Abstract—In this paper, we present a tightly-coupled monocular visual-inertial navigation system (VINS) using points and lines with degenerate motion analysis for 3D line triangulation. Based on line segment measurements from images, we propose two sliding window based 3D line triangulation algorithms and compare their performance. Analysis of the proposed algorithms reveals 3 degenerate camera motions that cause triangulation failures. Both geometrical interpretation and Monte-Carlo simulations are provided to verify these degenerate motions which prevent triangulation. In addition, commonly used line representations are compared through a monocular visual SLAM Monte-Carlo simulation. Finally, real-world experiments are conducted to validate the implementation of the proposed VINS system using the “closest point” line representation.

I. INTRODUCTION AND RELATED WORK

Over the past decade, visual-inertial navigation systems (VINS) have seen a great increase in popularity due to their ability to provide accurate localization solutions while utilizing only low-cost inertial measurement units (IMUs) and cameras. The affordability, size, and light-weight nature of these sensors make them ideal for deployments in a wide-range of applications such as unmanned aerial vehicles (UAVs) [1] and mobile devices [2].

When fusing camera and IMU data, a key question is how to best utilize the rich amount of information available in the images. In particular, most VINS can be categorized by whether they use indirect or direct image processing techniques. Direct visual methods provide motion estimates by minimizing costs involving the raw pixel intensities captured by the camera [3, 4]. By contrast, indirect systems typically extract geometric features from the pixels and track their motion across the image plane with the most common being points [5–7] and/or lines [8–13]. Point features correspond to 3D positions in the space that are detected as “corners” on the image plane. Lines, which are most commonly seen in human-built environments, are detected as straight edges in the image. Quite a few navigation works have been devoted to utilizing both geometric features to achieve robust and accurate estimation.

In particular, leveraging the multi-state constraint Kalman filter (MSCKF) framework [5], Kottas et al. [8] proposed the use of a unit quaternion with distance scalar to model a line and incorporate line features into visual-inertial odometry (VIO). In addition, they provided an observability analysis showing that VINS with a single line feature suffers from 5 unobservable directions. Later on, Kottas et al. [9] and Guo et al. [14] proposed the utilization of line constraints (e.g., parallel lines and vertical lines) to improve the estimation accuracy in structured environments such as indoor or Manhattan world, where all lines lay along the 3 major directions. Yu et al. [11] designed a VINS system with line features suitable for rolling-shutter cameras. Utilizing the Plücker representation and orthonormal error states [15] for lines, Heo et al. [12] implemented an invariant MSCKF with point and line features using a Lie group representation of the state. Zheng et al. [13] proposed representing a line with two 3D endpoints and designed a stereo visual-inertial navigation system leveraging both points and lines. He et al. [10] used IMU preintegration [16] to design a batch-optimization based visual-inertial SLAM with point and line features, where line features are represented in Plücker coordinates [17].

In many of these previous works, different representations for line features were used without considering their impacts on estimator performance. Leveraging our previous work [18], which offered a brief survey of line representations and proposed a “closest point” representation, in this paper we conduct a performance evaluation for the commonly used line parameterizations. In addition, we investigate and identify the degenerate motions for line feature triangulation which cause poor line estimates and have large practical impacts for monocular VINS which might leverage line features. The main contributions of this paper can be listed as follows:

- We present a tightly-coupled monocular visual-inertial navigation system which leverages point and line features and performs online spatial and temporal calibration.
- We propose two sliding window based line feature triangulation algorithms and compare their performances. We identify 3 degenerate motions that cause these line triangulation methods to fail.
- We investigate commonly used line representations and numerically compare their performances in a line feature based monocular visual-inertial SLAM scenario.
- Real-world experiments are performed to validate the designed system with the CP line representation, and the performances are shown to be improved than the system using points only.

