LineSLAM: Visual Real Time Localization Using Lines and UKF

Eduardo Perdices, Luis M. López, and José M. Cañas

Universidad Rey Juan Carlos, Fuenlabrada, Spain
{eperdices,jmplaza}@gsyc.es, luismiguel.lopez@urjc.es
http://jderobot.org

Abstract. In visual simultaneous location and mapping (SLAM) with a single camera, the use of 3D points as a basic feature has been shown sufficient to reliably estimate the camera position and orientation. Nevertheless, the resultant maps are not clear enough for certain applications, even for a large amount of point features. We propose a novel SLAM technique that uses lines as basic features, and the unscented Kalman filter (UKF) as a tracking algorithm. This paper discusses the mathematical foundations as well as the practical implementation of this technique, along with the results of preliminary experiments.

Keywords: Cameras, simultaneous localization and mapping, SLAM, visual localization, line detection, Plücker coordinates, unscented Kalman filter.

1 Introduction

The automatic extraction of relevant information from the image flow of a camera is one of the main challenges of computer vision. One of the main successful techniques in this field are the visual simultaneous localization and mapping (SLAM) algorithms. They provide 3D location of the camera in real time taking as input only the image flow. An important precedent of SLAM is structure from motion (SFM), which has been extensively studied with impressive results over the past three decades [1]. However, the developed approaches usually didn't work in real time since its applications didn't require it. New applications such as augmented reality (AR) have motivated new approaches that focus in real-time algorithms.

Many real-time AR applications with mobile cameras have been used in videogames (e.g. Invizimals$^{\text{TM}}$), in industrial environments (e.g. Seabery's Augmented Training technology for welding) and in medical scenarios. Wearable vision, the new Google glasses and robotics are other scenarios where these algorithms can be applied.

Monocular Simultaneous Localization and Mapping (MonoSLAM) [2] was the first approach achieving good performance working in real time. This approach was based on sequential Bayesian estimation with an extended Kalman filter

(EKF). Other approaches aimed at improving the MonoSLAM technique providing a new parameterization to work with features in the infinite [3], adding new techniques to tolerate spurious features [5] or using particle filtering methods to scale better with the number of features [4]. Another issue to take into account before using previous approaches is linearization; even though EKF is able to manage non linear systems, it can be imprecise or unstable when applied on a highly non-linear model. The UKF [6] is an alternative that can cope with these non-linearities; however, it has more computational complexity that the EKF. Novel works [7] state that in this case a square-root UKF can be used, which maintains the same order of magnitude (O^2) as an EKF.

Afterwards, a new kind of algorithm appeared to solve SfM problem in real time. This new approach, called parallel tracking and mapping [8], introduced a new point of view using optimization methods and decoupling mapping and tracking into two different threads, since most applications need to work in real time only for tracking and not for mapping. In [10] both methods MonoSLAM and PTAM were compared and they came to the conclusion that EKF based approaches work better for small feature sets, whereas PTAM based approaches are more suitable elsewhere. Recently, PTAM approach has been improved with new techniques to make it more robust [9].

Most of the state of art explained before is based on point features. Each landmark is modeled as a point in a 3D space; 2D interesting points are detected in each observation (e.g. with a corner detector) and their surrounding small image patches are stored. Recent research suggests that the use of more complex geometrical features such as straight lines can improve the performance of the MonoSLAM algorithms. See [11] and [12] for EKF implementations to use straight lines and points at the same time; and [13] for an analysis of different parameterizations to work with point and line features.

So far, there are no visual SLAM implementations that use lines as a basic feature and make use of the UKF for tracking at the same time. This paper presents a novel algorithm, named LineSLAM, that covers this gap. Hopefully lines would improve accuracy, robustness to blur and fast movements, and help in the construction of more abstract and reusable maps. In addition, the proposed line representation has been carefully designed to facilitate an efficient and powerful management. The use of lines may be the first step towards more complex objects that may improve the localization quality.

Several algorithms to detect straight lines in an image are available. A basic technique [16] is to apply an edge filter on an image and then use a Hough transform to detect lines, but it consumes too much time to work in real time. More efficient and precise implementations are available, [17,18]; the detector chosen for the present work [18] is explained in Section 3.3.

The rest of the paper is organized as follows: Section 2 of the paper describes the proposed line representation and some related operations. In Section 3, the UKF-, line-based SLAM algorithm is detailed, along with the image pressing required. Some preliminary experiments performed with the software implementation of our visual SLAM are shown in Section 4. Conclusions and future lines in section 5 wrap up the paper.

2 Line Representation

We aim to take advantage of environments that contain straight lines by parameterizing them in both the 3D world and the 2D image plane. This section describes the main mathematical tools (structures and operations) that will be used to model the environment of the camera. In structured real environments, especially indoors, straight lines are ubiquitous as they are formed by the intersection of 2 planes (such as walls) and they are present between corners of artificial objects such as pieces of furniture.

Some of the advantages of using lines instead of (or additionally to) points are: i) lines having homogeneous visual features along themselves are easy to detect and parameterize; ii) the schemes can be adapted to deal with incomplete observations (segments) resulting from occlusions and image-plane cutting; iii) the combination of corners (points) and edges (lines) can help detect and identify objects in the scene; iv) scale- and perspective- invariance characteristics of edge features [14] can be exploited to make the algorithm more robust.

