Controllable Spoofing Attacks on Visual SLAM in Robotic Vehicles

Yuan Xu[1], Gelei Deng[1], Guanlin Li[1], Xingshuo Han[2], Shangwei Guo[3], Tianwei Zhang[1]
[1]Nanyang Technological University. {xu.yuan, gelei.deng, guanlin001, tianwei.zhang}@ntu.edu.sg
[2]Nanjing University of Aeronautics and Astronautics. xingshuo.han@nuaa.edu.cn
[3]Chongqing University. swguo@cqu.edu.cn

***Abstract*—Visual Simultaneous Localization and Mapping (vSLAM) systems are fundamental to autonomous platforms such as self-driving vehicles, drones, and robotics. However, existing research has predominantly focused on improving the accuracy and performance of these systems, leaving their physical-world vulnerabilities underexplored. To bridge this critical gap, we propose `SLAMTricker`, a novel and practical attack framework that systematically exploits vSLAM vulnerabilities through a two-stage approach: an offline attack scenario search to identify frames highly susceptible to spoofing and an online attack vector generation to deploy malicious vectors. Unlike prior works, `SLAMTricker` offers high controllability by leveraging environment-aware attack vectors, making it highly effective in various scenarios.**

We evaluate `SLAMTricker` on four widely-used public datasets and three state-of-the-art vSLAM systems, alongside real-world experiments using a physical robot. Experimental results show that `SLAMTricker` significantly outperforms existing attacks in feasibility, transferability, and success rate, achieving diverse attack goals such as pose deviation, relocalization errors, map point manipulation, and incorrect loop closures. Video demos can be found in our anonymous website: https://sites.google.com/view/slamtricker.

1. Introduction

Simultaneous Localization and Mapping (SLAM) has been a prominent topic with widespread applications in robotic vehicles (RVs), such as robot vacuums (iRobot Roomba [1]), automatic guided vehicles (Nvidia Drive [2]), and unmanned aerial vehicles (DJI Tello [3]). SLAM serves as a critical technology for enabling accurate localization, particularly in indoor environments where GPS signals are unavailable, and it also aids in outdoor navigation by providing supplementary localization data. Although SLAM can be implemented by using diverse sensor data, like cameras [4], [5], LiDAR [6], [7], and radar [8], [9], camera-based approaches, commonly referred to as Visual SLAM (vSLAM), stand out due to their affordability, precision, and simplicity in system design, and emerge as an effective and widely adopted method in various real-world scenarios [10]–[14].

To accurately estimate position using camera data, a vSLAM system typically follows four key steps: (1) extracting features from each captured frame, (2) identifying stable matching pairs from extracted features across consecutive frames, (3) estimating position by optimizing deviations across all matching pairs, and (4) correcting accumulated estimation errors using bundle adjustment techniques. Based on these steps, several robust and widely adopted vSLAM frameworks have been developed in the robotics domain. For instance, ORB-SLAM2 [15] introduced the first comprehensive vSLAM framework, leveraging these four steps to compute both the RV's position and the camera's trajectory. Building on this foundation, DynaSLAM [16] extended ORB-SLAM2, enhancing its robustness in dynamic environments. More recently, ORB-SLAM3 [17] further advanced the field by introducing a multi-map data association technique, significantly improving the accuracy and resilience.

While recent work [46] surveys sensor attack hardness in general, little attention has been paid to spoofing attacks targeting vSLAM systems. In this paper, we pose the following question: *Given the prevalence of spoofing attacks on GPS-based localization systems [18]–[22], can we implement a* ***controllable*** *camera spoofing attack capable of misleading a vSLAM system into a specified position?* Existing techniques [23]–[26] have shown the feasibility of attacking vSLAM systems in either autonomous driving or mobile robots. However, these approaches primarily aim to introduce errors into the camera's trajectory and face several critical limitations: (1) **Limited Control over Outcomes**: Current attacks only induce localization errors without providing precise control over the spoofed locations. This significantly restricts both the flexibility and impact of the attack. (2) **Incomplete Attack Coverage**: Existing methods fail to comprehensively analyze the attack surface of vSLAM systems. They focus narrowly on injecting spoofing vectors and targeting a single module, neglecting other strategies (e.g., covering, shifting) and components (e.g., relocalization, loop closure). This leaves key vulnerabilities unexplored. (3) **Lack of Environmental Optimization**: These attacks overlook the varying contributions of different scenarios to attack success, which greatly impacts effectiveness. Due to internal systematic mechanisms, e.g., octree-based feature distribution, bin-based feature matching and outlier filtering, many adversarial points concentrated within a patch are often discard, reducing the impact of patches.

To address these limitations, we first perform a comprehensive analysis of the position estimation mechanism in vSLAM systems. Unlike GPS-based localization, we observed

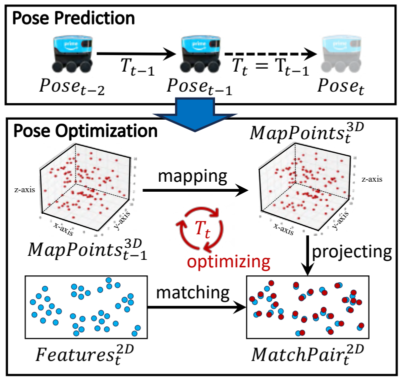


Figure 1: The process of pose prediction and optimization.

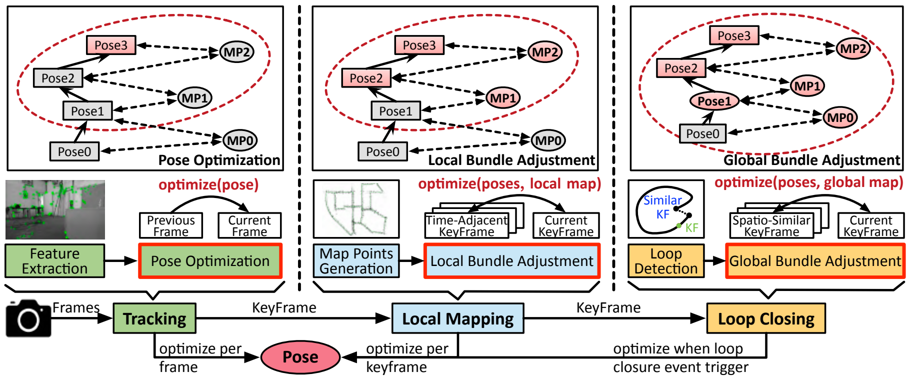


Figure 2: Overview of the optimizing pipeline in the vSLAM system. The variables and fixed states are shown in red and gray, respectively.

that vSLAM systems predict and optimize position estimates over time through processes like bundle adjustment, which enhance consistency and accuracy. As a result, a position misestimation at a specific moment may later be corrected by the system's optimization process. However, this iterative optimization also expands the attack surface, allowing us to target multiple stages of the optimization pipeline. Furthermore, we observe that the accuracy of vSLAM systems is highly dependent on the distribution of features within the surrounding environment. In certain scenarios, even minor changes in extracted features can directly or indirectly affect localization and mapping accuracy across different stages of optimization. Existing research often treats vSLAM systems as monolithic, overlooking the vulnerabilities unique to each stage and the influence of feature distributions. Critical questions, such as "*Which features are critical for position estimation at different stages?*", and "*What environmental characteristics can make attacks more practical to design and deploy?*" remain largely unanswered. For example, imagine a scenario where a large number of features are extracted and scattered across different parts of a frame. Since only changes to a majority of feature points involved in optimization can significantly affect the final pose estimation, a successful attack in such a scenario might require generating multiple spoofing vectors and strategically deploying them across different areas. This approach, however, is impractical in real world due to physical constraints.

Building on the insights and analysis above, we propose four new vSLAM spoofing attacks targeting different stages of the vSLAM pipeline: the *pose deviation attack*, *relocalization attack*, *map point manipulation attack*, and *illusion creation attack*. These attacks are designed not only to forge a malicious position for the victim RV but also to compromise the map's integrity and impose high time costs for restoring the system to its normal state. To achieve this, we must address three key challenges: (1) **Spatio-temporal Consistency**: Existing vSLAM systems rely on spatio-temporal consistency (e.g., temporal feature matching and spatial similarity searching) to estimate and optimize the RV's position, thereby enhancing robustness and accuracy. This makes designing an effective spoofing attack more complex than simply manipulating features over a few frames. We must analyze all frames along the victim RV's route to craft a comprehensive and effective attack. (2) **Feature Validation Mechanisms**: Most vSLAM systems implement mechanisms to filter out suspicious or erroneous features, such as bin filtering and least-squares optimization. To ensure a successful spoofing attack, we need to generate features that can bypass these validation mechanisms while remaining feasible for real-world deployment. (3) **Attack Transferability**: The real-world environment introduces uncertainties and dynamic conditions. To ensure the practicality of our attack, we must address its transferability. Specifically, our attack should remain effective across different vSLAM systems, hardware configurations (e.g., image resolutions, vehicle speeds), and environmental contexts (e.g., varying routes and the presence of movable objects).

To address the challenges outlined above, we present SLAM-Tricker, the *first* controllable camera spoofing attack framework for vSLAM. `SLAMTricker` is capable of automatically generating potential attack vectors and deploying them within an identified environment. To pinpoint vulnerable frames in the target environment, we first extract vulnerabilities associated with each position optimization stage. Building on this, we design an innovative attack scenario search scheme that analyzes all spatio-temporal frames across the route, identifying key frames for targeted attacks. For generating spoofing vectors, we formulate the vSLAM spoofing attack as an optimization problem and propose a spoofing vector generation scheme. This scheme computes the physical 3D positions of spoofing vectors, deploys them, and triggers different attacks at runtime.

To ensure practicality and transferability of our proposed attack, we identify two key components for modeling vSLAM attacks: (1) **Perturbation Function**: Maps the spoofing vector (e.g., a malicious object or feature point) in the physical world to the model's input space. (2) **Objective Function**: Quantifies the attack's goal, guiding the optimization process. For the perturbation function, we formalize the transformations involved in the mapping process, including spatial mapping, projection, feature searching, and modification. Additionally, we incorporate the attacker's capabilities and environmental constraints to define the feasible bound-

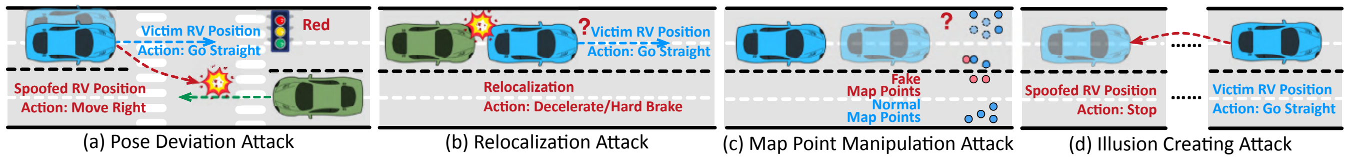


Figure 3: Our proposed four spoofing attacks base on the identified vulnerabilities of each module in vSLAM.

aries for vSLAM spoofing attacks. Our methodology also enables analysis of scenarios where vSLAM employs non-linear optimization models across varying spatio-temporal granularities. This comprehensive approach ensures precise and practical deployment of spoofing vectors while accounting for the complexities of real-world environments.

