Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2006-003 January 9, 2006

Wide-Area Egomotion Estimation from Known 3D Structure

Olivier Koch and Seth Teller

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Wide-Area Egomotion Estimation from Known 3D Structure

Olivier Koch
MIT Computer Science and Artificial Intelligence Laboratory
Cambridge MA 02139
koch@csail.mit.edu

Seth Teller
MIT Computer Science and Artificial Intelligence Laboratory
Cambridge MA 02139
teller@csail.mit.edu

Abstract

We describe an algorithm that takes as inputs a coarse 3D model of an environment, and a video sequence acquired within the environment, and produces as output an estimate of the camera's 6-DOF egomotion expressed in the coordinates of the 3D model. Our method has several novel aspects: it performs line-based structure-from-motion; it aligns the local line constellation to the known model; and it uses off-line visibility analysis to dramatically accelerate the alignment process.

We present simulation results demonstrating the method's operation in a multi-room environment. We show that the method can estimate metric egomotion accurately and could be used for for many minutes of operation and thousands of video frames.

1. Introduction

Robust, wide-area egomotion estimation is a longstanding goal of computer vision. Existing methods typically handle only short-duration, short-excursion sequences. We wish to develop an egomotion estimation capability suitable for long-duration, long-excursion use, to track the 6-DOF rigid body pose (attached to a user's head, body, or hand-held device) as it is moved within an extended environment. Our design target is to estimate egomotion with an accuracy of 2 centimeters and 0.1 degrees, over several hours of walking-speed motion inside a building containing hundreds of rooms.

1.1. Method Overview and Assumptions

We can frame egomotion estimation as an online 6-DOF localization task alternating between two operating phases, both of which occur after a one-time, off-line precomputation. All three steps are described briefly here:

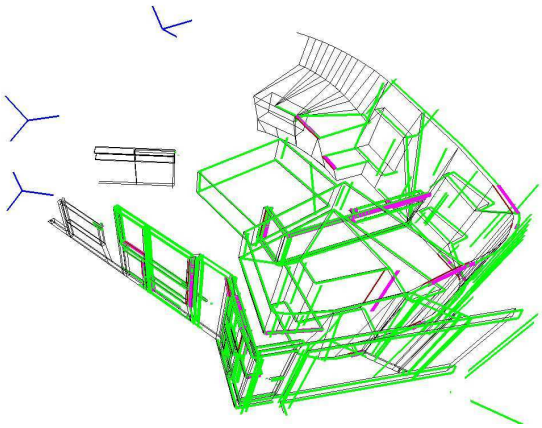

Figure 1. 3D reconstruction from line correspondence between three images. In blue: reconstructed camera positions; in black: original 3D structure; in green: reconstructed structure; in pink: the edges used for correspondence.

- **Initialization** When the camera pose is known poorly or not at all, e.g., at the start of processing, or after "loss of lock," determine a valid current camera pose estimate;
- **Maintenance** When an accurate camera pose estimate is available for all but the most recently acquired image, update the estimate to account for recent camera motion; and
- **Precomputation** A visibility analysis of the environment that makes both Initialization and Maintenance much more effi cient.

Our approach makes assumptions about the information available to the algorithm, and about the character of the camera's motion through the environment. The remainder of this section describes these assumptions, and how our algorithms and system depend upon them. We defer, to Section 5, discussion of what happens when our assumptions are not met in practice.

First, we assume that an approximately accurate **geometric model** of the environment's "coarse structure" – which we define as its walls, floors, ceilings, doors and windows – is available. Such a model could be provided by the building's architects, or produced independently by a post-construction modeling method. This enables us to frame egomotion estimation (*i.e.*, 6-DOF localization) as a 3D structure-to-structure matching problem with a long-term drift-free solution, rather than as a generally divergent integration of many short-term visual odometry estimates as in typical structure-from-motion approaches. We cannot, and do not, assume that the environment model is completely accurate, nor that it is comprehensive in its inclusion of detail elements such as door jambs, window moldings, furniture, *etc*. We have formulated our method to be robust in the face of such "unmodeled clutter" (Section 3).

