PL-VIWO: A Lightweight and Robust Point-Line Monocular Visual Inertial Wheel Odometry

Zhixin Zhang[1], Wenzhi Bai[1], Liang Zhao[2], Pawel Ladosz[3]

Abstract— This paper presents a novel tightly coupled Filter-based monocular visual-inertial-wheel odometry (VIWO) system for ground robots, designed to deliver accurate and robust localization in long-term complex outdoor navigation scenarios. As an external sensor, the camera enhances localization performance by introducing visual constraints. However, obtaining a sufficient number of effective visual features is often challenging, particularly in dynamic or low-texture environments. To address this issue, we incorporate the line features for additional geometric constraints. Unlike traditional approaches that treat point and line features independently, our method exploits the geometric relationships between points and lines in 2D images, enabling fast and robust line matching and triangulation. Additionally, we introduce Motion Consistency Check (MCC) to filter out potential dynamic points, ensuring the effectiveness of point feature updates. The proposed system was evaluated on publicly available datasets and benchmarked against state-of-the-art methods. Experimental results demonstrate superior performance in terms of accuracy, robustness, and efficiency. The source code is publicly available at: https://github.com/Happy-ZZX/PL-VIWO.

I. INTRODUCTION

Achieving robust and accurate localization in complex outdoor environments remains a critical challenge for ground robot technology. Visual Odometry systems which rely solely on image data, often struggle in outdoor navigation, leading to unstable and inaccurate localization. To address these limitations, integrating interoceptive sensors, such as IMUs and wheel encoders has been proven to be an effective approach. This integration not only improves system robustness [1] but also mitigates errors caused by unobservable states during degenerate motions [2]. Furthermore, Multi-sensor-fusion-based systems can provide stable estimation results based on measurements of interoceptive sensors in texture-less, dynamic, and poor illumination environments. These advantages have made Visual-Inertial-Wheel Odometry a widely adopted solution for outdoor robot localization.

Point features are widely used in Visual-Odometry (VO) [3] and Visual-inertial-Odometry (VIO) systems [4], [5], [6] where descriptors or optical flow are employed to track point features in 2D images for camera pose estimation. However,

[1]Zhixin Zhang, Wenzhi Bai are with the Department of Electrical and Electronic Engineering, University of Manchester, Manchester M13 9PL, U.K. Email: {zhixin.zhang, wenzhi.bai}@manchester.ac.uk

[2]Liang Zhao is with the School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, U.K. Email: liang.zhao@ed.ac.uk

[3]Pawel Ladosz is with the Department of Mechanical, Aerospace and Civil Engineering, University of Manchester, Manchester M13 9PL, U.K. Email: pawel.ladosz@manchester.ac.uk
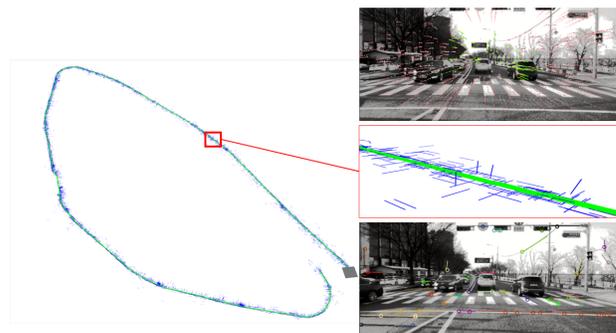
Fig. 1: PL-VIWO in KAIST Urban32. **Left**: Top-down view of the trajectory with line mapping results. **Top-Right**: Point tracking results, where red dots connected by lines represent observations of point features within the sliding window, and green indicates those failed in triangulation or MCC. **Middle-Right**: Zoomed-in trajectory (green) with lines (blue). **Bottom-Right**: Point-line pairing results, where circles with the same color line indicate point features that lie on the corresponding line feature.

in low-texture and dynamic environments, such as congested urban roads or highways, obtaining sufficient reliable visual point features is challenging. To address this limitation, incorporating additional structural features, such as lines, provides more constraints for state estimation. Similar to point features, line features require detection, tracking, and triangulation, all of which demand additional computational resources and pose challenges to real-time performance [7]. Furthermore, line triangulation is particularly susceptible to certain degenerate motions, especially in 2D motion, further complicating the use of line features in ground robots [8]. Another approach to enhancing localization accuracy in outdoor environments is the removal of dynamic point features. A common method is to employ object detection algorithms to generate image masks that identify dynamic objects [9]. Point features within masked regions are excluded from the visual update process. However, this technique is computationally demanding and susceptible to missed detections. As an alternative, Motion Consistency Check (MCC) assesses the temporal consistency of point features over a short time window. Points that fail MCC are classified as dynamic and subsequently excluded from visual updates, improving the accuracy of estimation.

To solve the aforementioned challenges, we propose a novel monocular visual-wheel-inertial odometry (VIWO)

system tailored for ground robots navigating in outdoor scenarios. To enhance visual constraints in texture-less and highly dynamic scenarios, we integrate line features with an efficient processing method. The proposed pipeline exploits geometric point-line relationships in 2D images for faster line matching while avoiding degenerate motion issues in triangulation. This approach not only accelerates line feature processing but also ensures robust line triangulation. Furthermore, during point feature selection, we employ the MCC to filter out potential dynamic points. The contributions of this work can be summarized as follows:

- A filter-based tightly coupled monocular visual-inertial-wheel odometry system for accurate localization in complex outdoor environments is proposed. Our novel pipeline leverages point-line relationships to enhance real-time performance and robustness.
- MCC is incorporated into our MSCKF-based visual update framework to maximize the utilization of valid point features in dynamic scenes.
- Comprehensive experiments demonstrate that our system achieves superior accuracy, robustness and efficiency compared to existing state-of-the-art methods.

II. RELATED WORKS

In this section, related works are presented and categorised into: Visual-Inertial-Wheel Odometry, Line-aided VIO, and VIO in dynamic environments.

