Real-Time Monocular SLAM with Straight Lines

Paul Smith*, Ian Reid* and Andrew Davison†
*Department of Engineering Science, University of Oxford, UK
†Department of Computing, Imperial College London, UK
[pas,ian]@robots.ox.ac.uk, ajd@doc.ic.ac.uk

Abstract

The use of line features in real-time visual tracking applications is commonplace when a prior map is available, but building the map while tracking in real-time is much more difficult. We describe how straight lines can be added to a monocular Extended Kalman Filter Simultaneous Mapping and Localisation (EKF SLAM) system in a manner that is both fast and which integrates easily with point features. To achieve real-time operation, we present a fast straight-line detector that hypothesises and tests straight lines connecting detected seed points. We demonstrate that the resulting system provides good camera localisation and mapping in real-time on a standard workstation, using either line features alone, or lines and points combined.

1 Introduction

The use of a camera as a real-time position sensor has become practical in recent years, aided by the emergence of algorithms in both the vision and robotics communities and the inevitable increasing performance of personal computers. Real-time camera tracking systems have a wide range of potential applications in robotics and augmented reality, and a system that simply requires a standard camera and personal computer is both cost-effective and readily-available.

Real-time camera-tracking is a challenging task, and many methods rely on significant prior knowledge of the environment, for example instrumenting the scene with surveyed markers [10] or using a three-dimensional model of the scene [6]. A system that can estimate the scene structure while tracking is much more flexible, and significant advances have been made in this area. Estimating both the camera's egomotion and the structure in an online system is known as Simultaneous Localisation And Mapping (SLAM), and features strongly the robotics literature. SLAM is closely related to computer vision work in structure from motion, the off-line solution of which is now routine [8], and while such structure-from-motion systems can be made to work in real time (e.g. [15]), they do not guarantee repeated localisation. As a result, much recent work in real-time camera localisation has concentrated on bringing SLAM into the computer vision fold.

Practical real-time monocular SLAM was first demonstrated by Davison [5], who uses the Extended Kalman Filter, a mainstay of SLAM literature. He resolves the problem of real-time operation by careful maintenance of the map to ensure that it is sparse but sufficient, and by using the map uncertainty to guide feature matching. More recently,

Pupilli and Calway [16] have demonstrated real-time camera tracking using a particle filter, which provides a good robustness, but theirs is predominantly a tracking system; its mapping ability is currently rudimentary, which restricts its range of applications. Eade and Drummond [7] have developed a system based on the FastSLAM algorithm, which combines particle filtering for localisation with Kalman filtering for mapping. FastSLAM has the advantage that it scales better with the number of features, but the absence of an explicit full covariance matrix can make loop-closing more difficult.

All of these SLAM algorithms operate using point features, which are relatively easy to localize and characterize. However, their appearance changes substantially with the camera's location, and while invariant feature descriptors can be used (e.g. SIFT [11]), they can be expensive to compute and with more invariance comes less discrimination. One solution that is possible within a SLAM framework is to augment the map of the world with estimates of the planes upon which each feature lies. These can then be used to predict the deformation in the feature, as in Molton *et al*'s extension of Davison's system [12], and this gives much-improved performance.

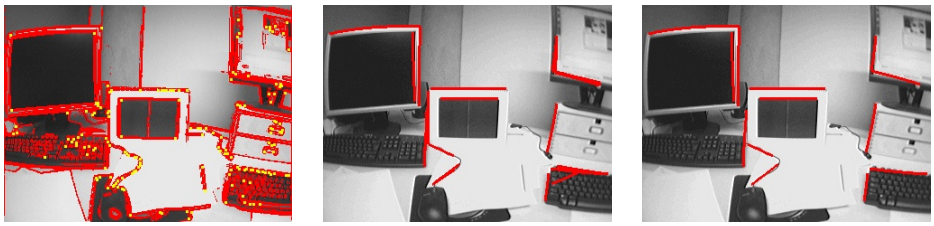
Straight lines are common in man-made environments and are arguably better features to track than points. Described just by a step change in intensity (which does mean that they lack discrimination), they are trivially stable under a wide range of viewing angles, and a number of measurements can be made along their length to localise them accurately. As a result, many camera-tracking systems have used line features (e.g. [9, 10, 6]). Lines also provide a great deal of information about the structure of the scene, and off-line structure from motion using lines is well-known (e.g. [18, 2]). Lines have also been used for some time in SLAM systems, but only using either multiple (or specialist) cameras [1, 4, 3] or alternative sensors [14] to detect and localize them. When using vision, no real-time system has yet been demonstrated.

