University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Cybersecurity

Visible Light Communication Attacks on Autonomous Vehicle vSLAM Systems

Supervisor
Prof. Alessandro Brighente
University of Padova

Master Candidate
Vinu Varshith Alagappan

Student ID
2041267

Academic Year
2023-2024

Abstract

Autonomous vehicles (AVs) heavily depend on Visual Simultaneous Localization and Mapping (vSLAM) systems for real-time navigation, obstacle detection, and decision-making. vSLAM systems rely on camera inputs to interpret road features, such as traffic lights, vehicle brake lights, and turn signals. This dependency introduces vulnerabilities to adversarial light-based attacks, particularly in scenarios where these visual cues can be manipulated. This project investigates the potential of exploiting Visible Light Communication (VLC) to subtly modulate external light sources, such as headlights and brake lights of nearby vehicles, to deceive the AV's vSLAM system. These manipulations could lead to false perceptions of braking, lane changes, or traffic congestion, resulting in unsafe AV responses, such as abrupt braking, improper lane changes, or incorrect navigation adjustments. This study presents a threat model detailing attacker objectives and capabilities, followed by a simulation-based experimental framework to evaluate the effects of light-based disruptions on vSLAM systems. The findings underscore the critical need for improved security in AV perception systems to safeguard against adversarial light-based manipulations.

Contents

Listing of figures

Listing of tables

Listing of acronyms

AV ..... Autonomous Vehicles

SLAM ..... Simultaneous Localization and Mapping

vSLAM ..... Visual Simultaneous Localization and Mapping

SIFT ..... Scale-Invariant Feature Transform

ORB ..... Oriented FAST and Rotated BRIEF

ADAS ..... Advanced Driver Assistance Systems

VLC ..... Visible Light Communication

V-VLC ..... Vehicle-to-Vehicle Visible Light Communication

LSD-SLAM .... Large-Scale Direct Simultaneous Localization and Mapping

PTAM ..... Parallel Tracking and Mapping

ICP ..... Iterative Closest Point

LOAM ..... Lidar Odometry and Mapping

FGR ..... Fast Global Registration

GPS ..... Global Positioning System

IMU ..... Inertial Measurement Unit

1 Introduction

The domain of autonomous vehicles (AVs) is undergoing rapid evolution, situated at the convergence of artificial intelligence, robotics, and transportation systems. These vehicles represent a significant technological advancement that has the potential to fundamentally change human interaction and mobility within various environments. The potential benefits of autonomous vehicles include improvements in road safety, reductions in traffic congestion, and decreased environmental impact, drawing the attention of researchers, industries, and policymakers. Autonomous vehicles (AVs) attain this level of independence through the integration of various sensors and algorithms. These components enable the vehicles to navigate intricate environments, identify obstacles, and execute decisions regarding speed, lane changes, and braking maneuvers. A key element of this system is Visual Simultaneous Localization and Mapping (vSLAM), enabling autonomous vehicles (AVs) to generate a real-time map of their environment through the analysis of visual data captured by cameras. The data generally involves factors including lane markers, road signs, other vehicles, and various environmental features.

The vSLAM system is particularly reliant on detecting the behaviors of nearby vehicles through their headlights and brake lights. Headlights provide crucial information about the presence and movement of oncoming or nearby vehicles, especially in low-visibility conditions, while brake lights signal that a vehicle is slowing down. These visual cues are integral to the AV's decision-making process, enabling it to maintain safe distances, avoid collisions, and operate smoothly in traffic. However, this dependence on visual inputs also makes vSLAM systems vulnerable to light-based attacks.

The vSLAM system depends especially on detecting the actions of other cars using their brake lights and headlights. Particularly in low-visibility conditions, headlights provide vital information regarding the presence and movement of approaching or surrounding cars; brake lights indicate that a vehicle is slowing down. The AV's decision-making process depends on these visual signals; they help it to keep safe distances, prevent collisions, and run smoothly in traffic. But this reliance on visual cues also renders vSLAM systems open to light-based attacks.

Though driverless cars depend on vSLAM systems, they are not without flaws. By means of subtle attacks such as adversarial patching, which can compromise feature detection, adversaries can leverage these systems. This therefore compromises the accuracy with which the system maps and localizes. Research by Baodong Chen et al. [4] for example showed that these adversarial patterns can significantly change vSLAM trajectories, therefore causing mistakes over seven times higher than usual (SLAM). Analogous to this, the "Phantom of the ADAS" study emphasizes how transient visual cues such as false road signs or pedestrians may mislead Advanced Driver-Assistance Systems (ADAS) into forming incorrect interpretations Phantom) [5]. These results highlight precisely how sensitive camera-based systems are to such focused manipulations, so it is imperative to create techniques resistant to such attacks.

At the same time, visible light communication (VLC) technologies, which use vehicle headlights and taillights to transmit information, are becoming an exciting addition to vehicular communication. Including VLC into current lighting systems is not easy, though. Careful thought is needed to make sure these systems stay free from vulnerabilities, consistent with laws, and safe (IR) [6]. These difficulties taken together emphasize the pressing need for strong plans to safeguard autonomous vehicle systems, therefore ensuring their dependability even in the face of physical and hostile disturbances.

1.1 Background on Autonomous Vehicles (AVs)

Over decades of study and development, autonomous cars have developed from simple basic notions into working prototypes. Defined by the Society of Automotive Engineers (SAE), these cars fall under several degrees of autonomy: Level 0 (no automation) to Level 5 (complete automation). At Level 5, an AV is totally self-sufficient and capability of running in any surroundings free of human intervention. (SAEs International)

One of the main features of AVs is their dependence on a set of sensors comprising:

- **Cameras:** For visual recognition and obstacle detection.

- **LiDAR:** To generate detailed 3D maps of surroundings.
- **Radar:** For detecting objects in various weather conditions.
- **Ultrasonic Sensors:** For close-range obstacle detection.
- **GPS:** For global positioning and route tracking.

Combining these sensors with sophisticated machine learning algorithms forms AVs' perception module. Real-time processing of the gathered data helps one to understand the surroundings, project the behavior of other entities, and guide actions. Successful deployment of autonomous cars in controlled environments including mines, ports, and some metropolitan areas has shown their possibilities for more general uses.

AVs have significant social influence. The World Health Organization (WHO) estimates that over 90% of road accidents worldwide are caused by human mistakes, so they want to reduce them. AVs can also save carbon emissions by streamlining routes and cutting idle times, therefore supporting environmental sustainability. (World Health Organization)

1.2 Visible Light Communication in Vehicles

Visible Light LED-based headlights and taillights used as transmitters as well as photodiodes (PD) and cameras used as receivers allows vehicles to communicate with one another. Active safety systems in cars include taillights and headlights. They enable the drivers to control the road ahead (i.e., forward lighting) and to transmit information to the traffic on the existence, dimensions, as well as the present maneuver of the vehicle. Their adaption for communication needs several difficulties, then. First, any changes to the lighting modules should not compromise the main purposes of llumination and signaling. Neither should the lifetime of the lighting modules nor the illumination quality. Second, automobile lighting modules are tightly controlled in line with the safety system. This adds more limitations about possible modifications, particularly if they make lighting modules dangerous for highways. Under such circumstances, the whole V-VLC idea becomes unfavorable.

Apart from system-level consequences, more crucial are the issues related to communication. As was already indicated, V-VLC has specific properties that would benefit vehicle communications as a complementing access method. As a rather new technology, there are still unresolved issues that must be addressed before V-VLC is accepted as safe for use on roadways.

1.3 Simultaneous Localization and Mapping(SLAM)

SLAM [7], which stands for Simultaneous Localization and Mapping, is a cutting-edge technology that allows robots and autonomous vehicles to navigate and understand their surroundings. Essentially, SLAM enables these systems to create a map of an unknown environment while simultaneously figuring out their location within it. This dual functionality is a game-changer for navigating new and dynamic spaces efficiently.

Imagine a robot vacuum cleaner as an example. Without SLAM, it would move aimlessly around the room, missing spots or repeatedly cleaning the same area, wasting both time and battery life. With SLAM, the vacuum uses its sensors, like cameras or laser scanners, to see the space around it. It builds a map of obstacles, furniture, and open areas while keeping track of its position in real-time. This allows it to clean methodically, avoid redundant paths, and conserve battery power.

But the applications of SLAM go far beyond household robots. In warehouses, fleets of robots equipped with SLAM can arrange shelves, transport items, and streamline inventory processes. In the skies, drones use SLAM to navigate and deliver packages in uncharted territories. For autonomous vehicles, SLAM is essential for parking in tight spaces, safely navigating busy streets, and avoiding obstacles in real-time.

SLAM has become more practical and accessible in recent years due to advancements in computing power and the availability of low-cost sensors like lidar and cameras. Technologies such as visual SLAM (using cameras), lidar SLAM (using lasers), and multi-sensor SLAM (combining data from multiple sensors) have made this technology adaptable for a wide range of uses. Developers can leverage tools like MATLAB and Simulink to integrate SLAM with other functions such as object tracking, sensor fusion, and path planning, enabling them to build robust systems for real-world applications.

In summary, SLAM is at the heart of many modern robotic and autonomous systems, enabling them to perform complex tasks like navigating unknown environments, planning routes, and avoiding obstacles with precision. Whether it's for a simple home vacuum or a sophisticated self-driving car, SLAM is the foundation of smarter, more efficient navigation.

1.3.1 Visual SLAM

Visual SLAM, or vSLAM, relies on images captured from cameras and other imaging sensors to perform simultaneous localization and mapping. It can use a variety of cameras, including

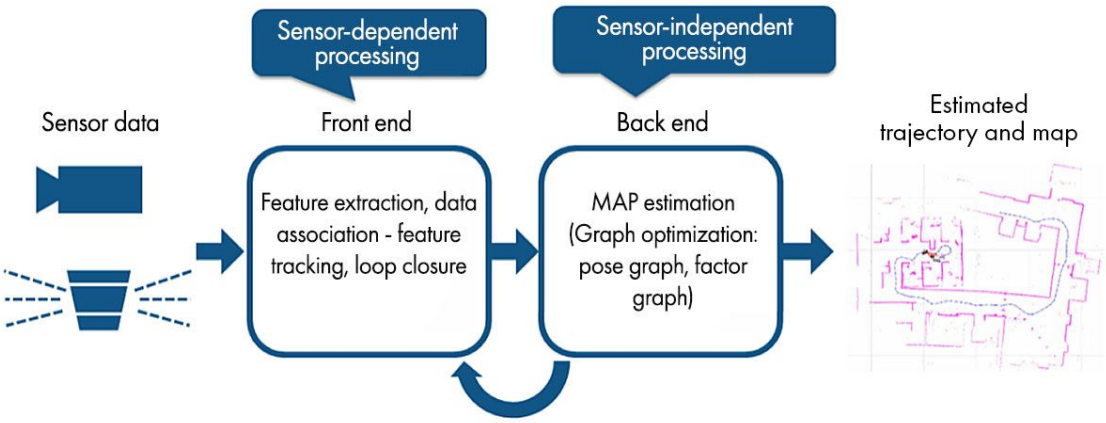

Figure 1.1: SLAM Model [1]

simple ones like wide-angle, fish-eye, or spherical cameras, as well as more advanced options like compound eye cameras (stereo and multi-camera setups) or RGB-D cameras, which include depth-sensing capabilities such as ToF (Time-of-Flight).