II. PROBLEM FORMULATION

In order to properly contextualize our line analysis, we first formulate the standard visual-inertial odometry problem...
when using both point and line features. We define the state vector for visual-inertial odometry as:
\[
\mathbf{x} = \begin{bmatrix} \mathbf{x}_I^T & \mathbf{x}_{\text{calib}}^T & t_d & \mathbf{x}_c^T \end{bmatrix}^T
\]
where \(\mathbf{x}_I\) denotes the IMU state, \(\mathbf{x}_{\text{calib}}\) denotes the rigid transformation between the IMU and camera, \(t_d\) represents the time-offset and \(\mathbf{x}_c\) represents the cloned IMU states. At time step \(k\), the state vector can be written as:
\[
\mathbf{x}_k = \begin{bmatrix} \mathbf{l}_G^q \mathbf{q}_k^T & \mathbf{l}_k^b_k & \mathbf{G}_k^T & \mathbf{l}_k^b_a & \mathbf{G}_k^p_h \end{bmatrix}^T
\]
(2)
where \(\mathbf{l}_G^q\) denotes JPL quaternion [19], representing the rotation from global frame \{G\} to IMU frame \{I\} at time step \(k\). \(\mathbf{G}_k^I\) and \(\mathbf{G}_k^p\) represent the IMU velocity and position in the global frame at time step \(k\), respectively, \(\mathbf{l}_k^b\) and \(\mathbf{l}_k^b\) represent the gyroscope and accelerometer biases. We define the error states of the IMU state as:
\[
\tilde{\mathbf{x}}_k = \begin{bmatrix} \tilde{\mathbf{q}}_k^T \mathbf{G}_k^T \mathbf{l}_k^b \mathbf{l}_k^b_a \mathbf{G}_k^p_h \end{bmatrix}^T
\]
(3)
where the error, \(\tilde{\mathbf{x}}\), is defined as the difference between the true state, \(\mathbf{x}\), and estimated state \(\hat{\mathbf{x}}\), that is, \(\tilde{\mathbf{x}} := \mathbf{x} - \hat{\mathbf{x}}\). However, for the JPL quaternion, the error state is defined as \(\delta \mathbf{q}\), where:
\[
\tilde{\mathbf{q}} = \delta \mathbf{q} \times \hat{\mathbf{q}} \simeq \begin{bmatrix} 1^T & \delta \mathbf{q} \end{bmatrix}^T \delta \mathbf{q}
\]
(4)
with \(\otimes\) denotes the quaternion multiplication [19].

In addition to this IMU navigation state, we also estimate the spatial calibration, \(\mathbf{x}_{\text{calib}}\), between the IMU and camera. In particular, our state vector contains the rotation from the IMU frame to the camera frame, \(\mathbf{R}_G^c\), as well as the translation from camera to IMU, \(\mathbf{G}_c^p\). Explicitly, we have:
\[
\mathbf{x}_{\text{calib}} = \begin{bmatrix} \mathbf{G}_c^p \mathbf{R}_G^c \end{bmatrix}^T
\]
(5)
Moreover, due to the nature of electronic hardware (e.g., asynchronous clocks, data transmission delays and electronic triggering delays), the timestamps reported by each of the sensors will differ from the “true” time that the measurements were recorded. In this work, we treat the IMU clock as the true time and estimate the offset of the aiding sensor relative to this base clock [20, 21]. We model the time offset \(t_d\) as a constant value:
\[
t_d = t_C - t_I
\]
(6)
where \(t_C\) is the time recorded on the sensor measurements, and \(t_I\) is the corresponding true IMU time.

If there are \(m\) cloned IMU poses at the time step \(k\) (corresponding to the poses of the IMU at the true imaging times), then \(\mathbf{x}_k\) can be written as:
\[
\mathbf{x}_k = \begin{bmatrix} \mathbf{G}_k^{l_{k-1}} \mathbf{q}^{l_{k-1}} & \cdots & \mathbf{G}_k^{l_{k-m}} \mathbf{q}^{l_{k-m}} & \mathbf{G}_k^{p_{h-1}} & \mathbf{G}_k^{p_{h-m}} \end{bmatrix}^T
\]
(7)