In perhaps the earliest work in visual SLAM using lines, Ayache and Faugeras [1] used a stereo pair of calibrated cameras to directly extract the three-dimensional location of line segments and filtered these within an EKF SLAM framework. More recently, Dailey and Parnichkun [4] use a trinocular rig and FastSLAM, while Bosse *et al.* [3] use a single omnidirectional camera to detect and track parallel lines, both with reasonable results. Neither system operates in real-time, and modifying their line detection and matching (between cameras or between frames) to operate at frame-rate would be a challenge. A practical real-time system must detect line features very quickly and should ideally use only a single standard camera. In this paper we present methods to achieve both of these.

Point and line features are complementary in a camera localisation system: point features provide good discrimination, but are view-dependent, while line features are robust to viewing changes, but are more fragile. This idea has been studied recently by Rosten and Drummond [17], who bootstrap line-tracking (using a prior accurately-known three-dimensional model) with detected point features. We show in this paper that monocular tracking with the fusion of point features and line features, and model-building with both types of feature, can be performed within a standard SLAM framework.

2 Fast Straight Line Detection

A line-based real-time SLAM system can not afford to spend too long finding lines in the image, so we shall first consider the problem of fast line detection; in Section 3 we



(a) (b) (c)

Figure 1: Fast line detection. (a) FAST corners (in yellow) and thresholded Sobel edgels (in red); (b) All possible straight lines between corners that are longer than 30 pixels and lie over Sobel edgels; (c) Straight lines after removing overlapping lines. From 166 corners, 13 lines are detected.

describe how to integrate lines into a SLAM framework.

An important realisation is that a line-detection algorithm for monocular SLAM does not need to be complete or repeatable: its job is simply to find enough good lines in a frame that sufficient can be initialised for tracking. The lines of interest are straight in the image (after allowing for radial distortion—see later), since straight lines are likely to represent genuine, straight, three-dimensional features, and good structure in the world to track. Algorithms such as fitting straight lines to edges found with the Canny edge detector or using the Hough transform are too slow for real-time operation. Instead, the algorithm presented here makes the assumption that such straight, structural, lines are terminated with sharp corners. The algorithm starts by finding corners in the image, and then hypothesizes and tests all possible lines between them.

To find corner features in the image, Rosten's FAST feature detector is used [17]. This uses a very efficient pixel test to identify corner features: on the 320×240 images used in our system, corner extraction takes 0.6ms, including performing non-maximal suppression (all timings in this paper are for a 3.4GHz dual-core Pentium 4 computer). The features found are marked in yellow on Figure 1(a), a representative video frame, and it can be seen that the assumption that most structural lines end in corners is a valid one (for this typical office scene). Figure 1(a) also shows, in red, edgels in the image detected using the Sobel operator, thresholded with a generous threshold. These will be used to test possible lines (although, for speed, Sobel values are only computed for the pixels tested—about 20% of the image).

All possible straight lines between pairs of corners are tested, which for the example in Figure 1, with 166 corners, means 13,695 line hypotheses. The speed of the algorithm comes from the ability to reject most of these hypotheses very quickly. Firstly, any line that is shorter than 30 pixels is rejected (since experience shows that shorter lines are less likely to be genuine, or be tracked as reliably by the SLAM system). Secondly, the mid-point of the hypothesized line is tested to see whether it lies on an image edgel, and if so, the quarter- and three-quarter-points are also tested. Only if all of these tests pass is the final test started, a Bresenham walk in the image from one end of the line to the other, testing for an edgel at each pixel. For wide-angle cameras, pixel locations can be computed in undistorted co-ordinates and mapped back into the image for testing, at little additional cost. If fewer than 95% of the pixels tested during the walk are edgels, that line is rejected. Of the 13,695 line hypotheses in the example, only 434 pass the first two tests and need to be walked, and the whole process takes 4.2ms, including detecting the initial

seed corners (or 5.3ms with radial distortion correction), and 78 lines are detected, shown in Figure 1(b).