Points in our model are represented using the homogeneous coordinates: the 2D point $[u, v]^T$ corresponds to the equivalence class $\lambda[u, v, 1]^T$ $\forall\lambda \neq 0$. A 2D point corresponds to a subspace of a 3D space. Similarly, the 3D point $[x, y, z]^T$ corresponds to the class $\lambda[x, y, z, 1]^T$ $\forall\lambda \neq 0$ (subspace of a 4D space).

Representing 2D lines by their slope and origin ordinate renders vertical lines impossible to represent, since their slope equals infinity. To avoid such an ill-posing, a line in the image plane is associated with the equation $ax + by + c = 0$, different choices of $[a, b, c]$ giving rise to different lines. Thus, a line may naturally be represented by the vector subspace $k[a, b, c]^T$ $\forall k \neq 0$.

The homogeneous representation of lines and points allows us to write some properties and line-point operations as simple vector operations.

This paragraph addresses planes and lines in 3D (see [1] for a more detailed explanation). Expressing the plane as an equation similarly to the 2D line equation, the vector equation $\boldsymbol{\pi}^T\mathbf{x} = 0$ expresses that the point $\mathbf{x}$ is on the plane defined by the 4-component vector $\boldsymbol{\pi}$. Having this in mind, note that lines in 3D cannot be represented by one scalar equation: at least two equations are required. Moreover, the 3D line can be intuitively represented as the join of 2 separate points or, equivalently, the intersection of 2 non-parallel planes. As a consequence, a line can be represented by a 4×4 skew-symmetric homogeneous matrix. In particular, the line joining the two points $\mathbf{A}, \mathbf{B}$ is represented by the Plücker matrix $\mathbf{L} = \mathbf{A}\mathbf{B}^T - \mathbf{B}\mathbf{A}^T$. The matrix $\mathbf{L}$ is independent of the points $\mathbf{A}, \mathbf{B}$ used to define it and has the required 4 degrees of freedom for a line in 3D. Point and plane transformations defined by matrices can easily be adapted to define a line transformation. A line in Plücker format can be more compactly represented as a 6-entry vector corresponding to the 6 nonzero elements of $\mathbf{L}$, namely $\mathcal{L} = [L_{12}, L_{13}, L_{14}, L_{23}, L_{42}, L_{34}]$. This form is more convenient for the

Kalman filter formulation as using it is equivalent to updating only the relevant components of the original 4×4 Plücker matrix.

2.1 Point, Plane and Line Operations

Some basic operations concerning points, planes and lines are presented next which are relevant to the line-based SLAM (LineSLAM) formulation.

The most important operation associated with the camera model is the observation function. This function is characterized by the 3×4 projective[1] matrix $\mathbf{E}$ that can be computed as $\mathbf{E} = \mathbf{KRT}$, where $\mathbf{K}, \mathbf{R}, \mathbf{T}$ are the projection, rotation and translation matrix. While $\mathbf{R}$ and $\mathbf{T}$ are 4×4 matrices that convert absolute (world) coordinates into camera frame coordinates, $\mathbf{K}$ is the 3×4 projection matrix that converts camera frame coordinates into image plane (2D) coordinates. More precisely, while $\mathbf{R}$ and $\mathbf{T}$ are time variant and depend on the camera state (position and orientation) vector; $\mathbf{K}$ is constant and it depends on the camera intrinsic parameters.

Once we have defined the observation projective matrix $\mathbf{E}$, we describe the basic transformations associated with it.

– Point projection. If we have a point $\mathbf{p}$ in absolute coordinates, its projection on the image plane is $\mathbf{p'} = \mathbf{Ep}$
– Line projection. If we have a line $\boldsymbol{L}$ in 3D absolute coordinates and in Plücker matrix format, its projection on the image plane is $\boldsymbol{L}' = \mathbf{E}\boldsymbol{L}\mathbf{E}^{\mathrm{T}}$. To convert the resulting 3×3 matrix into a homogeneous line vector, we take $\boldsymbol{l} = [L'_{23}, L'_{13}, L'_{21}]^{\mathrm{T}}$.

$$\begin{aligned} \boldsymbol{L}' &= \mathbf{E}\boldsymbol{L}\mathbf{E}^{\mathrm{T}} \\ \boldsymbol{l} &= \left[L'_{23}\ L'_{13}\ L'_{21}\right]^{T} \end{aligned} \tag{1}$$

– Finding the φ-plane corresponding to an observed line. When the true position of a line is not available, but a projection of it (from a known camera state) is available, its coordinates cannot be estimated just from one view (observation), but it is possible to find the plane that contains all the lines that could give rise to that view. Since that plane also contains the *focal* point of the camera in the position of the current observation, we call that plane the *φ-plane*. Given the projection matrix $\mathbf{E}$ and the 2D line (the view) $\boldsymbol{l}$, the φ-plane is computed as $\boldsymbol{\varphi} = \mathbf{E}^{\mathrm{T}}\ \boldsymbol{l}$. The proof is straightforward and is omitted for space limitations. This representation is useful because observations of the same line from different viewpoints give rise to a pencil of φ-planes that intersect on the same line. As a consequence, a set of φ-planes that match the same line and their viewpoints have enough parallax can be used to estimate the line vector. This will be further explained in Sect. 2.2.