One of the major advantages of visual SLAM is its cost-effectiveness, as it can be implemented using relatively inexpensive cameras. Cameras also provide a wealth of information, making them suitable for detecting landmarks previously measured positions within the environment. This capability can be further enhanced by combining landmark detection with graph-based optimization, allowing for a flexible SLAM implementation.

Monocular SLAM is a specific approach within vSLAM that uses only a single camera. Although this makes defining depth a challenge, the problem can be addressed by detecting known objects such as AR markers or checkerboards in the image for localization, or by integrating camera data with other sensors like inertial measurement units (IMUs), which track velocity and orientation. Related technologies include structure from motion (SfM), visual odometry, and bundle adjustment.

Visual SLAM algorithms generally fall into two categories. Sparse methods match feature points in images using algorithms like PTAM and ORB-SLAM, while dense methods analyze overall image brightness with algorithms like DTAM, LSD-SLAM, DSO, and SVO.

1.3.2 Lidar SLAM

Light Detection and Ranging (lidar) is a technology that relies on laser sensors, or distance sensors, to measure distances with exceptional precision. Compared to cameras, Time-of-Flight

(ToF) sensors, and other alternatives, lidar is significantly more accurate, making it ideal for applications involving high-speed vehicles like self-driving cars and drones. The data captured by lidar sensors typically consists of 2D (x, y) or 3D (x, y, z) point clouds. These point clouds provide highly precise distance measurements, making them particularly effective for constructing maps with SLAM algorithms.

Lidar-based SLAM estimates movement by sequentially registering point clouds. The computed motion, representing the traveled distance, is used to localize the vehicle. Relative transformations between point clouds can be determined using registration algorithms such as iterative closest point (ICP) or normal distributions transform (NDT). Alternatively, feature-based approaches like Lidar Odometry and Mapping (LOAM) or Fast Global Registration (FGR), which leverage Fast Point Feature Histograms (FPFH), can also be employed. Point cloud maps can be represented as either grid maps or voxel maps.

To address challenges in localization, lidar data is often fused with other measurements, such as wheel odometry, global navigation satellite systems (GNSS), and inertial measurement unit (IMU) data. While 2D lidar SLAM is commonly used for applications like warehouse robots, 3D point clouds are typically employed for UAVs and autonomous driving systems.

1.3.3 Multi-Sensor SLAM

Multi-sensor SLAM is an advanced type of SLAM algorithm that combines data from various sensors, such as cameras, inertial measurement units (IMUs), GPS, lidar, radar, and more, to improve precision and reliability. By leveraging the unique strengths of each sensor and compensating for their individual weaknesses, multi-sensor SLAM delivers enhanced performance. For example, cameras provide rich visual details but may struggle in low-light or high-speed conditions, whereas lidar performs reliably across different lighting scenarios but can face challenges with certain surface materials. By integrating data from multiple sources, multi-sensor SLAM creates a more robust and dependable solution compared to single-sensor approaches.

A key component of multi-sensor SLAM is the factor graph framework, which is both modular and adaptable. It integrates diverse sensor inputs like cameras, IMUs, and GPS and can accommodate custom data sources such as lidar and odometry by converting them into pose factors. This flexibility supports various configurations, including Monocular Visual-Inertial SLAM and Lidar-IMU SLAM, making multi-sensor SLAM a versatile choice for complex applications.

SLAM systems are designed to handle real-world challenges such as dynamic environments,

sensor noise, and computational constraints. For instance, in urban settings, SLAM must navigate complex scenarios involving moving pedestrians, vehicles, and changing traffic signals.

2 Background

2.1 Visual Simultaneous Localization and Mapping (vSLAM)

This chapter covers the basics of building a Visual SLAM algorithm. It starts by explaining how images are represented in computer systems, helping to understand how visual data is processed and used. Next, it discusses feature extraction and matching key techniques for identifying and tracking important points in an image. These concepts form the foundation of how a Visual SLAM system maps its environment and keeps track of its position.

2.1.1 Image representation

In this initial section, we will explain how the key information required for processing is represented. Additionally, we will describe the primary sensor that will be utilized and provide its mathematical representation.

Images

In computational systems, images are commonly represented as matrices, where each element corresponds to a pixel in the image. The value stored in each matrix entry depends on the type

of image and the information it conveys. For this project, three types of images are utilized, each serving a distinct purpose.

Colored images store the intensity values of three color components blue, green, and red for each pixel. The order of these components is crucial, as different systems may use varying standards. For example, OpenCV's 'imshow' function defaults to a BGR representation, whereas other components, such as neural networks used for depth estimation, assume an RGB format. Typically, each color intensity is represented using 8 bits, providing sufficient detail for applications requiring rich environmental information.

Grayscale images, formatted similarly to colored images, reduce the data stored in each matrix entry to 8 bits, representing the intensity of gray captured by the camera for each pixel. These images are significantly lighter in memory compared to colored images while still providing enough information for visual SLAM algorithms to compute descriptors and track camera movement. However, tasks such as depth estimation often require colored images, as they carry more comprehensive environmental details, simplifying the neural network's learning process.

Depth images resemble grayscale images but typically use 16 bits per pixel to encode depth information. This allows them to represent distances in millimeters, covering ranges of up to 65 meters. Since depth images often contain higher resolution data, they need to be rescaled to a range of 0 to 255 for visualization purposes. This transformation ensures compatibility with display systems while preserving the critical depth information for mapping and localization tasks. Together, these image types contribute to processing and understanding the visual environment in SLAM applications.

2.1.2 Feature extraction and Matching

This section delves into one of the fundamental algorithmic elements required for implementing a feature-based visual SLAM algorithm. Visual SLAM methods are generally categorized into two major groups: feature-based methods and direct methods.

Feature-based methods are the focus of this thesis, as ORB-SLAM3 belongs to this category. These methods extract features and compute descriptors for a specific frame, then attempt to identify the same points in subsequent images. By analyzing how these points shift between frames, the algorithm computes the camera's trajectory. In contrast, direct methods operate on the entire image, bypassing feature extraction. These methods aim to find a transformation matrix that minimizes the photometric error, differing from feature-based methods, which focus on minimizing the reprojection error.

A core component of any feature-based SLAM algorithm is the features used within the program. A feature is simply a specific point in an image represented in a straightforward manner. A feature point must meet the following criteria:

- Repeatability: The same point should be easily re-detected across multiple images taken from different perspectives.
- Distinctive: The point should be distinguishable from other points in the image, particularly from other features within the same image.
- Efficiency: Features should form a small subset of the image's pixels and must be identifiable using a computationally efficient algorithm, which is especially crucial for SLAM applications.
- Locality: Identifying a feature should require only a small subset of the image's pixels.

Various types of features can be extracted from an image. A feature is associated with corresponding key points (the feature's position in the image), the method used to detect them, and their descriptors (usually a vector describing the key point). In this thesis, SIFT features will be introduced first as a foundational concept. ORB features, which were developed more than a decade after SIFT, will then be discussed as a more efficient alternative.

Understanding these features and their characteristics is essential for building robust feature-based SLAM algorithms, as they form the foundation for tracking and mapping within the SLAM framework.

SIFT. To discuss SIFT (Scale-Invariant Feature Transform), we first need to understand how feature points are detected. The process begins by computing the scale space of the image, which is defined as:

$$S(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

where

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

is a Gaussian function, and I(x,y) represents the image. The scale space allows us to analyze the image at multiple scales, making it possible to identify features that are consistent regardless of size or scale.

Next, we compute $\sigma^2 \nabla^2 S(x, y, \sigma)$, which is the Laplacian of Gaussian applied to the scale space. By identifying local maxima in this function for a given (x,y,σ), we can locate candidate feature points. However, not all points are retained; only the most promising candidates

are kept to reduce noise and ensure the quality of features. Importantly, this process also determines the characteristic scale σ for each feature point, enabling SIFT to achieve scale invariance.

After detecting a candidate feature point, the next step is to determine its principal orientation. This is done by calculating the gradient of pixels near the candidate point, where "near" is defined relative to the characteristic scale. Using these gradients, an orientation histogram is created, and the most frequent orientation is selected as the principal direction of the feature. This step ensures orientation invariance.

The final step in SIFT is constructing a descriptor, or "signature," for each feature point. The descriptor is generated by building normalized histograms of gradients within the region corresponding to the feature point. The region is rescaled according to the characteristic scale and rotated to align with the principal direction. The result is a vector representation of the feature point, and when two descriptors are sufficiently similar, a match can be established.

SIFT features are versatile and effective, finding applications in areas beyond SLAM, such as stitching images to create panoramic photographs. However, while SIFT is a powerful feature extractor and descriptor, it comes with a high computational cost. Over the years, efforts have been made to create more efficient alternatives. The first significant improvement came with SURF (Speeded-Up Robust Features), but ORB (Oriented FAST and Rotated BRIEF) emerged later as a superior option. ORB offers better performance and is widely used, including in ORB-SLAM3, the focus of this thesis. Further details on SIFT can be found in the original paper introducing the method. [8]

ORB. To describe the ORB feature, we begin by explaining how features are extracted and then how descriptors are computed. ORB relies on FAST for keypoint detection. FAST is an efficient corner detection algorithm that extracts points using the following steps:

- Step 1: Select a pixel and check its brightness.
- Step 2: Consider the 16 pixels nearest to the selected pixel.
- Step 3: If N consecutive points among these 16 pixels have a brightness greater than the brightness of the selected pixel plus a threshold, or smaller than the brightness of the selected pixel minus the threshold, then that pixel is selected as a feature point.

Since FAST does not inherently provide information about scale or rotation, ORB modifies this detector to achieve scale and rotation invariance. ORB builds a pyramid of subsampled images and detects features at each level of the pyramid. Matches can then occur between points

detected in different levels of the pyramid across different images.ORB uses the intensity centroid method to determine rotation. The centroid is calculated as:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right)$$

where

$$m_{pq} = \sum_{x,y \in B} x^p y^q I(x,y), \quad p,q \in \{0,1\}$$

The rotation direction of the feature is obtained by connecting the geometric center of the feature with the centroid.