We evaluate the susceptibility of three prevalent vSLAM frameworks (ORB-SLAM2 [15], DynaSLAM [16], and ORB-SLAM3 [17]) to our four new attacks. Utilizing KITTI [27], [28], 4Seasons [29], EuRoC [30] and TUM [31] datasets, we identify vulnerable frames and strategically implement spoofing vectors. Our results show that `SLAMTricker` can modify the RV's estimated pose or decrease map points. Additionally, we assess the practicality and adaptability of these attacks in real-world conditions with a physical RV. Well-positioned spoofing vectors can lead to malicious state estimation, causing navigation disruptions, collision risks, and substantial recovery time. We further estimate the influence of various image resolutions, vehicle speeds, environmental dynamic and routes on these attacks. The results corroborate `SLAMTricker`'s enduring robustness in all conditions.

2. Studies on vSLAM Systems

2.1. Position Estimation Overview

Accurate localization and environment perception are essential tasks for robotic vehicles (RVs). As shown in Figure 1, modern vSLAM systems employ a two-stage framework to continuously correct the estimated position.

Pose Prediction. In the first stage, an approximate pose is predicted using a constant velocity motion model. This model assumes the RV maintains a constant velocity, meaning the current pose transformation, T_t, from time $t-1$ to t can be approximated as the pose transformation, T_{t-1}, from time $t-2$ to $t-1$. Based on this assumption, the pose at the current time t is estimated by combining the pose at time $t-1$ with T_t. While this approach provides a quick approximation, errors from environmental dynamics or deviations from the assumption often accumulate.

Pose Optimization. The second stage refines the approximate pose by correcting these errors through: (1) *Map Points Mapping*. The 3D map points from time $t-1$ are transformed to the current time t using the approximated pose transformation T_t. (2) *Map Points Projecting*. The transformed 3D map points are projected onto the current 2D frame using the camera's intrinsic parameters and configurations. (3) *Feature Matching*. Extracted 2D features from the current frame are matched with the projected 2D map points. This step identifies the best matching pairs based on similarity and spatial distance. (4) *Error Optimizing*. The mean error across all matched feature pairs is minimized by fine-tuning the pose transformation T_t, producing a more accurate estimate.

2.2. Vulnerability Analysis

Existing vSLAM systems are typically composed of three core modules: (i) tracking, (ii) local mapping, and (iii) loop closing. As illustrated in Figure 2, these modules collectively contribute to pose optimization, albeit at different update frequencies and scopes. Below, we describe each module, analyze their roles in the optimization process, and identify vulnerabilities exploitable for spoofing attacks.

Module 1: Tracking. The tracking module serves as the foundation of the vSLAM system by processing each perceived frame to estimate the RV's pose in real time. As shown in the left part of Figure 2, it extracts features from the current frame and matches them with projected 2D map points to optimize the RV's pose. This module operates at a high frequency (i.e., per frame) to ensure responsiveness to motion. However, the pose optimization in this stage focuses exclusively on the current pose (highlighted in red), while map points and poses from other frames (shown in gray) remain fixed. If insufficient matching pairs are found during optimization, the system transitions into relocalization mode. In this mode, the RV slows down or turns around to collect additional features, significantly affecting its functionality.

► **Potential Attacks.** The tracking module involves two primary operations related to pose estimation: pose optimization per frame and relocalization when few matching pairs are available. These operations present two attack opportunities: (1) *Pose Deviation Attack*. This attack aims to introduce cumulative errors into pose estimation by manipulating features over multiple consecutive frames. By shifting the majority of features in matching pairs to a specific direction, attackers can cause the pose optimization to output a spoofed deviation. The effectiveness of this attack depends on the number of frames processed before the system transitions to the local mapping module, which could correct accumulated errors based on newly generated keyframes. The conversion of a frame to a keyframe is determined by its difference from the previous keyframe. Scenarios where the RV remains stationary for extended periods—such as waiting at a red traffic light—create opportunities for this attack. For example, according to the National Association of City Transportation Officials [32], traffic lights typically remain red for 60 to 90 seconds, translating to $600 \sim 900$ frames in the 10fps KITTI dataset. These prolonged periods allow attackers to inject substantial errors, which may be further amplified during braking or startup times.

(2) *Relocalization Attack*. This attack seeks to force the vSLAM system into relocalization mode by disrupting the initial pose optimization. By reducing the number of matching pairs or manipulating their characteristics to appear as outliers, attackers can trigger relocalization, causing the RV to slow down or stop for reinitialization. This disruption may result in navigation delays or even rear-end collisions. To execute this attack, attackers must understand the conditions under which matching pairs are discarded as outliers. For instance, features with edges that deviate significantly from the majority during optimization are flagged as outliers and excluded from further processing. By obscuring extracted features or altering them to appear inconsistent, attackers can decrease the number of valid matching pairs, effectively destabilizing the system.

Module 2: Local Mapping. The local mapping module processes keyframes to refine the local map and maintain consistency. As illustrated in the center of Figure, it performs two key tasks: creating new map points and optimizing the poses of nearby keyframes and associated map points. During the map point generation process, stable features observed in multiple views are triangulated to compute the 3D coordinates of new map points, thereby enriching the map structure. Once a new keyframe is added, the system executes local bundle adjustment, which minimizes projection errors between map points and their observed 2D positions in keyframes. Unlike the tracking module, this optimization adjusts not only the pose of the current frame but also the poses of co-visible keyframes and the map points observed by those frames. By combining these operations, the local mapping module ensures local consistency and accuracy. Operating at an intermediate frequency, it serves as a bridge between the high-frequency tracking module and the less frequent global corrections performed by loop closing.

► **Potential Attacks.** The local mapping module is susceptible to the *Map Point Manipulation Attack*, which aims to disrupt pose estimation by altering matched features during optimization. This attack can either eliminate valid map points or introduce malicious, unstable features that are incorrectly converted into map points. To execute this attack, it is crucial to understand the mechanisms of map point creation and culling. A map point is considered valid if it survives three keyframes after its creation; otherwise, it is culled. The system removes a map point if the ratio of frames tracking it to the expected number of frames falls below a threshold or if fewer than two keyframes observe it after three keyframes. Attackers can exploit these rules by obscuring stable features across multiple frames, triggering map point culling to discard otherwise valid points. Alternatively, attackers can introduce "stable" but malicious features to generate fake map points, misleading the optimization process. By disrupting the local mapping module in this way, the attack compromises the system's ability to maintain local consistency and can introduce errors to other modules, affecting the overall vSLAM performance.

Module 3: Loop Closing. The loop closing module ensures long-term consistency by identifying and correcting accumulated drift through loop detection. It operates at a low frequency, focusing on global optimization to improve the entire map structure. When a loop is detected, the module identifies spatio-similar keyframes and triggers global bundle adjustment. This process adjusts all keyframe poses and associated map points, correcting errors accumulated over time. By refining the global pose graph, the module ensures that the overall map remains consistent and accurate, even as the RV revisits previously mapped areas, effectively mitigating drift over extended trajectories.

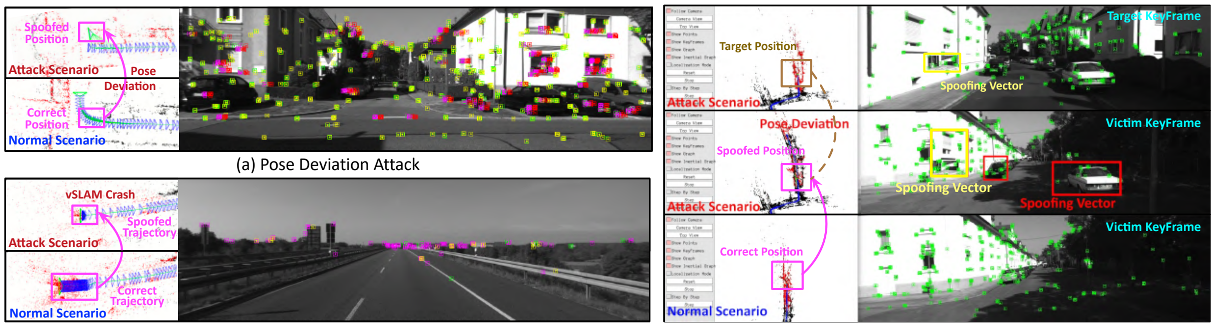
► **Potential Attacks.** The loop closing module is susceptible to two types of spoofing attacks: (1) *Loop Closing Failure Attack*, where the robot revisits a previously observed location, but the module fails to detect the loop, and (2) *Illusion Creating Attack*, where the RV is misled into believing it has revisited a location it has not actually encountered. The first type of attack is difficult to execute effectively, as subsequent observations of multiple keyframes often allow for re-detection of loop closures, though it may be facilitated through manipulations like map point corruption. The second attack is more critical, as it introduces false positive loop detection. This results in significant errors between poses, misleading the robot and disrupting its global map structure. False positives are particularly dangerous, as they can propagate large-scale inaccuracies throughout the pose graph, posing a severe threat to the system's reliability.

2.3. Preliminary Verification and Issues

To validate our analysis, we conducted a set of preliminary experiments using the KITTI dataset, as illustrated in Figure 4. For the *pose deviation attack* and *relocalization attack*, we directly modified the positions of extracted features. In the *pose deviation attack*, shown in Figure 4(a), the normal scenario depicts a vehicle waiting at an intersection and then turning right. Under the attack scenario, a subset of features (marked in purple) was randomly selected and slightly shifted to the right (marked in red) over 50 consecutive frames. This caused significant errors in the vehicle's estimated pose, deviating from its correct trajectory. For the *relocalization attack*, shown in Figure 4(b), we removed the majority of extracted features (marked in purple), which led to the vSLAM system crashing due to insufficient matching pairs for reliable pose estimation.

For the *map point manipulation attack* and *illusion creating attack*, we adopted patches and object manipulations from previous work [33]. As Figure 4(c) shows, we poisoned both the target and victim spaces with these patches. This caused the system to incorrectly detect a loop closure at an intersection, resulting in the vehicle being relocated to the target frame. Furthermore, the system added numerous erroneous map points to the map, corrupting the overall structure. Although these experiments successfully demonstrate the vulnerabilities of vSLAM systems, they also highlight a critical limitation: *directly modifying features or employing such conspicuous and abnormal patches is **IMPRACTICAL** in real-world scenarios.*

Previous study [23] has shown the feasibility of generating patches through dynamic adjustment models. However,

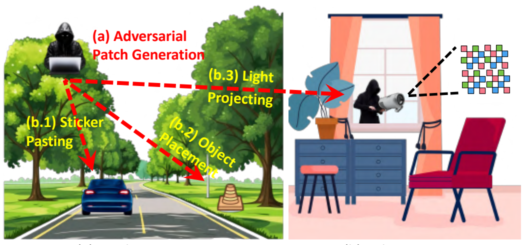


(a) Pose Deviation Attack

(b) Relocalization Attack

(c) Map Point Manipulation Attack and Illusion Creating Attack

Figure 4: Proposed attacks by directly modifying features (a,b) or using the patches (c) from previous work [33]



(a) Outdoor Scenario

(b) Indoor Scenario

Figure 5: The Attack Scenario: (a) a malicious attacker and (b.1-3) three types of attack vectors.

such approaches focus solely on hiding mismatched features, limiting their effectiveness in spoofing attacks. Specifically, these methods cannot control the positions of extracted features, making them unsuitable for pose deviation and relocalization attacks. Furthermore, they fail to provide a sufficient number of consistent matches for executing an illusion creating attack. In this paper, we propose an alternative strategy: *can we exploit specific scenarios to achieve our goals with the **LEAST** amount of natural attack vectors?*

3. Models and Assumptions

3.1. Threat Model

Attack Goal. This work focuses on a scenario where an attacker seeks to manipulate the localization and mapping outcomes of vSLAM systems in autonomous vehicles or robots by executing a camera spoofing attack. The attacker's objective is not merely to disrupt the system but to impose a controllable and deliberate alteration. This level of control enables the attacker to achieve specific malicious outcomes, such as steering the vehicle off its intended path, creating false perceptions of revisited locations, or introducing critical errors that could lead to accidents. Unlike prior work that primarily focuses on unstructured or generalized attacks, our approach emphasizes targeted attacks that guide the system toward predefined and attacker-controlled outcomes, posing a significant threat to safety and reliability.