A. Visual-Inertial-Wheel-Odometry

Visual-inertial sensors are among the most widely used sensor combinations for mobile robot state estimation, valued for their complementary characteristics and cost-effectiveness. Numerous representative works have been proposed in recent years, which can generally be categorized into two approaches: filter-based and optimization-based. Prominent filter-based VIO systems, such as S-MSCKF [11], and OpenVINS [4], employ Kalman filters to update system states. In contrast, optimization-based methods, including VINS-Mono [5], and ORB-SLAM3 [6], formulate state estimation as an optimization problem and solve for the optimal system state through iterative refinement.

In VIO system for ground mobile robots, certain movements introduce additional unobservable states, leading to degraded estimation accuracy [2]. Integrating with wheel odometer addresses this issue by making the scale observable and significantly improving estimation accuracy. Lee et al. [12] tightly coupled the wheel with the VIO system based on the MSCKF framework, incorporating online calibration. Similarly, [13] utilized an optimization-based approach and introduced a robust initialization method leveraging wheel measurements. Other studies [14], [15], [16] have further explored planar motion constraints to improve state estimation of ground robots moving in 2D environments.

B. Line-aided VIO

Line features provide additional stable and robust geometric information, allowing the VO or VIO system to improve

accuracy and robustness [7], [8], [17]. However, maintaining real-time performance with line features is often challenging. PL-VINS [18] addresses this by separating feature extraction and optimization into different threads, enabling real-time line-aided VIO. Based on this, [19] proposed PLF-VINS which use point-line coupled and parallel lines relation in the optimization process to enhance accuracy. Additionally, structured line features have demonstrated effectiveness in specific scenarios by leveraging horizontal and vertical lines to impose additional geometric constraints [20], [21]. These methods handle points and lines independently, disregarding the relationship between them. AirVo [22] introduced a line feature pipeline that utilizes learning-based point feature tracking for line tracking and triangulation, improving performance in illumination-challenging environments. Inspired by their work, we propose a novel method that integrates point and line features with their geometry relationship in 2D images, enabling efficient line tracking and robust triangulation in complex outdoor scenarios.

C. VIO in Dynamic Enviroment

In VIO systems, dynamic points often lead to inaccurate estimations. Benefiting from deep learning technology, some works employ semantics segmentation algorithms to classify the potential dynamic points in 2D images [9], [10]. IMU integration which provides short-term 3D position estimation results can serve as a reference for MCC [23]. Similarly, Ground-Fusion [24] uses the wheel pre-integration results as the reference to filter the dynamic features. However, relying solely on a single sensor presents several challenges. For instance, IMU accelerometer noise accumulates significantly after double integration while wheel measurements are limited to 2D transformation information. Therefore, we utilize the historical poses in the sliding window as the reference for MCC to enhance system performance in dynamic scenes.

III. OVERVIEW

For clarity of proposed method, this section first defines the system state, then briefly introduces the MSCKF algorithm, and finally presents the proposed system framework.

A. System Dynamic Model

Similar to the classical MSCKF-Based VINS system [4], the state vector of the proposed system consists of current IMU state $\mathbf{x}_{I_k}$ and n historical IMU clones $\mathbf{x}_{H_k}$.

$$\mathbf{x}_{I_k} = (_G^{I_k}\mathbf{R}, \ ^G\mathbf{p}_{I_k}, \ ^G\mathbf{v}_{I_k}, \ \mathbf{b}_g, \ \mathbf{b}_a) \tag{1}$$

$$\mathbf{x}_{H_k} = (_G^{I_{k-1}}\mathbf{R}, \ ^G\mathbf{p}_{I_{k-1}}, \ \ldots, \ _G^{I_{k-n}}\mathbf{R}, \ ^G\mathbf{p}_{I_{k-n}}) \tag{2}$$

where $_A^B\mathbf{R}$ is the rotation matrix from frame A to B and $^A\mathbf{p}_B$ is position of B in A. The terms $\mathbf{b}_g$ and $\mathbf{b}_a$ denote the gyroscope and accelerometer bias, respectively. In this work, we define the frames as follows: G is the global frame, I is the IMU frame, and C corresponds to the camera frame.

For the Inertial-aided Navigation System (INS), IMU measurements $\mathbf{w}_{m_k}$ and $\mathbf{a}_{m_k}$ at timestamp t_k are used to propagate the state from t_k to t_{k+1}.

$$\mathbf{x}_{I_{k+1}} = \mathbf{f}(\mathbf{x}_{I_k}, \ \mathbf{w}_{m_k}, \ \mathbf{a}_{m_k}) \tag{3}$$
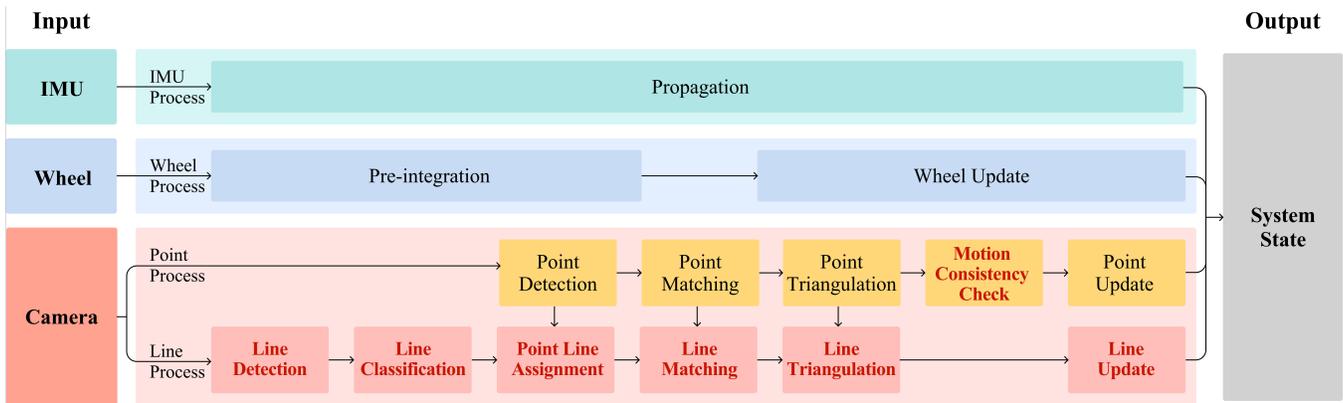
Fig. 2: The framework of PL-VIWO system is divided into four components, each represented by different color-coded boxes. The boxes highlighted with bold red text indicate the key contributions of this work.