Having achieved scale and rotation invariance through modifications to FAST, the next step is computing the descriptor for each feature point. ORB uses the BRIEF descriptor, which is a 128-bit array. Each element in this array (either 1 or 0) is determined by selecting two random points within the feature and comparing their intensities. If the first point has a higher intensity, the value is 1; otherwise, it is 0. ORB slightly modifies BRIEF to improve rotation invariance, which explains the name "Oriented FAST and Rotated BRIEF."

Once features are extracted, feature matching becomes a significant challenge. A naive approach would involve matching features in two images based on a distance metric, such as Euclidean distance for SIFT descriptors or Hamming distance for ORB descriptors. However, this can lead to issues. For instance, in images with repetitive textures, many feature points with similar descriptors may exist, resulting in incorrect matches. A potential solution to this problem is discussed in [9].

In conclusion, ORB combines FAST for efficient feature detection with modifications to achieve invariance and BRIEF for compact and robust descriptors, making it a powerful alternative for feature-based SLAM. For a detailed explanation, refer to the original paper [10].

2.1.3 Monocular visual SLAM

One of the pioneering real-time monocular visual SLAM implementations was "MonoSLAM" by Davison (2007) [11]. This algorithm used the Extended Kalman Filter (EKF) paradigm to achieve real-time localization at 30 Hz over a small area, creating a sparse map of highly consistent features. MonoSLAM utilized an inexpensive IEEE 1394 webcam with a wide-angle lens. Since depth information cannot be derived from a single image, the main challenge was localizing observed features in 3D space. MonoSLAM approached this by initializing every feature

as a 3D line extending from the camera's optical center toward the feature's image plane projection. Discrete depth hypotheses were distributed along this line, and their likelihoods were iteratively updated based on subsequent observations. Once the probability distribution became sufficiently peaked, the feature was fully initialized as a 3D Gaussian distribution. Feature detection was performed using the Shi and Tomasi detector, with robust feature matching enhanced by storing small image windows around detected features. These windows were warped over time to accommodate parallax and scale changes. Despite its groundbreaking achievement in enabling real-time monocular SLAM on inexpensive hardware, MonoSLAM's maps were sparse, and EKF-SLAM was prone to divergence in dense maps due to false data associations and its quadratic computational complexity.

Kwok, Ha, and Fang (2007) [12] introduced a cost function-based method for data association in bearing-only visual SLAM. Unlike range-and-bearing systems, bearing-only systems lack depth information, making data association more challenging. Traditional approaches rely on the Mahalanobis distance computed from the innovation error (the difference between the predicted and observed landmark state) and the innovation covariance, which accounts for uncertainties in both the known state and measurements. This work introduced a likelihood-based approach that incorporated uncertainty in both landmark state and bearing measurements. A cost function was calculated for each potential association, and the candidate pair with the minimal cost was selected, enhancing accuracy in data association for bearing-only SLAM.

Kootstra, de Jong, and Wedema (2009) [13] compared the EKF-SLAM and FastSLAM paradigms to highlight the limitations of the Extended Kalman Filter approach. Using a Pioneer 2 DX robot equipped with a single camera, they observed the same environment in a loop under reliable and unreliable feature matching settings. In EKF-SLAM, incorrect data associations propagated errors across the entire pose history due to its single-hypothesis tracking mechanism, severely degrading performance. FastSLAM, in contrast, was more resilient to false associations, as its particle-based approach distributed the impact of errors. Depth information was derived through triangulation across buffered image observations. The study demonstrated that FastSLAM outperformed EKF-SLAM in both reliable and unreliable settings.

Joan Sola, Monin Devy, and Lemaire (2008) [14] proposed an approach for undelayed initialization of landmarks in bearing-only SLAM. The method leveraged bearing information to reduce directional uncertainty without requiring immediate depth estimation. Landmarks were modeled as a series of Gaussians, with the most probable hypothesis selected over subse-

quent observations. Once sufficient information was gathered, the landmark was initialized as a full 3D Gaussian distribution.

Civera, Davison, and Martinez Mondiel (2008) [15] addressed the limitations of traditional landmark initialization by introducing an inverse depth parametrization. Standard Gaussian distributions fail to accurately represent landmark uncertainty at first observation, as the landmark's position lies on a cone centered at its projection on the image plane. The inverse depth parametrization described landmarks with a six-element vector, including the camera's pose at the time of first observation, the direction angles, and the inverse depth. This representation allowed landmarks to provide useful orientation information even before sufficient parallax was achieved for accurate depth estimation. Once enough parallax was observed, the landmark could be converted to a Euclidean representation to improve computational efficiency.

Li, Hong, Cai, and Luo (2008) [16] presented a refinement of particle filter SLAM, similar to FastSLAM, by incorporating current measurements into the prediction step. This refinement allowed particles to be sampled more effectively to approximate the true distribution before updating landmark states. The approach improved both pose estimation and map consistency compared to standard particle filter methods, aligning with advances seen in FastSLAM 2.0 by Thrun and Montemerlo.

Mucientes, and Regueiro (2009) Gamallo, Mucientes, and Regueiro (2009) [17] explored the use of omnidirectional cameras in monocular FastSLAM. Using a fish-eye lens-equipped camera with a 185° field of view, they adapted FastSLAM 2.0 to improve temporary particle sampling based on current measurements. The environment, a museum, featured ceiling-mounted lights as landmarks, which were observed with an infrared-filtered camera. The omnidirectional camera enabled comprehensive environmental mapping and efficient SLAM performance.

2.2 Simulation Software

This section outlines the software utilized in the development of this thesis, detailing the specific roles each one played.

2.2.1 MATLAB (Ver. R2024b)

MATLAB [18] is a programming platform built on the MATLAB language, which is designed for matrix-based computations. It provides an intuitive way to express and process mathemati-

cal calculations. First released commercially in 1984, MATLAB has become a cornerstone tool for engineers and scientists focused on algorithm development, data analysis, and the creation of models and applications. Its popularity is attributed to its user-friendly interface (especially compared to other programming languages), its extensive library of packages and toolboxes tailored to specific fields (as discussed in Section 3.3.1), and its strong community support through forums and online platforms. These attributes have also made MATLAB a preferred choice in academia. For instance, the University of Padua offers students a Campus-Wide license, enabling installation on personal computers for academic use.

The decision to use MATLAB in this thesis is driven by these advantages, along with its suitability for tackling Simultaneous Localization and Mapping (SLAM) problems. MATLAB provides a comprehensive set of algorithms for addressing SLAM challenges, making it an ideal platform for implementing these solutions. Additionally, it serves as the central tool to which other software solutions are integrated, ensuring seamless workflow and robust problem-solving capabilities.

Matlab Add-ons

As mentioned earlier, MATLAB's functionality can be enhanced by adding packages and toolboxes tailored to specific fields of interest. Below is an overview of the toolboxes used in this work:

- **Automated Driving Toolbox:** This toolbox offers algorithms and tools for designing, simulating, and testing advanced driver-assistance systems (ADAS) and autonomous driving systems. It was used to generate a 2D map of the environment and track the path for the drone to follow. [19]

- **Computer Vision Toolbox:** Designed for creating and testing computer vision, 3D vision, and video processing systems, this toolbox provides a rich set of algorithms, functions, and applications. It was utilized to implement the V-SLAM methodology required for this project. [20]

- **Image Processing Toolbox:** This toolbox includes a wide range of reference-standard algorithms and workflow applications for image processing, analysis, visualization, and algorithm development. It played a supporting role in tasks involving image manipulation and analysis. [21]

- **Control System Toolbox:** This toolbox provides essential tools for modeling, analyzing, and designing control systems. It supports tasks such as linear system analysis, time and

frequency response computations, and control design, including PID tuning and state-space methods. [22]

- **Robotics System Toolbox:** Focused on robotic system development, this toolbox includes algorithms and functions for motion planning, kinematics, dynamics, and manipulation. It was particularly valuable for designing and testing robotic components of the project. [23]

These toolboxes collectively enabled the successful development and implementation of various algorithms and methodologies required for this work.

2.3 Implement Visual Simultaneous Localization and Mapping in Matlab

Visual simultaneous localization and mapping (vSLAM) involves determining the position and orientation of a camera relative to its environment while simultaneously constructing a map of the surroundings. This process relies exclusively on visual inputs from the camera. vSLAM has various applications, including augmented reality, robotics, and autonomous driving.

Visual SLAM algorithms are typically divided into two categories based on how they estimate camera motion. Indirect methods, also known as feature-based methods, rely on feature points in images to minimize the reprojection error. Direct methods utilize the overall brightness of images to minimize photometric error. MATLAB's Computer Vision Toolbox provides algorithms and functions for implementing feature-based visual SLAM workflows. It also includes the monovslam object, which encompasses the full visual SLAM pipeline. The workflow comprises steps like map initialization, tracking, local mapping, loop detection, and drift correction.

To construct a feature-based visual SLAM pipeline using a sequence of images, the following steps are performed:

1. Initialize Map - Begin by initializing the 3D map points from two image frames. Use triangulation based on 2D feature correspondences to compute the 3D points and the relative camera pose.

2. Track Features - For each new frame, estimate the camera's pose by matching the current frame's features to those in the last key frame.

3. Create Local Map - When a frame is identified as a key frame, create a new 3D map of points. Refine the camera pose and 3D points using bundle adjustment.

4. Detect Loops - Identify loops for each key frame by comparing the current frame to all prior key frames using the bag-of-features method.

5. Correct Drift - Optimize the pose graph to correct drift in the camera poses across all key frames.

The workflow, as depicted in Figure 2.1, illustrates a typical feature-based vSLAM pipeline. It highlights the points where data is stored or retrieved from objects that manage the data, ensuring an efficient and structured approach to visual SLAM implementation.