A. System Dynamic Model

The linear circular acceleration \(a\) and angular velocity measurements \(\omega\) are modeled with additive noises and biases:
\[
\mathbf{a}_m = \mathbf{a} + \mathbf{G} \mathbf{a} + \mathbf{b}_a + \mathbf{n}_a
\]
(8)
\[
\mathbf{\omega}_m = \mathbf{\omega} + \mathbf{b}_\omega + \mathbf{n}_\omega
\]
(9)
where \(\mathbf{a}_m\) and \(\mathbf{\omega}_m\) are the continuous-time Gaussian noises that contaminate the IMU readings. The dynamic model for this system is given by [19]:
\[
\begin{aligned}
\dot{\mathbf{q}}_k(t) &= \frac{1}{2} \Omega \left( \mathbf{q}_k(t) \right) \dot{\mathbf{q}}_k(t), \\
\mathbf{G}_k^p(t) &= \mathbf{G}_k^p(t), \\
\mathbf{G}_k^p(t) &= \mathbf{G}_k^p(t) + \mathbf{n}_w, \quad \mathbf{b}_a(t) = \mathbf{n}_w(t), \quad t_d = 0 \\
\mathbf{x}_{\text{calib}}(t) &= 0_{6 \times 1}, \quad \dot{\mathbf{x}}(t) = 0_{6m \times 1}
\end{aligned}
\]
(10)
where \(\mathbf{n}_w\) and \(\mathbf{n}_w\) denote the zero-mean Gaussian noises driving the IMU gyroscopic and accelerometer biases, \(\mathbf{G}^p\) denotes gravity, \(\mathbf{\Omega} \left( \mathbf{q} \right)\) represents the skew matrix, \(\mathbf{\Omega} \left( \mathbf{q} \right) \triangleq \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}\) and continuous time variable \(\mathbf{x}(t)\) denotes the state value at time \(t\). After linearization, the error state dynamic equation are:
\[
\dot{\tilde{\mathbf{x}}} (t) \simeq \begin{bmatrix} \mathbf{F}_I(t) & \mathbf{0}_{15 \times (6m+7)} \\ \mathbf{G}_I(t) & \mathbf{0}_{(6m+7) \times 12} \end{bmatrix} \tilde{\mathbf{x}}(t) + \begin{bmatrix} \mathbf{G}_I(t) \\ \mathbf{0}_{15 \times (6m+7)} \end{bmatrix} \mathbf{n}(t)
\]
(11)
where \(\mathbf{F}_I(t)\) and \(\mathbf{G}_I(t)\) are the continuous-time IMU error state and noise Jacobians matrices, respectively, and \(\mathbf{n}(t) = \begin{bmatrix} \mathbf{n}_p^T & \mathbf{n}_w^T \end{bmatrix}^T\) represents the system noise models as a zero-mean white Gaussian process with autocorrelation \(\mathbb{E}[\mathbf{n}(t) \mathbf{n}^T(t-\tau)] = \mathbf{Q}(\tau - \tau)\). To propagate the covariance \(\mathbf{P}_{k|k}\) at time step \(k\), the state transition matrix \(\mathbf{F}_{k|k+1}\) from time \(t_k\) to \(t_{k+1}\) can be computed by solving \(\mathbf{F}_{k|k+1} = \mathbf{F}(t_k)\mathbf{F}_{k|k+1}\) with identity initial conditions. Thus, the discrete-time noise covariance and the propagated covariance can be written as:
\[
\mathbf{Q}_k = \int_{t_k}^{t_{k+1}} \mathbf{G}(t) \mathbf{Q} \mathbf{G}^T(t) \mathbf{F}_k(t) \mathbf{d}t
\]
(12)
\[
\mathbf{P}_{k+1|k} = \mathbf{F}(t_k) \mathbf{P}_{k|k} \mathbf{F}_k^T + \mathbf{Q}_k
\]
(13)
B. Measurement Model

As the camera moves through an environment, point feature measurements can be extracted and tracked between images. These camera measurements are described by:
\[
\mathbf{z}_p = \Pi \left( \begin{bmatrix} \mathbf{c}_{\mathbf{x}_p} + \mathbf{n}_p \end{bmatrix}, \mathbf{f} \right) = \begin{bmatrix} \mathbf{x}^T \mathbf{y} \end{bmatrix}
\]
(14)
where \(\mathbf{c}_{\mathbf{x}_p}\) represents the 3D position of the point feature expressed in the camera frame. According to our time offset definition (6), the feature \(\mathbf{c}_{\mathbf{x}_p}\) in the sensor frame with reported time stamp \(t\) corresponds to the time \(t - t_d\) in the IMU base clock. Hence, we have:
\[
\mathbf{c}_{\mathbf{x}_p} = \int_{t_d}^{t} \mathbf{R}_G^c \mathbf{R}(t - t_d) \left( \mathbf{G}_{\mathbf{x}_p} - \mathbf{G}_{\mathbf{p}_I(t-t_d)} \right) + \mathbf{c}_{\mathbf{p}_I}
\]
(15)
where \(\mathbf{R}_G^c \mathbf{R}(t - t_d) \left( \mathbf{G}_{\mathbf{x}_p} - \mathbf{G}_{\mathbf{p}_I(t-t_d)} \right)\) represent the IMU pose at time \(t - t_d\), which will be denoted as time step \(k\) for simplicity in the ensuing derivations.