Second, we assume that the environment contains many large, opaque surfaces, and that consequently the **visibility is limited**; *i.e.*, that the number of environment surfaces (and edges) visible to any observer will be a small fraction of the total number of geometric primitives within the environment [7]. When the camera's approximate location is known, this visibility assumption greatly reduces the number of candidate environment features that may be matched against features in the short-sequence SFM solution.

Third, we assume that the **camera motion is smooth**, and that the camera never moves through an opaque (*i.e.* impenetrable) surface. When these assumptions are met, the input imagery will have "high overlap;" *i.e.*, frames acquired at nearly the same time will tend to observe common environment geometry. Thus our system will usually operate in its relatively computationally inexpensive maintenance mode, and will only rarely have to initialize (or reinitialize) its pose estimate. Moreover, it is intuitively clear that the smoother the motion, the higher the inter-frame overlap, and thus the smaller the search space of candidate pose updates will be.

Fourth, we assume that the camera is intrinsically **calibrated**, and that its motion path is such that the 3D edges in the environment remain sufficiently numerous, visible, and nearby to support accurate short-sequence metric SFM. Section 4 gives a quantitative sufficiency criterion for each of these conditions.

We emphasize that we do *not* make assumptions about built structure that form the foundation of other vision-based localization systems. For example, we do not assume the presence of vertical lines, horizontal lines [4], vanishing points [2], or right angles [1]. We do not assume any knowledge of surface color or reflectance attributes in the environment, or indeed any "appearance" information beyond the environment geometry. Finally, we do not assume a static environment, *i.e.* one free of time-dependent attributes, such as changing lighting or transient or moving objects.

1.2. Paper Overview

After this introduction, and a description of related work (§2), the bulk of the paper consists of descriptions of several technical components of our method:

- An image-space edge tracker based on nearest-neighbor search (§3.3);
- A short-sequence projective structure from motion (SFM) algorithm using trifocal tensor minimization [9] constrained by three-frame line correspondences (§3.5.1);
- A stratified reconstruction algorithm [6] to transform the projective SFM solution into a metric reconstruction up to an unknown scale factor (§3.5.1);
- A visual odometry estimator and online baseline selection method for more stable trifocal tensor geometry;
- A 3D-to-3D matching method that registers the metric SFM solution (and thus the current egomotion estimate) to the environment model (§3.6);
- A precomputation phase that determines the 3D edges visible from each of a set of sample camera positions.

The paper then continues with empirical studies of the algorithm's performance for a variety of synthetic datasets 4, a discussion of the algorithm's failure modes (§5), and a conclusion (§6).

2. Related Work

Existing methods for vision-based localization basically split into two categories: those using geometric features such as lines, points and SIFT features, and those using a machine-learning approach. In the latter, observations are statistically matched against a database of images collected during a training phase. In the former, geometric constraints are extracted from observations in the hope of finding a unique solution for the camera pose.

Optical flow and feature tracking [11] can both be used for egomotion estimation, but unavoidably exhibit drift, *i.e.* egomotion estimates that diverge from ground truth over time.

Yagi *et al*. estimated the azimuth of edges extracted from a conic image sensor to refine robot odometry estimates [15]. Antone and Teller demonstrated extrinsic camera refinement from many intrinsically calibrated real-world images, assuming sufficiently many vanishing points [1]. Bosse *et al*. used vanishing points coupled with an efficient 3D line tracker to estimate camera motion from an omnidirectional video sequence [2].

Mouaddib and Marhic presented a geometric-based approach in which vertical lines are detected using a goniometric sensor and matched against a map designed during an off-line process [14]. Se, Lowe and Little introduced 3D SIFT landmarks and RANSAC matching for robot localization [16]. Cauchois demonstrated real-world panoramic image matching against a set of synthetic images generated from a 3D model of the environment [5]. Jogan and Leonardis defined the problem of localization as the recognition of a panoramic view among a dataset of known panoramic views. Using an image sampling with 60cm spacing, they achieve localization accuracy of 60cm in an occlusion-free environment [12]. Matsumoto *et al*. used a similar approach in [13]. Winters *et al*. demonstrated localization is performed using topological maps [17] by matching omni-directional images against a PCA-compressed database of images collected during a training phase. These methods require a substantial learning phase, and do not scale well with the size of the environment.