System Model. This paper primarily focuses on exploring the vulnerabilities of widely adopted vision-based SLAM (vSLAM) systems used in RV applications. These systems rely on camera-captured frames to perform real-time localization and mapping in both indoor and outdoor environments. Specifically, during the visual odometry (VO) process, vSLAM systems extract features from carefully selected key pixels in each frame. These features are then mapped and tracked across consecutive frames to establish correspondences. Finally, leveraging the optimization methods outlined in § 2.1, the vSLAM system accurately performs simultaneous localization and mapping, ensuring robust trajectory estimation and environmental reconstruction for autonomous navigation.

3.2. Attacker Assumptions

Attack Scenario. To execute a practical attack in the real world, this paper assumes a two-stage attack scenario:

(1) Offline Stage. In this stage, the attacker employs a data acquisition vehicle or smartphone camera to traverse the victim RV's route and capture environmental visual contexts, including landmarks and scene elements. The collected image datasets are analyzed by `SLAMTricker` to identify potential target attack scenarios—specific sequences of consecutive frames where the attacker can exploit vulnerabilities using minimal or highly covert attack vectors. This analysis ensures the feasibility of deploying undetectable or stealthy attack patterns in the subsequent stage.

(2) Online Stage. Leveraging the insights from the offline analysis, the attacker use `SLAMTricker` to fine-tune the position and shape of the attack vectors based on the feature distribution of each target scenario. During this phase, the attacker carefully deploys these malicious vectors, ensuring their alignment with the visual context of the target environment. The attack is activated when the victim RV enters the predefined target scenario, exploiting the spatio-temporal consistency expected by the RV's perception system.

Attacker Capability. During the *offline stage*, we assume the attacker has knowledge of the victim's driving environment and hardware platform, consistent with prior work on localization-based spoofing attacks [23]–[26]. Specifically, the attacker is aware of the general driving route and environmental context of the victim's RV but does not have access to camera frames from the victim's system. Instead, the attacker collects environmental data independently using identical or similar hardware to replicate the visual context of the target area before launching the attack. Our attack operates under a **grey-box** access assumption, where the attacker knows the victim employs the target vSLAM system but lacks detailed internal technical information such as proprietary algorithms or configurations. The attacker can infer critical system behaviors through preliminary experiments (e.g., deploying patches to observe their effects on performance), as demonstrated in Figure 4. This grey-box assumption is considered reasonable and practical, as attackers can often obtain such knowledge through reconnaissance and testing without requiring privileged access to the victim's internal data.

Stealthiness Consideration. A critical requirement for a successful spoofing attack is to maintain its visual stealthiness. Unlike adversarial attacks in the digital domain, which focus on pixel-level indistinguishability, our approach aims to ensure that the deployed attack vectors are *semantically and contextually appropriate* within the target environment. By adhering to existing physical attack methodologies [21], [22], [34]–[39], we achieve this stealthiness through careful selection and deployment of attack vectors,. Specifically, **(b.1) Sticker Pasting**: The attacker can implement unnoticeable adversarial patches on commonly seen real-world objects, such as a moving vehicle, making them appear natural in the environment. **(b.2) Object Placement**: The attacker can deploy malicious 3D objects strategically in areas out of direct human attention but within the victim system's perception range, such as roadside or hidden locations. **(b.3) Light Projecting**: In indoor scenarios, the attacker can use a projector positioned outside a window to project malicious patches onto walls or other surfaces. This approach enables dynamic attacks without leaving physical traces.

4. SLAMTricker

4.1. Framework Overview

SLAMTricker is a comprehensive attack framework designed to exploit vulnerabilities in vSLAM systems by seamlessly integrating offline and online attack stages.

4.2. Offline Attack Scenario Search

The goal of offline attack scenario search is to identify frames that are vulnerable to spoofing in a given environment. A vulnerable frame should satisfy two conditions: (1) it can easily trigger the four proposed spoofing attacks, and (2) it requires minimal attack vectors to successfully execute the attack. To achieve this, we first formalize the workflow of vSLAM systems, then analyze environmental constraints to quantify the relationship between environmental contexts and attack vectors, and finally propose a search scheme to identify vulnerable frames for each type of attack.

4.2.1. vSLAM Formulation. The vSLAM module consists of the following components: (1) Graph optimization model G. (2) Pre-processing feature extraction function $\Phi : \mathbb{R}^{n*3} \to \mathbb{R}^{n*2}$, which maps 3D points into 2D pixels. (3) Projection function $\rho : \mathbb{R}^{n*3} \times \mathbb{R}^{6} \to \mathbb{R}^{n*2}$, which maps 3D map points into 2D map point coordinates in the current frame using the estimated RV pose. (4) Feature searching function $\Theta : \mathbb{R}^{n_1*2} \times \mathbb{R}^{n_2*2} \to \mathbb{R}^{n_2*2}$, which identifies the most relative feature from a map point. Let $p \in \mathbb{R}^6$ be the robot's pose estimation, including 3D coordinates and angles. $M = (M_1, M_2, \ldots, M_{n_m})$ is the pristine 3D map points. Then we have $m = \rho(M, p)$, where $m = (m_1, m_2, \ldots, m_{n_m})$, is the corresponding 2D map point coordinates. Let $K = (K_1, K_2, \ldots, K_{n_k})$ be the pristine 3D features. Then we also have $k = \Theta(\Phi(K), m)$, where $k = (k_1, k_2, \ldots, k_{n_m})$ is the identified relative 2D features. Notations used in SLAMTricker are summarized in Table 2 in Appendix.

4.2.2. Environment Constrains. We define the environment constraint ($\mathcal{C}$) based on the distribution of map points. A higher proportion of maliciously matched map points correlates with a higher likelihood of causing pose estimation errors. Frames with denser map points provide greater opportunities for attackers, as spoofing fewer features can yield significant disruptions. To quantify the concentration of map points, we introduce the **density metric** d_{m_i}, which measures the local density around a map point m_i:

$$\begin{aligned} d_{m_i} = &\mathrm{Card}(\{m_j | (m_1, m_2, \ldots, m_{n_m}) = \rho(M, p), \\ &\|m_i - m_j\|_2 \le \tau, j \in [n_m]\}), \end{aligned} \tag{1}$$

where $\|m_i - m_j\|_2$ denotes the l_2-normed distance between a current map point m_i and another map point m_j in the same frame. If the distance is less than a certain threshold τ, m_j can be regarded as a neighbor of m_i. The number of all neighbors $\mathrm{Card}(\cdot)$ is defined as the density of m_i.

We also define the **attack ratio** R, which is the proportion of high-density map points ($d_{m_i} \ge \gamma$), n'_m, to the total number of map points n_m:

$$\max\ R \quad \text{w.r.t. } R = \frac{n'_m}{n_m} \quad \triangleright\text{all methods} \tag{2}$$

$$\text{and} \quad \begin{cases} \min \|p - p'\|_2 & \triangleright\text{features adding method} \\ \max \|p - p'\|_2 & \triangleright\text{features shifting method} \end{cases}$$

A higher R and a larger pose offset ($\|p - p'\|_2$) increase the likelihood of achieving attack goals. Deployment methods like features adding aim to minimize pose offset, while features shifting maximizes it for greater deviation.

4.2.3. Attack Scenario Search Scheme. We propose an Attack Scenario Search Scheme (Algorithm 1) to identify vulnerable frames based on the vulnerabilities analyzed in § 2.2.

Pose Deviation Attack (Lines 2-6): To avoid misestimated position being corrected by the next optimization in local mapping module, we record the ID of each frame that is selected as a keyframe (the trigger of next optimization). When the number of frames between two keyframes exceeds a specified threshold T_1 and the attack ratio R exceeds the threshold T_2, these frames are considered suitable for attack.

Relocalization Attack (Lines 7-8): The number of input pairs in each frame, excluding outliers and the selected high-density map points, is also calculated. The comparison of the

Algorithm 1: Attack Scenario Search Scheme

```
Input: $F$                  ▷ A set of frames in the given environment
       $T_1 \sim T_5$        ▷ Thresholds of different attacks
Output: $F_v$               ▷ Target victim frames
        $KF_v$              ▷ Target victim keyframes
1  foreach Frame $f_i \in F$ do
2      if $f_i$ is a Keyframe then
3          $j = i$;
4          $KF.add(f_j)$
5      if $((i - j) > T_1) \vee (R > T_2)$ then
6          $F_v.add(f_j)$ ;          // pose deviation atk
7      if $f_i.nmatch - f_i.outliers - f_i.n'_m < T_3$ then
8          $F_v.add(f_i)$ ;          // relocalization atk
9  foreach KeyFrame $kf_i \in KF$ do
10     $kf_j.spoofpoints =$
           HgihDensityCompute($kf_j.newMapPoints$);
11     if $(R > T_2) \vee (kf_j.nmatch < T_4)$ then
12         $KF_v.add(kf_i)$ ;    // map point manip. atk
13     if $kf_j = DetectLoop(kf_i)$ then
14         if $SearchByProjection(kf_i, kf_j) > T_5$ then
15             $KF_v.add(kf_i)$ ;    // illusion creating
                   atk
```

calculated result with the threshold T_2 is used to determine if the frame can effectively trigger this attack.

Map Point Manipulation Attack (lines 9-12): Upon receiving a keyframe from the local mapping module, all high-density map points should be used as the attack target, as the attacker can manipulate more points with the same attack vector. So if R exceeds the threshold T_2, it is deemed that the exclusion of the map points in this keyframe will result in a more significant effect. Furthermore, if the number of model input pairs in the keyframe itself is less than the threshold T_4, it is believed that adding map points will have a more pronounced attack effect.

Illusion Creating Attack (Lines 13-15): We iterate through all keyframes to identify analogous scenarios across different spaces. Due to the insufficient similarity, these spatially separated keyframes do not trigger the loop correction in subsequent optimization. The number of matching pairs of these keyframes is counted. If it exceeds the threshold T_5, the attack is considered to be executed with minimal cost.

Thresholds $T_1 \sim T_5$ can be obtained through reconnaissance and testing without requiring privileged access to the victim's internal data. In our study, $T_1 \sim T_5$ are set to 20, 0.6, 10, 20, 40, respectively.

4.3. Online Spoofing Vector Generation

4.3.1. Attack Formulation. We first formulate the vSLAM attacks. The attacker aims to launch a spoofing attack, which manipulates the features to make vSLAM malfunction. Let $S = (S_1, S_2, \ldots, S_{n_S})$ be the malicious features generated from the spoofing vectors. S must satisfy the constraints from the attack capability $\mathcal{A}$ and environment constraint $\mathcal{C}$: $S \in \mathcal{A} \cap \mathcal{C}$. Let $m^s = (m^s_1, m^s_2, \ldots, m^s_{n_s})$ be the 2D map points influenced by the attack. We introduce $\Psi : \mathbb{R}^{n_1 * 2} \times \mathbb{R}^{n_2 * 2} \to \mathbb{R}^{n_3 * 2}$ to represent the attack's behavior on the pristine features k and malicious features s. Then the vSLAM attack is formulated as the following optimization problem:

$$\begin{aligned} \min \quad & \mathcal{L}_{atk}(\Psi(k, s), m, p; G) \\ \text{s.t.} \quad & s \in \{s | \Theta(\Phi(S), m^s), S \in \mathcal{A} \cap \mathcal{C}\}, \\ & k = \Theta(\Phi(K), m), m = \rho(M, p) \end{aligned} \tag{3}$$

where $\mathcal{L}_{atk}$ is the spoofing loss to achieve the attack goal.