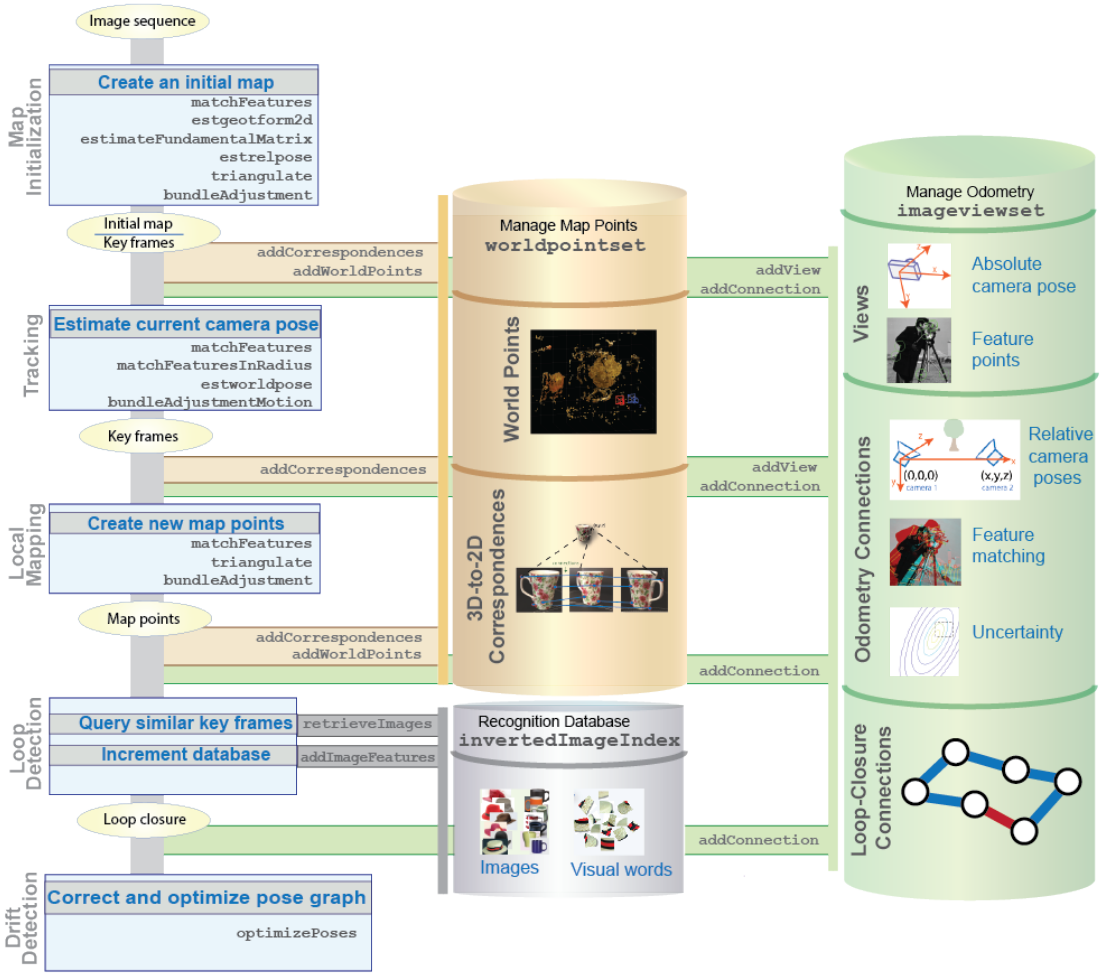

Figure 2.1: Visual SLAM Workflow [2]

2.3.1 Map Initialization

The first crucial phase in the SLAM (Simultaneous Localization and Mapping) system is map initialization, which creates a basis for later mapping and localization. In Visual SLAM, this involves identifying and storing important visual features from the camera's initial frames taken. These features serve as landmarks so the system may orient itself relative to the environment. Usually, the technique guarantees robustness in several contexts by use of feature identification methods such as ORB (Oriented FAST and Rotated BRIEF) or SIFT (Scale-Invariant Feature Transform). MATLAB provides the basis for creating the pose graph by means of the `imageviewset` object, therefore preserving the relationship between camera views and their poses.

Once the key features are found, their spatial linkages help to design the structure of an initial map. This stage usually consists in the computation of the camera's pose using matched features from the surroundings, therefore establishing a reference frame for future operations. MATLAB's functions, such as `addView` and `addConnection`, allow for effectively managing the relationships between views and preserve a constant representation of the initial map. The result is a framework ready to integrate new data as the system navigates its environment.

2.3.2 Tracking

Tracking is the continuous process of estimating the camera's pose as it moves through the environment by comparing consecutive frames. It detects important elements in the present frame and matches them to previously found landmarks on the map. This mechanism guarantees that the system keeps a proper awareness of its orientation and position. For this reason, one often uses optical flow, feature matching, or descriptor-based methods. In MATLAB, functions like `matchFeatures` are pivotal in finding correspondences between frames, and `estimateGeometricTransform` is used to compute the transformation matrix describing camera motion.

The tracking phase is essential for maintaining map consistency and ensuring accurate localization in dynamic scenarios. It constantly changes the camera's orientation in respect to the existing map, therefore allowing exact navigation and environmental interaction. The `imageviewset` object of MATLAB can dynamically add connections and new frames, therefore enabling smooth mapping of tracking results into the map. This stage also guarantees that the SLAM system responds properly to real-world obstacles, so guiding local mapping and loop closure.

2.3.3 Local Mapping

Local mapping integrates fresh data acquired throughout the tracking phase, therefore incrementally creating the map. This involves enhancing the local area to lower mistakes and adding new features or frames to the current map. Local mapping guarantees that the SLAM system stays strong and consistent by adding high accuracy features from the nearby surroundings to the map. Whereas local bundle adjustment maximizes feature alignment and lowers uncertainty, MATLAB's `imageviewset` object helps effective management of views and connections.

Maintaining a thorough and accurate representation of the surroundings depends on this stage. It ensures that the system changes with the times or extra knowledge acquired during navigation. Maintaining a well-organized pose graph helps local mapping also ready the system for global optimization chores including loop closure. MATLAB's tools for feature addition, pose adjustment, and graph optimization simplify this process so that developers may dynamically refine the map in accordance with real-time requirements.

2.3.4 Loop Detection

Loop detection identifies when the system revisits a previously mapped location, allowing it to correct accumulated errors and align the map more accurately. This process is vital for ensuring the global consistency of the SLAM system, as it prevents the map from drifting due to incremental pose estimation errors. The detection of loops relies on comparing features from the current view with those from earlier frames. MATLAB provides functions like `findViewConnections` to facilitate this comparison and establish relationships between matching frames.

Once a loop is detected, the pose graph is updated to incorporate the additional constraints, ensuring that the map aligns correctly across revisited locations. This step not only improves the system's understanding of its surroundings but also enhances its ability to handle large-scale environments. The integration of loop closures using MATLAB's `optimizePoses` function ensures that the pose graph remains consistent, minimizing the impact of accumulated drift and enabling the SLAM system to maintain high accuracy over time.

2.3.5 Drift Correction

Drift correction addresses the errors that accumulate over time as the SLAM system estimates its pose incrementally. These errors, if unchecked, can lead to significant discrepancies in the map, making it unreliable for navigation. Drift correction uses techniques like pose graph optimization to align the system's estimated trajectory with observed landmarks and loop closures. MATLAB's `optimizePoses` function plays a central role here, adjusting the pose graph to minimize inconsistencies and align the map with the real environment.

In addition to reducing errors, drift correction enhances the robustness and scalability of the SLAM system, ensuring it remains reliable even in complex or large-scale environments. By utilizing the `createPoseGraph` function in MATLAB, the pose graph can be converted into a `digraph` object for detailed analysis and modification. This enables developers to fine-tune the system, ensuring that the final map is both accurate and efficient for real-world applications.

2.3.6 Visualization

Visualization is an integral part of the SLAM process, providing a clear representation of the map, trajectory, and pose graph. It allows developers to analyze the performance of the SLAM system, identify potential issues, and refine its components. MATLAB offers various tools for visualization, including the plot function for pose graphs and `pcshow` for 3D point clouds. These tools enable the real-time rendering of the system's progress, making it easier to debug and optimize.

Effective visualization is not only crucial for development but also for demonstrating the system's capabilities. By dynamically updating visualizations with functions like `updatePlot`, developers can monitor changes to the map and trajectory as new data is integrated. This ensures that the SLAM system remains transparent and its operations comprehensible, making it easier to refine and deploy in practical scenarios.

3 Models

3.1 Impact of Visible Light on Cameras

Autonomous vehicles (AVs) depends heavily on camera sensors for their Visual Simultaneous Localization and Mapping (vSLAM) systems. These cameras track and analyze light coming from many sources, including the headlight and brakelight of surrounding cars. Operating over a wide range of visible light (400–700 nm), cameras used in AVs translate light signals into pixel intensity data, which the AV's algorithms then use to detect and classify objects, motions, and signals on the road.

This attack's main objective is to exploit how cameras perceive changes in light intensity. Like most digital cameras, AV cameras are sensitive to both steady-state light and changes in light, hence they are vulnerable to modulated light signals. Using Visible Light Communication (VLC), an attacker can subtly change the light intensity, so manipulating the camera's perception and causing the AV to misinterpret normal light signals such as brake lights or turn lights as cues to act (e.g., lane changes).

3.2 Attacker's Objective

The objective of the attacker is to exploit the AV's vSLAM system by modulating the headlight or brake light of their vehicle in a way that deceives the target vehicle into reacting to false

visual cues. The attacker uses Visible Light Communication (VLC) technology to achieve this modulation, subtly adjusting the intensity or flicker rate of the lights to simulate behaviors such as braking, accelerating, or approaching.

In a typical driving scenario, an AV relies on the camera to observe the brake lights or headlights of surrounding vehicles to gauge speed, distance, and environmental conditions. When the brake lights illuminate, the AV interprets this as a signal to slow down or stop. Similarly, variations in headlight patterns, such as dimming or brightening, may prompt the AV to infer environmental or vehicle behavioral changes and adjust its driving actions accordingly. By controlling these visual signals through light modulation, the attacker can induce incorrect responses from the AV without any actual change in driving behavior.

For instance, the attacker might subtly modulate the brake lights to suggest frequent braking or decelerating rapidly. The AV behind, relying on this visual input, may brake unnecessarily, disrupt traffic flow, or even cause rear-end collisions if the behavior is unexpected. Alternatively, modulating the headlights to simulate dimming conditions could cause the AV to misinterpret an obstacle or environmental change, resulting in dangerous maneuvers.

The attacker's goal is not to disable the AV but to cause localized disruptions that affect its real-time decision-making. The subtle nature of the attack ensures that it remains undetected by human drivers, who are accustomed to variations in vehicle lighting, and potentially also by onboard monitoring systems, which do not typically look for patterns of malicious modulation in headlights or brake lights. This ability to operate covertly while causing significant disruptions makes the attack particularly dangerous.

3.3 System model and assumptions

The autonomous vehicle (AV) is equipped with a Visual Simultaneous Localization and Mapping (vSLAM) system that relies heavily on camera-based inputs to interpret its environment and make real-time decisions. This system assumes that visual cues, such as brake lights and headlights from other vehicles, are authentic and unmanipulated. The attack scenario presumes typical driving conditions where the AV operates predominantly based on its visual perception capabilities, such as in daylight or adequately illuminated nighttime scenarios. The surrounding environment includes dynamic elements like road markings and other vehicles, providing contextual visual data critical to the AV's navigation.