[1] Note that the projective matrix that maps from 3D to 2D is 3×4 because of the homogeneous coordinates. The same applies to the dimension of $\mathbf{K}, \mathbf{R}$ and $\mathbf{T}$.

2.2 Line Initialization

This section is devoted to the process of initializing newly observed features (lines), which is key to the creation and maintenance of a map from the processed visual information.

Mathematically, this can be defined as robustly estimating an unknown 3D line which is the "rolling axis" of a pencil of planes, from noisy observations (projections) of the planes belonging to such pencil. This is useful because, despite all φ-planes associated with observations of a single line intersect in that line, the observations are usually corrupted with noise and that noise makes the planes not intersect on the same line. Moreover, errors in the visual line matching could give rise to outliers in the set of φ-planes. A measure of the reliability of the estimated line is needed to know when the available φ-plane observations are "different enough" to reliably compute a 3D line.

To estimate the line most likely to give rise to a set of φ-planes, we rely on the singular value decomposition (SVD) method. To do so, we construct an $n \times 4$ matrix $\mathbf{M}$, such that each line of the matrix is a φ-plane in the set, and we factorize it as $\mathbf{M} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}}$. Intuitively, the 4×4 matrix $\mathbf{U}$ contains 4 "representative" φ-planes, and the diagonal, 4×4 matrix $\boldsymbol{\Sigma}$ contains the singular values associated to these representative planes. The estimated line is the intersection of the φ-planes associated with the 2 largest singular values.

In a scenario where the set of φ-planes is not corrupted with noise and there are at least 2 different φ-planes, $\mathbf{M}$ has rank 2, and therefore, the resulting $\boldsymbol{\Sigma}$ has only 2 nonzero values. In the case of noisy observations, if the different points of view (location of the camera associated with an observation) of the line have enough parallax, then $\mathbf{M}$ will have 2 large SVs (singular values) and 2 SVs close to 0. If there is only 1 large singular value, that means that the observations have not enough parallax and the line cannot be reliably estimated. If there are 3 or 4 large singular values, that means that the amount of noise is too high. In practice we say that the matrix has 2 large singular values when the 2nd SV divided by the 3rd SV is greater than a pre specified threshold value. This threshold has been set to 20 in experimental tests with reasonably good results. This provides a means to decide when to initialize a line, i.e. incorporate it to the set of known lines that are used to refine the estimated camera location.

3 LineSLAM

LineSLAM is a SLAM algorithm that, starting from N_i known (landmark) 3D lines, keeps these lines under track from the image flow of a single camera and creates a map with N 3D lines. It also incorporates the detected lines in the scene to the set of landmark lines. This algorithm is inspired by MonoSLAM approaches, with the novelty of using lines as features instead of isolated points.

Most MonoSLAM approaches deal with their non-linear dynamic models by using an EKF, which basically linearizes the non-linear functions replacing these functions with their Jacobians where needed. Instead of using an EKF, LineSLAM has been implemented with a UKF. The associated dynamic model's functions need not be linearized; instead, a deterministic sampling technique known as unscented transform is used to generate a reduced set of samples around the mean of statistic distributions.

Figure 1 shows the main steps of the algorithm. At each iteration, a new video frame is obtained and lines are detected from it [cf. Sect. 3.3]. Afterwards, the UKF prediction step is performed, what involves predicting the probability distribution of the current camera state and landmark line vectors. The predicted line observations (resulting from projecting the known landmark lines into the predicted camera image plane) are matched with the 2D lines detected (this steps will be further explained in Sects. 3.2 and 3.4). The last step consists in creating new "candidate" lines from unmatched observations and initialize them as valid 3D lines whenever they fulfill the conditions explained in Sect. 2.2. The newly initialized 3D lines are added to the initial map with the aim of improving the accuracy of the camera localization.

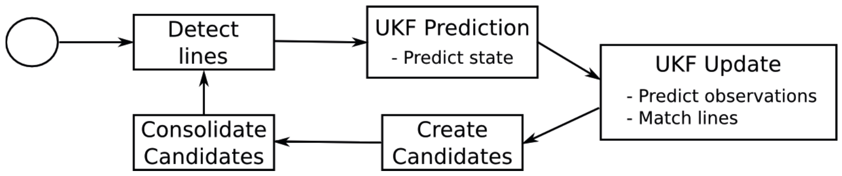

Fig. 1. Main LineSLAM algorithm design

3.1 State and Observation Models

The dynamic model's estimated state vector $\hat{x}$ contains the camera state $\hat{x_c}$ and N line states $\hat{l_i}$ corresponding to the known landmarks. Generally not all known landmarks will be in the camera field of view, so that the state vector size will change dynamically depending on the number of observed lines.