4.3.2. Input Perturbation Modeling. After analyzing the spoofing attack capability $\mathcal{A}$ (§ 3.2) and environment constraint $\mathcal{C}$ (§ 4.2.2), there is still a big gap between spoofed features (S) and model inputs. We close this gap by first modeling three key functions (Φ, ρ, Θ) in vSLAM, and then formulating the attack behavior Ψ based on these models.

Modeling Mapping Function (Φ). The key to modeling the mapping function is to identify the features of the current frame and the downsampling level at which each feature is located. We use x and l to denote an image and the downsampling level. Thus, the extracted 2D malicious features $\Phi(S)_l$ at level l can be written:

$$x_l = \text{Resize}(x, \epsilon, l), \quad \Phi(S)_l = \text{FastDetect}(x_l, S), \tag{4}$$

where function Resize is a bi-linear interpolation downsampling method, reducing the resolution of x by the factor ϵ^l. FastDetect is a function composition, including a fixed 3D-to-2D mapping based on the sensor's setting, extracting features from spoofed vectors at corresponding x_l. Note this process is deterministic, i.e., extracted features from one image multiple times are the same.

Modeling Projection Function (ρ). This function projects a 3D map point in the physical world to the 2D coordinate in the current frame with an estimated pose. Such a process requires two steps: (1) transforming the past 3D coordinates of the map point to the current 3D coordinates, and (2) projecting the current 3D coordinates to the 2D coordinates in the frame. The step (1) can be described as follows:

$$\begin{aligned} & [M_t, 1]' = T \cdot [M_{t-1}, 1]' \\ & \text{w.r.t.} \quad T = \begin{bmatrix} R & r \\ 0' & 1 \end{bmatrix}, \quad \begin{aligned} M &= (M_1, M_2, \ldots, M_{n_m}) \\ M_i &= (M^x_i, M^y_i, M^z_i), \end{aligned} \end{aligned} \tag{5}$$

where M_t and M_{t-1} denote the 3D map points at time t and $t-1$. T is a 4×4 transformation matrix to represent the transformation relationship from M_{t-1} to M_t. Specifically, the transformation matrix is a diagonal matrix containing a 3×3 rotation matrix R and a 3×1 translation matrix r.

Once obtaining M_t, we can estimate the 2D map point coordinate in the current frame using the equation below:

$$\begin{aligned} & (m^x_i, m^y_i, 1)' = I_C \cdot (\frac{M^x_i}{M^z_i}, \frac{M^y_i}{M^z_i}, 1)' \\ & \text{w.r.t.} \quad I_C = \begin{bmatrix} f_x & 0 & c_x \\ 0' & f_y & c_y \\ 0' & 0 & 1 \end{bmatrix}, \quad \begin{aligned} m &= (m_1, m_2, \ldots, m_{n_m}) \\ m_i &= (m^x_i, m^y_i), \end{aligned} \end{aligned} \tag{6}$$

Here (m^x_i, m^y_i) is the 2D map point coordinates in the current frame. I_C is the camera intrinsic matrix containing the focal length along x-axis f_x and y-axis f_y, and the

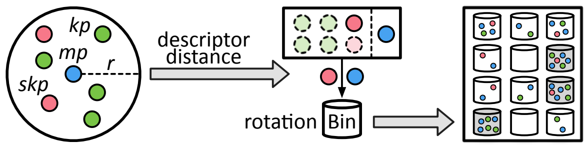


Figure 6: The process of feature searching function Θ.

optical center offset along x-axis c_x and y-axis c_y. Note all the camera intrinsic parameters are fixed and can be estimated by attackers.

Modeling Feature Searching Function (Θ). This function converts extracted features and projected map points into inputs of the graph model, as shown in Figure 6. IIt traverses all map points first, and then creates a container for each map point, storing all features within a circular area of radius r from the map point.

Let $k'_l = \Phi(K)_l$. Because m is calculated from M, and M is transferred from K in vSLAM, we can further provide each m_i a level l_{m_i} in m, based on the corresponding k'_l. The final features generation follows a rule that level l_{k_i} of the selected features k_i must be the same as level l_{m_i} of the corresponding map point m_i or the difference is 1, which is formalized as:

$$\begin{aligned} \mathcal{K}_i = \{k_j = (k_j^x, k_j^y) | (m_i^x - k_j^x)^2 + (m_i^y - k_j^y)^2 \le r^2, \\ i \in [n_m], j \in [n_k]\} \\ \text{s.t.} \quad l_{k_j} \in \{l_{m_i} - 1, l_{m_i}, l_{m_i} + 1\}, \text{for each} \quad k_j \in \mathcal{K}_i \end{aligned} \tag{7}$$

For each feature in the container, the feature searching function computes the Hamming distance of descriptors between all features $k_j \in \mathcal{K}_i$ and the related map point m_i. The feature with the shortest distance $k^{\text{best}} \in \mathcal{K}_i$ will form a pair with the map point m_i and be put into the corresponding histogram bin according to the rotation angle difference between them. Histogram bins are twelve containers for storing matching pairs of different angular differences (interval: 30°). Finally, the three bins with the most matching pairs (m_i, k^{best}) are selected as inputs to the model, while others are discarded. We can formulate this process as following:

$$\begin{aligned} (m_i, k^{\text{best}}) \in \{(m_i, k^{\text{best}}) \quad | \\ \text{BinIdx}((m_i, k^{\text{best}})) \in \text{Top3BinIdx}\} \end{aligned} \tag{8}$$

$$\begin{aligned} \text{w.r.t.} \quad k^{\text{best}} = \min \text{HDist}(k_j, m_i), k_j \in \mathcal{K}_i \\ \text{BinIdx}(m_i, k^{\text{best}}) = \frac{|\text{angle}_m - \text{angle}_{k^{\text{best}}}|}{30} \end{aligned} \tag{9}$$

Here HDist and BinIdx denote the functions of computing the Hamming distance and the bin index. Top3BinIdx is the set of top three bin indexes.

Formulating Attack Behavior (Ψ). This function simulates the influence after adding malicious features to the original features. For features adding and covering, Ψ represents a union operation between two variables with selection:

$$\begin{aligned} \Psi(k, s) = (k_{i_1}, k_{i_2}, \dots, k_{i_j}, s_{i_{j+1}}, \dots, s_{i_n}) \\ \text{s.t.} \quad \text{rotation invariance of } k_{i_j} \text{ and spatial similarity} \end{aligned} \tag{10}$$

Algorithm 2: Spoofing Vector Generation Scheme

```
Input: $m$                              ▷ Map points in the current frame
Output: $R$                                        ▷ The attack ratio
        $C_{\mathcal{S}}$                        ▷ The number of spoofing vectors
        $S$  ▷ The 3D coordinates of a set of malicious features
in the physical world
/* Potential Spoofing Vector Discovery */
$C_{\mathcal{S}} = C_s = 0$;
foreach map point $m_i$ belongs to $m$ do
    $d_{m_i} = 0$;
    foreach map point $m_j$ belongs to $m$ do
        if $(\|m_i - m_j\|^2 \le \tau) \wedge (\|l_{m_i} - l_{m_j}\| \le 1) \wedge (i \ne j)$ then
            $d_{m_i} = d_{m_i} + 1$ ; // density statistics
            $N_i.add(j)$ ; // set to store neighbors
    if $d_{m_i} \ge \gamma$ then
        V.add($m_i, N_i$) ; // vector to store victims
        $C_s = C_s + 1$ ;                                  // $n'_m$
$R = \frac{C_s}{n_m}$ ;                               // Attack Ratio
/* Vector Position Calculation */
foreach map point $m_i, N_i \in V$ do
    $s_i^x = m_i^x + r \cdot \sin \frac{2\pi}{\alpha}$ ; // atk point coordinates
    $s_i^y = m_i^y + r \cdot \sin \frac{2\pi}{\beta}$ ;          // in sample frame
    $l_{s_i} \in \{l_{m_i} - 1, l_{m_i}, l_{m_i} + 1\}$;
    /* upsampling restores 3D coordinates */
    if features shifting attack then
        $S_i^x, S_i^y, S_i^z = \text{upsample}(s_i^x, s_i^y, l_{s_i})$;
    else
        $S_i^x, S_i^y, S_i^z = \text{upsample}(m_i^x, m_i^y, l_{m_i})$;
    /* merging based on intersections */
    $j = i$;
    while $j! = V.size$ do
        if $(N_i \cap N_j) \ne \varnothing$ then
            $V' = N_i \cup N_j$ ;       // disjoint set $V'$
            V.erase(j);
        else
            $j++$;
$C_{\mathcal{S}} = \text{Card}(V')$ ;                       // Card($\mathcal{S}$)
```

For features shifting, Ψ represents a union operation with selection combining with an element-wise perturbing operation:

$$\Psi(k, s) = (k_{i_1} + d_1, k_{i_2} + d_2, \dots, k_{i_j} + d_j, \dots, s_{i_n} + d_n) \tag{11}$$

where $d_j = (d_j^x, d_j^y)$ is the coordinate shifting along x-axis and y-axis.

Spoofing Analysis. The selection of a malicious feature as an input to the model is influenced by four factors: the projected map point coordinates (m), the search radius (r), the descriptor of the malicious feature, and the bin index of the matching pair. The projected map point coordinates can be calculated using Equation 6, while the search radius is normally fixed as 15 in mainstream vSLAM frameworks (ORB-SLAM2, DynaSLAM, ORB-SLAM3). To minimize the descriptor distance in Equation 8, the attacker can employ solutions like designing the surrounding pixels based on a pre-defined pattern in feature extraction or copying the original feature area and changing its position. The histogram bin index, which is dependent on the distribution of rotation angle differences between map points and other features, can be estimated by analyzing the victim robot's trajectory.

4.3.3. Objective Function Design. We introduce our objective function $\mathcal{L}_{atk}$. Unlike previous spoofing works that perform the analysis on machine learning models, it is hard to design adversarial examples to fool a graph optimization model due to the wide distribution of model inputs. Therefore, given a graph optimization model G, inputs (s, k, m) and the corresponding correct pose output p, the attacker aims to generate spoofing vectors $\mathcal{S} = \mathcal{A} \cap \mathcal{C}$, such that $G(s, k, m) = p'$, where p' is a malicious pose deviated from the correct pose p. The objective function of a vSLAM spoofing attack can be formulated as below:

$$\min \mathcal{L}_{atk} \Rightarrow \min \text{ Card}(\mathcal{S}) \quad \text{s.t.} \quad G(s, k, m) = p' \tag{12}$$

where $G(s, k, m) = p'$ is the target attack goal and s is the result obtained after a series of pre-processing (Φ, ρ, Θ) of the spoofing vectors $\mathcal{S}$. Equation 12 aims to minimize the number of potential spoofing vectors (Card($\mathcal{S}$)) to achieve our goal. The goal of minimizing the spoofing loss $\mathcal{L}_{atk}$ (Equation 3) can be also equivalent to maximizing the attack ratio R, shown in Equation 2.