B. MSCKF based Visual Update

For feature measurements, the error state model can be generically represented as :

$$\tilde{\mathbf{z}} = \mathbf{z} - \mathbf{h}(\hat{\mathbf{x}}_{I_k}, {}^G\hat{\mathbf{x}}_f) \tag{4}$$

where $\hat{\mathbf{x}}_{I_k}$ is the estimated state and ${}^G\hat{\mathbf{x}}_f$ is the estimated feature position vector in the global frame. Then this error modal can be linearized as:

$$\tilde{\mathbf{z}} = \mathbf{H}_x \hat{\mathbf{x}}_{I_k} + \mathbf{H}_f {}^G\hat{\mathbf{x}}_f + \mathbf{n}_z \tag{5}$$

where $\mathbf{H}_x$ and $\mathbf{H}_f$ are the Jacobians with respect to system state $\mathbf{x}_{I_k}$ and feature ${}^G\mathbf{x}_f$, respectively. And $\mathbf{n}_z$ is the measurement noise. By decomposing $\mathbf{H}_f$, a new measurement modal independent of feature error can be derived :

$$\begin{aligned} \tilde{\mathbf{z}}' &= \mathbf{Q}_n^T \tilde{\mathbf{z}} \\ &= \mathbf{Q}_n^T \mathbf{H}_x \hat{\mathbf{x}} + \mathbf{Q}_n^T \mathbf{H}_f {}^G\hat{\mathbf{x}}_f + \mathbf{Q}_n^T \mathbf{n}_f \\ &= \mathbf{H}'_x \hat{\mathbf{x}}_{I_k} + \mathbf{n}'_z \end{aligned} \tag{6}$$

where $\mathbf{Q}_n^T$ is the null space of $\mathbf{H}_f$ and $\mathbf{Q}_n^T \mathbf{H}_f = \mathbf{0}$. Then the standard EKF update can be performed.

C. System Overview

The proposed framework is illustrated in Fig. 2. Our PL-VIWO system consists of four main components: IMU, Wheel, Point, and Line Process. Built upon MINS [1], our system retains the same IMU and Wheel Process. For the point update, we integrate the MCC module to maximize the inclusion of reliable visual point features in dynamic environments. Additionally, line features are incorporated to reinforce visual constraints with a novel pipeline which exploits the relationship between point and line features for efficient line tracking. To address the challenge of degenerate motions in ground robots, which often prevent line triangulation, we propose a robust line triangulation method that leverages the point-line relationship and point triangulation results. For the line update, MSCKF is employed in the same manner as for the point update.

IV. Methodology

Here, we describe the key components of the proposed approach, with a focus on the contributions.

A. Motion Consistency Check

Outdoor ground robot localization often suffers from estimation inaccuracies due to dynamic objects. A common approach in MSCKF-based VIO to address this issue is to discard point features with large reprojection errors or those failing the Chi-square test during updates. However, in highly dynamic environments, this strategy may result in an insufficient number of valid point features for visual updates, even if numerous point features are extracted, which ultimately degrades localization accuracy.

VIO systems typically extract and track a large number of feature points and only a subset of these features is selected for state update. We introduce MCC during the point feature selection to ensure a sufficient number of static points are used for updates. Specifically, for a point feature ${}^G\mathbf{p}_f$ with n measurements, the average projection error $\mathbf{r}$ is defined as:

$$\mathbf{r} = \frac{1}{n} \sum_{i=1}^{n} \left\| \mathbf{z}_i - \boldsymbol{\pi}({}^{C_i}_G \mathbf{R}({}^G\hat{\mathbf{p}}_f - {}^G\mathbf{p}_{C_i})) \right\| \tag{7}$$

where $\mathbf{z}_i$ is the i_{th} image measurement and ${}^G\hat{\mathbf{p}}_f$ is the 3D position of point feature derived from triangulation. The function $\boldsymbol{\pi}$ is the camera projection function. For feature points whose error $\mathbf{r}$ are greater than the threshold, they are considered dynamic points and excluded from the update.

B. 2D Line Process

The pipeline for 2D line process consist of Line Detection, Line Classification, Point-Line Assignment and Line Matching.

1) Line Detection: Line Segment Detector (LSD) [25] is a classical 2D line detection algorithm that identifies the two endpoints of all line segments in an image. Since the line detection speed is influenced by the image size, the image is downsampled to 0.25 times to enhance efficiency in our system. Additionally, as detected line segments vary in length, longer segments tend to be more accurate and easier

