Drift-Free Real-Time Sequential Mosaicing

Javier Civera
jcivera@unizar.es
Universidad de Zaragoza

Andrew J. Davison
ajd@doc.ic.ac.uk
Imperial College London

Juan A. Magallón
magallon@unizar.es
Universidad de Zaragoza

J.M.M. Montiel
josemari@unizar.es
Universidad de Zaragoza

March 14, 2008

Abstract

We present a sequential mosaicing algorithm for a calibrated rotating camera which can for the first time build drift-free, consistent spherical mosaics in real-time, automatically and seamlessly even when previously viewed parts of the scene are re-visited. Our mosaic is composed of elastic triangular tiles attached to a backbone map of feature directions over the unit sphere built using a sequential EKF SLAM (Extend Kalman Filter Simultaneous Localization And Mapping) approach.

This method represents a significant advance on previous mosaicing techniques which either require off-line optimization or which work in real-time but use local alignment of nearby images and ultimately drift. We demonstrate the system's real-time performance with real-time mosaicing results from sequences with 360 degrees pan. The system shows good global mosaicing ability despite the challenging conditions: hand-held simple low-resolution webcam, varying natural outdoor illumination, and people moving in the scene.

1 Introduction

Mosaicing is the process of stitching together data from a number of images, usually taken from a rotating camera or from a translating camera observing a plane, in order to create a composite image which covers a larger field of view than the individual views. While a variety of approaches have been published for matching up sets of overlapping images with a high degree of accuracy, all have relied on off-line optimization to achieve global consistency. Other previous methods which operate sequentially and in real-time suffer from the accumulation of drift. The goal of this paper is drift-free real-time mosaic building from a *live* camera, connected to a PC.

We propose a method for mosaicing which can achieve real-time operation while benefitting from feature correspondences across arbitrarily long time periods and automatic re-capturing of features as areas are re-visited such that drift is eliminated. For image alignment we propose the probabilistic filtering approach familiar from Simultaneous Localisation and Mapping (SLAM) in mobile robotics research which has not been previously applied to image mosaicing. SLAM is the generic problem which has become well-defined in the mobile robotics field as the challenge a moving sensor platform faces when neither its motion nor the structure of the surrounding scene is known in advance and the goal is to estimate both. They build a persistent, *probabilistic* representation of the state of the sensor and scene map which evolves in response to motion and new sensor measurements. For mosaicing, we specifically use an Extended Kalman Filter (EKF) approach to SLAM with a state vector consisting of stacked parameters representing the 3D orientation and angular velocity of the camera and the directions (i.e. view-sphere coordinates, since no depth can be estimated for the scene points) of a set of automatically acquired features, none of which need to be known in advance.

To summarize our complete mosaicing algorithm, we take the image stream from a rotating camera, build an efficient, persistent SLAM map of infinite points — *directions* mapped onto the unit sphere — and use these as the anchor points of a triangular mesh, built sequentially as the map eventually covers the whole view-sphere (see Figure 1.) Every triangle in the mesh is an elastic tile where scene texture is accumulated to form a mosaic. As each new image arrives, the probabilistic map of infinite points is updated in response to new measurements and all the texture tiles are re-warped accordingly. So, every measurement of a point potentially improves *the whole mosaic*, even parts not currently observed by the camera thanks to probabilistic knowledge of the correlations between estimates of different features. This attribute is especially valuable when a loop is closed because the whole map benefits from a large correction, removing drift.

This paper has two main contributions. First, we present a real-time EKF SLAM algorithm for estimating the motion for a rotating camera observing points at infinity — this work was previously published in conference paper [18] as a 'visual compass'. The sec-

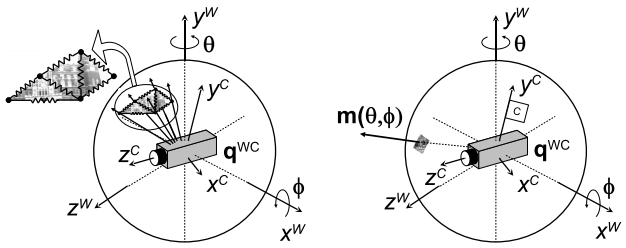

Figure 1: Left, An elastic textured triangular mesh is built over a map of scene directions over the unit sphere. Right a scene feature, $\mathbf{y}_i$, stamped with its texture patch.

ond unpublished and novel contribution is to use this map of *widely spread but high quality features* as the basis for a real-time, drift-free mosaicing system. We believe that this is the first algorithm which can build drift-free mosaics over the whole viewsphere in seamless real-time: no expensive backtracking, batch optimization or learning phase is required, and arbitrarily long image sequences can be handled without slow-down.