3. Our Method

3.1. Preliminaries

We denote 3D points and lines as upper case italics (*e.g*. X), 2D points and edges as lower case italics (*e.g*. x), and matrices in bold (*e.g*.$\mathbf{K}$). We denote indices with subscripts (*e.g*. $x = (x_1, x_2, x_3)^\top$). We represent each projective camera as a 3×4 matrix $\mathbf{P}$ equal to [9]:

$$P = K \begin{bmatrix} \mathbf{R} & -\mathbf{R}t \end{bmatrix} \tag{1}$$

where $\mathbf{R}$ is a 3×3 rotation matrix, t a 3×1 translation vector, and $\mathbf{K}$ the 3×3 camera calibration matrix

$$K = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where α_x and α_y represent the focal length in pixels and x_0 and y_0 describe the principal point in pixels. We assume that the skew factor is zero.

With this model, any 3D point X can be projected onto a camera $\mathbf{P}$ using the formula $x = \mathbf{P}X$. If an inverse projection matrix $\mathbf{P}^{-1}$ is known, the image point can be back-projected into a 3D ray $X = \mathbf{P}^{-1}x$. Note that an inverse projection matrix is a 4x3 matrix satisfying the relation $\mathbf{P} \cdot \mathbf{P}^{-1} = \mathbf{I}_3$ where $\mathbf{I}_3$ is the 3×3 identity matrix.

Finally, we use a standard radial polynomial to correct radial distortion [10].

3.1.1 Environment Model

The 3D environment is represented as a set of 3D vertices, edges and faces. For simplicity of processing, we subdivide all input faces into triangles or quadrilaterals. No other appearance information (*e.g*., color, material, texture, transparency) is stored in the model or used by our system.

3.1.2 Geometric Distance

We use the Euclidean distance metric $|\cdot|$ for points in space. For line segments, we define two distance metrics. The first metric, d_o, captures the fractional degree of overlap between two line segments $l_1 = (x_{11}, x_{12})$ and $l_2 = (x_{21}, x_{22})$. If d_{PL} is the shortest distance between a point and a line segment, then:

$$d_o(l_1, l_2) = min(d_{PL}(x_{11}, l_2) + d_{PL}(x_{12} \cdot l_2), \\ d_{PL}(x_{21}, l_1) + d_{PL}(x_{22} \cdot l_1)).$$

The second distance, d_a, captures the angular alignment between the segments. If m_1 is the midpoint of l_1 and m_2 the midpoint of l_2, then:

$$d_a(l_1, l_2) = |m_2 - m_1| \cdot (1 + |sin(angle(l_1, l_2))|). \tag{3}$$

3.1.3 Frames and Images

We define a *frame* as a collection of ordinary images taken from the same 3D location, but with different camera orientations. Below, we refer to the "frames" collected by an omnidirectional video camera that integrates six standard cameras in a rigid mechanical structure.

3.2. Main Idea

The main idea of our system is to use short-sequence SFM to build a "submap" of the structure near the moving camera, then align this submap with the known 3D model. The pose estimation system has two major modes: *initialization* and *maintenance*. Initialization mode is entered whenever the system has a poor estimate of the camera location, such as at the start of processing or after loss of lock. Maintenance mode is entered when an approximate pose estimate is available, the inaccuracy of which is due to short-term camera motion since the most recent successful localization. Figure 2 depicts the system's processing architecture.

The initialization phase takes as input a 3D model $\mathcal{M}$ and a short-sequence SFM solution $(\mathcal{P}_R, \mathcal{S}_R)$ where $\mathcal{P}_R$ represents the camera pose and $\mathcal{S}_R$ the recovered 3D structure. The index $_R$ is used to refer to a *relative* coordinate

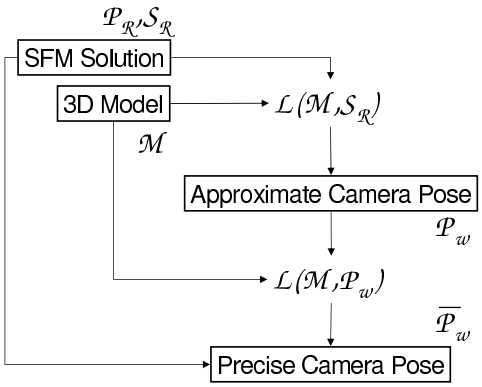

Figure 2. System Overview

frame. The output is an approximate camera pose $\mathcal{P}_W$ - where the index $_W$ is used to refer to a *global* (or world) coordinate frame. The maintainance phase takes as input the approximate camera pose $\mathcal{P}_W$, the model $\mathcal{M}$ and the SFM solution $(\mathcal{P}_R, \mathcal{S}_R)$. The output is an accurate camera pose estimate $\overline{\mathcal{P}}_W$ in the *global* coordinate frame.