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{\mathbf{x}}_c \\ \hat{l_1} \\ \hat{l_2} \\ \dots \\ \hat{l_N} \end{bmatrix} \; ; \; \hat{\mathbf{x}}_c = \begin{bmatrix} r^W \\ v^W \\ q^{WC} \\ \omega^C \end{bmatrix} \; ; \; \hat{\mathcal{L}}_i = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \\ l_5 \\ l_6 \end{bmatrix}$$

The camera state vector $\hat{\mathbf{x}}_c$ is composed of its 3D position r^W, its orientation quaternion q^{WC} and its linear and angular velocities v^W and ω^C. Each line's state is in turn a 6-dimension Plücker vector [cf. Sec. 2]. Coordinates are defined in Fig. 2.

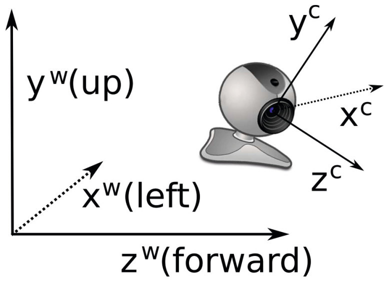

Fig. 2. Camera and world coordinates

The state-transition (movement) model F_t in a time step Δt is constant for each line. The camera transition model F_{t_c} is defined by the following equation:

$$F_{t_c}(\mathbf{x}_c) = \begin{bmatrix} r^W_{|t-1} + v^W_{|t-1}\Delta t \\ v^W_{|t-1} \\ q(\omega^C_{|t-1}\Delta t) \times q^{WC}_{|t-1} \\ \omega^C_{|t-1} \end{bmatrix}$$

The observation vector H_t is formed by concatenating all visible (2D) lines at time t. The observation model associated to this is as follows: for each line $\mathcal{L}_i$ defined by its 6-dimension Plücker vector, a 3D vector H_{t_i} is computed converting the Plücker vector into a $4x4$ matrix:

$$\mathcal{L}_i = \begin{bmatrix} L_{12}\ L_{13}\ L_{14}\ L_{23}\ L_{42}\ L_{34} \end{bmatrix}^T \Rightarrow \begin{bmatrix} 0 & L_{12} & L_{13} & L_{14} \\ -L_{12} & 0 & L_{23} & -L_{42} \\ -L_{13} & -L_{23} & 0 & L_{34} \\ -L_{14} & L_{42} & -L_{34} & 0 \end{bmatrix} = \mathbf{L}$$

and then projecting the line according to (1).

3.2 UKF Prediction and Update

As it was stated before, the UKF uses a deterministic sampling technique to handle non-linear dynamic equations. For a state x with N_p parameters, whose state prediction is $\hat{x}$ and its covariance P, the UKF needs to calculate $2N_p + 1$ states χ_i (called "sigma" vectors) and its corresponding weights $W_i^{(m)}$ and $W_i^{(c)}$ using the following formulas:

$$\begin{cases} \chi_i = \hat{x} \text{ if } i = 0 \\ \chi_i = \hat{x} + (\sqrt{(N_p + \lambda)P})_i \text{ if } i = 1, ..., N_p \\ \chi_i = \hat{x} - (\sqrt{(N_p + \lambda)P})_i \text{ if } i = N_p + 1, ..., 2N_p \end{cases}$$
$$\begin{cases} W_i^{(m)} = \frac{\lambda}{N_p+\lambda} \text{ if } i = 0 \\ W_i^{(m)} = \frac{1}{2(N_p+\lambda)} \text{ if } i = 1, ..., 2N_p \end{cases}$$
$$\begin{cases} W_i^{(c)} = \frac{\lambda}{N_p+\lambda} + (1 - \alpha^2 + \beta) \text{ if } i = 0 \\ W_i^{(c)} = \frac{1}{2(N_p+\lambda)} \text{ if } i = 1, ..., 2N_p \end{cases}$$

where $\lambda = \alpha^2(N_p + \kappa) - N_p$ is a scale factor, α represents how each sigma will be spread around $\hat{x}$, and κ and β are scale factors that can be tuned depending on the a priori knowledge about the distribution of x.

Once the vectors χ_i and its corresponding weights $W_i^{(m)}$ and $W_i^{(c)}$ have been computed, the Kalman prediction step is as follows:

$$\begin{aligned} \hat{x}_{t|t-1} &= \textstyle\sum_{i=0}^{2N_p} W_i^{(m)} F(\chi_i) \\ P_{t|t-1} &= \textstyle\sum_{i=0}^{2N_p} (W_i^{(c)}(F(\chi_i) - \hat{x}_{t|t-1})(F(\chi_i) - \hat{x}_{t|t-1})^T) + Q_t \end{aligned}$$

where $\hat{x}$ is the state vector, P the state covariance matrix and Q the covariance error matrix. The Kalman update step is in turn computed through these equations:

$$\begin{aligned} Y_t &= \textstyle\sum_{i=0}^{2N_p} W_i^{(m)} H(\chi_i) \\ S_t &= \textstyle\sum_{i=0}^{2N_p} (W_i^{(c)}(H(\chi_i) - Y_t)(H(\chi_i) - Y_t)^T) + R_t \\ C_t &= \textstyle\sum_{i=0}^{2N_p} (W_i^{(c)}(\chi_i - \hat{x}_{t|t-1})(H(\chi_i) - Y_t)^T) \\ \mathcal{K}_t &= C_t S_t^{-1} \\ \hat{x}_t &= \hat{x}_{t|t-1} + \mathcal{K}_t(z_t - Y_t) \\ P_t &= P_{t|t-1} - \mathcal{K}_t S \mathcal{K}_t^T \end{aligned}$$

where Y is the prediction mean vector, S the prediction covariance matrix, C the prediction state cross-covariance matrix, $\mathcal{K}$ the Kalman gain matrix, z the real observation vector and R the observation error matrix.