C. Line Measurement Model

Similar to [15], we adopt a simple projective line measurement model which describes the distance of two line endpoints, \(\mathbf{x}_l = [u_l \nu_l 1]^T\) and \(\mathbf{x}_l = [u_v \nu_v 1]^T\), to the projected line segment in the image:
\[
\mathbf{z}_l = \begin{bmatrix} \frac{s_1}{\sqrt{l_1^2 + l_2^2}} \frac{s_1}{\sqrt{l_1^2 + l_2^2}} \end{bmatrix}^T
\]
(16)
where \((u,v)\) are the coordinates of the point on the image plane, while \(l = [l_1 \ l_2 \ l_3]^\top\) is the 2D image line projected from the 3D line \(C_x\) expressed in the camera frame.

**D. Multi-State Constraint Kalman Filter (MSCKF)**

Each feature measurement, whether it be a point (14) or line measurement (16), can be written generically as:

\[
\mathbf{z} = \mathbf{h}(x, G\mathbf{x}_f) + \mathbf{n}_z
\]

(17)

where \(G\mathbf{x}_f\) represents the feature (either a point \(G\mathbf{x}_p\), or line \(G\mathbf{x}_l\)). If \(G\mathbf{x}_f\) is kept in the state vector, the problem size (and thus the computational burden) will grow unbounded, quickly preventing real-time estimation. One way to avoid this is to use a null space operation [22] to marginalize these features.

To perform this, the MSCKF maintains a sliding window of stochastically cloned historical IMU poses corresponding to past imaging times in the state vector, and accumulates the corresponding feature measurements collected over this window. In our implementation, we clone these poses at the “true” imaging times in order to estimate the time offset as \(\hat{\delta}_t\).

Further, we consider a line measurement (16), can be written generically as:

\[
\mathbf{z} = \hat{\mathbf{h}}(x, G\mathbf{x}_l) + \mathbf{n}_z
\]

(18)

where \(\mathbf{H}_f\) and \(\mathbf{H}_l\) represent the Jacobians w.r.t. the state \(x\) and feature, respectively. Decomposing \(\mathbf{H}_l\) with QR factorization gives the following:

\[
\mathbf{H}_l = [Q_e \ Q_d] [\mathbf{R}_\Delta] \mathbf{Q} = \mathbf{Q} \mathbf{R}_\Delta
\]

(19)

Note that \(Q_e\) is the left null space of \(H_f\), i.e., \(Q_e^\top H_f = 0\). Multiplying (18) to the left by \(Q_e^\top\) yields a new measurement model independent of the feature error:

\[
\tilde{z}_e' \simeq \mathbf{H}_e^\top \mathbf{x} + n'_e
\]

(20)

As this new measurement relates only to quantities contained in the state vector, the standard EKF update can be performed. Utilizing this null space operation for every feature tracked over the sliding window keeps the problem size (and thus the computational cost) bounded.

**III. LINE REPRESENTATION AND JACOBIANS**

**A. Line Representation**

In order to incorporate line measurements into the estimator, we need to find an appropriate representation for these features.

In our previous work [18], we summarized several line error representations (including orthonormal, quaternion, and closest point) shown in Table I and Figure 1a.