(a) Line detection

(b) Line classification

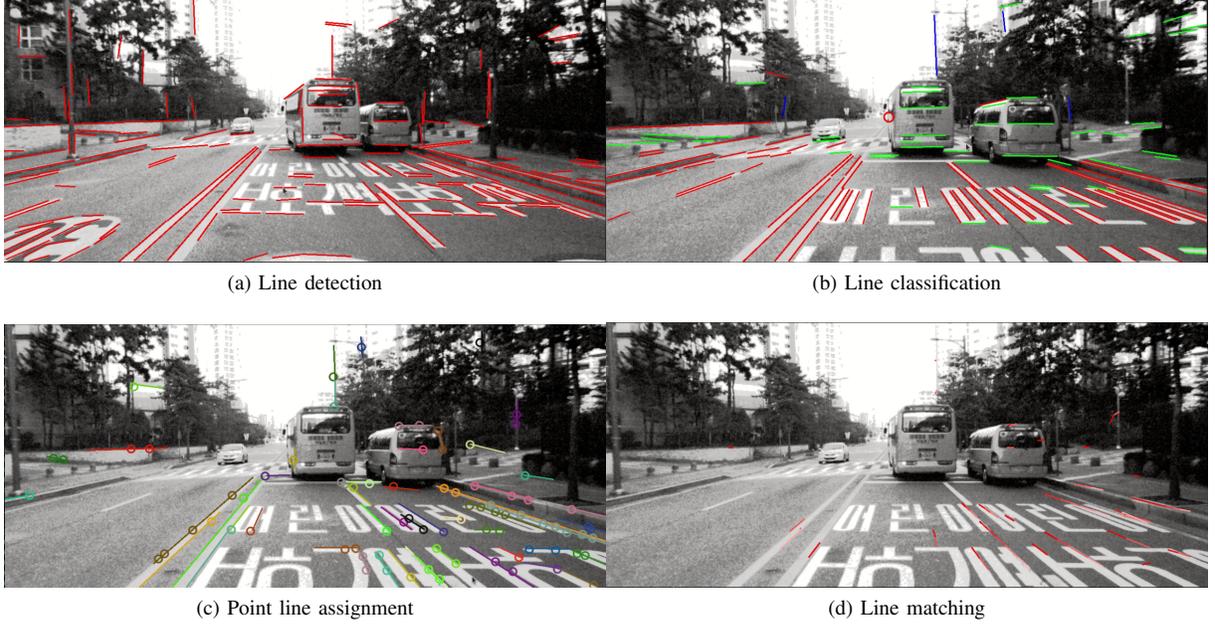(c) Point line assignment

(d) Line matching

Fig. 3: 2D Line process results in KAIST Urban 28. (a) The red line segments represent the detection result. (b) The colored lines represent the classification results: red, green, and blue indicate lines parallel to the x-, y-, and z-axes of the IMU frame, respectively. Non-parallel lines are omitted from the diagram. The red circle around the centre of the image represents the vanishing point corresponding to the x-axis while y and z lie outside the image. (c) Circles lying on a line with the same color represent point features assigned with a line. (d) The colored lines in the figure are formed by connecting the midpoints of all observed line segments within the sliding window. The redder line means more recent.

to track. Therefore, we discard the line segments shorter than 30 pixels and the detection result is shown in Fig. 3 (a).

2) Line Classification: The detected 2D line segments correspond to straight lines in 3D space with various orientations. However, lines aligned with the motion direction suffer from degenerate motion, preventing reliable triangulation via plane intersections from consecutive frames [8]. Therefore, we utilize vanishing points to classify 2D line segments based on their parallelism to different IMU frame axes. During triangulation, the classified lines are assumed to align with their respective coordinate axes.

The vanishing point is the point where parallel lines in 3D space appear to converge in a 2D perspective projection which is widely used to identify parallel relationships in 3D space from 2D images [20], [21]. The vanishing points $\mathbf{vp}_x$ along x-axis, can be formulated as:

$$\mathbf{vp}_x = \boldsymbol{\pi}(^C_I\mathbf{R}\mathbf{u}_x) \qquad (8)$$

where $^C_I\mathbf{R}$ represents the extrinsic rotation matrix and $\mathbf{u}_x$ is the unit vector along x-axis. The same calculation method is applied to obtain the vanishing point for the y and z-axis. By connecting the midpoints of the detected line segments to different vanishing points, new straight lines are formed. The alignment quality is then evaluated by computing the angle error e_{angle} and distance error e_{dist} between these generated lines and the originally detected segments as follows:

$$e_{angle} = \text{atan}(\frac{\mathbf{p}_s(1) - \mathbf{p}_e(1)}{\mathbf{p}_s(0) - \mathbf{p}_e(0)}) - \text{atan}(\frac{\mathbf{p}_m(1) - \mathbf{vp}(1)}{\mathbf{p}_m(0) - \mathbf{vp}(0)}) \quad (9)$$

$$e_{dist} = \frac{\mathbf{n}^\top \begin{bmatrix} \mathbf{p}_s(0) & \mathbf{p}_s(1) & 1 \end{bmatrix} + \mathbf{n}^\top \begin{bmatrix} \mathbf{p}_e(0) & \mathbf{p}_e(1) & 1 \end{bmatrix}}{2\sqrt{\mathbf{n}(0)^2 + \mathbf{n}(1)^2}}$$

(10)

where $\mathbf{p}_s$, $\mathbf{p}_e$, $\mathbf{p}_m$ are the startpoint, endpoint and midpoint, respectively. And $\mathbf{n}$ is the norm vector formed by vanishing point and midpoint:

$$\mathbf{n} = \begin{bmatrix} \mathbf{p}_m(0) & \mathbf{p}_m(1) & 1 \end{bmatrix}^\top \times \begin{bmatrix} \mathbf{vp}(0) & \mathbf{vp}(1) & 1 \end{bmatrix}^\top \quad (11)$$

A line segment and a vanishing point are considered collinear only if both errors are below the threshold. The classification result is shown in Fig. 3(b).

3) Point-Line Assignment: By analyzing the distance between points and lines in a 2D image, the correspondence between them can be established directly. Since the points tracking results have already been obtained during the point feature update, the tracking results of line features can be determined based on the associated point tracking results. This approach eliminates the need for line feature descriptors, significantly accelerating line matching.

For point $\mathbf{p}$: $\begin{bmatrix} u_p & v_p \end{bmatrix}^\top$ and line $\mathbf{l}$: $\begin{bmatrix} u_s & v_s & u_e & v_e \end{bmatrix}^\top$ on 2D image, the distance between them can be computed by:

$$d = \begin{cases} \sqrt{(u_p - u_s)^2 + (v_p - v_s)^2}, & \text{if } cross \leq 0 \\ \sqrt{(u_p - u_e)^2 + (v_p - v_e)^2}, & \text{if } cross > len^2 \\ \frac{|(v_e - v_s)u_p + (u_s - u_e)v_p + (u_e v_s - u_s v_e)|}{\sqrt{(v_e - v_s)^2 + (u_s - u_e)^2}}, & \text{otherwise} \end{cases}$$
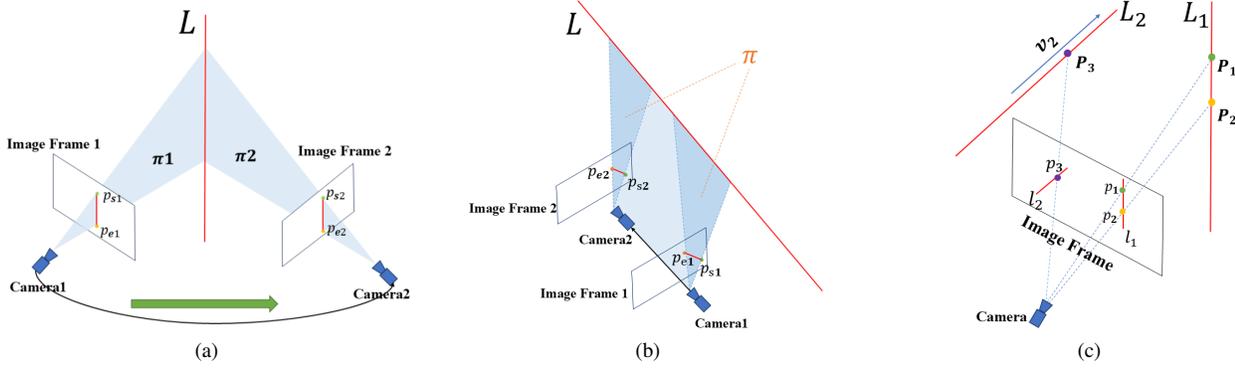
Fig. 4: Line triangulation (a) Triangulation from planes. (b) Degenerate motion for line feature triangulation. Cameras stay in the same plane π. (c) Triangulation from points and direction.