3.3 Line Detection

Each video frame is analyzed to detect segments of straight lines. We have adapted the approach described by Solis et al. [18], which uses a compilation of different image processing schemes involving normalization, Gaussian smoothing, thresholding, and Laplace edge detection to extract edge contours from input images; then, these contours are labeled according to its orientation and segments are recognized joining consecutive contours with similar orientations. Figures 3 and 4 shows the algorithm results in indoor and outdoor image samples.

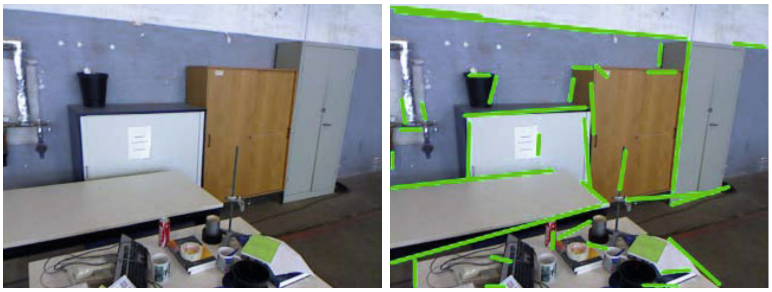

Fig. 3. Line detection in an indoor scenario, 36 lines detected in 4.5 ms

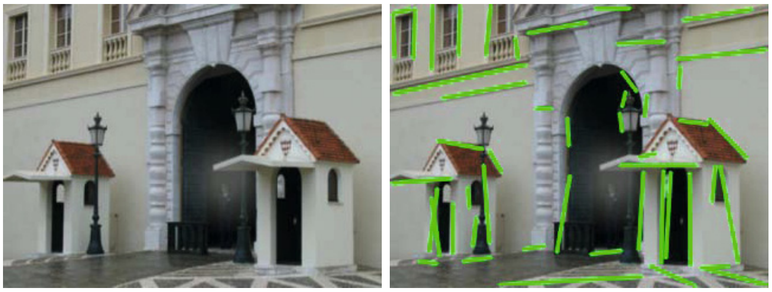

Fig. 4. Line detection in an outdoor scenario, 49 lines detected in 4.9 ms

This solution is far more accurate than other line detection algorithms; it also exhibit robustness and obtains good results in a few milliseconds, what is essential to maintain our algorithm running in real time. The execution time of the algorithms vary from 3 to 6 ms with a 320×240 image resolution depending on the image characteristics.

3.4 Line Matching

While carrying out the UKF update step, the lines stores in MonoSLAM must be matched with the segments detected in the image. To do so, each 2D segment is converted into the general equation form of the line solving the determinant with its extremes ($\mathbf{p}_s$ and $\mathbf{p}_e$):

$$\mathbf{v}_i = (x_i, y_i, z_i) = \begin{vmatrix} i & j & 1.0 \\ p_{s_x} & p_{s_y} & 1.0 \\ p_{e_x} & p_{e_y} & 1.0 \end{vmatrix}$$

These 2D lines are compared with the known 3D lines projecting each 3D line into the image plane to obtain a 2D vector $v_l = (x_l, y_l, z_l)$ [cf. Sec. 2.1]. To match each detected line with the LineSLAM 3D lines we have implemented a method to compare two 2D lines. The first step is projecting the 2D vector into a subset of $\mathbb{R}^3$ with:

$$\mathbf{w} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \Rightarrow \begin{bmatrix} x/\sqrt{x^2+y^2} \\ y/\sqrt{x^2+y^2} \\ z/\sqrt{x^2+y^2} \end{bmatrix}$$

This subspace makes easier to compare distances and angles among 2D lines. Thus, comparison between two 2D lines converted to this subspace (w_i and w_l) is made by the equation:

$$d = \begin{bmatrix} a_i - a_l \\ b_i - b_l \\ c_i - c_l \end{bmatrix}^T \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \alpha \end{bmatrix} \begin{bmatrix} a_i - a_l \\ b_i - b_l \\ c_i - c_l \end{bmatrix}$$

where α balances the importance between line-to-line distance and orientation. Even with this equations, consecutive segments with the same orientation are difficult to match because both segments might have the same general equation form of the line. To rule out these cases we also store representative image patches of the 3D line the first time it is observed and calculate the correlation between theses patches and current segments patches with a zero-mean sum of squared differences algorithm (ZMSSD), a technique that is fast to compute, and invariant to brightness.

To validate a match between lines both d and paths correlation must be below certain thresholds. These thresholds are set depending on image resolution, image quality and current environment.

4 Experiments

We have tested our LineSLAM implementations first in simulated environments with Matlab and afterwards with real observations from one single camera.