Note that \(\hat{q}_l\) is a unit quaternion and \(\tilde{q}_l = [q_l^1 \ q_l^2 \ q_l^3]^\top\). Given the 3D positions of two points \(p_1\) and \(p_2\) (expressed in the same frame) corresponding to the same line \(x\), we can obtain its Plücker coordinate (Model 1 in Table I) as [15, 23]:

\[
\begin{bmatrix}
\mathbf{n}_l \\
\mathbf{v}_l
\end{bmatrix} = \begin{bmatrix}
p_1 \\
(p_2 - p_1)
\end{bmatrix}
\]

(21)

Please refer to Appendix I for detailed derivations.
IV. LINE TRIANGULATION

In order to utilize line features in the MSCKF, an estimate of its 3D line parameters is needed to linearize the measurement model (17). From the above section, it is clear that we need the basic geometric elements (e.g., \(v_e, v_c\) and \(d_l\)) for a line feature.

Image line detectors such as Line Segment Detector (LSD) [24] extract two arbitrary endpoints of the line segment in each image. Consider line feature, \(x_i\), which has been detected and tracked over the sliding window, yielding a collection of line segment endpoint pairs, we propose two algorithms for line feature triangulation based on these stacked endpoint measurements.

A. Algorithm A

Denoting the endpoints for a line in the \(i\)-th image in the sliding window as \(x_{si}\) and \(x_{ei}\), see Figure 1b, we obtain the normal direction of the plane \(\pi_i\) formed by the line \(x_i\) and the \(i\)-th camera center:

\[
C_i n_{ei} = \frac{[x_{si} x_{ei}]}{||x_{si} x_{ei}||}
\]  

Since line \(x_i\) resides on every plane \(\pi_i\), we have the following constraint:

\[
\left[ \begin{array}{c} \vdots \\ C_i n_{ei}^T C_i R_i^T \\ \vdots \\ N \end{array} \right] C_i v_{ei} = 0
\]

(31)

Therefore, \(C_i v_{ei}\) can be found as the unit vector minimizing the error on this constraint, which is given by the eigenvector corresponding to the smallest eigenvalue of \(N^T N\).

The transformation of a line expressed in frame \(C_i\) to a representation in frame \(C_j\) is given by:

\[
\begin{bmatrix} C_i d_l C_i n_{ei} \\ C_i v_{ei} \end{bmatrix} = \begin{bmatrix} C_j R_j & C_j p C_j | C_i R_i \\ 0_3 & C_j R_i \end{bmatrix} \begin{bmatrix} C_i d_l C_i n_{ei} \\ C_i v_{ei} \end{bmatrix}
\]

(32)

\[
\Rightarrow C_i d_l C_i n_{ei} = C_j d_l C_j n_{ei} = [C_j p C_j | C_i R_i] C_i v_{ei}
\]

(33)

\[
\Rightarrow C_i d_l [C_i v_{ei}] = [C_i d_l C_j R_j C_i n_{ei}] = [C_j d_l C_j R_j C_i v_{ei}]
\]

(34)

where \(b_i = [C_i v_{ei} | C_i R_i n_{ei}]\) is a unit vector perpendicular to \(C_i R_i n_{ei}\). Given all the measurements from \(i = 2 \ldots m\), we build a linear system as:

\[
C_i d_l \begin{bmatrix} \vdots \\ b_i^T C_i n_{ei} \\ \vdots \\ N \end{bmatrix} = \begin{bmatrix} \vdots \\ b_i^T C_i n_{ei} \\ \vdots \\ 0 \end{bmatrix}
\]

(35)

By solving the above system, we obtain \(C_i d_l\). After this step, we have recovered all the required line parameters; \(C_i n_{ei}\) eq. (30), \(C_i v_{ei}\) eq. (32) and \(C_i d_l\) eq. (35). The triangulated 3D line feature can be transformed to other frames (i.e. global frame \(G\)) as needed. Note that this is a generic algorithm and we have made no assumptions on the correspondences of the endpoints.

B. Algorithm B

One of the classical methods to triangulate line features is based on the two intersecting planes (e.g., \(\pi_1\) and \(\pi_2\)). The dual Plücker matrix \(L^*\) can be computed as:

\[
L^* = \pi_1 \pi_2^T - \pi_2 \pi_1^T = \begin{bmatrix} C_1 v_{ei}^{(i)} & \ldots & C_m v_{ei}^{(i)} \\ \vdots & \ldots & \vdots \\ C_1 d_l^{(i)} & \ldots & C_m d_l^{(i)} \end{bmatrix}
\]