The attacker is assumed to have access to a vehicle with modifiable headlights and brake lights that can be controlled using Visible Light Communication (VLC) technology. They possess

sufficient technical expertise to embed subtle, undetectable modulations into these lights, leveraging the AV's reliance on visual cues to disrupt its behavior. The attacker's knowledge extends to understanding the AV's vSLAM-based decision-making processes and the behavioral implications of manipulated light inputs.

The AV is considered vulnerable due to a lack of advanced mechanisms to validate the authenticity of light-based visual inputs. It relies on these cues without sophisticated verification against malicious patterns. The AV operates with a safety-first principle, responding immediately to perceived environmental changes, such as braking in response to brake lights or altering lane position in reaction to headlight variations. The attacker's intent is limited to causing localized disruptions in the AV's behavior, such as unnecessary braking or lane shifts, without attempting to cause physical damage or exploit internal systems. The attack does not require direct access to the AV's hardware or software and is executed solely through manipulation of external light signals.

The target system is an autonomous vehicle (AV) equipped with a Visual Simultaneous Localization and Mapping (vSLAM) system, which is integral to the vehicle's ability to navigate and avoid obstacles. This system leverages data from cameras mounted on the AV to interpret its surrounding environment. By analyzing visual inputs such as road markings, traffic signs, surrounding vehicles, and other objects, the vSLAM system constructs a dynamic map of the environment and localizes the vehicle's position within it.

The vSLAM system relies heavily on accurately detecting key visual signals, such as headlight variations and brake lights from nearby vehicles, to make real-time driving decisions. The AV's camera captures these visual cues, which the vSLAM algorithm interprets to guide the vehicle's responses. For instance, the system detects illuminated brake lights as an indication that the leading vehicle is slowing or stopping, prompting the AV to reduce its speed or halt. Similarly, fluctuations in the brightness of headlights, such as dimming or flashing patterns, may provide information about environmental conditions, signaling potential hazards or changes in the behavior of other vehicles.

This system assumes that the visual inputs it receives are authentic and unmanipulated, making it susceptible to attacks that exploit this trust. An attacker can leverage Visible Light Communication (VLC) technology to modulate the headlight or brake light of their vehicle, embedding false visual signals into the light patterns. VLC allows data to be transmitted by subtly altering the light intensity or flicker rate in a manner indistinguishable to human observers but interpretable by the AV's cameras. These manipulated light signals are processed by the AV's vSLAM system as legitimate inputs, causing the vehicle to execute inappropriate actions, such

as sudden braking or unnecessary adjustments in its lane positioning.

By targeting the AV's reliance on headlight and brake light cues, the attacker can exploit the vSLAM system to disrupt the vehicle's behavior without any overt indications to human drivers or surrounding vehicles. This vulnerability underscores the critical need for enhanced validation of visual inputs in autonomous driving systems.

4

Dataset

4.1 KITTI Vision Benchmark Suite

The KITTI Vision Benchmark Suite [24] is a project developed by the Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago, providing a comprehensive collection of real-world computer vision benchmarks. To create the datasets, a station wagon equipped with two pairs of high-resolution cameras one capturing grayscale images and the other RGB images was driven around the German city of Karlsruhe and its surrounding areas. The vehicle was also fitted with a laser scanner and an inertial navigation system (GPS/IMU) to accurately estimate ground truth data.

The datasets consist of sequences of varying lengths recorded in diverse settings such as urban areas, residential neighborhoods, highways, and university campuses, featuring both static and dynamic subjects. The KITTI suite supports a range of computer vision tasks, including stereo vision, optical flow, visual odometry, 3D object detection, and 3D tracking.

Due to the wide variety of scenarios and the precision with which the data were collected, KITTI has become a highly popular resource in computer vision research. It is frequently used to test new algorithms, evaluate implementations, and compare alternative benchmarks, making it a valuable tool for advancing research in these fields.

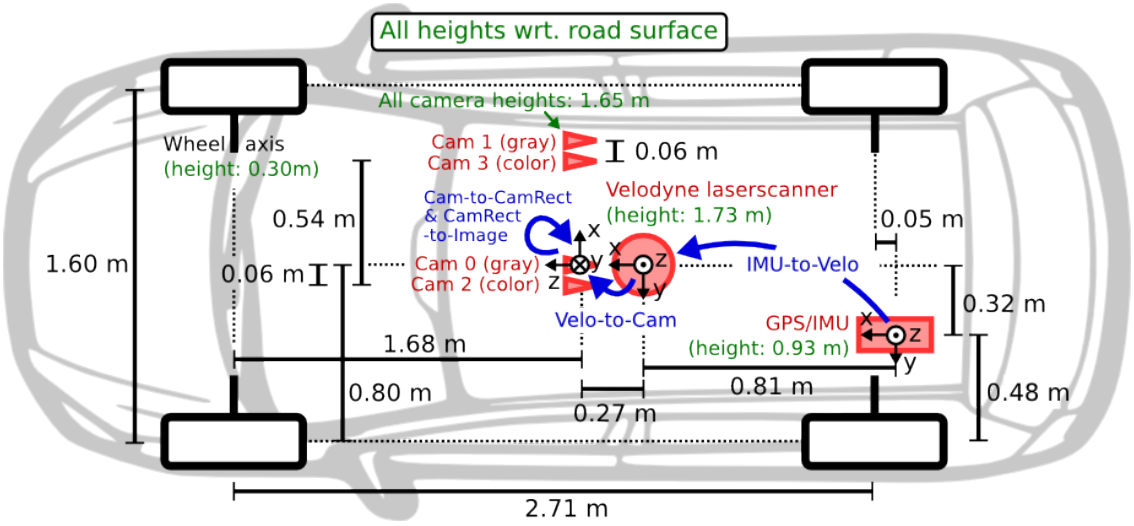

Figure 4.1: KITTI's full sensor setup [3]

4.1.1 SLAM Evaluation Dataset

Sensor Setup

Figure 4.1 shows the configuration of the fully equipped vehicle employed to record the dataset. The sensors used are:

- 1 Inertial Navigation System (GPS/IMU): OXTS RT 3003 ;
- 1 Laser scanner: Velodyne HDL-64E;
- 2 Grayscale cameras, 1.4 Megapixels: Point Grey Flea 2 (FL2-14S3M-C);
- 2 Color cameras, 1.4 Megapixels: Point Grey Flea 2 (FL2-14S3C-C);
- 4 Varifocal lenses, 4-8 mm: Edmund Optics NT59-917

The KITTI dataset features cameras mounted with a stereo baseline of 0.54 meters and positioned 1.65 meters above the ground. These cameras capture high-resolution raw images with a size of 1392 × 512 pixels, which are initially distorted and unrectified. The cameras operate at 10 frames per second (fps) and are synchronized with the Velodyne HDL-64E laser scanner, which also spins at 10 fps. In addition to the raw data, the dataset includes undistorted and rectified images cropped to 1242 × 375 pixels, ready for direct use. Detailed information about the sensor setup and data collection process is available in the KITTI dataset documentation [25].

Among its offerings, the city dataset is particularly significant for research in Visual SLAM (Simultaneous Localization and Mapping), autonomous navigation, and computer vision. It captures realistic, raw-world data from urban environments, featuring challenges such as dense traffic, varied lighting conditions, and dynamic objects like pedestrians and vehicles. These factors make it ideal for testing algorithms in environments that closely mirror real-world scenarios.

The raw dataset [26] provides synchronized streams from multiple high-resolution sensors mounted on a station wagon. These include two grayscale cameras and two RGB cameras, both running at 10 fps, a Velodyne HDL-64E laser scanner for detailed 3D mapping, and a GPS/IMU system for highly accurate ground truth data. The camera's stereo baseline and elevated positioning allow for a rich perspective of the environment, while the raw images retain their distortions, giving researchers the flexibility to apply their own preprocessing techniques such as rectification and undistortion.

The city dataset is invaluable for testing SLAM algorithms under challenging conditions, such as complex road layouts, varying lighting, and dynamic urban elements. By working with the raw data, researchers can perform custom calibrations and test the robustness of their methods against real-world complexity. This versatility and fidelity make the KITTI city dataset an essential resource for developing scalable and reliable systems for autonomous vehicles and other advanced computer vision applications.

5

Experimental Overview and Results

This chapter introduces the experimental model designed to simulate different high-beam headlight and tailight effects on image frames. The purpose is to understand how these artificial lighting conditions impact feature detection and the resulting accuracy of the SLAM system. Using the KITTI dataset, the experiment involves initializing a map with frames, applying high-beam effects, and comparing performance metrics, including feature counts and trajectory accuracy.

Visual SLAM (Simultaneous Localization and Mapping) systems are highly dependent on the quality and reliability of feature detection. Features distinctive points or patterns in an image serve as landmarks for building maps and determining the camera's position in space. However, real-world conditions are rarely ideal. Lighting anomalies, such as glare from vehicle headlights, can distort the visual information captured by cameras, making it difficult for SLAM systems to operate accurately.

5.1 Methodology and Evaluation metric

5.1.1 Map Initialization

The first step in any SLAM process is to initialize the map. In this experiment, the map is initialized using the first two frames of the dataset. These frames are analyzed to extract distinctive features using the ORB (Oriented FAST and Rotated BRIEF) feature detection algorithm.

Feature matching between the frames is then performed to establish correspondences that help compute the relative transformation between the frames.

The initialization succeeds only if there are enough reliable matches between the two frames. If the transformation is valid, the SLAM system declares the map initialized and moves on to the next stage. For this experiment, the system successfully initialized the map, as confirmed by the output: "Map initialized with frame 1 and frame 2."