4.1 Simulated Data

The main objective of the simulations is to validate the numerical procedures described in the body of this article. Moreover, they are intended to give insights on the impact of different parameters such as noise, number of detected lines, and the distance of the lines to the camera (relative to their length).

The results shown in this section are the results of a Monte Carlo estimation of the mean square error of the camera position and orientation, as a function of different parameters. The methodology of the experiments is as follows: for each tuple of parameters, several sets of random 3D lines are generated. The 3D lines are contained in a unitary sphere and the simulated camera traces a complete circle around the center of that sphere. The position and orientation of the camera is estimated from the noisy observations of those lines. The mean square error is averaged over all the independent runs.

The first set of simulations shows the effect of the noise in the camera localization error for different numbers of detected lines. The radius of the camera circular trajectory is fixed at 28 cm. The error curves show that the performance of the filter is limited for low levels of noise. Note that the slope of the logarithmic error curve decreases as the noise tends to zero, meaning that it is not possible to arbitrarily improve the quality of the estimation just by reducing the noise in the observations (that would need an unaffordable increase in the camera resolution). The results also suggest that, in order to improve the reliability of the estimated camera's state, the effect of reducing the noise by a factor of 10 is similar to that of increasing the detected lines by a factor of 2. Another conclusion is that a reasonable performance can be obtained by using as less as 4 (non-coplanar) lines.

The second set of simulations shows the effect of the average distance from the camera to the detected lines. As expected, the error increases monotonically with the distance to the observed pattern.

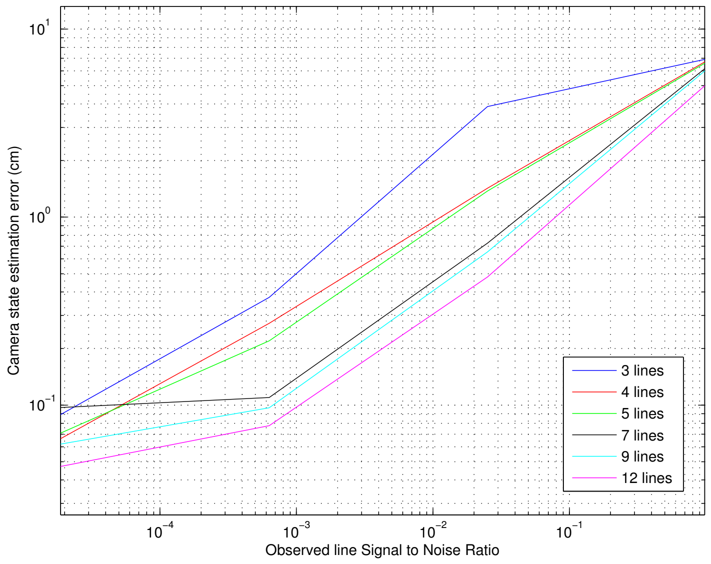

Fig. 5. Error in camera's state estimation vs. observation noise for different number of detected lines

4.2 Experiments with Real Data

We have implemented our LineSLAM approach in C++ to perform experiments with real images in real time. For these real experiments we have used a Logitech QuickCam Pro 9000 camera whose calibration matrix **K** is:

$$\mathbf{K} = \begin{bmatrix} 277.2 & 0 & 162.0 & 0 \\ 0 & 274.9 & 121.0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

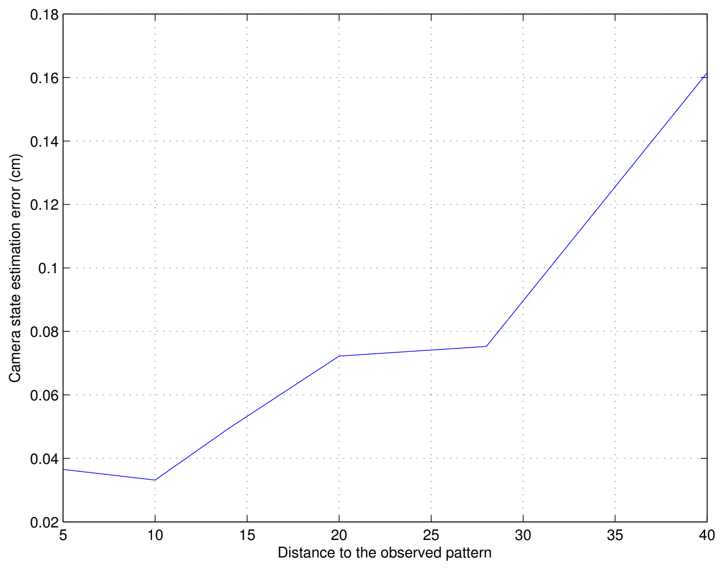

Fig. 6. Error in camera's state estimation vs. average distance to the observed lines

The design of the algorithm is the same than in Matlab but we face new problems such as less accuracy in line detection, what predictably will take to a worse state estimation.