Section 2 is devoted to a literature review and comparison between off-line and sequential approaches. Section 3 covers the essentials of the SLAM map of infinite points, and Section 4 describes how features are added to or removed from the map. Section 5 is devoted to the mesh for the mosaic given a map of feature directions. Finally experimental results and conclusions are presented in Sections 6 and 7.

2 Related Work

2.1 Image Mosaicing

Mosaic building requires estimates of the relative rotations of the camera when each image was captured. Classically, the computation of such estimates has been addressed as an off-line computation, using pair-wise image matching to estimate local alignment and then global optimization to ensure consistency. The goal has normally been to produce visually pleasing panoramic images and therefore after alignment blending algorithms are applied to achieve homogeneous intensity distributions across the mosaics, smoothing over image joins. This paper focuses only on the alignment part of the process, achieving registration results of quality comparable to off-line approaches but with the advantage of sequential, real-time performance. Blending or other algorithms to improve the aesthetic appearance of mosaics could be added to our approach straightforwardly, potentially also running in real-time.

Szeliski and Shum in [25] presented impressive spherical mosaics built from video sequences, explicitly recognizing the problem of closing a loop when building full panoramas (from sequences consisting of a single looped pan movement). However, their method needed manual detection of loop closing frames. The non-probabilistic approach meant that the misalignment detected during loop closing was simply evenly distributed around the orientation estimates along the sequence.

Sawhney et al. [22] tackled sequences with more complicated zig-zag pan motions, requiring a more general consideration of matching frames which were temporally distant as loops of various sizes are encountered. Having first performed pairwise matching of consecutive images they could estimate an approximate location for each image in the sequence. This was followed by an iterative hypotheses verification heuristic, applied to detect matching among geometrically close but temporally distant images to find the topology of the camera motion, and then finally global optimization. Both [25] and [22] use whole image intensities without feature detection for the optimization.

Capel and Zisserman in [3] proposed methods based on matching discrete features; RANSAC is applied to detect outlier-free sets of pairwise matches among the images. A global bundle adjustment optimization produces the final mosaic, achieving super-resolution. Agapito et al. [9] also used robust feature matching to perform self-calibration and produce mosaics from sequences from a rotating and zooming camera.

Brown and Lowe in [2] considered the problem of mosaicing not video sequences but sets of widely separated, uncalibrated still images. Their method used SIFT features to perform wide-baseline matching among the images, and automatically align them into a panorama, optimizing over the whole set for global consistency using bundle adjustment.

The recent work of Steedly et al. [24] is perhaps closest to the current paper because it explicitly considers questions of computational cost in efficiently building mosaics from long video sequences (on the order of a thousand frames), though not arriving at real-time performance. The key to efficient processing in their system is the use of automatically assigned keyframes throughout the sequence — a set of images which roughly span the whole mosaic. Each frame in the sequence is matched against the nearest keyframes as well as against its temporal neighbors. In fact, the idea of building a persistent map of keyframes as the backbone of the mosaic can be thought of as very similar to our sparse SLAM feature map, though their work lacks sequential probabilistic filtering to guide matching.

To our knowledge, up to now mosaicing algorithms which truly operate in real-time have been much more limited in scope. Several authors have shown that the straightforward approach of real-time frame to frame image matching can produce mosaics formed by simply concatenating local alignment estimates. Marks et al.

[16] presented a real-time system for mosaicing underwater images using correlation-based image alignment, and Morimoto and Chellappa a system based on point feature matching which estimated frame-to-frame rotation at 10Hz [19]. It should be noted that while Morimoto and Chellappa used the EKF in their approach, the state vector contained only camera orientation parameters and not the locations of feature points as in our SLAM method.

The clear limitation of such approaches to real-time mosaicing is that inevitable small errors in frame to frame alignment estimates accumulate to lead to misalignments which become especially clear when the camera trajectory loops back on itself — a situation our SLAM approach can cope with seamlessly.

Some recent approaches have attempted to achieve global consistency in real-time by other means — Kim and Hong [14] demonstrated sequential real time mosaic building by performing 'frame to mosaic' matching and global optimization at each step. However, this approach is limited to small-scale mosaics because the lack of a probabilistic treatment means that the cost of optimization will rise over time as the camera continues to explore. Zhu et al. [27] on the other hand combine a frame to frame alignment technique with an explicit check on whether the current image matches to the first image of the sequence, detection of which leads to the correction of accumulated drift. Again, the method is not probabilistic and works only in the special case of simple panning motions.

2.2 SLAM

The standard approach to SLAM is to use a single Gaussian state vector and covariance to represent stacked sensor and feature estimates and to update this with the Extended Kalman Filter (EKF) — this approach was called the 'stochastic map' when proposed initially by Smith and Cheeseman in [23]. It has been widely used in mobile robotics with a range of different sensors; odometry, laser range finders, sonar, and vision among others (e.g. [5, 11, 20]). This amounts to a rigorous Bayesian solution in the case that the sensor and motion characteristics of the sensor platform in question are governed by linear processes with Gaussian uncertainty profiles — conditions which are closely enough approximated in real-world systems for this approach to be practical in most cases, and in particular for small-scale mapping.