where $cross$ is the projection of the point on the line segment and len is the length of line segment.

$$len = \sqrt{(u_s - u_e)^2 + (v_s - v_e)^2}$$
$$cross = (u_e - u_s)(u_p - u_s) + (v_e - v_s)(v_p - v_s) \quad (12)$$

For $cross < 0$, the point project on the left side of the line startpoint $\mathbf{p}_s$. Conversely, its projection lies on the right side of endpoint $\mathbf{p}_e$ when $cross > d^2$. A 2D point is considered to be on a line only if its projection falls between the two endpoints and the distance to the line is less than 3 pixels. The point line assignment result is shown in Fig. 3(c).

4) Line Matching: The line features can be tracked by using point matching and point line assignment results. AirVO [22] scores matches based on the number of points which is unsuitable for outdoor complex scenes. A possible reason is that line features in such environments are scattered and numerous, making it unlikely for multiple lines to share the same points. Therefore, we adopt two simpler rules:

- Two line segments in adjacent frames are considered the same 3D line if they share at least two point features.
- If only one point is assigned, the differences in position and direction between the two line segments are calculated. If both differences are below the threshold, they are considered to be the same. (We assume a line maintains the same direction and undergoes limited positional changes between adjacent frames).

Fig. 3(d) shows the line matching results.

C. Line Triangulation

This paper uses the Plücker coordinate to represent a 3D spatial line $\mathbf{L}$, as it facilitates spatial transformations. It consists of a normal vector $\mathbf{n}$ and a direction vector $\mathbf{v}$.

$$\mathbf{L} = \begin{bmatrix} \mathbf{n} & \mathbf{v} \end{bmatrix}^\top \quad (13)$$

Classical line triangulation methods [8] project 2D line segments into 3D as planes. The intersection of planes forms the 3D line, as illustrated in Fig. 4(a). However, this triangulation method suffers from degenerate motion. When the camera moves parallel to the line direction, the two formed planes coincide making it impossible to recover the spatial

line, like in Fig. 4 (b). This issue is particularly common in 2D urban navigation for ground robots, where line features such as lane markings and curbs cannot be triangulated when the robot moves in a straight line. Consequently, even if most line features are successfully tracked, they cannot be used for visual updates without accurate triangulation.

To address this, we introduce two novel line triangulation methods to handle the cases when classical triangulation fails. Since points and lines are associated in IV-B.3, and the direction of some lines has been determined in IV-B.2, this additional information is leveraged in our methods. Firstly, if more than two triangulated points are assigned to a line $\mathbf{L}$, then choose the closest two (ie. $\mathbf{p}_1$ and $\mathbf{p}_2$) and a 3D line can be obtained, as illustrated in Fig. 4(c) right part.

$$\mathbf{L} = \begin{bmatrix} \mathbf{n} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1 \times \mathbf{p}_2 \\ \frac{\mathbf{p}_1 - \mathbf{p}_2}{|\mathbf{p}_1 - \mathbf{p}_2|} \end{bmatrix}^\top \quad (14)$$

However, triangulating points in outdoor dynamic scenes is challenging, there are cases where only one point $\mathbf{p}_3$ has been triangulated. In such cases, the second triangulation method will be used. If the line direction is parallel to x-axis of the IMU coordinate, the direction vector can be directly obtained from the rotation matrix $^I_G\mathbf{R}$, as shown in Fig. 4(c) left:

$$\mathbf{v}_x = {}^I_G\mathbf{R}\mathbf{u}_x \quad (15)$$

Where $\mathbf{u}_x$ is the unit vector along x-axis. Lines parallel to y and z-axis can be determined in the same way. Then the norm vector $\mathbf{n}$ can be derived by using point $\mathbf{p}_3$:

$$\mathbf{n} = \mathbf{p}_3 \times \mathbf{v} \quad (16)$$

D. Line Measurement Model

The line measurement model describes the distance between the observed two endpoints $\mathbf{p}_s$ and $\mathbf{p}_e$, to the projected line $\mathbf{l}$ in the 2D image coordinate:

$$\mathbf{z_l} = \begin{bmatrix} \mathbf{d}_s & \mathbf{d}_e \end{bmatrix}^\top = \begin{bmatrix} \frac{\mathbf{p}_s^\top \mathbf{l}}{\sqrt{l_1^2 + l_2^2}} & \frac{\mathbf{p}_e^\top \mathbf{l}}{\sqrt{l_1^2 + l_2^2}} \end{bmatrix}^\top \quad (17)$$

where $\mathbf{p_s} = \begin{bmatrix} u_s & v_s & 1 \end{bmatrix}^\top$ and $\mathbf{p_e} = \begin{bmatrix} u_e & v_e & 1 \end{bmatrix}^\top$. The projected line $\mathbf{l}$ can be derived by the camera pose and Plücker coordinates of 3D line in the global frame $^G\mathbf{L}$:
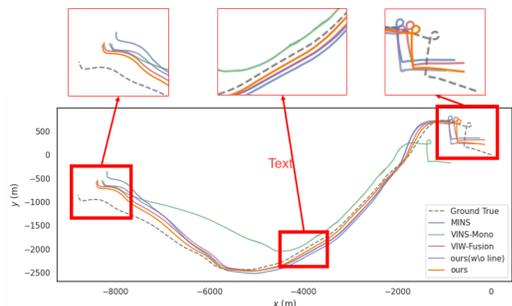
Fig. 5: The KAIST Urban 31 trajectory (PL-VINS failed).

