potential hazards in decision-making and navigation. Specifically, In an obstacle hiding attack, an obstacle may suddenly disappear from the robot's perception due to an erroneous state estimate. When this occurs, the corresponding bounding box is no longer detected in the camera feed. To address this, we use the local transformation matrix T_t^L to predict the expected position of the bounding box for the next time step. We then compute the local mean error e_{t+1}^L for the predicted bounding box position. If the error e_{t+1}^L below a certain threshold T_4, it indicates that the object should still be present in the scene, but its detection has failed. This discrepancy allows us to detect that an obstacle hiding attack has occurred.

On the other hand, in an object misclassification attack, the semantic label of an object is incorrectly changed, such as mistaking a pedestrian for a signpost. In such cases, the object's visual appearance may remain consistent across consecutive frames, and the local mean error e_{t+1}^L might not increase significantly. However, if we observe that the semantic label of the object has changed while the local mean error e_{t+1}^L remains below the threshold T_4, we can detect that an object misclassification attack has occurred. This combination of low local error and semantic inconsistency indicates a misclassification.

Obstacle Hiding: $e_{t+1}^L < T_4$ and Obstacle Disappears
Object Misclassification: $e_{t+1}^L < T_4$ and Object Changes

All these threshold T_1, T_2, T_3 and T_4 are determined empirically based on system performance and the robot's mechanical parameters. IHowever, based on extensive experiments conducted on different platforms, we have verified that various metrics exhibit significant differences between normal and attack scenarios, making it easy to identify attacks using predefined threshold values. In this paper, we choose 40, 20, 20, and 20 as the default values for these thresholds.

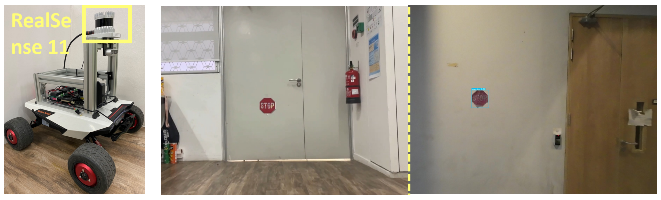


Fig. 4. Experiment setup of real-world scenarios.

IV. Evaluation

A. Experiment Setup

We employ our methodology as an evaluation tool, integrating it with the simulation and real-world scenarios.

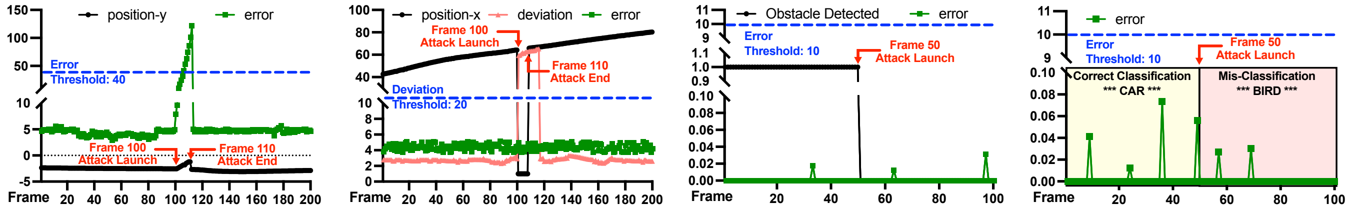
Simulation. As depicted in Figure 3, we use Apollo 7.0 and the Carla simulator, each running in separate Docker environments. Apollo collects environmental and robot data from Carla to plan routes and calculate velocities, while Carla handles the actual robot movement. Our attack detector operates in the same Docker container as Apollo, receiving sensor data such as images and pose information through a bridge using the Apollo Cyber module, which converts and transfers data between components. All simulations are run on a high-performance server with an Intel i9-10900 CPU, 32GB of RAM, and an NVIDIA GeForce RTX 2080Ti GPU.

In our attack model, we assume the attacker has the ability to manipulate the robot's internal state directly. This is achieved by identifying and hijacking target channels in Apollo, redirecting data to malicious nodes. The red dotted box in Figure 3 illustrates how the attacker remaps sensor data channels, injecting malicious data into the system. This approach allows us to simulate robust and stealthy sensor spoofing attacks that are difficult to detect, providing a thorough test of our detection system's robustness. Additionally, this method gives us flexibility to simulate various attack scenarios at different times and locations, allowing for comprehensive evaluation under diverse conditions.