4.3.4. Optimization. To optimize the objective function $\mathcal{L}_{atk}$, we need to obtain a set of key information for each frame, including the number of spoofing vectors, the position of each malicious features in the physical world and the attack ratio R. Thus, we propose an automatic spoofing vector generation method (Algorithm 2) to solve the above challenges. Specifically, the method consists of two steps. The first step is the potential spoofing vector discovery. As described in § 4.2.2, we design a novel metric called density (d_{m_i}) to denote the distribution of map point m_i in the frame. Here, we calculate the density of each map point (Lines 2-6) and create a container to store the indexes of other map points associated with the current map point (Line 7). Note that the conditional statement contains the judgement of the downsampling level of map points if using the sticker or projection vectors, since the feature searching function Θ only searches the features of the level adjacent to the map point level (Equation 7). Once the density exceeds a pre-defined threshold γ, the map point will be selected and added into the victim map point container V (Lines 8-10), which is a set. The threshold γ determines the attack concealment: a smaller value enables more attack points but increasing the dispersion and likelihood of being detected. We can obtain the attack ratio R by the number of map points in V, and the number of map points in this frame (Line 11).

The second step is the vector position calculation, which aims to get the position of each malicious feature in the physical world and the number of spoofing vectors. Specifically, we traverse all map points in the victim map point container V. For each map point, we estimate the position of malicious features under three constraints: (1) the distance between the malicious feature and associated map point cannot exceed the searching radius r (Equation 7); (2) the downsampling level of the malicious feature must be near the level of the associated map points (Equation 7); (3) the rotation histogram bin where the malicious feature is located must be within the top three bins with the largest number (Equation 8). We use r, l_{m_i}, α and β to denote the boundary of the above three constraints (Lines 13-15). To maximize $\|p - p'\|_2$ in Equation 2, we set r as the maximum searching radius in the feature search function. The level of malicious features can be chosen based on the real-world condition. To ensure the third constraint, α and β for each malicious feature should be the same so that all malicious features can be collected in one rotation histogram bin, thus increasing the possibility of being chosen as one of the top three bins. We upsample the malicious features to restore their 3D coordinates S in the physical world (Lines 16-19). From Equation 2, we select different 2D coordinates to restore based on features deployment methods, and count the number of spoofing vectors through merging map point neighbor containers into disjoint sets (Lines 20-27).

5. Evaluation

5.1. Attack on Public Datasets

In this subsection, we evaluate proposed `SLAMTricker` attack using several well-established public datasets: KITTI [27], [28], 4Seasons [29], EuRoC [30], and TUM [31]. These datasets are widely regarded as benchmarks for testing vSLAM performance in applications such as autonomous driving, drone navigation, and automated guided vehicles. We target three state-of-the-art vSLAM frameworks, namely ORB-SLAM2 [15], DynaSLAM [16], and ORB-SLAM3 [17]. These frameworks represent cutting-edge vSLAM techniques with significant applications in both academic research and industrial use cases. Details of the datasets and the targeted vSLAM frameworks are provided in Appendix B. We mainly illustrate the attack results of ORB-SLAM2 on these datasets, if not specified. Quantitative results on all the frameworks and datasets are presented in § 5.1.5. Table 3 in Appendix shows the attack contexts of our experiments on datasets.

5.1.1. Pose Deviation Attack. Once an attack scenario is identified, Algorithm 2 is employed to determine the optimal number and positions of spoofing vectors required for the attack. Our observations reveal that pose deviation attack scenarios frequently occur during turning or parking maneuvers. As shown in Figure 7, we utilize the feature shifting method, wherein a truck with adversarial patches (acting as the spoofing vector) slowly moves in front of the victim vehicle. The patches are designed to manipulate features (green boxes) detected by the vSLAM system, causing deviations in the estimated trajectory. The left panel of Figure 7 illustrates the comparison between the normal scenario and the attack scenario. In the normal scenario, the victim vehicle maintains its correct trajectory, waiting at an intersection without deviations. Conversely, in the attack scenario, the victim's trajectory is spoofed, causing significant pose deviation.

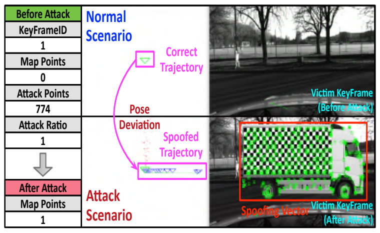


Figure 7: Pose Deviation Attack.

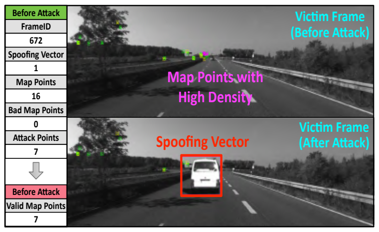


Figure 8: Relocalization Attack.

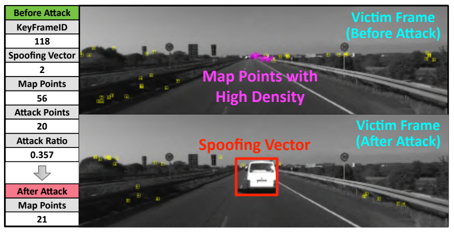


Figure 9: Map Point Manipulation Attack.

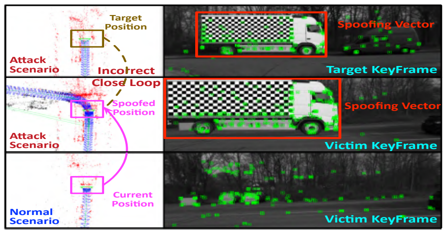


Figure 10: Illusion Creating Attack.

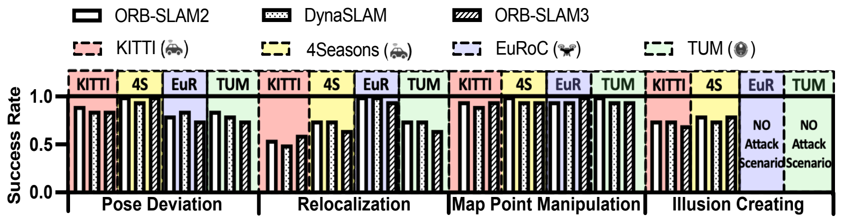


Figure 11: Attack success rates on three vSLAM frameworks.

5.1.2. Relocalization Attack. Relocalization attacks are evaluated using the features that cover the deployment method as depicted in Figure 8. The top image displays the map point in yellow and the corresponding searched feature in green prior to optimization. The bottom image shows the frame after the relocalization attack has been executed. Results indicate that a single spoofing vector can effectively obscure seven input pairs in the identified frame, resulting in a reduction in valid input pairs from 16 to 9, thereby triggering the relocalization. During the attack stage, the impact on the victim input pairs can be exacerbated by deploying a white van in front of the victim vehicle, as the van can obscure more points.

5.1.3. Map Point Manipulation Attack. The feasibility of the map point manipulation attack through map points culling is evaluated in Figure 9. Upon identification of the attack scenario, the features covering deployment method is used to obscure the high-density victim map points. Algorithm 2 is employed to calculate the position of the 3D object spoofing vector, which is then deployed in adjacent keyframes of the attack scenario. In this scenario, there are 56 map points in the target victim keyframe, with 20 of them being high-density points that can be covered by 2 spoofing vectors, resulting in an attack ratio of 0.357. For simplicity, a white van is utilized to cover these victim map points. We can observed that the victim map points fail to be identified and the total number of map points in this scenario is reduced from 56 to 21.

5.1.4. Illusion Creating Attack. Once an attack scenario is identified, Algorithm 2 is applied to locate regions with a high density of map points, enabling the deployment of spoofing vectors to induce loop closure errors. In this attack, we employ the feature adding deployment method to increase the similarity between the victim's victim and target keyframes. As illustrated in Figure 10, the similarity is enhanced by introducing a truck with adversarial patches (highlighted by the red boxes) in both the target and victim keyframes. Unlike prior works that rely on impractical setups [33], our approach minimizes environmental modifications by carefully selecting spoofing vectors with dense features and strategically placing them in the target region. The left panel of Figure 10 demonstrates the impact of the attack on the victim vehicle's trajectory. In the normal scenario, the victim vehicle correctly follows its current path. However, in the attack scenario, the vehicle mistakenly identifies a loop closure at an intersection, causing it to relocate its position to the target frame. This results in the victim system perceiving the spoofed position as its current location, demonstrating the effectiveness of this attack.

5.1.5. Quantitative Evaluation. Figure 11 shows the success rate of the four attacks on three mainstream vSLAMs. In particular, each attack is repeated 20 times on four popular datasets: KITTI, 4Seasons, EuRoC and TUM.
Observations. (1) The pose deviation attack maintains similar success rates on different datasets. Higher concentration of map points in the attack scenario and ratio of all map points lead to higher attack success rate. (2) For the relocalization attack, success rates differ across datasets due to variations in map-point distributions and concentrations. For example, the attack scenario in EuRoC is in a dark hall, and most of the map points are concentrated on the bright windows, so we can easily trigger relocalization as long as we cover some windows. However, the scenarios in other datasets still have some scattered map points that are difficult to remove. (3) The map point manipulation attack maintains high success rates on different datasets. This is because the generation and distribution of map points are relatively stable and will not be affected by randomness. (4) The illusion creating attack relies on the similarity of different locations in the dataset. Since both EuRoC and TUM are collected in a single hall, identifying similar places in this small area is difficult.

5.2. Attack on Physical Robotic Platform

In this subsection, we evaluate the impact of the proposed four attacks on an physical robotic platform through a

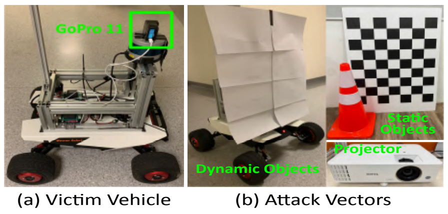


Figure 12: Experiment setup.

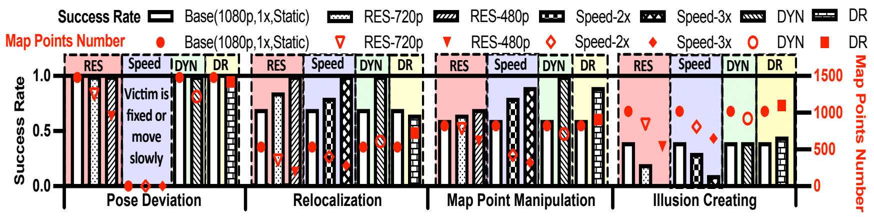


Figure 13: Transferability analysis in the physical world.

series of real-world case studies. Table 4 in Appendix shows the attack contexts of our experiments in the physical world.

Experiment Setup. We adopt a physical unmanned ground vehicle (UGV), which is equipped with a GoPro HERO 11 camera [40] to capture 720p images, and an onboard chip to run Robot Operating System (ROS [41]) and ORB-SLAM3. As shown in Figure 12, three attack vectors are selected: (1) a dynamic object represented by a mobile robot with eight pieces of white paper attached; (2) static objects including traffic cones and whiteboard marking with mosaics; (3) a Benq TH685P projector with 3500 lumens.

5.2.1. Attack Feasibility Analysis. In the offline stage, the UGV is controlled to navigate in our university and record the surrounding environment. The recorded video is then processed using `SLAMTricker`, to identify the target scenarios for each of the four spoofing attacks, as illustrated in Figure 18. Next, the position of the spoofing attack is computed based on the density of map points in the frame. In the online stage, the appropriate spoofing vectors are selected and deployed into the computed position. The navigation task is launched, causing the victim robot to autonomously move towards the designed attack scenario. Each scenario is described in detail in Appendix B.