Many of the lines detected overlap each other, mainly due to the FAST detector finding multiple features along edges. In addition, for real-time SLAM we want to ignore lines near to existing lines in the map, so that the map remains sparse. For both these reasons, a post-processing stage is performed that checks for overlapping lines. This renders each line into a Boolean image, starting with the existing lines from the SLAM system, and then rendering the new lines in order of length, starting with the longest first (since we prefer longer lines to shorter ones). Any line which overlaps an already-rendered line is rejected. Figure 1(c) shows the 18 remaining lines after this process (with no existing lines in this case). This post-processing takes a further 0.2ms, bringing the total time for line detection to 4.4ms.

This line detection algorithm is easily fast enough to use as part of a line-based SLAM system. It does not detect every line in a frame, but detects more than enough in each frame to select a few lines to initialise. Although it scales with the square of the number of corners detected, a cap can be placed on the time taken by only keeping the strongest n corners: in the system presented here, we limit the number of corners to $n = 175$.

3 EKF SLAM using Straight Lines

Having developed a fast means of detecting lines they then need to be included in a SLAM system, and here we follow the approach introduced by Davison [5]. This stores the estimate of the camera location $\boldsymbol{x}_v$ (including its velocity) and the three-dimensional position vectors representing the location of the (in his case) point features $\boldsymbol{y}_i$ in a single state vector $\boldsymbol{x}$, with the uncertainty as a full covariance matrix $\mathtt{P}$:

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_v \\ \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \end{pmatrix} \qquad \mathtt{P} = \begin{bmatrix} \mathtt{P}_{xx} & \mathtt{P}_{xy_1} & \mathtt{P}_{xy_2} & \dots \\ \mathtt{P}_{y_1x} & \mathtt{P}_{y_1y_1} & \mathtt{P}_{y_1y_2} & \dots \\ \mathtt{P}_{y_2x} & \mathtt{P}_{y_2y_1} & \mathtt{P}_{y_2y_2} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{1}$$

As each frame arrives, the camera's new position is predicted according to a motion model, and from that the image location of each point feature is predicted. Each point feature is characterized by the image patch around it when it was first observed. A set of visible features is selected for measurement, and their image patches are matched by cross-correlation within a confidence region around their predicted image locations. The innovation between the predicted and observed location of each feature is used to update the Extended Kalman Filter and refine the estimates of both the camera and feature locations and reduce their uncertainties. On a modern personal computer, Davison's system takes around 10ms per video frame, with the major elements being: feature matching 2ms; Kalman Filter update 2ms; feature initialisation 5ms. With a time budget of 33ms per frame at 30Hz, there is spare processing time available to add lines into this system.

3.1 Representing straight lines

Lines in our system are represented by the locations of the two line end-points. Measurements are only ever made normal to the line (due to the aperture problem), which means

that only the location and direction of the corresponding infinite line can be estimated. The component of the line end-points' locations along the line will remain uncertain, but this makes no difference to the accuracy of the camera localisation or the other line parameters. In theory, such an infinite line could be parameterized by any two points on it, but in this system the end-points are used to determine suitable places for measuring the line (i.e. the segment between the two points), so should ideally be close to the physical line end-points. For two three-dimensional points $\boldsymbol{y}_{iA}$ and $\boldsymbol{y}_{iB}$ with corresponding image projections $\boldsymbol{g}_{iA}$ and $\boldsymbol{g}_{iB}$, a line is therefore represented as

$$\boldsymbol{y}_i = \begin{pmatrix} \boldsymbol{y}_{iA} & \boldsymbol{y}_{iB} \end{pmatrix}^T \qquad \text{and} \qquad \boldsymbol{g}_i = \begin{pmatrix} \boldsymbol{g}_{iA} & \boldsymbol{g}_{iB} \end{pmatrix}^T \tag{2}$$

in the map and image respectively. This is clearly not a minimal representation, but does simplify the implementation greatly, as well as being more linear than some other representations, such as the angles used in [4], and hence better for use in the EKF.