(36)

where \(\pi_i = [C_i n_{si} | 0]^T\) and \(\pi_i = [C_i n_{ei} | C_i v_{ei} | C_i p_{ci}]^T\). The line geometric elements \(C_i d_l^{(i)}, C_i n_{ei}^{(i)}\) and \(C_i v_{ei}^{(i)}\) are computed based on \(\pi_1\) and \(\pi_i\). In this way, we offer a generalization of this method for \(m\) measurements. In particular, we solve for the line parameters using:

\[
C_i n_{ei} = \sum\limits_{i=2}^{m} C_i n_{ei}^{(i)} \|\sum\limits_{i=2}^{m} C_i n_{ei}^{(i)}\|^{-1}
\]

(37)

\[
C_i v_{ei} = \sum\limits_{i=2}^{m} C_i v_{ei}^{(i)} \|\sum\limits_{i=2}^{m} C_i v_{ei}^{(i)}\|^{-1}
\]

(38)

\[
C_i d_l = \frac{\sum\limits_{i=2}^{m} C_i d_l^{(i)}}{m-1}
\]

(39)

After linear triangulation, we perform nonlinear least squares to refine the line estimates, noting again that we have made no assumption of the endpoint correspondences.
C. Degenerate Motion Analysis for Triangulation
When using a monocular camera, the ability to perform line feature triangulation is heavily dependent on the sensor motion. In particular, we identify degenerate motions that cause the line feature parameters to become unobservable, thereby causing triangulation failures (see Fig. 1c and Tab. II). Letting $C$ denote the center of the camera frame and $L$ be the line feature and formulate a plane $\pi$ as shown in Fig. 1c.

- If the monocular camera moves along the direction $v_e$ of $L$, or toward $L$ with direction $v_e/n_e$, the camera will stay in the same plane, $\pi$. As a result, each of the $C_i v_{ei}$ will be parallel to each other, causing matrix $N$ in (32) to become rank 1, thereby causing ambiguity in the solution for $v_1 v_{e1}$. In addition, without $c_1 v_{e1}$, $c_1 d_1$ becomes unsolvable.
- If the monocular camera undergoes pure rotation (no translation), the camera will also stay in the plane $\pi$, causing degeneracy as in the previous case.
- The effective motion for line triangulation is the motion $n_e$ perpendicular to the plane $\pi$ as shown in Fig. 1c.

Note that for a monocular camera, any combination of the listed 3 degenerate motions (e.g., planar motions in plane $\pi$ shown in Fig. 1c) will also cause triangulation failure.

Interestingly, we see that for stereo cameras, if both cameras remain in the plane during the motion (such as when the platform translation and camera to camera offset remain in the plane), we will still have degenerate motion. This is because triangulation requires that we measure $L$ from different views along $n_e$. In such a case, even stereo vision cannot guarantee proper line triangulation.

<table>
<thead>
<tr>
<th>Motion</th>
<th>Solvable</th>
<th>Unsolvable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Along line direction $v_e$</td>
<td>$n_e$</td>
<td>$v_e$ and $d$</td>
</tr>
<tr>
<td>Toward line $v_e/n_e$</td>
<td>$n_e$</td>
<td>$v_e$ and $d$</td>
</tr>
<tr>
<td>Pure rotation</td>
<td>$n_e$</td>
<td>$v_e$ and $d$</td>
</tr>
<tr>
<td>Perpendicular to plane $n_e$</td>
<td>$n_e,v_e$ and $d$</td>
<td>-</td>
</tr>
<tr>
<td>Random motion</td>
<td>$n_e,v_e$ and $d$</td>
<td>-</td>
</tr>
</tbody>
</table>

V. SIMULATIONS
We performed two Monte-Carlo simulations to verify the proposed line feature triangulation algorithms and different line representations.