5.2.2. Transferability Study. In this subsection, we aim to examine the transferability of our proposed attack methodology by evaluating the influence of three critical factors on the efficacy of `SLAMTrick`. Specifically, we target four critical factors, including image resolutions, vehicle speeds, dynamic environments and different routes. Figure 13 presents the success rate and map points number of the four physical attacks. The experiments are repeated 20 times, and the number of successful outcomes is recorded. The details of each factor are as follows:

Image Resolution (Figure 19). Three image resolutions are selected as input for ORB-SLAM3: 1080p (1980×1080), 720p (1280×720), and 480p (640×480). As vSLAM relies on pixel differences to extract features, reducing the resolution leads to a decrease in the number of features extracted per frame, consequently reducing the number of stable map points. It is worth noting that an image ratio of 4:3 is used in 480p resolution, contrasting with 16:9 in 1080p or 720p resolutions. This deviation leads to the exclusion of peripheral pixels on the right and left sides, which can consequently contribute to a decrease in the quantity of stable map points.

Vehicle Speed (Figure 20). To ensure the control group's consistency, the robot moved at a slower speed in each scenario. Subsequently, the recorded 1080p video is accelerated by factors of 2 and 3 by sampling frames. Since map points are generated using stable features extracted from multiple consecutive keyframes, higher vehicle speeds result in a decrease in the number of keyframes along the same route, leading to a reduction in the generated map points.

Dynamic Environments (Figure 21.ab). This experiment is conducted at the normal vehicle speed and 1080p image resolution. However, a dynamic disturbance is introduced to the environment in the first scenario by having a person move in front of the victim robot during the attack. The impact of this attack on the number of map points is minimal since moving objects do not generate stable map points. While dynamic disturbances may momentarily obscure some stable map points, these points swiftly reappear after the disturbance is gone, thus exhibiting a level of robustness in the face of temporary disruptions.

Different Routes (Figure 21.ac). This experiment is also conducted at the normal vehicle speed and 1080p image resolution. However, the route variations introduced for the victime robot lead to minor differences in the distribution of map points. As shown in Figure 21a, the robot proceeds linearly towards a wall. In contrast, Figure 21b shows the robot navigates in a different route, first approaching a door and then taking a right turn to face the wall. Despite these variations in the angles for observing attack vectors, both scenarios preserve a high attack success rate.

Quantitative Evaluation. Figure 13 presents the success rate and number of map points for four physical attacks. Each experiment is repeated 20 times, and the average result is recorded. We can observe that the pose deviation attack exhibits good robustness across various parameters. This stems from the attack's reliance on the victim map points' ratio rather than their absolute count. The success rate of the relocalization attack rises as the number of map points decreases. This can be attributed to the attack's efficacy in diminishing the map points by covering them until the reduced count triggers relocalization. Similarly, the success rate of the map point manipulation attack also escalates with a decrease in the number of map points. This effect arises due to the requirement of a sufficient number of matching similar map points in a series of temporally continuous keyframes across distinct spaces for triggering a closed loop. The reduction in map points diminishes such similar matches, resulting in the failure of the closed loop. Consequently, the success rate of the illusion creating attack experiences a decrease as the number of map points increases. Notably, all the proposed attacks demonstrate robustness under dynamic environments and different routes.

6. Related Work

Related Attacks on ORB-SLAM. Recently, Wang et al. [24] employed 74 infrared lights in a parking lot to orchestrate a spoofing attack on vSLAM, which resulted in the generation of discontinuous and uneven trajectories. Ikram et al. [25] discovered that a straightforward patch implementation can effectively by-pass all geometric checks within the ORB-SLAM framework, leading to erroneous loop closures in indoor environments. Chen et al. [23] designed an adversarial patch generation method and improved the robustness of these patches by hiding them on common real-world objects. All of the above works can be regarded as a specific use case of our proposed map point manipulation attack through adding spoofing vectors. Furthermore, through a comprehensive analysis of attack scenarios and the utilization of diverse attack vectors, our proposed techniques can control the spoofed position of each attack goal and increase attack success rate, which could not be achieved by prior works.

Related Attacks on DNN-based vSLAM. The integration of deep learning models into vSLAM has gained significant attention within the research community recently [42]–[44]. These works aim to enhance robustness in dynamic environments by replacing a part of geometric vision tasks in the vSLAM pipeline, such as depth estimation and optical flow prediction. However, DNN-based vSLAM inherently introduces increased computational overhead and is vulnerable to adversarial attacks [26], [45]. Both the targets and methods of these attacks differ significantly from this paper.

7. Discussion and Limitation

Attack Adaptability. The appropriate scenario for each attack highly depends on the map points in the environment and how robots behave, but widely exist in real-world situations. Specifically, the pose deviation attack happens when a robot stays in one place for a long time, e.g., waiting at traffic lights or parking in a lot. The relocalization attack is more common in places with fewer map points, such as highways and indoor areas. The map point manipulation attack can also take place in areas with dense map point distributions, such as parking lots and suburbs. The illusion-creating attack occurs in areas with high similarities, which is typical in office spaces and urban areas due to similar furniture or building layouts.

Countermeasures. We discuss the potential defense strategies at two levels. The first category is at the sensor level. Sensor fusion, a technique for intelligently combining data from multiple sensors, could be utilized as a defense against vSLAM spoofing attacks. RV systems often have multiple sensors, such as cameras, Lidar and IMU, to provide additional information to detect and counter an vSLAM attack. There have been several sensor fusion algorithms proposed with a focus on security and safety [47]–[50]. However, it is important to note that this defense approach relies on the majority of the sensors functioning correctly. While not foolproof, incorporating sensor fusion can greatly increase the difficulty of executing a successful attack.

The second category is at the model level. Since vSLAM spoofing attacks rely heavily on the distribution of map points, one potential defense is to introduce randomness in the input pair selection for the nonlinear graph optimization model. By doing so, the attacker is not able to predict when and where to inject spoofing vectors into the victim frame. To achieve this, one approach is to randomly select a feature from the group of high-scoring candidates for each map point instead of always selecting the feature with the highest score. An alternative is to randomly choose some map point-feature pairs from all inputs for optimization. While these defense strategies have the potential to mitigate the risks of vSLAM spoofing attacks, it is worth noting that the first solution reduces the robustness of vSLAM, while the second increases the likelihood of triggering relocalization.

Limitations. While SLAMTricker successfully demonstrates controllable and systematic spoofing attacks against multiple vSLAM systems, several limitations remain to be addressed in future work: (1) **Route dependence**: SLAMTricker assumes predictable trajectories for offline analysis; effectiveness may drop if the victim deviates significantly from the planned route. (2) **Environmental robustness**: The attack remains reliable under moderate illumination or occlusion changes but requires further validation under extreme outdoor conditions such as strong sunlight, rain, or crowded scenes. (3) **Generalization**: Evaluation is limited to sparse-feature-based vSLAM systems. Extending the framework to dense or neural SLAM remains future work. Nevertheless, the systematic analysis of `SLAMTricker` deepens the understanding of fundamental vulnerabilities in SLAM pipelines and provides insights for designing more resilient algorithms.

8. Conclusion

This paper focuses on the security of vision-based vSLAM for position estimation tasks in RVs. We propose the first camera spoofing attack framework for vSLAM. We demonstrate the effectiveness of `SLAMTricker` by conducting experiments on three popular vSLAM methods. Our results highlight the importance of addressing security risks in RVs relying on vSLAM and the need for robust vSLAM modules to prevent malicious attacks.

Acknowledgements

We thank the reviewers for their insightful suggestions and comments. This work is supported by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme and CyberSG R&D Cyber Research Programme Office. Any opinions, findings and conclusions or recommendations expressed in these materials are those of the author(s) and do not reflect the views of National Research Foundation, Singapore, Cyber Security Agency of Singapore as well as CyberSG R&D Programme Office, Singapore.

References

[1] “irobot brings visual mapping and navigation to the roomba 980.” https://spectrum.ieee.org/irobot-brings-visual-mapping-and-navigation-to-the-roomba-980, 2023.

[2] “Nvidia drive.” https://on-demand.gputechconf.com/gtc-cn/2019/pdf/CN9618/presentation.pdf, 2023.

[3] “Dji tello ros2.” https://github.com/tentone/tello-ros2, 2023.

[4] A. Pumarola, A. Vakhitov, A. Agudo, A. Sanfeliu, and F. Moreno-Noguer, “Pl-slam: Real-time monocular visual slam with points and lines,” in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4503–4508.

[5] W. Wang, D. Zhu, X. Wang, Y. Hu, Y. Qiu, C. Wang, Y. Hu, A. Kapoor, and S. A. Scherer, “Tartanair: A dataset to push the limits of visual slam,” in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 4909–4916.

[6] K. Liu, A. Xiao, J. Huang, K. Cui, Y. Xing, and S. Lu, “D-lc-nets: Robust denoising and loop closing networks for lidar slam in complicated circumstances with noisy point clouds,” in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 12 212–12 218.

[7] X. Zhao, S. Yang, T. Huang, J. Chen, T. Ma, M. Li, and Y. Liu, “Superline3d: Self-supervised line segmentation and description for lidar point cloud,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, vol. 9, 2022, pp. 263–279.

[8] G. Kim, Y. S. Park, Y. Cho, J. Jeong, and A. Kim, “Mulran: Multimodal range dataset for urban place recognition,” in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2020, pp. 6246–6253.

[9] P. Gao, S. Zhang, W. Wang, and C. X. Lu, “Dc-loc: Accurate automotive radar based metric localization with explicit doppler compensation,” in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2022, pp. 4128–4134.

[10] S. Zheng, J. Wang, C. Rizos, W. Ding, and A. el Mowafy, “Simultaneous localization and mapping (slam) for autonomous driving: Concept and analysis,” *Remote Sensing*, vol. 15, no. 4, p. 1156, 2023.

[11] J. Cheng, L. Zhang, Q. Chen, X. Hu, and J. Cai, “A review of visual slam methods for autonomous driving vehicles,” *Engineering Applications of Artificial Intelligence*, vol. 114, p. 104992, 2022.

[12] K. L. Lim and T. Bräunl, “A review of visual odometry methods and its applications for autonomous driving,” *CoRR*, vol. abs/2009.09193, 2020.

[13] A. Tourani, H. Bavle, J. L. Sánchez-López, and H. Voos, “Visual slam: What are the current trends and what to expect?” *Sensors*, vol. 22, no. 23, p. 9297, 2022.

[14] M. Filipenko and I. Afanasyev, “Comparison of various slam systems for mobile robot in an indoor environment,” in *Proceedings of the IEEE Conference on Intelligent Systems*, 2018, pp. 400–407.

[15] R. Mur-Artal and J. D. Tardós, “Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras,” *IEEE Transactions on Robotics*, 2017.

[16] B. Bescós, J. M. Fácil, J. Civera, and J. Neira, “Dynaslam: Tracking, mapping, and inpainting in dynamic scenes,” *IEEE Robotics Autom. Lett*, 2018.

[17] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, “Orb-slam3: An accurate open-source library for visual, visual-inertial and multi-map slam,” *IEEE Transactions on Robotics*, 2021.

[18] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, “Unmanned aircraft capture and control via gps spoofing,” *Journal of Field Robotics*, 2014.

[19] D. A. Schmidt, K. Radke, S. A. Çamtepe, E. Foo, and M. Ren, “A survey and analysis of the gnss spoofing threat and countermeasures,” *ACM Computing Surveys*, 2016.

[20] S. Z. Khan, M. Mohsin, and W. Iqbal, “On gps spoofing of aerial platforms: a review of threats, challenges, methodologies, and future research directions,” *PeerJ Computer Science*, 2021.

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

[22] K. C. Zeng, S. Liu, Y. Shu, D. Wang, H. Li, Y. Dou, G. Wang, and Y. Yang, “All your gps are belong to us: Towards stealthy manipulation of road navigation systems,” in *USENIX Security Symposium*, 2018.