Since only measurements normal to a line are practicable, the measurement vector $\boldsymbol{h}_i$ for a line feature $\boldsymbol{y}_i$ is defined to be the projection of the end-points onto the line normal:

$$\boldsymbol{h}_i = \begin{pmatrix} \hat{\boldsymbol{n}}_i \cdot \boldsymbol{g}_{iA} & \hat{\boldsymbol{n}}_i \cdot \boldsymbol{g}_{iB} \end{pmatrix}^T \tag{3}$$

where $\hat{\boldsymbol{n}}_i$ is the unit normal to the line in the image. In fact, $\boldsymbol{h}_i$ itself is also not measured: instead the innovation, the normal error between the predicted end-points and the image edge, $\boldsymbol{v}_i = \begin{pmatrix} v_{iA} & v_{iB} \end{pmatrix}^T$, is measured directly (see Section 3.2).

This choice of representation makes the implementation of line features in an EKF very simple. The EKF filter measurement update step requires the Jacobian relating the measurement vector $\boldsymbol{h}_i$ to the system state $\boldsymbol{x}$, and in this system this simplifies to the two independent Jacobians $\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}_v}$ and $\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{y}_i}$. If both of these are already known for point features, the Jacobians for line features can be easily stated:

$$\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}_v} = \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{g}_i}\frac{\partial \boldsymbol{g}_i}{\partial \boldsymbol{x}_v} = \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{g}_i}\begin{bmatrix} \frac{\partial \boldsymbol{g}_{iA}}{\partial \boldsymbol{x}_v} \\ \frac{\partial \boldsymbol{g}_{iB}}{\partial \boldsymbol{x}_v} \end{bmatrix} \qquad \text{and} \qquad \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{y}_i} = \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{g}_i}\frac{\partial \boldsymbol{g}_i}{\partial \boldsymbol{y}_i} = \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{g}_i}\begin{bmatrix} \frac{\partial \boldsymbol{g}_{iA}}{\partial \boldsymbol{y}_{iA}} & 0 \\ 0 & \frac{\partial \boldsymbol{g}_{iB}}{\partial \boldsymbol{y}_{iB}} \end{bmatrix} \tag{4}$$

where Jacobians in $\boldsymbol{g}_{iA}$ and $\boldsymbol{g}_{iB}$ are the end-point measurement Jacobians, and $\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{g}_i}$ is the Jacobian of the projection of the points onto the line normal:

$$\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{g}_i} = \begin{bmatrix} \hat{\boldsymbol{n}}_i^T & 0 \\ 0 & \hat{\boldsymbol{n}}_i^T \end{bmatrix} \tag{5}$$

3.2 Making Line Measurements

The innovation for a line is measured using a set of sample points arrayed along the line, as is common in line-tracking applications [9, 6]. Each sample point performs a one-dimensional search in the image, in the direction normal to the line, for the closest pixel with a high intensity gradient in that direction. The sample point measurements (the error distances) along the line are then combined, using least-squares (or a more robust technique) to produce a final estimate of the end-point innovation $\boldsymbol{v}_i$ and also a covariance matrix for that measurement, $\mathtt{R}_i$. The sample point measurements could alternatively be fed directly into the system's EKF, but in this case guaranteeing that every line provides

a two-dimensional measurement vector simplifies the implementation, and opens up the possibility, later, of using more sophisticated robust methods when making the measurement.

The innovation covariance $\mathtt{S}_i$ provides a natural means by which to define the size of the search region (as is also used by Davison [5]). This matrix is given by

$$\mathtt{S}_i = \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}_v} \mathtt{P}_{xx} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}_v}^T + \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}_v} \mathtt{P}_{xy_i} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{y}_i}^T + \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{y}_i} \mathtt{P}_{y_i x} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}_v}^T + \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{y}_i} \mathtt{P}_{y_i y_i} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{y}_i}^T + \mathtt{R}_i \qquad (6)$$

where the matrices $\mathtt{P}_{ij}$ are the relevant parts from the EKF state covariance matrix defined in (1). Choosing a 3σ confidence level, this covariance defines a maximum distance for each sample point to search. If no match is found, a sample point is not used, and if less than a certain fraction of a line's sample points find a match (currently set to 66%), the line's measurement is deemed to have failed.