Visual sensing has been relatively slow to come to the forefront of robotic SLAM research. Davison and Murray [8] implemented a real-time system where a 3D map of visual template landmarks was build and observed using fixating stereo vision. Castellanos et al. [4] built a 2D SLAM system combining straight segments from monocular vision and odometry, and trinocular straight segments and odometry. In [7] Davison demonstrated 3D SLAM using monocular vision as the only sensor, also using a smooth motion model for the camera to take the place of odometry this system exhibited unprecedented demonstrable real time performance for general indoors scenes observed with a low cost hand-held camera. [6] proposed inverse depth coding for points allowing to deal with distant, even at infinite, features. Recently, many other interesting 3D SLAM systems which rely only on visual sensing have started to emerge (e.g. [10, 15, 13]).

2.3 Off-line SFM vs. EKF SLAM

In order to directly compare the previous off-line approaches to mosaicing with our sequential one, we will focus on methods which rely on discrete feature matching. The off-line approaches generally match features between images in a pairwise fashion (usually between temporal neighbours) and then perform a final global bundle adjustment optimization to achieve good drift-free solutions.

In our sequential approach, scene structure — a set of selected points we call a map — and the camera motion are estimated by iterating the following steps:

1. Predict the locations of *all* the map features in the next image, along with a gated search region for each. The information from all previous images is implicitly accumulated in this state-based predection and this leads to tight search regions.
2. Use template matching to exhaustively search for the feature match only within its search region.
3. Update estimates of the locations of *all* the mapped features and the current camera location.
4. Map maintenance, adding new features when new scene areas are explored and deleting features predicted to be visible but persistently not matched.

The resulting sparse set of map features, autonomously selected and matched has — when compared with 'first match and then optimize' approaches — the following desirable qualities for producing consistent mosaics:

Long tracks — Only features persistently observed when predicted to be visible are kept in the map, and are allowed to build up immunity to future deletion. Highly visible and identifiable features tend to live on in this process of 'survival of the fittest'.

Loop closing tracks — When the camera revisits an area of the scene previously observed, it has the natural ability to identify and re-observe 'old', loop closing, features seamlessly.

To achieve the highest accuracy in every update of a scene map after matching image features, the update step would ideally be done using an iterative non-linear bundle adjustment optimization, but this would be prohibitively expensive. Instead, we apply the EKF as a sequential approximation to bundle adjustment. Update by bundle adjustment after processing every single image means a non linear optimization for *all camera locations* and all scene features, so processing long image sequences results in an increasing number of camera locations and hence an increasing dimension for the bundle adjustment of $(3m_k + 2n_k)$, where m_k is the number of images, and n_k is the number of scene points. Also, calculating search regions requires the inversion of a matrix of dimension $3m_k + 2n_k$ to compute the estimation covariance.

To compare the EKF with bundle adjustment, it should be considered that, as stated in [26], the EKF is a sequential approximation to bundle adjustment where:

1. A motion model is included to relate one camera location with the next.

2. The EKF is just bundle adjustment's first half-iteration because only the most recent camera location is computed. The estimated state is reduced, subsuming all historic camera location estimates in the feature covariance matrix. The estimated state, dimension $(7 + 2n_k)$, is composed of the last camera pose and all map features. In our model the camera state vector has dimension 7: an orientation quaternion and 3D angular velocity.

3. At each step, information about the previous camera pose is subsumed in the covariance matrix. In doing so, linearization for previous camera poses is not computed as in the optimal solution, and hence linearization errors will remain in the covariance matrix. However, mosaicing with a rotating camera is a very constrained, highly linear problem, so are convinced that the results we can obtain in real-time are highly accurate, only falling a little short of the result a full optimization would produce.

3 Geometrical Modeling

We start the exposition of our method by explaining the mathematical models used for features, camera motion and the measurement process, before proceeding in Section 4 to the EKF formulation.

Feature Model Feature points are modeled by storing both geometric and photometric information (see Figure 1). Geometrically, the feature's direction relative to world frame W is parameterized as an angular azimuth/elevation pair:

$$\mathbf{y}_i = \begin{pmatrix} \theta_i & \phi_i \end{pmatrix}^\top . \tag{1}$$

To represent each infinite point photometrically, a texture patch of fixed size is extracted and stored when the point is imaged for the first time. This patch is used for correlation-based matching.