[23] B. Chen, W. Wang, P. Sikorski, and T. Zhu, “Adversary is on the road: Attacks on visual slam using unnoticeable adversarial patch,” in *USENIX Security Symposium*, 2024.

[24] 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 Conference on Computer and Communications Security*, 2021.

[25] M. H. Ikram, S. Khaliq, M. L. Anjum, and W. Hussain, “Perceptual aliasing++: Adversarial attack for visual slam front-end and back-end,” *IEEE Robotics Autom. Lett*, 2022.

[26] Y. Nemcovsky, M. Jacoby, A. M. Bronstein, and C. Baskin, “Physical passive patch adversarial attacks on visual odometry system,” in *ACCV*, 2022.

[27] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *Computer Vision and Pattern Recognition*, 2012.

[28] “Visual odometry / slam evaluation 2012,” http://www.cvlibs.net/datasets/kitti/eval_odometry.php, 2022.

[29] “4seasons dataset,” https://cvg.cit.tum.de/data/datasets/4seasons-dataset, Tech. Rep., 2024.

[30] “The euroc mav dataset,” https://projects.asl.ethz.ch/datasets/doku.php?id=kmavvisualinertialdatasets, 2023.

[31] “Rgb-d slam dataset and benchmark,” https://cvg.cit.tum.de/data/datasets/rgbd-dataset, 2022.

[32] “Urban street design guide - signal cycle lengths.” https://nacto.org/publication/urban-street-design-guide/intersection-design-elements/traffic-signals/signal-cycle-lengths/, 2022.

[33] 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.

[34] 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 *ACM Conference on Computer and Communications Security*, 2019.

[35] 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.

[36] 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.

[37] 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 Conference on Computer and Communications Security*, 2020.

[38] 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.

[39] T. Sato, J. Shen, N. Wang, Y. Jia, X. Lin, and Q. A. Chen, “Dirty road can attack: Security of deep learning based automated lane centering under physical-world attack,” in *USENIX Security Symposium*, 2021.

[40] “Gopro hero 11,” https://gopro.com/en/us/shop/cameras/hero11-black/CHDHX-111-master.html, 2022.

[41] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *IEEE International Conference on Robotics and Automation*, 2009.

[42] L. Xiao, J. Wang, X. Qiu, Z. Rong, and X. Zou, "Dynamic-slam: Semantic monocular visual localization and mapping based on deep learning in dynamic environment," *Robotics and Autonomous Systems*, 2019.

[43] S. Milz, G. Arbeiter, C. Witt, B. Abdallah, and S. K. Yogamani, "Visual slam for automated driving: Exploring the applications of deep learning," in *CVPR Workshops*, 2018.

[44] R. Li, S. Wang, and D. Gu, "Deepslam: A robust monocular slam system with unsupervised deep learning," *IEEE Trans. Ind. Electron*, 2021.

[45] Z. Ali, M. L. Anjum, and W. Hussain, "Adversarial examples for handcrafted features," in *BMVC*, 2019.

[46] H. Kim, R. Bandyopadhyay, M. O. Ozmen, Z. B. Celik, A. Bianchi, Y. Kim, and D. Xu, "A systematic study of physical sensor attack hardness," in *IEEE Symposium on Security and Privacy*, 2024.

[47] G. Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart, "Fusion of imu and vision for absolute scale estimation in monocular slam," *Journal of Intelligent and Robotic Systems*, 2011.

[48] W. Huang and H. Liu, "Online initialization and automatic camera-imu extrinsic calibration for monocular visual-inertial slam," in *IEEE International Conference on Robotics and Automation*, 2018.

[49] Y.-S. Shin, Y. S. Park, and A. Kim, "Direct visual slam using sparse depth for camera-lidar system," in *IEEE International Conference on Robotics and Automation*, 2018.

[50] Y. Zhu, C. Zheng, C. Yuan, X. Huang, and X. Hong, "Camvox: A low-cost and accurate lidar-assisted visual slam system," in *IEEE International Conference on Robotics and Automation*, 2021.

Appendix

1. Distribution of Map Points

In this section, we aim to generalize the common features of the target scenario in order to enhance the efficiency of identifying potential attack opportunities. The target scenario for `SLAMTricker` is dependent on the distribution of map points, and thus, this study conducts an in-depth evaluation of various real-world environments to identify and analyze the factors that determine the distribution of map points. The scenarios in our evaluation is collected from the KITTI dataset. Furthermore, additional scenarios were recorded using a GO Pro 11 camera in order to expand the scope of the evaluation. The ORB-SLAM2 framework was used to analyze the distribution of map points in each scenario. It is crucial to note that for a pixel in a frame to be converted into a map point, it must first be extracted as a feature and observed by multiple consecutive keyframes. As such, we regard the feature extraction and temporal continuity as key factors that influence the distribution of map points and conduct analysis. The results of this analysis can also help us better understand the attack vectors applicable to different scenarios (Table 1).

1.1. Feature Extraction. We investigate the correlation between feature extraction and map point distribution from three distinct perspectives: the complexity of the environment (Figure 14), the distance between the target object and victim RV (Figure 15), and the lighting conditions (Figure 17). To specifically examine the impact of the complexity of the environment on feature extraction, we selected three representative scenarios. (1) A centralized environment, characterized by a limited number of objects located close together, such as an indoor corridor, results in a concentrated distribution of features, making it a suitable target for our proposed attacks. (2) A linear environment, characterized by objects arranged in a linear structure from near to far, such as a highway, results in a mostly concatenated distribution of features, which makes it possible to attack more features by using a 3D object close to the victim RV. (3) A decentralized environment, characterized by objects scattered throughout the frame, results in a scattered distribution of features, making it unsuitable for our attacks.

The distribution of features at varying distances from both the victim RV and the target object are depicted in Figures 15a-c. A gradual decrease in distance is observed to coincide with an increase in the number of features present on the target object. This phenomenon can be attributed to the fact that as the distance between the two entities decreases, the number of pixels comprising the target object also increases, leading to a greater level of detail being present at close range. As the distribution of features is contingent upon the distance between the victim RV and the target object, it is important to consider the suitability of different distances for potential attack scenarios. At far distances, the absence of features renders the target object unsuitable for attack. Conversely, at close distances, the abundance of features presents challenges for certain types of attacks, such as the pose deviation and relocalization attacks. As such, it can be inferred that the optimal range for targeted attacks is at medium distances, where the number of features is sufficient for successful execution while still being manageable.

Moreover, the presence of shadows under varying lighting conditions must also be discussed when evaluating potential attack scenarios. As Figure 17 shows, shadow edges are readily identified as features, making them advantageous targets for attack. Note that projection vector is suitable for this scenario, and the shape of the shadow can be flexibility manipulated through a projector in the execution of such attacks.

1.2. Temporal Continuity. The ability to maintain consistency of features across consecutive keyframes is an important factor to consider in evaluating potential attack scenarios. The speed of the victim RV is a key determinant of this continuity. As depicted in Figure 20a, when the RV is driving at high speeds, small objects on the side of the road may be difficult to detect as map points due to the rapid changes in position of the target object. Conversely, at relatively low speeds (Figure 20b), small traffic signs can be easily detected as map points. Furthermore, large objects are guaranteed to be visible across multiple keyframes, eliminating this issue (Figure 20c). In light of these considerations, the scenario of small objects at high speeds can be disregarded as they are unlikely to provide features. Additionally, the pose deviation attack focuses on attacks within consecutive keyframes, making both the high and low speed of the RV unsuitable in this case.

2. Evaluation Details

Datasets *(1) KITTI.* The KITTI dataset, primarily geared towards the autonomous driving domain, is a widely acclaimed dataset used for benchmarking machine learning and computer vision algorithms. Collected predominantly in varied urban settings, this dataset provides a comprehensive suite of sensor data, encompassing 3D point clouds, and images (comprising both stereo and RGB imagery). The data in KITTI are collected from a standard station wagon with two high-resolution color and grayscale video cameras. Additionally, it includes object annotations for each scene. For each frame in the data trace, there are 15 cars and 30 pedestrians at most. The diversity and richness of data make KITTI uniquely positioned for tasks such as object detection and tracking, road detection, and semantic segmentation, crucial for the performance of autonomous vehicles. Overall, it is an outdoor dataset captured from cities and highways with automobiles.
(2) 4Seasons. The 4Seasons dataset is a versatile dataset specifically tailored for autonomous driving and urban navigation tasks under diverse and dynamic environmental conditions. It is collected in various urban and rural environments across multiple seasons, times of day, and weather conditions, providing a rich and varied data source for robust algorithm development. The dataset's emphasis on environmental variability and seasonal changes makes it uniquely suited for addressing challenges related to domain adaptation and robust autonomous driving in real-world conditions. Overall, it is an outdoor dataset captured in diverse urban and rural environments under varying conditions.
(3) EuRoC. Distinctly different from KITTI, the EuRoC dataset is tailored to meet the demands of robotics, specifically Micro Aerial Vehicles (MAVs). The dataset encompasses high-fidelity visual-inertial data recorded across a variety of challenging indoor environments. The data are collected in the ETH machine hall. Such an environment-specific data collection makes EuRoC an excellent resource for the development and evaluation of algorithms tackling visual-inertial odometry, 3D reconstruction, and vSLAM, all pivotal for the navigation and localization tasks of MAVs. Overall, it is an indoor dataset collected in the ETH machine hall with MAVs.
(4) TUM. The TUM dataset stands out for its extensive utility in developing and testing algorithms for visual odometry and vSLAM. Primarily gathered in different indoor scenarios using a small automatic vehicle with a RGB-D camera, this dataset is designed with a specific focus on indoor robotics, e.g., Automatic Guided Vehicle (AGV). What sets TUM apart is the inclusion of ground truth trajectory data, which is obtained from a high-accuracy motion tracking system. This ground truth data enables researchers to carry out detailed performance evaluations of algorithms, particularly those aimed at indoor robotic navigation and mapping. Overall, it is a dataset with various indoor scenes focusing on AGVs.
vSLAM Frameworks *(1) ORB-SLAM2.* ORB-SLAM2 is an efficient and popular vSLAM system. Notably, ORB-SLAM2 can concurrently compute the trajectory of the camera and construct a sparse 3D reconstruction of the scene. Its robustness extends to handling scenarios of intensive motion clutter, enabling wide baseline loop closing, and relocalization, supplemented with an automatic initialization function. The advancements over its predecessor, the original ORB-SLAM, are primarily manifested in an enhanced mapping and loop closing system, providing increased robustness and functionality.
(2) DynaSLAM. DynaSLAM is an advanced extension of the ORB-SLAM2 framework. It differs from ORB-SLAM2 in its proficiency in managing dynamic objects. Traditional vSLAM frameworks often stumble when dealing with non-static environments. DynaSLAM addresses this challenge by distinguishing and isolating dynamic objects from the static background. The outcome is a more dependable mapping and tracking of the environment.
(3) ORB-SLAM3. ORB-SLAM3 is the most popular open-source vSLAM framework. It introduces a novel, robust, real-time monocular vSLAM method capable of handling dynamic scenarios and improved relocalization. Besides, it introduces the concept of a multi-map system, which allows the system to start a new map when the previous map is lost and then merge the maps when the same area is revisited. This functionality is not present in ORB-SLAM2.
Real-world Scenarios *(1) Pose Deviation Attack* (Figure 18(a)). In this attack scenario, a corner is selected as the target. The UGV advances slowly, executing a right turn upon reaching the wall. Since features are extracted from multiple switches, we use a projector to project a complex image, a large tree, onto the wall, thereby generating a cluster of extra features. The features are converted into map points through fixing the image for a while. Then we animate the tree to the right after the local mapping module stores the features into the map. The estimated position of the robot undergoes a lateral shift, ultimately resulting in a collision with the wall on the left during an attempt to return to its original position.
(2) Relocalization Attack (Figure 18(b)). In this attack scenario, an indoor corridor is selected as the target. The UGV follows the wall while navigating. Features are obtained from the front door and ceiling. To effectively overwrite nearly all features and map points ahead, another UGV equipped with a piece of white paper is controlled to overtake from the left side of the victim UGV and move to its front. The map points number in the remaining right corner would be less than 10, and the robot re-estimates its pose through a relocation operation, which takes approximately 12 seconds.