The prediction step of the EKF predicts the new camera location and the image location of the lines. Of the visible lines (those with more than four sample points in the frame) up to ten are selected, and the measurements from each of these (if successful) are fed into the EKF update step.

3.3 Line Initialisation

If there are insufficient lines in a frame (fewer than ten), new lines are detected using the algorithm of Section 2. Up to four of these are selected to initialise at any one time and, of the lines detected, the ones selected are those which are most normal to the current camera velocity, since they are more likely to initialise quickly.

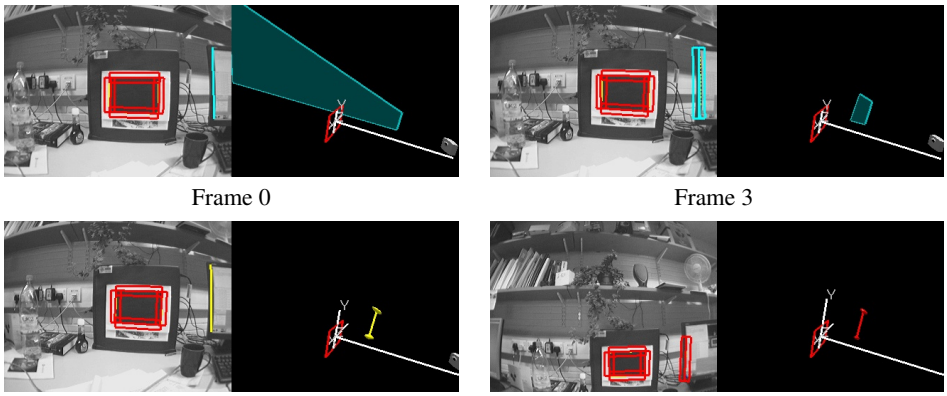
All monocular SLAM algorithms share the problem that a camera provides a two-dimensional measurement of a three-dimensional feature, and so the depth of a feature cannot be known until it has been observed from more than one viewpoint. The statistics of measuring this depth are non-Gaussian, so Davison [5] represents the uncertainty in new points using a one-dimensional particle filter along the world ray that corresponds to the detected feature and updates this until it has converged enough to be represented by a single Gaussian. A more efficient solution is given by considering inverse depth instead, for which a Gaussian distribution is acceptable since inverse depth varies linearly with disparity [7, 13], and as a result the EKF can still be used.

The two end-points of the line detected in the image define two rays in the world upon which the end-points lie, and the three-dimensional line feature lies on the plane between these rays (see Figure 2). A partially-initialised feature is parameterised in two parts:

$$\boldsymbol{y}_{pi} = \begin{pmatrix} \boldsymbol{r}_i & \hat{\boldsymbol{h}}_{iA} & \hat{\boldsymbol{h}}_{iB} \end{pmatrix}^T \qquad \text{and} \qquad \boldsymbol{\lambda}_i = \begin{pmatrix} \lambda_{iA} & \lambda_{iB} \end{pmatrix}^T \qquad (7)$$

where $\boldsymbol{y}_{pi}$ defines the plane using $\boldsymbol{r}_i$, the camera position when the feature was first observed, and $\hat{\boldsymbol{h}}_{iA}$ and $\hat{\boldsymbol{h}}_{iA}$, the unit vectors giving the directions of the two rays. The vector $\boldsymbol{\lambda}_i$ contains the reciprocals of the distances along each ray to the line end-points.