The system makes use of an alignment operation $\mathcal{L}$ that takes a model model $\mathcal{M}$ and a submap $\mathcal{S}$ and returns one or more locations within $\mathcal{M}$ where $\mathcal{S}$ exists as a sub-structure. The system also makes use of a visibility computation and store that takes a model $\mathcal{M}$ and an approximate camera pose $\mathcal{P}$ and returns the visible sub-structure $\mathcal{S}$ visible from $\mathcal{P}$ within $\mathcal{M}$.

3.3. Edge Tracker

Each frame of the video sequence is passed through a simple line detector based on a Hough transform. Edges shorter than a threshold t (in pixels) are discarded. The remaining line segments are matched against each other between consecutive frames using the distance d_a as defined in section 3.1. A dynamic queue of N correspondences is maintained as new frames are processed. For the first frame, the N longest segments are inserted into the queue. Then, for each new frame, a matching edge is sought for each edge on the queue. If a match is found, the queue element is updated. If no match is found, the element remains in the queue. If no match is found within p_{time} frames, the element is removed from the queue. Finally, if the queue is not full, each newly appearing (*i.e.*, unmatched) segment causes creation of a new queue element. Figure 3 summarizes our edge tracker algorithm.

1: First frame: Select the N longest line segments and put them into the queue Q.
2: **for** each new frame F_i **do**
3: **for** each element E_j in queue Q **do**
4: Search for a match to E_j in F_i:
5: If no match found: increment counter c in E_j
6: If a match found: update Q and reset c to zero
7: Cleanup: remove all elements of Q for which $c > p_{time}$.
8: New elements: if $size(Q) < N$, pick the longest unmatched segment in F_i and insert it into the queue.

Figure 3. Edge Tracker Algorithm

3.4. Visibility Precomputation

We wish to handle environments of arbitrary spatial extent and geometric complexity. In many architectural environments, visibility is limited; that is, only a small fraction of the environment geometry is visible from any point within the model. We perform a visibility precomputation that discretizes the space of viewpoints, and associates with each sampled viewpoint an approximation of the set of model edges visible from that point. We reason that, if the space of viewpoints is sampled sufficiently densely, a camera moving near any sample viewpoint $\mathcal{S}$ should observe very nearly the same model edges as those visible from $\mathcal{S}$. (The camera may observe fewer edges, due to occlusion by unmodeled clutter, image aliasing, or feature detection failure. The camera may observe either fewer or more edges due to errors in the geometric model itself, *i.e.*, inconsistencies between the actual environment and the geometric model of the environment.)

The precise set of polygons (and thus edges) visible from a varying viewpoint is a spatially complex function, and combinatorially expensive to compute [8]. We sidestep these difficulties both by replacing the exact computation with a discrete (sampled) representation, and by sampling the viewpoint space more finely where visibility changes more quickly.

We use an openGL-based algorithm [18] for visibility computation. We define a coarse initial grid of viewpoints (each node on the grid corresponds to a camera position at a standard height of 39 inches). From each sampled viewpoint, we render the known model, determine the set of visible edges, and store them into a lookup table. We then dynamically refine the grid according to visibility. The visibility difference (measured as the size of the set difference between the set of lines visible at each node) is computed. Whenever the variation between two nodes is larger than a threshold, a new node is created in the middle. The process goes on until all adjacent nodes have a difference smaller than the threshold. In addition, the adjacency between two nodes is discarded if a wall lies between them.

The algorithm works as follows. At each camera position, six synthetic cameras corresponding to the six faces of a cube are situated. For each of these synthetic cameras, the full 3D model is rendered, with hidden surfaces elimi-

nated, and the resulting pixel buffer is stored in an array P. In a second step, the edges of the 3D model are rendered in feedback mode, leading to a feedback buffer B. For each item in the buffer B, the z-buffer value is compared to the z-value stored in P. If these value match within some small threshold, the edge is classified as visible.