Real-World Scenarios. As illustrated in Figure 4, we utilize a physical unmanned ground vehicle (UGV) equipped with an Intel RealSense D435i front-facing camera, capturing 1080p images at 30fps, and a Bosch BMI055 6-axis IMU integrated into the onboard system. In our real-world experiments, we simulate both obstacle hiding and object misclassification attacks using adversarial patches applied to stop signs. For the obstacle hiding attack, the patch was placed on an indoor door, and the UGV was driven towards it to evaluate how well the system detected and reacted to the hidden obstacle. The object misclassification attack involved attaching the adversarial patch directly to a stop sign, causing the system to misidentify the object.

B. Evaluation on Simulation

Odometry Spoofing: In this attack, we incrementally added an offset (+0.2) to the robot's y–coordinate starting from the 100th frame, while leaving the x and z coordinates unchanged. As shown in Figure 5(a), this caused the AV to miscalculate its position and veer to the right to correct its lane, eventually colliding with a roadside obstacle. The



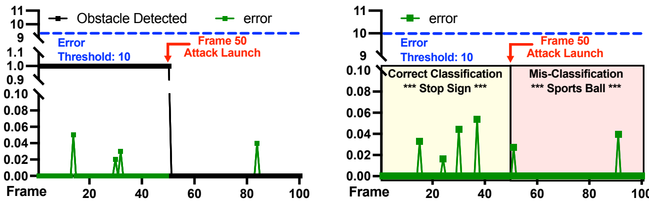
(a) Odometry Spoofing (b) Position Spoofing (c) Obstacle Hiding (d) Object Misclassification

Fig. 5. Experiment result of four attack goals in simulation.

TABLE I

SUCCESS RATE AND LATENCY COMPARISON.

Method	Success Rate		Latency (ms)			
	Without Foggy	Foggy	PS	OS	OH	OM
DetectorGuard [38]	74% (OH)	0%	✗	✗	119	✗
PerceptGuard [43]	99% (OM)	0%	✗	✗	✗	435
Our Detector	99.5% (Total)	0%	82	75	64	66



(a) Obstacle Hiding (b) Object Misclassification

Fig. 6. Experiment result of patched-based attack goals in real world.

y-coordinate was restored at the 110th frame. The corresponding optimization error surged to 132, far exceeding the threshold of 40, making the attack easily detectable.

Position Spoofing: For the position spoofing attack, the robot's x-coordinate was altered from its original value to 1 over ten frames, starting from frame 100. The y and z-coordinates remained unchanged. This manipulation resulted in a significant pose deviation, as shown by the red triangles in Figure 5(b), exceeding the threshold of 20. Despite the attack, the optimization error remained normal, as the spatial continuity of the environment was maintained.

Obstacle Hiding: We launched an obstacle hiding attack by injecting malicious data that shifted the position of a target robot, effectively making the bounding box disappear. Figure 5(c) demonstrates how the predicted box is used to extract keypoints for attack detection. The robot fails to detect the obstacle and continues moving, resulting in a collision. The error value remained below the threshold of 10, which allowed us to detect the attack in real-time.

Target Semantic Confusion: In this attack, the classification of a vehicle in front of the robot was maliciously changed from a car to a bird. As shown in Figure 5(d), the misclassification led the robot to ignore the forward vehicle and continue moving, eventually causing a collision. Despite the object's misclassification, the error between the projected 3D map points and extracted 2D keypoints remained low, as the object's spatial consistency was maintained. This behavior can be used to detect object misclassification attacks.

False Positives Analysis We evaluated the false positives of our detector under different environmental conditions (sunny, rainy night, light, and heavy fog) using the Carla platform. For each spoofing attack and scenario, we conducted ten simulations with varied starting positions and obstacles. We recorded the attack detection success rate and the delay between the attack initiation and successful detection. Our detector was compared against two state-of-the-art defense frameworks: DetectorGuard [38] and PercepGuard [43], which use certified defense and vision-based consistency checking, respectively.