Until a feature is shown to be reliable, it is not used for full measurements. Instead, each new feature has its own EKF which estimates only $\boldsymbol{\lambda}_i$, using line tracking in exactly the same way as described in Section 3.2. The other parameters ($\boldsymbol{y}_{pi}$) are placed in the main EKF state (which is necessary to correctly populate and maintain the off-diagonal elements in the covariance matrix), but the main EKF does not make measurements of



Frame 0 Frame 3

Frame 6 Frame 93

Figure 2: Line initialisation. Frame 0: A line feature from the image is selected for initialisation, and is represented by a semi-infinite plane in the world; Frame 3: observations have reduced the uncertainty in the plane's extent; Frame 6: the uncertainty has reduced enough for the partially-initialised feature to be converted to the usual line parameterization, i.e. end-points with uncertainty; Frame 93: further observations while tracking have reduced the uncertainty.

this feature. If the feature tracks successfully for some time (set at fifteen frames), and its covariance has shrunk below a threshold, it is then converted to a full feature $\boldsymbol{y}_i$ with covariances $\mathtt{P}_{y_i y_i}$ and $\mathtt{P}_{y_i x_v}$ using both $\boldsymbol{\lambda}_i$ and $\boldsymbol{y}_{pi}$ and their covariances; otherwise it is discarded. The new converted feature $\boldsymbol{y}_i$ replaces $\boldsymbol{y}_{pi}$ in the main EKF map and is made available for tracking. Figure 2 shows an example of this initialisation process.

As with the fully-initialised line representation (2), the representation for $\boldsymbol{y}_{pi}$ makes the EKF in $\boldsymbol{\lambda}_i$ straightforward. Defining $\boldsymbol{\lambda}_i'$ as the vector of (non-inverse) depths to the end-points, and writing the end-point locations given a particular choice of $\boldsymbol{y}_{pi}$ and $\boldsymbol{\lambda}_i'$ as $\boldsymbol{y}_i$, the measurement Jacobian $\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{\lambda}_i}$ is given by

$$\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{\lambda}_i} = \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{g}_i}\frac{\partial \boldsymbol{g}_i}{\partial \boldsymbol{y}_i}\frac{\partial \boldsymbol{y}_i}{\partial \boldsymbol{\lambda}_i'}\frac{\partial \boldsymbol{\lambda}_i'}{\partial \boldsymbol{\lambda}_i} = \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{g}_i}\frac{\partial \boldsymbol{g}_i}{\partial \boldsymbol{y}_i}\begin{bmatrix} \hat{\boldsymbol{h}}_{iA} & 0 \\ 0 & \hat{\boldsymbol{h}}_{iB} \end{bmatrix}\begin{bmatrix} -\frac{1}{\lambda_{iA}^2} & 0 \\ 0 & -\frac{1}{\lambda_{iB}^2} \end{bmatrix} \tag{8}$$

where $\frac{\partial \boldsymbol{g}_i}{\partial \boldsymbol{y}_i}$ and $\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{g}_i}$ were defined in (4) and (5) earlier.

4 Results

A demonstration of the complete system performing real-time SLAM using lines only is shown in Figure 3. Four lines (the edges of the central paper target) were provided as known structure (and scale), and then further features were acquired automatically. The sequence is 27 seconds long, with considerable camera rotation and changes of viewing angle, but the camera trajectory is well-tracked throughout (the trajectory is shown in yellow on the right-hand side of each frame in Figure 3). The standard Davison point-based system [5] fails after around 13 seconds, once the viewing angle becomes too shallow.

The coloured rectangles in the camera view on the left of each figure show the predicted line locations and the 3σ search range for each feature, while the lines on the

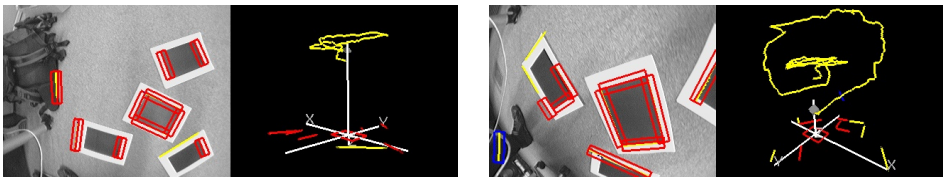


Figure 3: Two views from a 27-second sequence of a nearly-planar scene, tracked using only lines. The current map, on the right of each frame, shows that the structure has been accurately estimated.

map shown to the right of each figure are the corresponding estimated three-dimensional line locations. The features tracked all lie in the plane $z = 0$, which is apparent in the generated map, and this fact can be used to quantify the accuracy of the automatically-initialised lines. The standard deviation in the z-co-ordinate of the detected features is 7.5mm, which over the viewing volume of approximately 1m^3 is an error of less than 1%. Three pairs of detected lines lie on adjoining edges of the rectangles and hence should meet in right angles: the mean absolute angular error in these cases is 1.7°. Overall, we consider this accuracy to be very good for a system operating in real time.