Since only a portion of each edge may be visible, edges are actually subdivided into one inch-long sub-edges. Figure 4 shows an example of our visibility computation in a typical lab area.

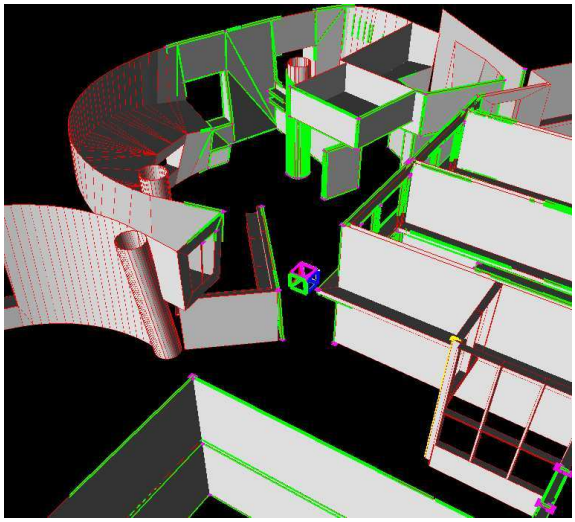

Figure 4. Visibility precomputation in a typical lab area. Visible edges are displayed in green. Note that many edges are partly occluded.

3.5. Short-Sequence SFM

3.5.1 3D structure estimate

Our SFM algorithm takes as input a set of line correspondences across three views and produces a metric reconstruction of the 3D structure and camera poses. Figure 5 shows the corresponding geometric configuration. In the first step, a projective reconstruction is obtained using the trifocal tensor minimization algorithm described in section 15.4 of [9]. For each line correspondence $l \leftrightarrow l' \leftrightarrow l''$ between the three images, we generate two correspondences $x_1 \leftrightarrow l' \leftrightarrow l''$ and $x_2 \leftrightarrow l' \leftrightarrow l''$ where x_1 and x_2 are the two endpoints of l. Since the trifocal tensor is of dimension 27 ($3 \times 3 \times 3$), 14 line correspondences are required.

The reconstruction obtained after the first step is correct up to a projective transformation of the 3D space. Without additional knowledge about the scene or the cameras, it is not possible to obtain a metric reconstruction of the scene. Figure 6 illustrates this ambiguity. In the case of calibrated cameras however, this ambiguity can be fully resolved up to a one dimension scaling ambiguity.

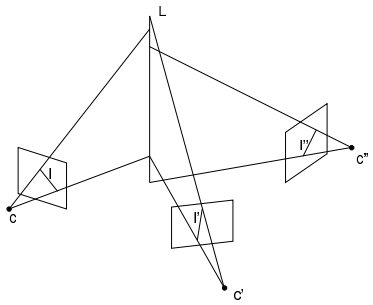

Figure 5. The projection of a 3D line segment L on three cameras C, C' and C''. If the camera positions are known, the 2D line segments l, l' and l'' can be back projected into a line in the 3D space.

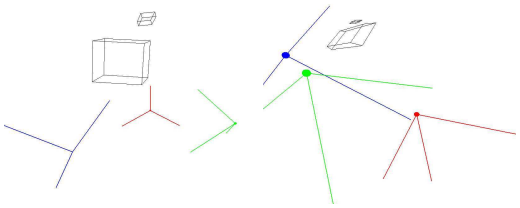

Figure 6. The projective reconstruction ambiguity: figures (left) and (right) show two projective reconstructions leading to the same 2D points on the three images. The three cameras are represented in red, green and blue. The ambiguity can be resolved up to a single scale factor if the cameras are calibrated.