Table I presents the results. Our detector outperformed the others in several key areas: (i) It achieved a 99.5% success rate for detecting all spoofing attacks, while the other methods showed lower success rates, particularly in foggy scenarios. False positives were observed in highway settings due to the reduced number of stable map points when the vehicle's speed increased, causing mismatches and triggering false alarms in global shifting attacks. However, all defense frameworks, including ours, failed under foggy conditions, primarily due to the limitations of YOLOv4 and the impaired performance of ORB-based feature extraction in such environments. This led to flickering bounding boxes and false detections. We believe future improvements in object detection algorithms and the integration of multi-modal techniques [55]–[57] will enhance robustness and reduce false positives. (ii) Our detector, with minimal computational overhead, can detect attacks within 100ms, providing a critical safety margin for the victim vehicle.

C. Case Study on Physical Scenarios

Figure 6 illustrates the variation of errors in our detector under different attack scenarios, along with the corresponding detection results. We can observe that even in different attack scenarios conducted in real-world environments, our proposed error metric demonstrates strong robustness. The triggers in the real-world setting did not significantly impact the detection outcomes of our detector. The rationale behind this observation is the invariance of spatio-temporal consistency in the trigger-based scenario. Since the trigger is deployed in the physical environment, it can be considered as a single entity with the environment.

V. CONCLUSION

In this paper, we introduced a novel detection mechanism for perception-based attacks using inconsistency modeling with the visual odometry. By quantifying both global and local inconsistencies, our method successfully detects various attacks, including odometry and position spoofing, obstacle hiding, and object misclassification. Extensive evaluations conducted in both simulated and real-world environments demonstrated the effectiveness of our approach, achieving a high detection success rate and low latency.

References

[1] Y. Xu, X. Han, G. Deng, G. Li, Y. Liu, J. Li, and T. Zhang, “Sok: Rethinking sensor spoofing attacks against robotic vehicles from a systematic view,” in *EuroSP*, 2023.

[2] Y. Son, H. Shin, D. Kim, Y.-S. Park, J. Noh, K. Choi, J. Choi, and Y. Kim, “Rocking drones with intentional sound noise on gyroscopic sensors,” in *USENIX Security Symposium*, 2015.

[3] T. Trippel, O. Weisse, W. Xu, P. Honeyman, and K. Fu, “Walnut: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks,” in *EuroSP*, 2017.

[4] Y. Tu, Z. Lin, I. Lee, and X. Hei, “Injected and delivered: Fabricating implicit control over actuation systems by spoofing inertial sensors,” in *USENIX Security Symposium*, 2018.

[5] Y. Zhao, H. Zhu, R. Liang, Q. Shen, S. Zhang, and K. Chen, “Seeing isn’t believing: Towards more robust adversarial attack against real world object detectors,” in *CCS*, 2019.

[6] Y. Jia, Y. Lu, J. Shen, Q. A. Chen, H. Chen, Z. Zhong, and T. Wei, “Fooling detection alone is not enough: Adversarial attack against multiple object tracking,” in *International Conference on Learning Representations*, 2020.

[7] P. Jing, Q. Tang, Y. Du, L. Xue, X. Luo, T. Wang, S. Nie, and S. Wu, “Too good to be safe: Tricking lane detection in autonomous driving with crafted perturbations,” in *USENIX Security Symposium*, 2021.

[8] Y. Cao, N. Wang, C. Xiao, D. Yang, J. Fang, R. Yang, Q. A. Chen, M. Liu, and B. Li, “Invisible for both camera and lidar: Security of multi-sensor fusion based perception in autonomous driving under physical-world attacks,” in *IEEE S&P*, 2021.

[9] Y. Xu, G. Deng, X. Han, G. Li, H. Qiu, and T. Zhang, “Physcout: Detecting sensor spoofing attacks via spatio-temporal consistency,” in *CCS*, 2024.

[10] Z. Wang, K. Wang, B. yang, S. Li, and A. Pan, “Sonic gun to smart devices,” in *Black Hat USA*, 2017.

[11] S. Nashimoto, D. Suzuki, T. Sugawara, and K. Sakiyama, “Sensor con-fusion: Defeating kalman filter in signal injection attack,” in *ACM AsiaCCS*, 2018.

[12] V. Dey, V. Pudi, A. Chattopadhyay, and Y. Elovici, “Security vulnerabilities of unmanned aerial vehicles and countermeasures: An experimental study,” in *International Conference on VLSI Design*, 2018.

[13] D. He, H. Liu, S. Chan, and M. Guizani, “How to govern the non-cooperative amateur drones?” *IEEE Network*, 2019.