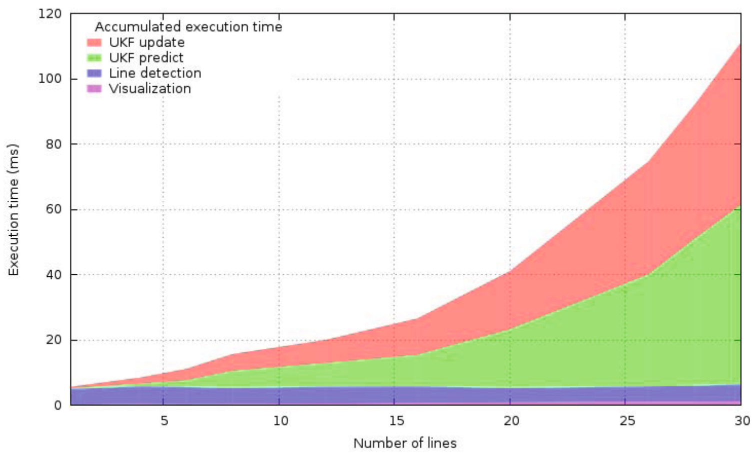

Fig. 7. Execution time vs. number of lines. It includes line detection (3-6ms), visualization (1ms) and UKF prediction and update.

Figure 7 shows how execution time increases with the number of lines N. Since we have used an UKF, whose complexity order is $O(n^3)$, the execution time increases quickly with N, being able to manage only until 18 lines in real time (30FPS). Execution time would improve if we used an EKF, which has a complexity order of $O(n^2)$.

Next experiment (Figs. 8 and 9) demonstrates the creation of new 3D lines with the method explained in Sect. 2.2. The experiment begins with 4 known 3D lines (labelled from 1 to 4), which are matched to their corresponding 2D

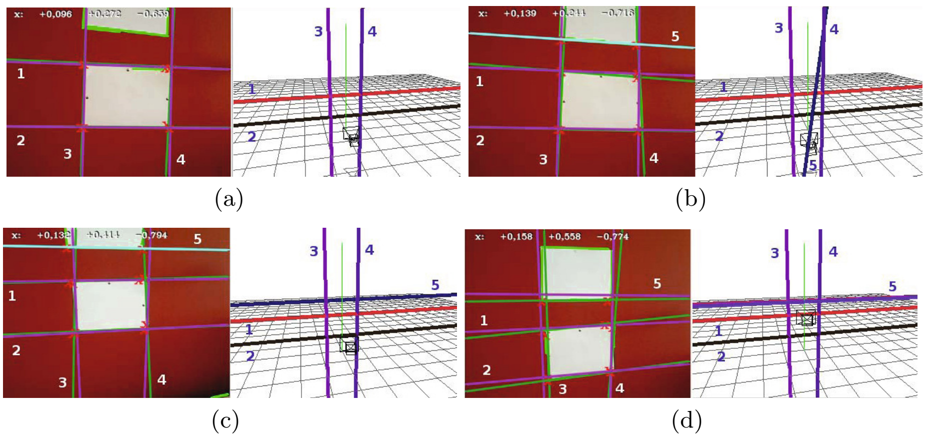

Fig. 8. Left images are real observations which contains detected segments (green), projections of consolidated lines (purple) and candidate lines (cyan). Right images show a 3D simulated environment with current camera position and lines stored.

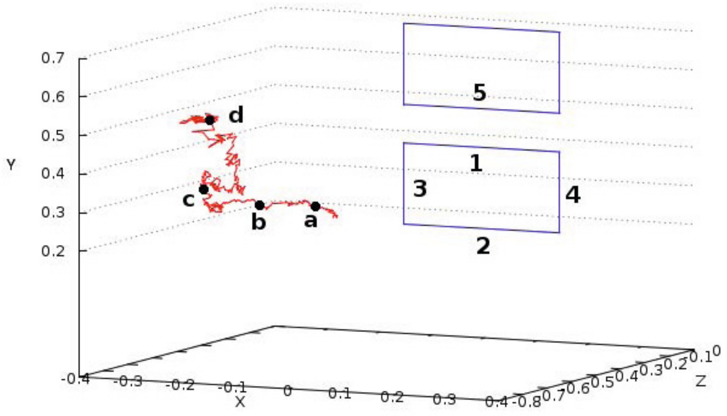

Fig. 9. Camera calculated trajectory (red) and real objects (blue). Labels 1 to 5 correspond to the lines stored in the map whereas labels a,b,c and d are the camera positions which correspond to the observations from Fig. 8.

detected lines and allow the UKF to estimate a reliable camera's state. Then (Fig. 8(b)), a new candidate line in 2D is selected (labelled as 5), but its 3D line is not properly calculated because the camera has not move enough and its parallax is too small to calculate a proper 3D line. After some iterations (Fig. 8(c)) the parallax is big enough and an accurate 3D line is calculated, and therefore the candidate is consolidated as a valid 3D line (Fig. 8(d)). From then on, this consolidated 3D line serves as a reliable and stable 3D line and helps the UKF to estimate its state vector.

5 Conclusions

This article has presented a novel approach to the monocular vision SLAM based on the detection of straight lines and the recursive estimation of the camera state by means of the UKF. The mathematical details of the line modeling, the signal processing (filtering) techniques as well as the image analysis tools related to the detection of lines in the image frame, have been presented, and a novel algorithm implementing all these aspects, LineSLAM, has been developed and experimentally validated.