A. Line Triangulation Simulation
We first used Monte-Carlo simulations to verify both the proposed sliding-window based line feature triangulation algorithms and our degenerate motion analysis. 8 lines were generated (see Fig. 2) and observed by a monocular camera from 20 poses in space, with the lines being placed about 2m in front of the poses. Similar to the real-world line segment detector (LSD), our simulated monocular camera collected the two endpoint measurements of lines in its view, with each endpoint bearing measurement corrupted by 2-pixel Gaussian random noise, while we assume no correspondences between these endpoints. 3 different camera motions (including straight line motion, planar motion and 3D sinusoidal motion) were simulated to verify the degenerate motions. During triangulation, we disturbed the true camera poses (both the orientation and position) with random noises as:

$$ \tilde{q}_m = \left[ \frac{\sigma_0}{1}, \tilde{q} \right] \otimes \tilde{q}, \quad p_c = p_c + n_p $$

where $n_0$ and $n_p$ represents the white Gaussian noises added to the camera pose estimates, with $\sigma_0 = 0.01 rad$ and $\sigma_p = 0.005 m$, respectively, while $\tilde{q}_m$ and $p_m$ represent the disturbed camera orientation and position estimates which were used by the triangulation algorithms. When evaluating the triangulated line errors, we transferred the estimated line parameters into CP form and computed the errors in Euclidean space.

For each motion profile we generated 30 sets of data and computed the root mean square error (RMSE) [25] for the line accuracy evaluation. As shown in Fig. 2, we see that for all the tested motion patterns, the proposed Algorithm A outperformed Algorithm B. Since lines 1, 5 and 8 are horizontal lines, when the camera performs straight line motion along this direction (shown in Fig. 2 left), their estimates can not be accurately triangulated. For the planar trajectory, the camera moved in the same plane formulated by the camera center and line 5, therefore, line 5’s triangulation still failed. For lines 1 and 8, their accuracy was slightly improved over the 1D motion case because this planar motion is not strictly degenerate for them. Finally, in the 3D motion case, all lines were successfully triangulated with relatively low error due to the fact that all degenerate cases were avoided.

B. Line Representation Simulation
When fusing line information into any estimator, it is vital to be able to determine which of those possible line representations yields the best performance. In order to test the effect of line representation choice, we performed a Monte-Carlo simulations using a visual line-SLAM system. A line map (64 lines in total) in an indoor room was generated while a monocular camera was simulated to follow a sinusoidal trajectory (with 150 simulated poses), as shown in the right of Fig. 3.

To simplify the simulation, we simulated relative pose odometry measurements for the camera (which were also disturbed with pose noises (40)). In order to test the robustness of the line representations, we performed tests using 3 different image is levels (2, 6 and 8 pixels) to corrupt the line endpoint measurements (as shown in the right of Fig. 3).

To simplify the simulation, the camera traversed a single loop of this trajectory as shown in the left of Fig. 3. The line features were triangulated using proposed Algorithm A. To solve the visual line-SLAM problem, we formulated it as a Maximum Likelihood Estimation (MLE) problem which can be computed as an instance of Nonlinear Least Squares [26]. In this simulation, we allowed 5 Gauss-Newton iterations for each representation for a fair comparison. We ran 30 Monte-Carlo simulations and computed the RMSE for the camera poses to evaluate the accuracy.
As shown in Fig. 3, all 3 representations gave similar SLAM performance. However, as we increase the measurement noise levels, the Plücker representation with orthonormal error states tends to perform slightly worse than the others (e.g., the CP and quaternion representations). Note that in all the noise levels tested, the CP and quaternion line representations yielded similar performance. These results indicate that one of these two representations should be used in practice, in particular for low-cost visual sensors which display a large amount of noise.

VI. EXPERIMENTS

For our real-world experiments, we implemented the proposed triangulation and MSCKF line feature update. A sparse point feature pipeline first extracts FAST [27] features which are then tracked frame to frame using KLT [28], after which 8-point RANSAC is used to reject outliers. For the line segments, we leverage the Line Segment Detector (LSD) [24] implementation within OpenCV [29] to detect lines and extract their descriptors using Line Band Descriptors (LBD) [30]. Incoming grayscale monocular images are first undistorted to allow for extraction of straight line segments in the image plane. We found that the line detection within OpenCV is the major bottleneck for processing time within the estimator, and limits the processing frame rate to 15 Hz, while the point-only MSCKF can run up to 50 Hz on an Intel Xeon E3-1505Mv6 with 3.00GHz.

We validated the designed system on the Vicon room.
We further evaluated the proposed method by looking at the relative pose errors (RPE) \cite{32} in Figure 4 and Table IV. This RPE is calculated over all dataset runs and thus shows the accuracy and variance for different distance segment lengths. It is clear that adding lines consistently decreases both the orientation and position error as the segment length grows. It is interesting that as the segment length grows, the difference in position accuracy also grows.