The visualization of this initialization is presented in Figure 5.1, where the matched features between the two frames are displayed. This serves as the foundation for the SLAM process, providing the baseline for all subsequent steps.

```
while ~isMapInitialized && currFrameIdx <= numel(imds.Files)
    currI = readimage(imds, currFrameIdx);

    [currFeatures, currPoints] = helperDetectAndExtractFeatures(currI, scaleFactor,
    numLevels);

    currFrameIdx = currFrameIdx + 1;

    % Find feature matches
    indexPairs = matchFeatures(preFeatures, currFeatures, Unique=true, ...
        MaxRatio=0.9, MatchThreshold=40);

    % If not enough matches are found, check the next frame
    minMatches = 100;
    if size(indexPairs, 1) < minMatches
        continue
    end

    preMatchedPoints  = prePoints(indexPairs(:,1),:);
    currMatchedPoints = currPoints(indexPairs(:,2),:);

    % Compute homography and evaluate reconstruction
    [tformH, scoreH, inliersIdxH] = helperComputeHomography(preMatchedPoints,
    currMatchedPoints);

    % Compute fundamental matrix and evaluate reconstruction
    [tformF, scoreF, inliersIdxF] = helperComputeFundamentalMatrix(preMatchedPoints,
     currMatchedPoints, intrinsics);

```

```
    % Select the model based on a heuristic
    ratio = scoreH/(scoreH + scoreF);
    ratioThreshold = 0.45;
    if ratio > ratioThreshold
        inlierTformIdx = inliersIdxH;
        tform          = tformH;
    else
        inlierTformIdx = inliersIdxF;
        tform          = tformF;
    end

    % Computes the camera location up to scale. Use half of the
    % points to reduce computation
    inlierPrePoints  = preMatchedPoints(inlierTformIdx);
    inlierCurrPoints = currMatchedPoints(inlierTformIdx);
    [relPose, validFraction] = estrelpose(tform, intrinsics, ...
        inlierPrePoints(1:2:end), inlierCurrPoints(1:2:end));

    % If not enough inliers are found, move to the next frame
    if validFraction < 0.9 || numel(relPose)>1
        continue
    end

    % Triangulate two views to obtain 3-D map points
    minParallax = 1; % In degrees
    [isValid, xyzWorldPoints, inlierTriangulationIdx] = helperTriangulateTwoFrames
    (...
        rigidtform3d, relPose, inlierPrePoints, inlierCurrPoints, intrinsics,
    minParallax);

    if ~isValid
        continue
    end

    % Get the original index of features in the two key frames
    indexPairs = indexPairs(inlierTformIdx(inlierTriangulationIdx),:);

    isMapInitialized = true;

    disp(['Map initialized with frame 1 and frame ', num2str(currFrameIdx-1)])
end % End of map initialization loop
```

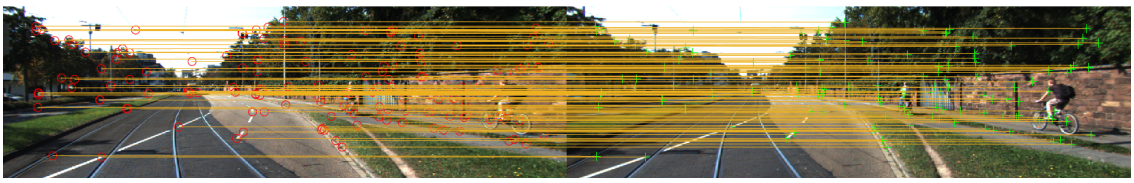

Figure 5.1: Map Initialization

5.1.2 Feature Detection

Feature detection is one of the most critical steps in the SLAM pipeline. In this experiment, features were first detected in an original image frame. These features are shown in Figure 5.2, where the green points highlight the detected features. These points act as landmarks that help the system track the camera's movement and construct a 3D map.

Figure 5.2: matched features.png

Next, to simulate adverse lighting conditions, a high-beam lighting effect was applied to the same image frame. The simulation added intense brightness at the center of the frame to mimic the glare caused by vehicle headlights. The manipulated image is shown in Figure 5.3, where the high-beam effect is clearly visible as a bright central region. This manipulation serves as the experimental condition to evaluate the robustness of the SLAM system.

The process of applying the high-beam effect is achieved using a custom function, applyHighBeamEffect. This function artificially brightens a defined region in the image, making it difficult for the SLAM system to detect and track features in that area.

```
% ---- Count Features in the Original Frame ----
% Count features before applying the high-beam effect
[origFeatures, origPoints] = helperDetectAndExtractFeatures(currI, scaleFactor,
    numLevels);
originalFeatureCounts(currFrameIdx) = origPoints.Count;  % Store feature count for
    original frame
```

```
5
6 % ---- Apply High-Beam Effect to Create Manipulated Frame ----
7 currI_manipulated = applyHighBeamEffect(currI, centerBeamPosition, beamWidth,
      beamHeight, lightIntensity);
8
9 % ---- Count Features in the Manipulated Frame ----
10 % Count features on the manipulated frame with the high-beam effect
11 [manipFeatures, manipPoints] = helperDetectAndExtractFeatures(currI_manipulated,
      scaleFactor, numLevels);
12 manipulatedFeatureCounts(currFrameIdx) = manipPoints.Count;  % Store feature count
      for manipulated frame
```

5.1.3 RMSE

The Root Mean Square Error (RMSE), also known as the Root Mean Square Deviation (RMSD), is a widely used metric for evaluating the accuracy of predictive models, estimating error in measurements, and comparing datasets. It quantifies the difference between observed and predicted values, making it a crucial tool in fields such as statistics, machine learning, geostatistics, and navigation. RMSE is particularly valued for its ability to provide a single summary statistic that represents the magnitude of error in a model or system.

RMSE is defined as the square root of the mean of the squared differences between predicted and observed values. Mathematically, it is expressed as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2} \tag{5.1}$$

5.2 Experiment and Results

5.2.1 Single High-Beam headlight Simulation on vSLAM

The first experiment focused on simulating a single high-beam headlight to study its impact on Visual SLAM (vSLAM) systems. High-beam headlights, commonly used in vehicles for better visibility at night, emit intense and focused light that creates a glare effect. This glare not only overwhelms the brightness of the affected area but also diminishes the visibility of other regions in the camera's field of view. Such conditions are representative of real-world challenges

faced by autonomous vehicles and robotics systems that rely on visual data for navigation and mapping.

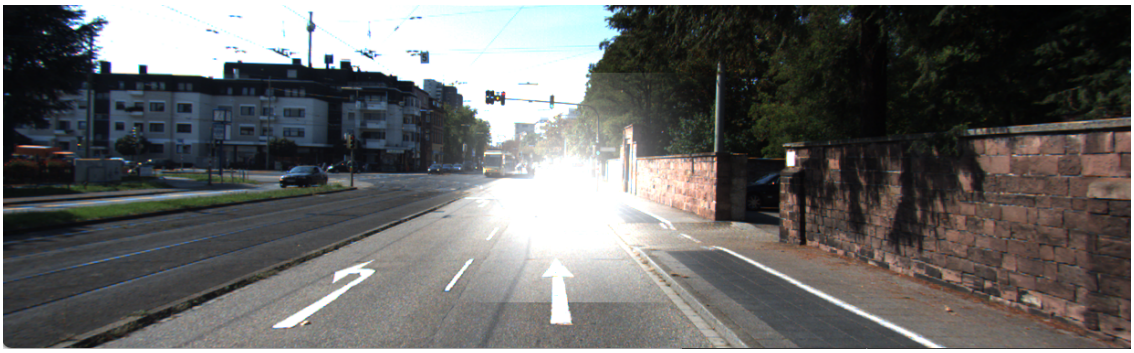

Figure 5.3: Single Headlight on vSLAM model

In this experiment, the simulation replicated the effect of a single high-beam headlight by introducing an artificially bright region in the central portion of the image. This effect was carefully calibrated to mimic the size, intensity, and distribution of light typically caused by a single vehicle headlight. The glare was created using parameters such as beam width, height, and intensity. The center position of the glare was strategically chosen to ensure that it directly impacted the area where the most critical features for vSLAM are usually detected.

5.2.2 Dual High-Beam headlight Simulation on vSLAM

The second experiment expanded on the first by simulating dual high-beam headlights, representing the glare typically produced by an oncoming vehicle with two active headlights. Dual headlights introduce a more complex visual anomaly compared to a single headlight, as they create two distinct glare zones that may overlap or spread across a significant portion of the image. This simulation aimed to evaluate how such conditions affect the ability of a Visual SLAM (vSLAM) system to detect features and maintain accurate trajectory estimation.

In this setup, two high-intensity glare zones were artificially introduced to the image, positioned on the left and right sides of the frame to mimic the typical alignment of a vehicle's headlights. Each glare zone was calibrated to have parameters such as beam width, height, and intensity, similar to the first experiment, but distributed across two regions instead of one. The overlap between these zones resulted in compounded brightness in the central area, creating a larger region of distortion and further limiting the system's ability to identify meaningful features.

Figure 5.4: Dual Headlight on vSLAM model

5.2.3 Intense Taillight Simulation on vSLAM

This experiment investigated the effect of simulated taillights, which are small but intense red light sources commonly observed at the rear of vehicles. Unlike headlight glare, taillights create a localized lighting anomaly, concentrated in specific areas of the image typically at the lower corners. This setup aimed to evaluate how such localized disruptions affect Visual SLAM (vSLAM) systems, particularly in scenarios where critical features are located near the affected regions.

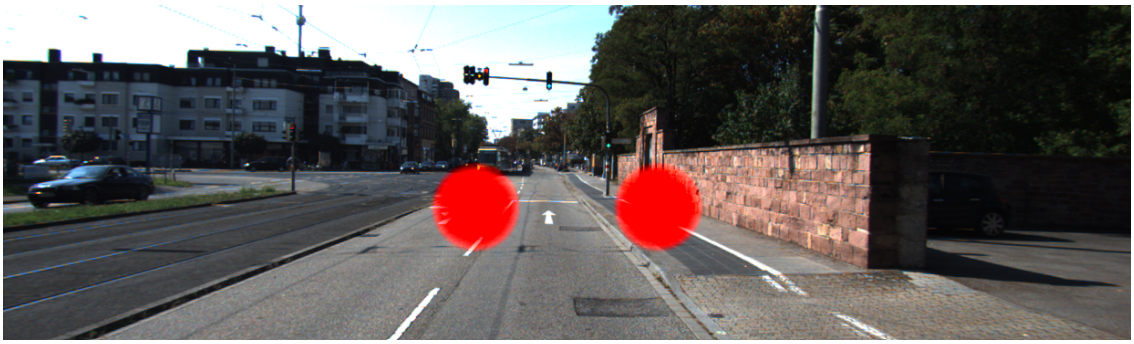

Figure 5.5: Intense taillight on vSLAM model

To simulate taillights, two bright red regions were artificially introduced to the bottom-left and bottom-right corners of the image. The intensity and size of these light sources were calibrated to mimic real-world taillight behavior, ensuring the disruption was both realistic and challenging for the SLAM system. Unlike the high-beam simulations, which introduced widespread glare, the taillight effect was confined to smaller, specific regions of the frame.