TABLE 1: The relationship between three attack vectors and different scenarios under each attack.
(sticker: 2D sticker ✳: projection ⬢: 3D object)

Scenario Type			Pose Deviation Attack	Relocalization attack	Map Point Manipulation Attack	Illusion Creating Attack
Feature Extraction	Complexity	Centralized Environment	🗒✳⬢	🗒⬢	🗒⬢	🗒✳⬢
		Linear Environment	⬢	⬢	⬢	⬢
		Decentralized Environment	-	-	-	-
	Distance	Close to the robot	-	-	🗒⬢	🗒✳⬢
		Middle Distance	🗒✳⬢	⬢	⬢	⬢
		Far from the robot	-	-	-	-
	Light	Shadow	✳	✳	✳	✳
Temporal Continuity	Velocity	High Speed (Small Object)	-	-	-	-
		Low Speed (Small Object)	-	🗒⬢	🗒⬢	🗒✳⬢
		High Speed (Large Object)	-	⬢	⬢	⬢

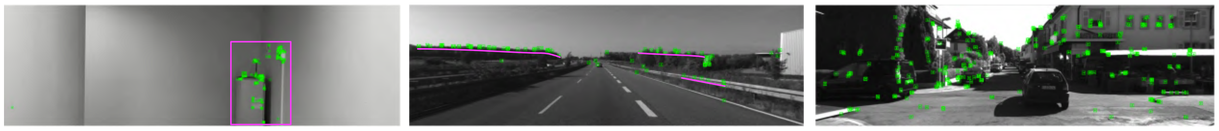

(a) Centralized Environment - Corner (b) Linear Environment - Highway (c) Decentralized Environment - City

Figure 14: The relationship between environmental complexity and map point distribution.

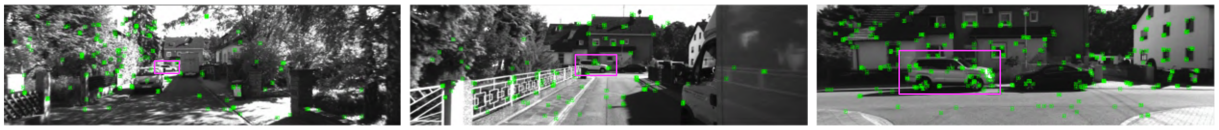

(a) Long Distance (b) Middle Distance (c) Short Distance

Figure 15: The relationship between distance and map point distribution.

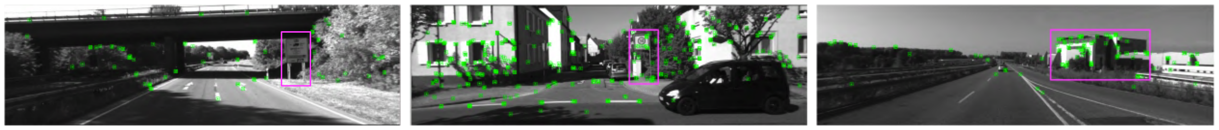

(b) High Speed - Small Object (a) Low Speed - Small Object (c) High Speed - Big Object

Figure 16: The relationship between vehicle speed and map point distribution.

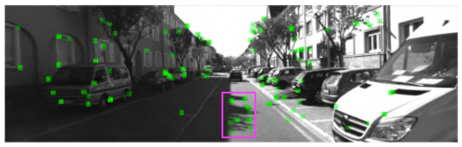


Figure 17: The relationship between shadow and map point distribution.

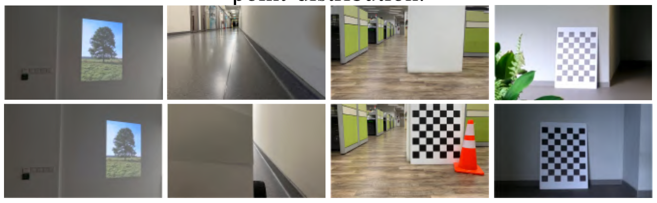

(a) Pose Deviation Attack (b) Relocalization Attack (c) Map Points Manipulation Attack (d) Illusion Creating Attack

Figure 18: The four proposed spoofing attack scenarios in physical world.

(3) Map Point Manipulation Attack (Figure 18(c)). In this attack scenario, a laboratory with a large area of white walls is selected as the target. The UGV moves around the laboratory and returns to its starting location. On its second pass through the attack area, a traffic cone and a whiteboard marking with mosaics are placed in front of the white walls. These spoofing vectors generate a large number of additional features, which are eventually converted into new map points after a while. The large discrepancy in the distribution of map points before and after the two passes results in the loop closing failure.

(4) Illusion Creating Attack (Figure 18(d)). In this attack scenario, two similar areas within an outdoor corridor are selected as targets, characterized by their extensive white wall sections resulting in sparse map points. The UGV initially moves in a direct route to location A, and then drives around and proceeds to location B. Under normal conditions, the UGV can successfully estimate its pose and delineate an accurate trajectory. We introduce identical features and map points to both areas by deploying a whiteboard board with mosaic markers. As the UGV reaches location B, the high similarity between the map points of two regions triggers the loop-closure module, causing a false loopback detection. This makes the UGV incorrectly estimate that it continues to navigate within location A, and draw a wrong map.

TABLE 2: Notations adopted in this work.

Notation	Description	Notation	Description
SLAMTricker	A vSLAM Attack Framework	G	Graph Optimization Model in vSLAM
Φ	Pre-processing Feature Extraction Function in vSLAM	ρ	Projection Function in vSLAM
Θ	Feature Searching Function in vSLAM	Ψ	Attack Behavior
HDist	Calculate Hamming Distance	BinIdx	Calculate Bin Index
$\mathcal{A}$	Adversary's Capability	$\mathcal{C}$	Environment Conditions
$\mathcal{L}_{atk}$	Spoofing Loss	V	Victim Map Point Container
Top3BinIdx	A Set of Top 3 Bin Indexes	p	Estimated robot's Pose
p'	Estimated robot's Pose under Attack	x	An Image
M	Pristine 3D Map Points	m	Calculated 2D Map Points
K	Pristine 3D Features	k	Calculated 2D Features
S	Malicious 3D Features from Spoofing Vectors	s	Calculated Malicious 2D Features
m^s	Influenced 2D Map Points by Attacks	$\Phi(S)_l$	Malicious 2D Features at Level l
k'_l	Extracted 2D Points at Level l	M_t	3D Map Points at Time t
T	4×4 Transformation Matrix	R	3×3 Rotation Matrix
r	3×1 Translation Matrix	I_C	Camera Intrinsic Matrix
n_m	Number of Map Points and 2D Features	n'_m	Number of High Density Map Points
n_k	Number of 3D Features	n_S	Number of Malicious 3D Features
n_s	Number of Influenced Map Points	d_{m_i}	Density for 2D Map Point m_i
l	Downsampling Level	l_{m_i}, l_{k_i}	Level for Point m_i, k_i
ϵ	Downsample Factor	ϵ^l	Downsample Factor at Level l
f_x, f_y	Focal Length	c_x, c_y	Optical Center Offset
R	Attack Ratio	α, β	Rotation Constraint
τ	Threshold of Distance between Map Points	γ	Threshold of Point Density
r	Searching Radius		

TABLE 3: Attack contexts of our experiments on datasets.

Spoofing Attack	**Target Module**	**Target Scenario (Dataset-ID)**	**Deployment Methods**	**Attack Goal**
Pose Deviation	Tracking	Three-way junction (4Seasons-Office Loop)	Features Shifting	Pose Deviation
Relocalization	Tracking	Suburban Highway (KITTI-01)	Features Covering	Localization Failure
Map Point Manipulation	Local Mapping	Suburban Highway (KITTI-01)	Features Covering	Map Points Reduction
Illusion Creating	Loop Closing	Three-way junction (4Seasons-Neighborhood)	Features Adding	Pose Deviation

TABLE 4: Robustness analysis in the physical world.

Spoofing Attack	**Target Module**	**Target Scenario**	**Attack Vectors**	**Deployment Methods**	**Attack Goal**
Pose Deviation	Tracking	Indoor Room	Light Projection	Features Shifting	Crash
Relocalization	Tracking	Indoor Corridor	Object Placement	Features Covering	Loss Time for Recovery
Map Point Manipulation	Local Mapping	Indoor Room	Object Placement	Features Adding	Loop Closing Failure
Illusion Creating	Loop Closing	Outdoor Corridor	Object Placement	Features Adding	Mapping Task Failure

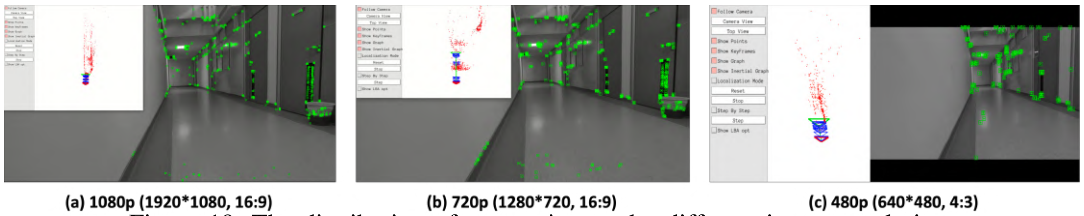


Figure 19: The distribution of map points under different image resolutions.

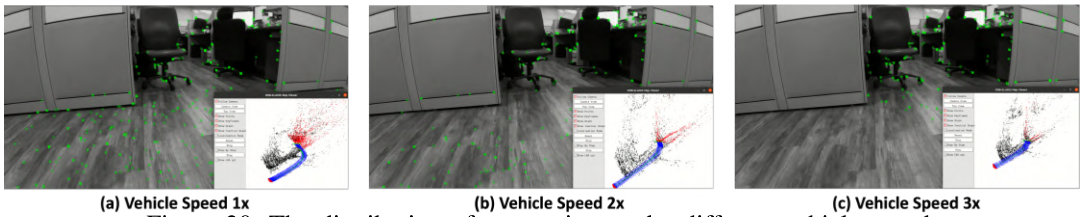
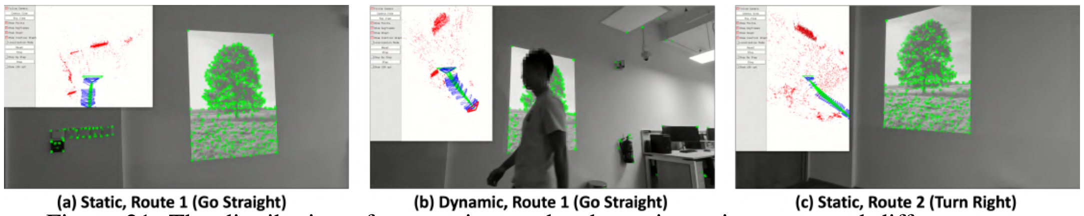


Figure 20: The distribution of map points under different vehicle speeds.


(a) Static, Route 1 (Go Straight)
(b) Dynamic, Route 1 (Go Straight)
(c) Static, Route 2 (Turn Right)
Figure 21: The distribution of map points under dynamic environment and different routes.