Fig. 6: Bar chart of average process time between frames and CPU usage across different algorithms.

$$l = \begin{bmatrix} \mathbf{K} & \mathbf{0}_3 \end{bmatrix} {}^C\mathbf{L} = \begin{bmatrix} \mathbf{K} & \mathbf{0}_3 \end{bmatrix} \begin{bmatrix} {}^C\mathbf{n} & {}^C\mathbf{v} \end{bmatrix}^\top$$
$$= \begin{bmatrix} \mathbf{K} & \mathbf{0}_3 \end{bmatrix} \begin{bmatrix} {}^C_G\mathbf{R} & [{}^C\mathbf{p}_G]_\times {}^C_G\mathbf{R} \\ \mathbf{0}_3 & {}^C_G\mathbf{R} \end{bmatrix} \begin{bmatrix} {}^G\mathbf{n} & {}^G\mathbf{v} \end{bmatrix}^\top$$

(18)

where $\mathbf{K}$ is camera intrinsic matrix for line projection and $[\mathbf{x}]_\times$ represent the skew-symmetric matrix of $\mathbf{x}$. The Jacobian matrices of the measurement model with respect to the IMU pose and line features follow a similar formulation as presented in the Appendix of [8].

V. EXPERIMENTS

We evaluated the PL-VIWO using the publicly available KAIST Complex Urban Dataset [26]. To assess the computational efficiency and localization accuracy, we compared the proposed PL-VIWO against state-of-the-art monocular VIO and VIWO methods. The benchmark includes three optimization-based algorithmes: VINS-Mono[5], PL-VINS[18], and VIW-Fusion[16], as well as filter-based: MINS[1]. For fairness, we configure MINS to use only monocular camera, IMU, and wheel information. Since PL-VIWO is built on MINS with MCC and line features, MINS serves as a key benchmark to highlight the effectiveness of proposed methods. PL-VINS is a VIO system built upon VINS-Mono integrated line features, while VIW-Fusion is a VIWO system that incorporates the wheel information into VINS-Mono. All experiments run on an Intel Core i7 CPU, 32 GB of RAM, and an NVIDIA A1000 GPU (6 GB VRAM) with Ubuntu 20.04.

Note that we initially used the same IMU parameters for both types which resulted in suboptimal performance for the optimization-based algorithms. To ensure fair and reliable evaluation, we tuned IMU noise in optimization-based methods for their best performance. Up to 150 points were used for visual updates. Additionally, we disabled the online calibration of both internal and external parameters, as well as the loop closure in all algorithms for fairness.

A. Localization Accuracy

Due to the varying test scenarios, we divided the localization experiments into two groups: highway and city scenes. The localization accuracy is evaluated using the root mean squared error (RMSE) of the absolute pose (position and orientation), computed by EVO [27]. Optimization-based algorithms are distinguished with a grey background in the result tables. Meanwhile, position errors above 300m or test not completed the test are listed as FAIL. The best results are **highlighted** and the second best are underlined.

1) Highway: Table I shows the localization results for highway scenarios. Since motion on highways is mostly uniform and linear, the VINS system suffers from degenerate motion. Additionally, visual constraints are weak due to the lack of texture information in highway environments. These factors lead to poor performance for both VINS-Mono and PL-VINS. VINS coupled with wheel achieves higher accuracy due to the incorporation of wheel information. Compared with MINS, our system utilizes more valid point features during the visual update process, resulting in improved accuracy even without incorporating line features. Our system further improves and achieves the best accuracy in 8 out of 11 sequences after introducing line features.

2) City: The localization results in city scenarios are shown in Table II. On urban roads, due to frequent changes in vehicle speed and the abundance of texture, VINS-Mono can achieve relatively accurate results. However, its accuracy remains lower than the system coupled with wheel in most sequences. PL-VINS fails in most of the scenarios, possibly due to dynamic lines, such as lines on moving vehicles, being incorrectly triangulated and added to the optimization process. Compared with MINS, our system demonstrates significant improvements in both translation and rotation accuracy due to the introduction of more effective visual constraints. These improvements are further enhanced after incorporating line features. VIW-Fusion achieves the best results in several sequences due to its optimization-based framework, which enhances robustness at the cost of higher computation. Overall our approach is the best in 5 out of 10 sequences and second best in 4 out of 10 sequences. The trajectories of different algorithms in Ubran 31 are shown in Fig. 5. The trajectory from PL-VIWO with line is the closest to the ground truth among all methods, further validating the effectiveness of our method.

B. Computation Efficiency Analysis

To assess the computational efficiency of the proposed method, we analyze the following two modules.

1) Feature Process Time: First, we record and compare the processing time of 2D image feature process in our

system with PL-VINS, as shown in Table III. Our system achieves faster performance in point extraction and tracking by employing a grid-based extraction method, which prevents redundant feature extraction in certain grid regions. For line extraction, we downscale the image, reducing the processing time to 7ms. Additionally, our line-matching approach avoids descriptor computation, achieving a processing time of just 0.5ms, whereas PL-VINS takes 6ms due to descriptor calculation. Overall, our proposed pipeline demonstrates greater efficiency in 2D image processing.

2) Total Process Time & CPU Usage: Additionally, the processing time and CPU usage of each algorithm are recorded in Fig. 6. For filtering-based algorithms, the processing time includes adjacent frame image processing and state updates. In contrast, in optimization-based algorithms, the processing time between adjacent keyframes consists of image processing and optimization. Filter-based algorithm frameworks require less processing time compared to optimization-based. Introducing line features and wheel data will add additional processing time. Regarding CPU usage, optimization-based methods employ additional threads for optimization. Consequently, their CPU usage is significantly higher than that of filtering-based algorithms, which run on a single thread. This makes our algorithm more suitable for deployment on resource-constrained edge devices.