5.2.4 Adversarial Patch with Dual Headlights on vSLAM

The final experiment combined two significant disruptions: an adversarial patch and dual high-beam headlights. This setup created one of the most challenging conditions for Visual SLAM (vSLAM) systems, designed to evaluate how the system handles both subtle and overt anomalies at the same time. The adversarial patch was strategically placed in the central region of the image to subtly manipulate feature detection. Interestingly, instead of reducing the number of features, the patch caused the system to detect more features in the manipulated image than in the original one. This happened because the patch introduced patterns that the system misinterpreted as valid features, inflating the feature count artificially.

Figure 5.6: Impact of Adversarial patch with Dual Headlight ON vSLAM

At the same time, the dual high-beam headlights introduced intense glare on the left and right sides of the image, creating overlapping brightness zones that overwhelmed and distorted features in those areas. While the adversarial patch tricked the system into thinking there were more features, the glare further complicated the situation by obscuring legitimate ones, making it harder for the system to accurately track and map the environment.

This experiment reveals how manipulations like adversarial patches can mislead the system, while lighting anomalies continue to degrade its overall performance, emphasizing the complexity of such combined challenges.

5.2.5 Results

Let's analyze the results of each experiment individually, focusing on feature comparisons between original and manipulated frames, along with the impact on the ground truth trajectory for each case.