After testing several different methods, the Solis 2D line [18] algorithm has been chosen as the best available technique for straight line detection. Regarding the filtering technique, the UKF is preferred to the EKF because with UKF does not require to compute the Jacobian matrix for the update step. The problems typically associated with linearization-based approximation are thus avoided.

The representation of lines in 2D and in 3D has been carefully chosen, because the quality of the estimated camera state depends on how they are represented. The Plücker matrix format has been chosen because the functions associated with this representation are expected to work well in conjunction with the family of EKF and UKF filters. Additionally, we have presented a robust algorithm to create new 3D lines from several 2D observations taken from known camera locations. This algorithm is based on SVD and it is the mathematical basis to incorporate 3D lines in the map. This is key to extend the built map to unknown areas, as well as addressing the map maintenance task. A map built in terms of 3D straight lines provides more abstracted information and can be more useful than a map based only on point features.

The performance of the developed algorithms has been assessed by two sets of preliminary experiments. The first one with synthetic data shows the effects of noise and camera-pattern distance in the camera location accuracy. The second one validates the system real-time functioning with a real camera and shows an example of addition of new lines in the map.

The obtained initial results encourage us to extend this work in several directions: i) making the algorithm able to deal with line segments of bounded length, instead of infinite lines; ii) using EKF and square root UKF for filtering instead of UKF to speed up the algorithm, and iii) making a thorough comparison between LineSLAM and point-based visual SLAM in terms of speed, robustness and potential for the closed-loop problem [15]. Two long-term future lines of work are: i) exploring the use of a wider class of objects (e.g. more abstracted shapes such as cubes or cylinders, widespread objects such as pens and books, or scenario-specific objects such as surgical instruments) as features for visual SLAM, and ii) the use of lines in bundle adjustment approaches like PTAM.

Acknowledgements. This research has been partially sponsored by the Community of Madrid through the RoboCity2030-II project (S2009/DPI-1559).

References

1. Hartley, R.I., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press (2004) ISBN: 0521540518
2. Davison, A.J.: Real-time simultaneous localization and mapping with a single camera. In: Proc. International Conference on Computer Vision (2003)
3. Montiel, J., Civera, J., Davison, A.: Unied inverse depth parametrization for monocular slam. In: Proceedings of Robotics: Science and Systems, Philadelphia, USA (August 2006)
4. Eade, E., Drummond, T.: Scalable monocular SLAM. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2006)
5. Civera, J., Grasa, O.G., Davison, A.J., Montiel, J.M.M.: 1-Point RANSAC for EKF Filtering: Application to Real-Time Structure from Motion and Visual Odometry. Journal of Field Robotics (2010)
6. Wan, E.A., Van Der Merwe, R.: The unscented Kalman filter for nonlinear estimation. In: The IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000, pp. 153–158. IEEE (2000)
7. Holmes, S., Klein, G., Murray, D.W.: A square root unscented Kalman filter for visual monoSLAM. In: Proc. of the IEEE International Conference on Robotics and Automation (2008)
8. Klein, G., Murray, D.: Parallel Tracking and Mapping for Small AR Workspaces. In: Proc. International Symposium on Mixed and Augmented Reality, ISMAR 2007, Nara (2007)
9. Strasdat, H., Davison, A.J., Montiel, J.M.M., Konolige, K.: Double Window Optimisation for Constant Time Visual SLAM (PDF format). In: ICCV (2011)
10. Strasdat, H., Montiel, J.M.M., Davison, A.J.: Visual SLAM: Why Filter? Image and Vision Computing (2012)
11. Smith, P., Reid, I., Davison, A.J.: Real-Time Monocular SLAM with Straight Lines (PDF format). In: BMVC (2006)
12. Jeong, W.Y., Lee, K.M.: Visual slam with line and corner features. In: Proc. IEEE/RSJ Int Intelligent Robots and Systems Conf., pp. 2570–2575 (2006)
13. Sol, J., Vidal-Calleja, T., Civera, J., Montiel, J.M.M.: Impact of Landmark Parametrization on Monocular EKF-SLAM with Points and Lines. International Journal of Computer Vision 97(3), 339–368 (2012)
14. Mikolajczyk, K., Zisserman, A., Schmid, C.: Shape recognition with edge-based features. In: British Machine Vision Conference, BMVC 2003 (2003)
15. Williams, B., Klein, G., Reid, I.: Automatic Relocalization and Loop Closing for Real-Time Monocular SLAM. IEEE Transactions on Pattern Analysis and Machine Intelligence 33(9), 1699–1712 (2011)

16. Duda, R.O., Hart, P.E.: Use of the Hough transformation to detect lines and curves in pictures. Communications of the ACM 15(1), 11–15 (1972)
17. Kǒsecká, J., Zhang, W.: Video compass. In: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (eds.) ECCV 2002, Part IV. LNCS, vol. 2353, pp. 476–490. Springer, Heidelberg (2002)
18. Solis, A., Nayak, A., Stojmenovic, M., Zaguia, N.: Robust Line Extraction Based on Repeated Segment Directions on Image Contours. In: Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defence Applications, Ottawa, Canada, July 8-10 (2009)