The line-only EKF SLAM system has a similar processing time to the point-only system, taking from 5 to 11ms per frame, depending on the number of lines being initialising, and the number of corners found in the frame. This breaks down as follows:

2ms	Feature measurements
2ms	Kalman Filter update
2–6ms	Line detection
1ms	Partially-initialised feature update

It is this speed of the line-based SLAM system, allied to the equally-fast point-based system that enables a combined point and line system to be contemplated. The parameterizations presented make it easy to use both lines and points in the same system, and putting both feature types and their measurements into the EKF automatically fuses the information from the two sources. Feature management in such a system takes a some care: both points and features need to be initialised at appropriate times to keep enough of each feature type visible, and a good spread in the image, and care needs taking to ensure that the selection of which features to use (where there is an abundance of features) is sensible. In out preliminary system we attempt to ensure equal numbers in all cases.

Figure 4 shows frames from an indoor test sequence, initialised with four point and four line features (the boundary and corners of the initial target). Thirteen further line and twelve point features are initialised by the end of this 28-second sequence, all in plausible locations, with the final map shown to the right of Figure 4 showing features that span the wall and the desk. This is a challenging sequence, and fails when using either points or lines alone. With point features, insufficient good points are detected to track when only the wall is in view, and the highly varied range of viewpoints in the scene means that point matching often fails (note the failed matches, in blue, in Figure 4). With line features, tracking is difficult for some of the lines, with failed matches a problem, which also means that line features do not initialise as quickly as points, and too slowly to cope with some rapid camera motions. The combined system tracks well, showing the benefit of using both features at the same time. The average processing time is 14.5ms per frame.

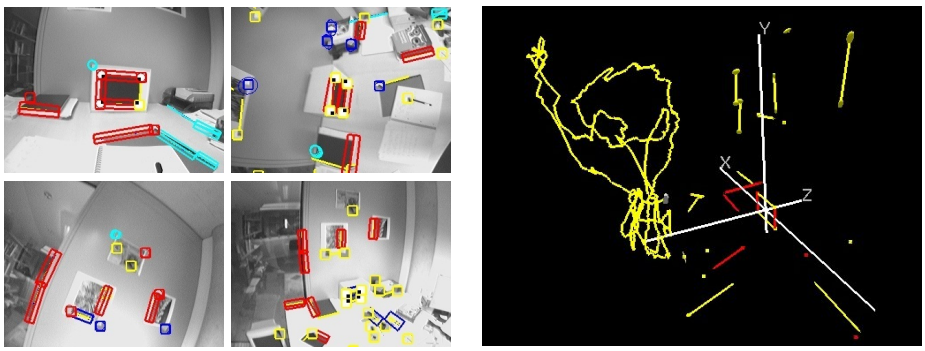


Figure 4: Office sequence. A sequence tracked using the combined point and line EKF SLAM system. Left: Four sample frames from the sequence, with the features overlaid. Right: The final map, showing 17 line and 16 point features, the positions of all corresponding well to the real locations. Features are colour-coded as follows: red=successfully measured in this frame; blue=failed measurement; yellow=not selected this frame; cyan=partially-initialised feature.

5 Conclusion

We have presented a monocular EKF SLAM system that uses straight lines as its features, while still comfortably operating at frame-rate. Line features provide a camera tracking system with natural illumination- and viewpoint-invariance, and when used as part of a SLAM system they provide a more intuitive map, and a more compact representation: lines are large-scale features and so can span the environment more efficiently than points. Lines are also much more useful features from which to start towards building a comprehensive scene description, such as detecting planes and reasoning about occlusion, which will be important as these monocular SLAM systems move away from the office corner and into larger environments. This richer representation of the camera's environment is one area for further research.