A. Remarks

It should be pointed out that, we empirically found that there are a few cases where the inclusion of free line features in VIO does not improve the estimation performance much, as shown in the V1.02 and V1.03 datasets. This is likely due to three key issues inherent to line features: (i) structure of the explored environment, (ii) motion of the monocular camera relative to the features, (iii) and poor visual line tracking. Through our experiments, we saw that free-line tracking and estimates of unstructured environments will be of low quality and provide little information to the estimator. We also found that line triangulation and estimation can suffer if the VIO platform traveled in the degenerate directions (see Table II), and these degenerate motions may often occur for free lines. To address these issues in the future, we will adopt long-term robust line features into our VIO state, avoid these degenerate triangulation cases and leverage the loop-closure properties of lines.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we designed a tightly-coupled monocular visual-inertial navigation system using point and line features with online spatial and temporal calibration. We also proposed two sliding window based 3D line triangulation algorithms and compared their performances. Based on the proposed algorithms, we identified 3 degenerate motions for monocular camera that cause the line triangulation to fail and validated their existence through simulation. Monte-Carlo simulations with a visual line SLAM setup were also performed to compare commonly used line feature representations, and the results showed that the CP and quaternion representations performed better than the orthonormal representation in scenarios with high measurement noises. Finally real world experiments were performed to verify the implemented system using the CP line representation. In the future we plan to investigate inverse depth line representations, different line outlier rejection methods, and feature constraints (e.g., parallel lines, point-on-lines, etc.).

APPENDIX

LINE MEASUREMENT JACOBIANS

The Jacobians for the CP line measurements are as follows:

\[
\frac{\partial \mathbf{z}}{\partial \mathbf{I}} = \frac{1}{l_n} \begin{bmatrix}
  u_1 - \frac{l_1 c_1}{l_n} & v_1 - \frac{l_1 e_1}{l_n} & 1 \\
  u_2 - \frac{l_2 c_1}{l_n} & v_2 - \frac{l_2 e_1}{l_n} & 1
\end{bmatrix}
\] (41)

\[
\frac{\partial \mathbf{I}}{\partial c} = \begin{bmatrix}
  f_v & 0 & 0 \\
  0 & f_u & 0 \\
  -f_v c_u & -f_u e_v & f_u f_v
\end{bmatrix}
\] (42)

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on May 22, 2021 at 02:11:36 UTC from IEEE Xplore. Restrictions apply.
\[
\frac{\partial \tilde{L}}{\partial \tilde{L}} = \begin{bmatrix}
\frac{\partial \tilde{L}}{\partial \tilde{L}} \\
\frac{\partial \tilde{L}}{\partial \tilde{L}} \\
\frac{\partial \tilde{L}}{\partial \tilde{L}} \\
\end{bmatrix}
\]

where \( e_1 = l^\top x_1, e_2 = l^\top x_e \). \( K \) is the projection matrix for line features, \( f_u, f_v, c_u \) and \( c_v \) are the corresponding camera intrinsic parameters.

\[
\frac{\partial \tilde{L}}{\partial \tilde{L}} = \begin{bmatrix}
\frac{\partial \tilde{L}}{\partial \tilde{L}} \\
\frac{\partial \tilde{L}}{\partial \tilde{L}} \\
\frac{\partial \tilde{L}}{\partial \tilde{L}} \\
\end{bmatrix}
\]

The Jacobians w.r.t. the pose can be written as:

\[
\frac{\partial \tilde{L}}{\partial \tilde{L}} = \begin{bmatrix}
\frac{\partial \tilde{L}}{\partial \tilde{L}} \\
\frac{\partial \tilde{L}}{\partial \tilde{L}} \\
\frac{\partial \tilde{L}}{\partial \tilde{L}} \\
\end{bmatrix}
\]

The Jacobians w.r.t. the CP line features can be written as:

\[
\frac{\partial \tilde{L}}{\partial \tilde{L}} = \begin{bmatrix}
\frac{\partial \tilde{L}}{\partial \tilde{L}} \\
\frac{\partial \tilde{L}}{\partial \tilde{L}} \\
\frac{\partial \tilde{L}}{\partial \tilde{L}} \\
\end{bmatrix}
\]

REFERENCES