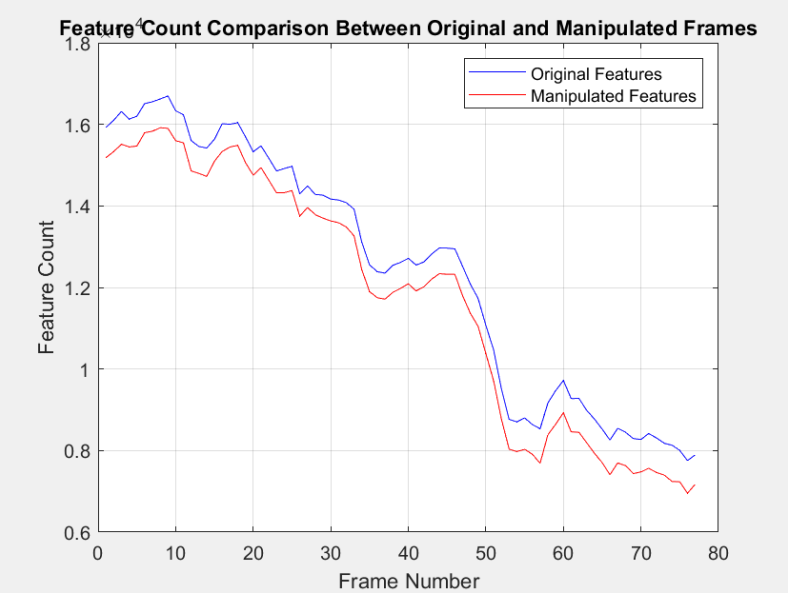

Figure 5.7: Feature Comparison Original and Manipulated Frames

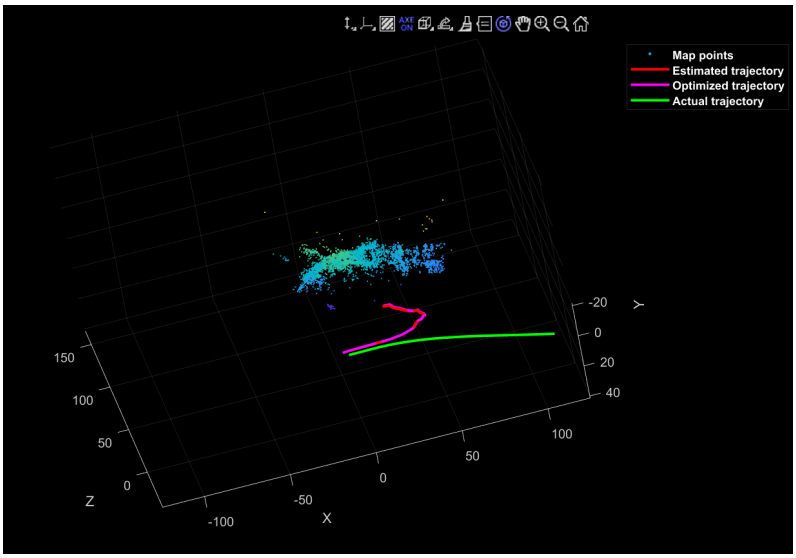

Figure 5.8: Ground Truth

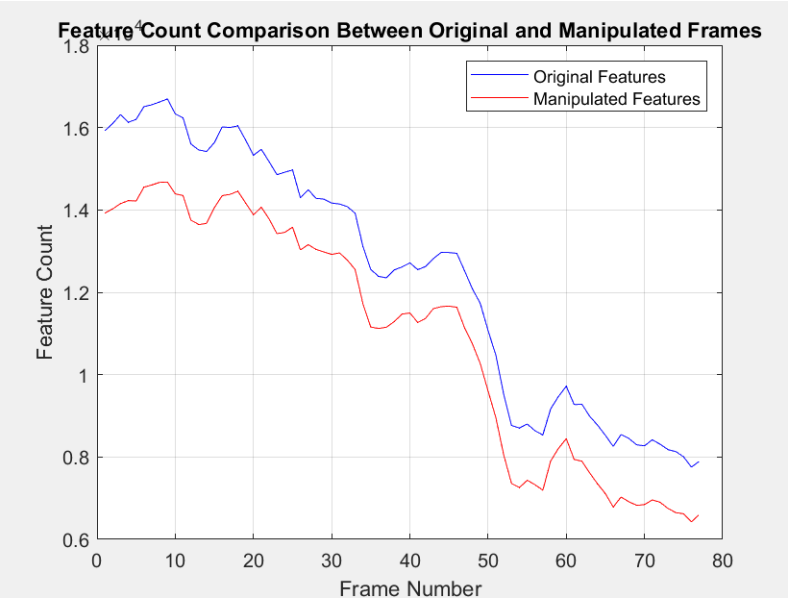

Figure 5.9: Feature Comparison Original and Manipulated Frames

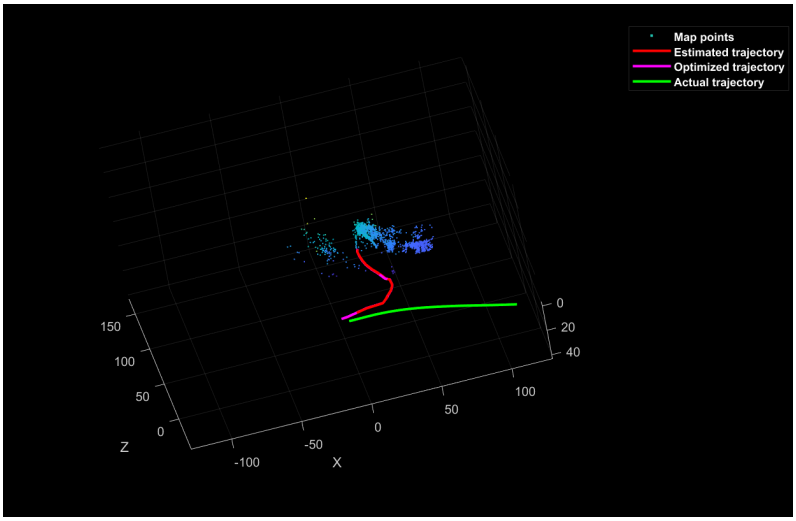

Figure 5.10: Ground Truth

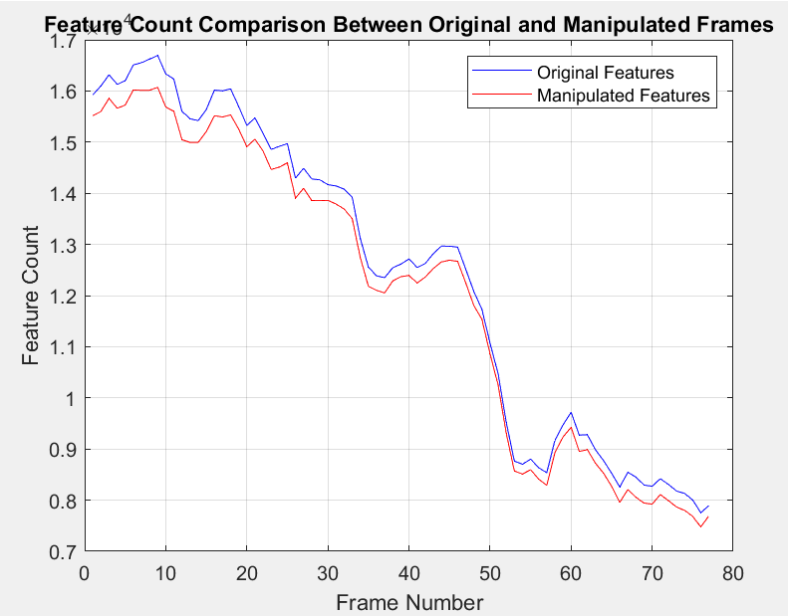

Figure 5.11: Feature Comparison Original and Manipulated Frames

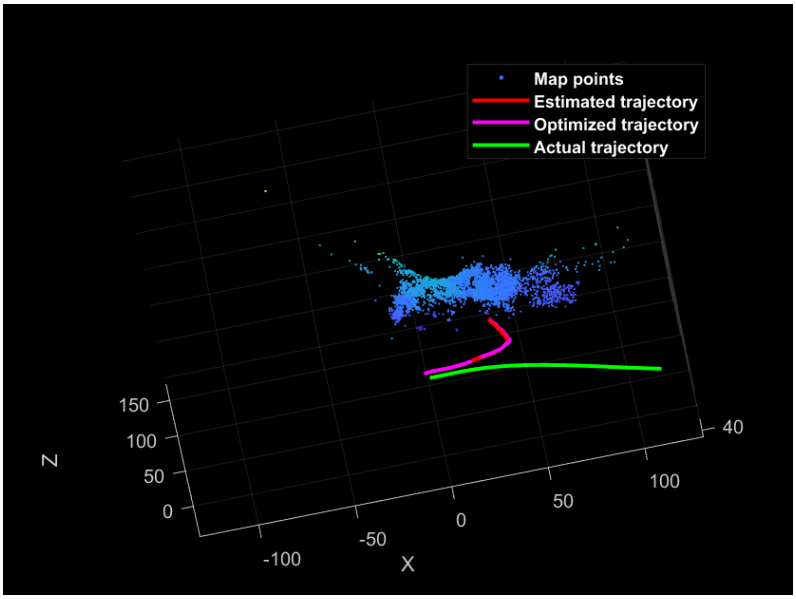

Figure 5.12: Ground Truth

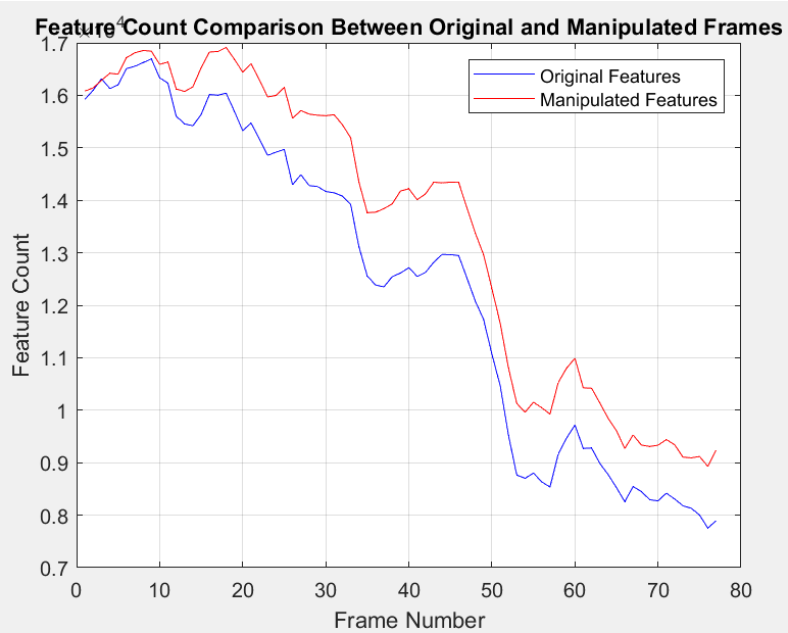

Figure 5.13: Feature Comparison Original and Manipulated Frames

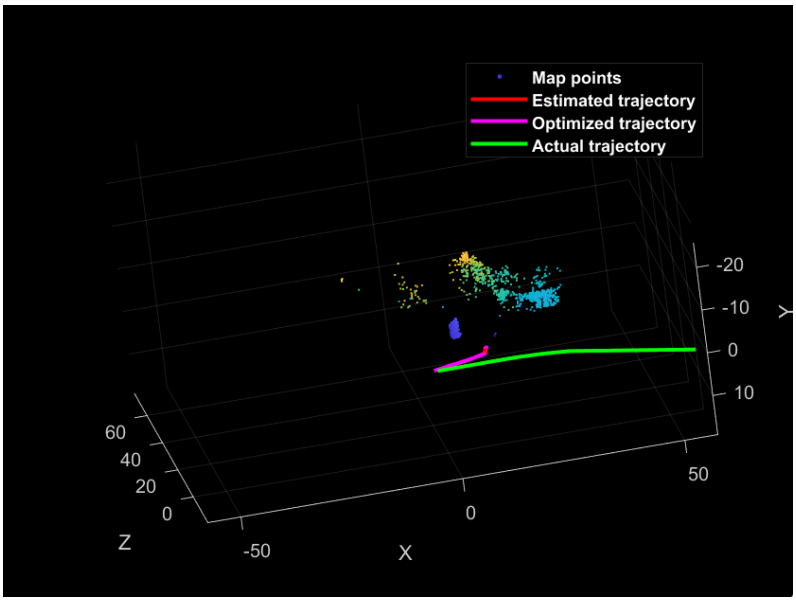

Figure 5.14: Ground Truth

The table 5.1 summarizes the comparison of features and RMSE (Root Mean Square Error) across various experimental lighting conditions in the manipulated frames relative to the original frames. Here the negative value in Adversarial patch denotes the features detected in manipulated frames is higher than the orginal frame

Experiment	Feature Comparison	RMSE
Single Headlight	697.64	25.44
Dual Headlight	1480.17	15.97
Intense Taillight	457.86	20.98
Adversarial Patch	-1049.70	13.29

Table 5.1: Feature Comparison and RMSE

6

Counter Measures

6.1 Defence Strategies to Strengthen vSLAM Systems

The experiments show that Visual SLAM (vSLAM) systems face significant challenges when operating under difficult visual conditions, such as intense lighting or adversarial disruptions. A key finding was the substantial drop in feature detection accuracy, leading to an RMSE of 15.9702 meters in trajectory estimation. To address these vulnerabilities and ensure reliability, several strategies can be adopted to make these systems stronger and more adaptable.

6.1.1 Improving Feature Detection

At the core of SLAM is the ability to detect features unique points in an image that the system tracks to navigate and map its environment. Traditional methods like ORB often falter in complex scenarios, such as extreme lighting or manipulated visuals. To tackle this, we can use advanced feature detectors like SuperPoint, which are powered by deep learning and trained to recognize features even in tough conditions. Another helpful approach is detecting features at multiple scales, capturing both fine details and larger patterns. Combining edge-based and corner-based methods can also ensure the system has a fallback if one type of feature is obscured.

6.1.2 Integrating Sensor Fusion

Relying solely on cameras makes SLAM systems vulnerable, especially in environments with glare, low light, or manipulation. Adding extra sensors can make the system much more robust. LiDAR, for example, provides depth information that remains reliable regardless of lighting. IMUs (Inertial Measurement Units) can assist in estimating motion when the camera struggles. Additionally, thermal or infrared cameras, which see beyond visible light, can help the system detect features even in total darkness or intense glare.

6.1.3 Image Preprocessing Techniques

Sometimes, the images themselves need a little help. Preprocessing techniques can make them easier for SLAM systems to interpret. For example, HDR (High-Dynamic-Range) imaging can balance brightness, ensuring no part of the image is too bright or too dark. Contrast enhancement techniques, like CLAHE, make subtle features stand out, while glare reduction filters can cut down on bright spots that obscure landmarks. These small adjustments can make a big difference in how well the system performs.

6.1.4 Defending Against Adversarial Attacks

Adversarial patches, which are designed to confuse feature detection, present a unique challenge. One way to counter this is by training SLAM systems on manipulated datasets so they can learn to spot and ignore such disruptions. Real-time anomaly detection can also help by flagging suspicious patterns in features and discarding false information. Cross-referencing features across frames ensures consistency, further safeguarding the system from manipulation.

7 Conclusion

The results of these experiments underscore the vulnerabilities of Visual SLAM (vSLAM) systems when exposed to challenging visual conditions such as intense lighting disruptions or adversarial manipulations. To overcome these issues, a multi-faceted approach is necessary, involving smarter feature detection, integration of complementary sensors, preprocessing enhancements, defenses against adversarial attacks, and robust trajectory optimization. These strategies collectively pave the way for vSLAM systems to operate reliably in diverse and unpredictable real-world environments.

While these solutions significantly strengthen the system, there remain areas for further exploration. Future research should focus on developing hybrid SLAM frameworks that integrate visual data with other sensing modalities, such as LiDAR or radar, to further reduce reliance on camera-only inputs. Additionally, adaptive learning algorithms can be implemented to allow SLAM systems to evolve in real-time, adapting to unseen scenarios by learning from their environments.

In conclusion, this thesis contributes to the field of SLAM research by identifying and quantifying the challenges posed by visual anomalies. By addressing these challenges, the findings pave the way for developing more resilient and reliable SLAM systems. As autonomous technology continues to evolve, ensuring the robustness of SLAM systems will be critical to their successful deployment in real-world applications, fostering safer and more efficient autonomous navigation.

References

[1] MathWorks, "Simultaneous localization and mapping (slam)," n.d., accessed: 2023-12-05. [Online]. Available: https://it.mathworks.com/discovery/slam.html

[2] (n.d.) Visual simultaneous localization and mapping (slam) overview. Accessed: 2023-12-05. [Online]. Available: https://it.mathworks.com/help/vision/ug/visual-simultaneous-localization-and-mapping-slam-overview.html

[3] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. (n.d.) Kitti vision benchmark suite: Dataset setup. Accessed: 2023-12-05. [Online]. Available: https://www.cvlibs.net/datasets/kitti/setup.php

[4] B. Chen, W. Wang, P. Sikorski, and T. Zhu, "Adversary is on the road: Attacks on visual {SLAM} using unnoticeable adversarial patch," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 6345–6362.

[5] 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 *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 293–308.

[6] 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 *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1930–1944.

[7] The MathWorks, Inc., "What is slam? simultaneous localization and mapping explained - mathworks," https://it.mathworks.com/discovery/slam.html, accessed: December 4, 2024.

[8] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2. Ieee, 1999, pp. 1150–1157.

[9] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[10] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[11] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[12] N. M. Kwok, Q. P. Ha, and G. Fang, "Data association in bearing-only slam using a cost function-based approach," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 4108–4113.

[13] G. Kootstra, S. de Jong, and D. Wedema, "Comparing the ekf and fastslam solutions to the problem of monocular simultaneous localization and mapping," 2009.

[14] J. Sola, A. Monin, M. Devy, and T. Lemaire, "Undelayed initialization in bearing only slam," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 2499–2504.

[15] J. Civera, A. J. Davison, and J. M. Montiel, "Inverse depth parametrization for monocular slam," *IEEE transactions on robotics*, vol. 24, no. 5, pp. 932–945, 2008.

[16] M. Li, B. Hong, Z. Cai, and R. Luo, "Novel rao-blackwellized particle filter for mobile robot slam using monocular vision," *International journal of intelligent technology*, vol. 1, no. 1, pp. 63–69, 2006.

[17] C. Gamallo, M. Mucientes, and C. Regueiro, "Visual fastslam through omnivision," *Towards Autonomous Robotic Systems*, pp. 128–135, 2009.

[18] The MathWorks, Inc., "Matlab - mathworks," https://it.mathworks.com/products/matlab.html, accessed: December 4, 2024.

[19] "Automated driving toolbox - mathworks," https://it.mathworks.com/products/automated-driving.html, accessed: December 4, 2024.

[20] The MathWorks, Inc., "Computer vision toolbox - mathworks," https://it.mathworks.com/products/computer-vision.html, accessed: December 4, 2024.

[21] "Image processing toolbox - mathworks," https://it.mathworks.com/products/image-processing.html, accessed: December 4, 2024.

[22] The MathWorks, Inc., "Control system toolbox - mathworks," https://it.mathworks.com/products/control.html, accessed: December 4, 2024.

[23] "Robotics system toolbox - mathworks," https://it.mathworks.com/products/robotics.html, accessed: December 4, 2024.

[24] A. Geiger, P. Lenz, and R. Urtasun, "The kitti vision benchmark suite," https://www.cvlibs.net/datasets/kitti, accessed: December 4, 2024.

[25] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[26] A. Geiger, P. Lenz, and R. Urtasun, "Kitti raw data benchmark," https://www.cvlibs.net/datasets/kitti/raw_data.php, accessed: December 4, 2024.