We have described a fast straight-line detector that uses seed corners to generate a set of candidate lines. By not insisting on detecting every straight line in the frame, we gain high-speed detection of sufficient lines for a SLAM system. We have demonstrated how lines can easily be included in a standard EKF SLAM framework, and that this provides a straightforward fusion of lines and points. Our preliminary results are promising. We have suggested the naive solution of using the two feature types equally at all times, but the optimal use of the two types of features within such a system is an interesting area for future research. Line features can track very well in a wide range of situations, but are also easily distracted by nearby edges, providing erroneous measurements. (Unlike the image patches used for point features, there is no obvious descriptor for lines to encourage correct association.) Point features are more difficult or expensive to maintain if good matches are required over the same range of camera motions, but match more reliably otherwise. The two have complementary properties, as also noted in [17], and a joint line- and point-based SLAM system that acknowledges this, and uses robust methods to ensure that the appropriate measurements are being made and used, should perform much better than a system using either alone.

Acknowledgements This work was supported by EPSRC Grant GR/T24685/01 and an EPSRC Advanced Research Fellowship to AJD. Thanks go to the members of the Oxford Active Vision Lab for many useful discussions.

References

[1] N. Ayache and O. D. Faugeras. Building, registrating, and fusing noisy visual maps. *International Journal of Robotics Research*, 7(6):45–65, 1988.

[2] A. Bartoli and P. Sturm. Structure-from-motion using lines: Representation, triangulation and bundle adjustment. *Computer Vision and Image Understanding*, 100(3):416–441, 2005.

[3] M. Bosse, R. Rikoski, J. Leonard, and S. Teller. Vanishing points and 3D lines from omnidirectional video. In *Proc. International Conference on Image Processing*, pages 513–516, Rochester, USA, 2002.

[4] M. N. Dailey and M. Parnichkun. Landmark-based simultaneous localization and mapping with stereo vision. In *Proc. Asian Conference on Industrial Automation and Robotics*, Bangkok, Thailand, 2005.

[5] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proc. IEEE International Conference on Computer Vision*, pages 1403–1410, 2003.

[6] T. Drummond and R. Cipolla. Real-time visual tracking of complex structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):932–946, 2002.

[7] E. Eade and T. Drummond. Scalable monocular SLAM. In *Proc. Conference on Computer Vision and Pattern Recognition*, pages 469–468, New York, USA, 2006.

[8] A. Fitzgibbon and A. Zisserman. Automatic camera recovery for closed or open image sequences. In *Proc. European Conference on Computer Vision*, pages 311–326, 1998.

[9] C. Harris and C. Stennett. RAPID, a video rate object tracker. In *Proc. British Machine Vision Conference*, pages 73–77, Oxford, UK, 1990.

[10] H. Kato and M. Billinghurst. Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *Proc. International Workshop on Augmented Reality*, pages 85–94, San Francisco, USA, 1999.

[11] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[12] N. Molton, A. Davison, and I. Reid. Locally planar patch features for real-time structure from motion. In *Proc. British Machine Vision Conference*, Kingston, UK, 2004.

[13] J. M. M. Montiel, J. Civera, and A. J. Davison. Unified inverse depth parametrization for monocular SLAM. In *Proc. Robotics: Science and Systems*, Philadelphia, USA, 2006.

[14] P. M. Newman, J. J. Leonard, J. Tardós, and J. Neira. Explore and return: Experimental validation of real-time concurrent mapping and localization. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1802–1809, Washington, USA, 2002.

[15] D. Nistér. Preemptive RANSAC for live structure and motion estimation. In *Proc. IEEE International Conference on Computer Vision*, pages 199–206, Nice, France, 2003.

[16] M. Pupilli and A. Calway. Real-time camera tracking using a particle filter. In *Proc. British Machine Vision Conference*, pages 519–528, Oxford, UK, 2005.

[17] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Proc. IEEE International Conference on Computer Vision*, pages 1508–1515, Beijing, China, 2005.

[18] J. Weng, T. S. Huang, and N. Ahuja. Motion and structure from line correspondences: Closed-form solution, uniqueness and optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(3):318–336, 1992.