C. Line Triangulation Results

To verify the effectiveness of the proposed line feature triangulation methods, we present the triangulation results
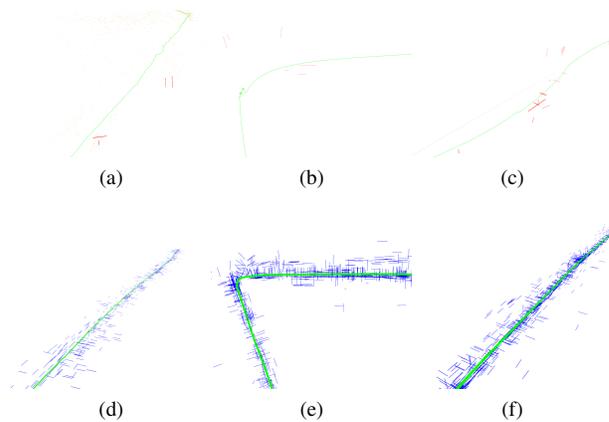

Fig. 7: Comparison of line triangulation results on various sequences. The top row presents the line triangulation results of PL-VINS with red lines, while the bottom row illustrates the results of PL-VIWO with blue lines. (a)(d) Urban23; (b)(e) Urban28; (c)(f) Urban32.

of PL-VINS and PL-VIWO on multiple sequences from the KAIST Urban datasets in Fig.7. PL-VINS triangulation relies solely on the intersection of two planes, and is degenerate for most 2D straight movements, resulting in a limited number of lines successfully triangulated. Ours incorporates the point-line relationship and the number of triangulated lines increases significantly. This indicates that more geometric constraints are utilized in the state update,

TABLE I: RMSE ATE: POSITION(m) & ORIENTATION($°$) FOR ALGORITHMS IN HIGHWAY SEQUENCES.

Algorithms	urban18 (3.9km)	urban19 (3.0km)	urban20 (3.2km)	urban21 (3.7km)	urban22 (3.4km)	urban23 (3.4km)	urban24 (4.2km)	urban25 (2.5km)	urban35 (3.2km)	urban36 (9.0km)	urban37 (11.8km)
VINS-mono	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL
PL-VINS	FAIL	FAIL	FAIL	FAIL	FAIL	208.26 7.10	FAIL	FAIL	256.70 159.09	FAIL	FAIL
VIW-Fusion	**43.60** 4.27	37.57 6.80	38.65 155.40	52.86 160.25	53.05 156.0	42.28 12.00	55.88 55.12	**29.46** **9.01**	36.77 10.35	116.98 20.02	174.87 158.55
MINS(I,C,W) -mono	50.58 2.94	37.19 6.02	40.67 9.39	46.33 12.67	**46.12** 16.71	42.53 5.95	**47.65** 36.93	36.60 37.30	55.01 15.68	117.89 6.34	197.53 4.16
PL-VIWO (w/o line)	52.50 2.86	37.08 3.27	40.40 10.18	47.37 **6.06**	48.55 **15.30**	43.00 2.01	48.76 24.57	31.38 28.98	40.01 10.37	116.63 4.85	178.50 4.05
PL-VIWO	47.99 **2.80**	**36.18** **3.22**	**36.86** **6.74**	46.15 6.43	49.77 15.80	**41.10** **1.00**	51.32 24.15	31.01 16.80	**36.09** **8.51**	105.46 **4.40**	136.04 **3.76**

TABLE II: RMSE ATE: POSITION(m) & ORIENTATION($°$) FOR ALGORITHMS IN URBAN SEQUENCES.

| Algorithms	urban26 (4.0km)	urban27 (5.4km)	urban28 (11.5km)	urban29 (3.6km)	urban30 (6.0km)	urban31 (11.4km)	urban32 (7.1km)	urban33 (7.6km)	urban34 (7.8km)	urban38 (11.4km)
VINS-mono	35.43 3.58	151.34 7.04	94.77 **3.58**	FAIL	110.40 7.35	FAIL	74.24 **5.90**	FAIL	FAIL	170.39 8.40
PL-VINS	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	154.41 8.83	134.07 7.38	291.33 18.47	284.88 18.03
VIW-Fusion	**23.86** 3.31	124.01 32.35	**32.28** 5.05	**42.26** **4.70**	**40.72** **5.56**	230.15 14.08	86.11 7.03	127.52 16.63	39.18 **3.62**	**48.12** 7.60
MINS(I,C,W) -mono	34.87 3.87	68.87 6.79	91.00 11.79	68.41 9.80	73.56 10.37	334.15 16.83	93.89 9.45	85.47 8.20	40.64 6.00	104.04 10.78
PL-VIWO (w/o line)	36.30 3.51	55.24 7.19	73.54 7.70	66.10 9.62	64.01 5.54	302.19 13.47	78.08 8.32	**75.29** 9.05	37.99 6.19	96.16 9.75
PL-VIWO	28.84 **2.27**	**47.67** **3.03**	49.19 7.00	47.34 6.83	65.67 6.63	**191.77** **11.15**	**65.96** 7.78	86.63 **6.88**	**33.60** 5.54	73.31 **6.98**
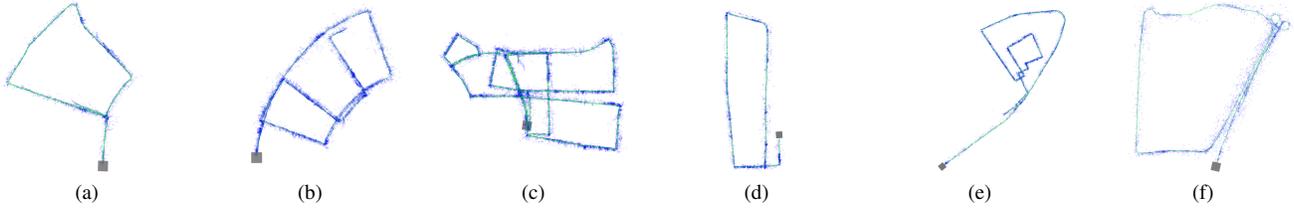
Fig. 8: Trajectories with lines in (a) urban26; (b) urban27; (c) urban28; (d) urban30; (e) urban33; (f) urban34.