Assuming that $\mathbf{K_1}$, $\mathbf{K_2}$ and $\mathbf{K_3}$ are the known three calibration matrices for the three camera matrices $\mathbf{P_1}$, $\mathbf{P_2}$ and $\mathbf{P_3}$, we compute the two essential matrices $\mathbf{E_{12}}$ and $\mathbf{E_{13}}$ corresponding to the normalize matrices ($\mathbf{K_1}^{-1}\mathbf{P_1}$,$\mathbf{K_2}^{-1}\mathbf{P_2}$) and ($\mathbf{K_1}^{-1}\mathbf{P_1}$,$\mathbf{K_3}^{-1}\mathbf{P_3}$). From each essential matrix, the camera matrices can be extracted up to a four-fold ambiguity. As described in [9], the four ambiguities correspond to reverting the camera translation vectors or flipping the cameras about the line joining the two centers. Therefore, $\mathbf{E_{12}}$ gives four solutions for ($\mathbf{P_1}$,$\mathbf{P_2}$), and $\mathbf{E_{13}}$ gives four solutions for ($\mathbf{P_1}$,$\mathbf{P_3}$), which leads to 16 possible configurations for the set of three cameras ($\mathbf{P_1}$,$\mathbf{P_2}$,$\mathbf{P_3}$). However, only one of them satisfies the following constraints:

1. the reconstructed 3D structure lies in front of the three cameras;

2. the reprojection error is minimized.

The best configuration is the one satisfying these two constraints, and it is the one we preserve.

3.5.2 Camera and Trifocal Tensor Refinement

Since the observed data is subject to noise, the line correspondences described in section 3.5.1 is relatively incorrect in practice. It is therefore necessary to run a minimization algorithm to find the best estimate for the trifocal tensor once an initial estimate of it has been obtained through a linear algorithm. Our algorithm minimizes the algebraic error using a Levenberg-Marquardt method as suggested in section 15.4 of [9]. Once an estimate has been obtained for the 3D structure and the camera poses, a Newton refinement algorithm as described in [3] is used to minimize the reprojection error. We minimize the error function

$$\xi = \sum_{i=1}^{n} \sum_{j=1}^{3} d_a(l_{ij} - proj(P_j, L_i)) \qquad (4)$$

where l_{ij} is the observed segment L_i on image j, $proj(P_j, L_i)$ is the projection of L_i on camera P_j and d_a is the distance between two line segments as defined in section 3.1. In practice, the reprojection error minimization turns out to bring significantly more improvement to the reconstruction than the trifocal tensor optimization.

3.6. Line Cloud Matching

Once the 3D structure has been reconstructed, it is necessary to align this structure with the known 3D model to recover the camera pose in the world coordinate frame. We present here a closed-form solution to the problem of aligning a set $\mathcal{S}$ of 3D lines onto another set $\mathcal{S}'$ of 3D lines. This problem has seven degrees of freedom: three for the translation $t_{\mathcal{S}\to\mathcal{S}'}$, three for the rotation $R_{\mathcal{S}\to\mathcal{S}'}$ and one for the scale factor $\lambda_{\mathcal{S}\to\mathcal{S}'}$.

Given a pair of *skew*[1] lines in each set, we construct a transformation that brings the first pair onto the second pair as follows. We define $C_{ll'}$ the shortest segment between two lines in 3D. Note that when the two lines are *skew*, this segment is unique and always defined. Given a pair of lines (l_1, l_2) in S and (l'_1, l'_2) in S', we define the scale factor $\lambda_{S\to S'}$ as the ratio of the lengths of $C_{l_1 l_2}$ and $C_{l'_1 l'_2}$. We define the translation $t_{S\to S'}$ as the vector $C_{l_1 l_2} C_{l'_1 l'_2}$. And we define the rotation $R_{S\to S'}$ as the rotation centered in the origin which brings the vector l_1 onto vector l'_1.

The transformation $\mathcal{T} = (\lambda_{\mathcal{S}\to\mathcal{S}'}, R_{\mathcal{S}\to\mathcal{S}'}, t_{\mathcal{S}\to\mathcal{S}'})$ is a good candidate for the searched transformation if (l_1, l_2) and (l'_1, l'_2) have the same skewness, where the skewness of two lines is defined as the angle made by these two lines on the plane perpendicular to the shortest segment between them.

This algorithm is wrapped into a RANSAC framework demonstrated in figure7.

[1] two lines are considered as skew if they are not parallel and do not intersect in space.