[14] D. He, Y. Qiao, S. Chen, X. Du, W. Chen, S. Zhu, and M. Guizan, “A friendly and low-cost technique for capturing non-cooperative civilian unmanned aerial vehicles,” *IEEE Network*, 2019.

[15] Z. Jin, X. Ji, Y. Cheng, B. Yang, C. Yan, and W. Xu, “Pla-lidar: Physical laser attacks against lidar-based 3d object detection in autonomous vehicle,” in *IEEE Symposium on Security and Privacy*, 2023.

[16] Y. Zhu, C. Miao, T. Zheng, F. Hajiaghajani, L. Su, and C. Qiao, “Can we use arbitrary objects to attack lidar perception in autonomous driving?” in *ACM CCS*, 2021.

[17] Z. Cheng, J. Liang, H. Choi, G. Tao, Z. Cao, D. Liu, and X. Zhang, “Physical attack on monocular depth estimation with optimal adversarial patche,” in *ECCV*, 2022.

[18] L. Huang, C. Gao, Y. Zhou, C. Xie, A. L. Yuille, C. Zou, and N. Liu, “Universal physical camouflage attacks on object detectors,” in *CVPR*, 2020.

[19] Z. Wu, S.-N. Lim, L. S. Davis, and T. Goldstein, “Making an invisibility cloak: Real world adversarial attacks on object detectors,” in *European Conference on Computer Vision*, 2020.

[20] K. Xu, G. Zhang, S. Liu, Q. Fan, M. Sun, H. Chen, P.-Y. Chen, Y. Wang, and X. Lin, “Adversarial t-shirt! evading person detectors in a physical world,” in *ECCV*, 2020.

[21] Y. Man and M. Li, “Ghostimage: Remote perception attacks against camera-based image classification systems,” in *International Symposium on Recent Advances in Intrusion Detection*, 2020.

[22] G. Lovisotto, H. Turner, I. Sluganovic, M. Strohmeier, and I. Martinovic, “Slap: Improving physical adversarial examples with short-lived adversarial perturbations,” in *USENIX Security Symposium*, 2021.

[23] K. Eykholt, I. Evtimov, E. Fernandes, A. R. Bo Li, C. Xiao, A. Prakash, T. Kohno, and D. Song, “Robust physical-world attacks on deep learning visual classification,” in *CVPR*, 2018.

[24] D. Song, K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, F. Tramèr, A. Prakash, and T. Kohno, “Physical adversarial examples for object detectors,” in *Workshop on Offensive Technologies*, 2018.

[25] Z. Kong, J. Guo, A. Li, and C. Liu, “Physgan: Generating physical-world-resilient adversarial examples for autonomous driving,” in *CVPR*, 2020.

[26] J. Wang, A. Liu, Z. Yin, S. Liu, S. Tang, and X. Liu, “Dual attention suppression attack: Generate adversarial camouflage in physical world,” in *CVPR*, 2021.

[27] C. Yan, Z. Xu, Z. Yin, X. Ji, and W. Xu, “Rolling colors: Adversarial laser exploits against traffic light recognition,” *CoRR*, 2022.

[28] B. Nassi, Y. Mirsky, D. Nassi, R. Ben-Netanel, O. Drokin, and Y. Elovici, “Phantom of the adas: Securing advanced driver-assistance systems from split-second phantom attacks,” in *ACM CCS*, 2020.

[29] B. Nassi, D. Nassi, R. Ben-Netanel, Y. Mirsky, O. Drokin, and Y. Elovici, “Phantom of the adas: Phantom attacks on driver-assistance systems,” *IACR Cryptology ePrint Archive*, 2020.

[30] R. Duan, X. Mao, A. K. Qin, Y. Chen, S. Ye, Y. He, and Y. Yang, “Adversarial laser beam: Effective physical-world attack to dnns in a blink,” in *CVPR*, 2021.

[31] W. Wang, Y. Yao, X. Liu, X. Li, P. Hao, and T. Zhu, “I can see the light: Attacks on autonomous vehicles using invisible lights,” in *ACM CCS*, 2021.

[32] W.-Y. Lin, F. Sheikholeslami, J. Shi, L. Rice, and J. Z. Kolter, “Certified robustness against physically-realizable patch attack via randomized cropping,” *ICLR Open Review*, 2021.