TABLE III: Comparison of Average Feature Processing Time of Principal Component with PL-VINS in Urban38.

	Point Extraction	Line Extraction	Point Matching	Line Matching	Total Time
PL-VINS	10 ms	23 ms	3.0 ms	6 ms	42 ms
PL-VIWO	5 ms	7 ms	2.5 ms	0.5 ms	15 ms

enhancing accuracy and robustness. The trajectories with line features across multiple challenging urban sequences are shown in Fig. 8, highlighting the effectiveness of the proposed triangulation method in complex environments.

VI. CONCLUSIONS

In this paper, we propose a low-cost monocular VIWO system that is robust and accurate for complex outdoor navigation. To enforce visual constraints, we integrate line features through a novel processing pipeline that leverages the point-line relationship. This approach enables efficient tracking and triangulation of line features, ultimately enhancing system accuracy. Additionally, to improve reliability in dynamic environments, we incorporate MCC to ensure a sufficient number of valid feature points for visual updates. Extensive experiments validate the efficiency and accuracy of our approach. In future work, we plan to extend PL-VIWO to a stereo version and further investigate constraints between points and lines, like point-on-line and parallel lines.

REFERENCES

[1] W. Lee et al., 'MINS: Efficient and Robust Multisensor-aided Inertial Navigation System', arXiv preprint arXiv:2309. 15390, 2023.

[2] K. J. Wu et al., 'VINS on wheels', in 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 5155–5162.

[3] R. Mur-Artal et al., 'ORB-SLAM: a versatile and accurate monocular SLAM system', IEEE transactions on robotics, vol. 31, no. 5, pp. 1147–1163, 2015.

[4] P. Geneva et al., 'Openvins: A research platform for visual-inertial estimation', in 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 4666–4672.

[5] T. Qin et al., 'Vins-mono: A robust and versatile monocular visual-inertial state estimator', IEEE transactions on robotics, vol. 34, no. 4, pp. 1004–1020, 2018.

[6] C. Campos et al., 'Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam', IEEE Transactions on Robotics, vol. 37, no. 6, pp. 1874–1890, 2021.

[7] Y. He et al., 'PL-VIO: Tightly-coupled monocular visual–inertial odometry using point and line features', Sensors, vol. 18, no. 4, p. 1159, 2018.

[8] Y. Yang et al., 'Visual-Inertial Odometry with Point and Line Features', in 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 2447–2454.

[9] C. Yu et al., 'DS-SLAM: A semantic visual SLAM towards dynamic environments', in 2018 IEEE/RSJ international conference on intelligent robots and systems (IROS), 2018, pp. 1168–1174

[10] J. Liu et al., 'RGB-D Inertial Odometry for a Resource-Restricted Robot in Dynamic Environments', IEEE Robotics and Automation Letters, vol. 7, no. 4, pp. 9573–9580, 2022.

[11] K. Sun et al., 'Robust stereo visual inertial odometry for fast autonomous flight', IEEE Robotics and Automation Letters, vol. 3, no. 2, pp. 965–972, 2018.

[12] W. Lee et al., 'Visual-inertial-wheel odometry with online calibration', in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020, pp. 4559–4566.

[13] J. Liu et al., 'Visual-inertial odometry tightly coupled with wheel encoder adopting robust initialization and online extrinsic calibration', in 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 5391–5397.

[14] F. Zheng et al., 'Visual-odometric localization and mapping for ground vehicles using SE (2)-XYZ constraints', in 2019 International Conference on Robotics and Automation (ICRA), 2019, pp. 3556–3562.

[15] M. Zhang et al., 'Pose estimation for ground robots: On manifold representation, integration, reparameterization, and optimization', IEEE Transactions on Robotics, vol. 37, no. 4, pp. 1081–1099, 2021.

[16] T. Zhuang, 'VIW-Fusion: visual-inertial-wheel fusion odometry', 2021. [Online]. Available: https://github.com/TouchDeeper/VIW-Fusion.

[17] A. Pumarola et al., 'PL-SLAM: Real-time monocular visual SLAM with points and lines', in 2017 IEEE international conference on robotics and automation (ICRA), 2017, pp. 4503–4508.

[18] Q. Fu et al., 'PL-VINS: Real-time monocular visual-inertial SLAM with point and line features', arXiv preprint arXiv:2009. 07462, 2020.

[19] J. Lee and S.-Y. Park, 'PLF-VINS: Real-time monocular visual-inertial SLAM with point-line fusion and parallel-line fusion', IEEE robotics and automation letters, vol. 6, no. 4, pp. 7033–7040, 2021.

[20] D. Zou et al., 'StructVIO: Visual-Inertial Odometry With Structural Regularity of Man-Made Environments', IEEE Transactions on Robotics, vol. 35, no. 4, pp. 999–1013, 2019.

[21] C. Pang et al., 'Structural Lines Aided Monocular Visual-Inertial-Wheel Odometry With Online IMU-Wheel Extrinsic Optimization on $\mathbb{S}^2$ Manifold', IEEE Transactions on Intelligent Vehicles, vol. 9, no. 2, pp. 4100–4114, 2024.

[22] K. Xu et al., 'Airvo: An illumination-robust point-line visual odometry', in 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2023, pp. 3429–3436.

[23] S. Song et al., 'DynaVINS: a visual-inertial SLAM for dynamic environments', IEEE Robotics and Automation Letters, vol. 7, no. 4, pp. 11523–11530, 2022.

[24] J. Yin et al., 'Ground-Fusion: A Low-cost Ground SLAM System Robust to Corner Cases', in 2024 IEEE International Conference on Robotics and Automation (ICRA), 2024, pp. 8603–8609.

[25] R. G. Von Gioi et al., 'LSD: A line segment detector', Image Processing On Line, vol. 2, 2012.

[26] J. Jeong et al., 'Complex urban dataset with multi-level sensors from highly diverse urban environments', The International Journal of Robotics Research, vol. 38, no. 6, pp. 642–657, 2019.

[27] M. Grupp, "EVO: Python package for the evaluation of odometry and SLAM," 2017. [Online]. Availbale: https://github.com/MichaelGrupp/evo, 2017