1: **for** run = 1,...,N **do**
2: pick two lines (l_1, l_2) in $\mathcal{S}$
3: pick two lines (l'_1, l'_2) in $\mathcal{S}'$
4: **skewness test**: If the two set of lines do not have the same skewness, jump to the next run.
5: Otherwise, compute the transformation $\mathcal{T}$ which transforms (l_1, l_2) into (l'_1, l'_2).
6: Apply $\mathcal{T}$ to the lines in $\mathcal{S}$ and match the result with $\mathcal{S}'$ in term of distance d_a.
7: **if** test = ok **then**
8: accept the transformation and return SUCCESS.
9: **else**
10: continue
11: return FAILURE.

Figure 7. Line Cloud Matching Algorithm

4. Results

We implemented our method in approximately 45,000 lines of Matlab code running on a 4-CPU 3.20Ghz 1Gb RAM laptop.

4.1. Line Cloud Matching

The RANSAC framework is parameterized by the maximum number of runs before FAILURE is returned. In our case, the input line cloud $\mathcal{S}$ contains as many lines as correspondences between the images. Typically, this number is about a few dozens. The output line cloud $\mathcal{S}'$ is the cloud of visible edges as returned by the function $\mathcal{L}$ defined in section 3.2. This cloud has a size of a few hundred lines. In practice, the match is found after a few thousands of runs. Most of the runs do not pass the skewness test, thus making the process relatively fast (a few seconds). A result of the line cloud matching is shown on figure 1.

4.2. SFM on Simulated Data

We use a 3D model of our lab space to generate sets of synthetic images. For each set of three images, we reconstruct the 3D structure and camera poses and compare them to the original data. We provide the following performance metrics to assess the efficiency of our method:

- $error_C$: the sum of the translation error for the three cameras (in inches).
- $error_\alpha$: the sum of the angle error for the three cameras (in degrees).
- $error_S$: the average error on the structure, defined as the d_a distance from the original structure to the reconstructed structure (in inches).
- $error_R$: the average reprojection error in pixels.

In the ideal case where noise is absent, all these metrics are equal to zero. In this section, we show the effect of various parameters on these metrics.

4.2.1 Effect of Noise

In our model, a Gaussian noise $\sigma = (\sigma_{trans}, \sigma_{rot})$ is applied to the line segments in term of translation and rotation. Table 4.2.1 summarizes the results. Figure 1 shows a successful reconstruction.

σ	$error_C$	$error_\alpha$	$error_S$	$error_R$
$0,0$	0.6	0.09	0.000	0.0
$1,1$	2.0	0.18	0.006	1.6
$1,3$	20.0	0.70	0.240	3.7
$2,4$	80.0	1.80	0.340	4.5

4.2.2 Increasing the Number of Correspondences

As stated in section 3.5.1, at least 14 line correspondences are required for a successful run of SFM. However, increasing the number of correspondences greatly improves the quality of the results, as demonstrated in table 4.2.2. In this experiment, we have chosen a noise $\sigma_{trans} = 1, \sigma_{rot} = 1$. Beyond a few hundred correspondences, the progress stales.

$\mathcal{N}$	$error_C$	$error_\alpha$	$error_S$	$error_R$
14	40.0	12.0	0.3000	23.0
32	2.0	0.20	0.006	1.60
50	2.0	0.18	0.002	0.52
120	1.7	0.16	0.002	0.31

4.2.3 Spacing between the Cameras

Our SFM algorithm is insensitive to the spacing between the cameras. We have run several sets of experiments for decreasing distances between the cameras - from 200 inches down to 10 inches. The quality of the reconstruction is the same, even when the cameras are close to each other.

4.2.4 Performance

The SFM reconstruction runs in 12 seconds on a 4-CPU 3.20Ghz 1Gb RAM laptop. However, we have spent little time on optimization and many leads can be followed to make the algorithms run faster.

5. Discussion

This section addresses the conditions under which our method operates successfully.

5.1. Genericity Assumptions

A variety of degenerate or anomalous conditions may cause our method to exhibit degraded performance, or to fail outright. These include:

- Degenerate camera path such as no motion, or only straight line motion.
- Degenerate environment structure such as bundles of parallel 3D lines. In this case, it is not possible to estimate the location along the line direction.
- Excessive clutter preventing a suffi cient observability of a priori structure.
- Insuffi cient light levels jeopardizing the feature measurements.
- Very rapid translation or rotation generating image blurring and loss of useful a priori location estimate.