[33] P. yeh Chiang, R. Ni, A. Abdelkader, C. Zhu, C. Studer, and T. Goldstein, “Certified defenses for adversarial patches,” in *ICLR*, 2020.

[34] A. Levine and S. Feizi, “(de)randomized smoothing for certifiable defense against patch attacks,” in *NeurIPS 2020*.

[35] C. Xiang, A. N. Bhagoji, V. Sehwag, and P. Mittal, “Patchguard: A provably robust defense against adversarial patches via small receptive fields and masking,” in *USENIX Security Symposium*, 2021.

[36] C. Xiang and P. Mittal, “Patchguard++: Efficient provable attack detection against adversarial patches,” in *CoRR abs/2104.12609*, 2021.

[37] J. H. Metzen and M. Yatsura, “Efficient certified defenses against patch attacks on image classifiers,” in *ICLR*, 2021.

[38] C. Xiang and P. Mittal, “Detectorguard: Provably securing object detectors against localized patch hiding attacks,” in *ACM CCS*, 2021.

[39] S. Li, S. Zhu, S. Paul, A. K. Roy-Chowdhury, C. Song, S. V. Krishnamurthy, A. Swami, and K. S. Chan, “Connecting the dots: Detecting adversarial perturbations using context inconsistency,” in *ECCV*, 2020.

[40] M. Yin, S. Li, Z. Cai, M. S. A. Chengyu Song, A. K. Roy-Chowdhury, and S. V. Krishnamurthy, “Exploiting multi-object relationships for detecting adversarial attacks in complex scenes,” in *IEEE ICCV*, 2021.

[41] N. M. Gürel, X. Qi, L. Rimanic, C. Zhang, and B. Li, “Knowledge enhanced machine learning pipeline against diverse adversarial attacks,” in *ICML*, 2021.

[42] C. Xiao, R. Deng, B. Li, T. Lee, B. Edwards, J. Yi, D. Song, M. Liu, and I. M. Molloy, “Advit: Adversarial frames identifier based on temporal consistency in videos,” in *IEEE ICCV*, 2019.

[43] Y. Man, R. Muller, M. Li, Z. B. Celik, and R. Gerdes, “That person moves like a car: Misclassification attack detection for autonomous systems using spatiotemporal consistency,” in *USENIX Security Symposium*, 2023.

[44] Q. Xiao, X. Pan, Y. Lu, M. Zhang, J. Dai, , M. Yang, and F. University, “Exorcising wraith: Protecting lidar-based object detector in automated driving system from appearing attacks,” in *USENIX Security Symposium*, 2023.

[45] J. Zhang, Y. Zhang, K. Lu, J. Wang, K. Wu, X. Jia, and B. Liu, “Detecting and identifying optical signal attacks on autonomous driving systems,” *IEEE Internet Things*, 2021.

[46] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, “The limitations of deep learning in adversarial settings,” in *EuroSP*, 2016.

[47] N. Carlini and D. A. Wagner, “Adversarial examples are not easily detected: Bypassing ten detection method,” in *AISec@CCS*, 2017.

[48] ——, “Towards evaluating the robustness of neural networks,” in *IEEE Symposium on Security and Privacy*, 2017.

[49] D. Nistér, O. Naroditsky, and J. R. Bergen, “Visual odometry,” in *CVPR*, 2004.

[50] D. Scaramuzza and F. Fraundorfer, “Visual odometry [tutorial],” *IEEE Robotics & Automation Magazine*, 2011.

[51] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, “Orb-slam: A versatile and accurate monocular slam system,” *IEEE Trans. Robotics*, 2015.

[52] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *ECCV*, 2014.
[53] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European Conference on Computer Vision*, 2006.
[54] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *IEEE ICRA*, 2011.
[55] Y. Zhang, J. Chen, and D. Huang, "Cat-det: Contrastively augmented transformer for multi-modal 3d object detection," in *CVPR*, 2022.
[56] Q. Cai, Y. Pan, T. Yao, C.-W. Ngo, and T. Mei, "Objectfusion: Multi-modal 3d object detection with object-centric fusion," in *ICCV*, 2023.
[57] L. Wang, X. Zhang, Z. Song, J. Bi, G. Zhang, H. Wei, L. Tang, L. Yang, J. Li, C. Jia, and L. Zhao, "Multi-modal 3d object detection in autonomous driving: A survey and taxonomy," *IEEE Transactions on Intelligent Vehicles*, 2023.