Our method succeeds in recovering the camera pose in a synthetic world. The case of partial occlusion has been tested and the results shown in section 4 are the same when only parts of the 3D lines are visible. Our method is currently unable to handle real data although the reconstruction results are very promising. We believe that this failure may have two root causes : an excessive noise in the observations and/or an inaccurate calibration of the camera.

5.2. Future Work

We are currently working on making the system more robust to noise so that data acquired in real conditions can be handled. The current implementation can also be optimized to reach a close to real-time performance. Some signifi cant progress can be made on various parts of the system. In particular, the initialization step, which is currently manual, could be replaced by an automatic method. The edge tracker could also be improved using a more sophisticated approach, such as the "stochastic nearest-neighbor gating" described in [2].

6. Conclusion

We have reframed 6-DOF localization as a visibility precomputation, followed by an initialization of a camera pose estimate, and online maintenance of that estimate. Our structure from motion algorithm trackes infi nite 3D lines – no line segments, no point features – to aid data association, robustness, and estimation accuracy.

We presented detailed empirical results for synthetic data showing a successful reconstruction of the 3D structure and the camera pose, even in the case of high occlusion level. We analyzed the effect of noise and correspondences on the reconstruction results. Finally, we discussed our system's

failure mode and how it might be made more robust and generically applicabled in the future.

References

[1] M. Antone and S. Teller. Scalable extrinsic calibration of omni-directional image networks. *Int. J. Comput. Vision*, 49(2-3):143–174, 2002. 2, 3

[2] M. Bosse, R. J. Rikoski, J. J. Leonard, and S. J. Teller. Vanishing points and three-dimensional lines from omni-directional video. *The Visual Computer*, 19(6):417–430, 2003. 2, 3, 7

[3] T. F. Coleman and Y. Li. An interior trust region approach for nonlinear minimization subject to bounds. *j-SIAM-J-OPT*, 6(2):418–445, May 1996. 6

[4] S. Coorg and S. Teller. Matching and pose refi nement with camera pose estimates. Technical Report MIT/LCS/TM-561, Massachusetts Institute of Technology, 1996. 2

[5] C. C. Eric. 3d localization with conical vision. 3

[6] O. Faugeras, Q.-T. Luong, and T. Papadopoulou. *The Geometry of Multiple Images: The Laws That Govern The Formation of Images of A Scene and Some of Their Applications*. MIT Press, Cambridge, MA, USA, 2001. 2

[7] T. A. Funkhouser and C. H. Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proc. of SIGGRAPH-93: Computer Graphics*, pages 247–254, Anaheim, CA, 1993. 2

[8] Z. Gigus and J. Malik. Computing the aspect graph for line drawings of polyhedral objects. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(2):113–122, 1990. 4

[9] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision, First Edition*. Cambridge University Press, ISBN: 0521623049, 2000. 2, 3, 5, 6

[10] J. Heikkilä and O. Silvén. A four-step camera calibration procedure with implicit image correction. In *CVPR*, pages 1106–, 1997. 3

[11] B. K. P. Horn and B. G. Schunck. Determining optical fbw. *Artificial Intelligence*, 16(1-3):185–203, 1981. 2

[12] M. Jogan and A. Leonardis. Robust localization using panoramic view-based recognition, 2000. 3

[13] Y. Matsumoto, K. Ikeda, M. Inaba, and H. Inoue. Visual navigation using omnidirectional view sequence, 1999. 3

[14] M. Mouaddib and B. Marhic. Geometrical matching for mobile robot localization, October 2000. 3

[15] Y. Y. Y. Nishizawa and M. Yachida. Map-based navigation for a mobile robot with omnidirectional image sensor copis. In *IEEE Trans. Robotics and Automation*, pages 634–648, October 1995. 2

[16] S. Se, D. Lowe, and J. Little. Local and global localization for mobile robots using visual landmarks. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 414–420, Maui, Hawaii, October 2001. 3

[17] N. Winters, J. Gaspar, G. Lacey, and J. Santos-Victor. Omni-directional vision for robot navigation. In *Proc. IEEE Workshop on Omnidirectional Vision - Omnivis00*, 2000. 3

[18] M. Woo, Davis, and M. B. Sheridan. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. 4