Camera Motion Model It is assumed that the camera translation is small compared with actual feature depths — true for any camera on a tripod, or well approximated by a camera rotated in the hand outdoors. The real-world camera dynamics are modeled as a smooth angular motion: specifically a 'constant angular velocity model', which states that at each time-step an unknown angular acceleration impulse drawn from a zero-mean Gaussian distribution is received by the camera. The camera state vector is:

$$\mathbf{x}_v = \begin{pmatrix} \mathbf{q}^{WC} \\ \omega^C \end{pmatrix} , \tag{2}$$

where ω^C is the angular velocity and $\mathbf{q}^{WC}$ is a quaternion defining orientation. See Figure 1.

At every time-step, an angular acceleration α^C having zero mean and fixed diagonal 'process noise' covariance matrix $\mathbf{P}_\alpha$ is assumed to affect the camera's motion. Therefore at each processing step of duration Δt the camera receives an impulse of angular velocity: $\Omega^C = \alpha^C \Delta t$.

So the state update equation is:

$$\mathbf{f}_v(\mathbf{x}_v, \mathbf{n}) = \begin{pmatrix} \mathbf{q}^{WC}_{new} \\ \omega^C_{new} \end{pmatrix} = \begin{pmatrix} \mathbf{q}^{WC} \times \mathbf{q}\left(\left(\omega^C + \Omega^C\right)\Delta t\right) \\ \omega^C + \Omega^C \end{pmatrix} \tag{3}$$

Measurement Model We consider first the projection of infinite points in a standard perspective camera. The camera orientation $\mathbf{q}^{WC}$ in the state vector defines the rotation matrix R^{CW}. So the coordinates of a point $\mathbf{y} = \begin{pmatrix} \theta & \phi \end{pmatrix}^\top$ on the unit sphere $\mathbf{m}$ expressed in frame C are:

$$\mathbf{m}^C = R^{CW} \begin{pmatrix} \cos\phi \sin\theta & -\sin\phi & \cos\phi\cos\theta \end{pmatrix}^\top \tag{4}$$

The image coordinates where the point is imaged are obtained applying the pinhole camera model to $\mathbf{m}^C$:

$$\mathbf{h}\left(\mathbf{m}^C\right) = \begin{pmatrix} u_u \\ v_u \end{pmatrix} = \begin{pmatrix} u_0 - \frac{f}{d_x}\frac{\mathbf{m}^C_x}{\mathbf{m}^C_z} \\ v_0 - \frac{f}{d_y}\frac{\mathbf{m}^C_y}{\mathbf{m}^C_z} \end{pmatrix} , \tag{5}$$

where u_0,v_0 define the camera's principal point, f is the focal length and d_x and d_y define the size of a pixel. Finally, a distortion model has to be applied to deal with real camera lenses. In this work we have used the standard two parameter distortion model from photogrammetry [17].

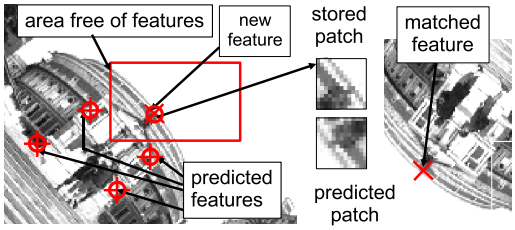

Figure 2: New features initialization and patch prediction.

4 Simultaneous Localization and Mapping

The algorithm to estimate camera rotation and scene point directions follows the standard EKF loop [1] of prediction based on the motion model (Equation 3), and measurement (Equation 5). All the estimated variables (the camera state $\mathbf{x}_v$ and all the estimated feature directions $\mathbf{y}_i, i = 1 \ldots, n$) are stacked in a single state vector $\mathbf{x} = \begin{pmatrix} \mathbf{x}_v^\top & \mathbf{y}_1^\top & \ldots & \mathbf{y}_n^\top \end{pmatrix}^\top$ with corresponding covariance $\mathbf{P}$ representing Gaussian-distributed uncertainty. Crucial to the method is the usage of the measurement prediction to actively guide matching. Next we explain in detail the matching process, the initialization of system state and feature initialization and deletion.

Matching Every predicted measurement of a feature in the map, $\hat{\mathbf{h}}_i$, and its corresponding innovation covariance, $\mathbf{S}_i$, define an gated elliptical acceptance image region where the feature should lie with high probability. In our experiments, defining acceptance regions at 95% probability typically produce ellipses 10–20 pixels in size.

The first time a feature is observed, we store both a texture patch and the current camera orientation. When that feature is later selected for measurement after camera movement, its predicted texture patch from the current camera orientation is synthesized via a warping of the original patch. This permits efficient matching of features from any camera orientation without the need for invariant feature descriptors. Figure 2 shows an example of the stored and predicted patches.

Search for a feature during re-measurement is carried out by calculating a normalized correlation score for every possible patch position lying within the search region. The position with the highest score, providing the match score is above a threshold, is considered to be matching measurement $\mathbf{h}_i$.

State Initialization We initialize the state of the camera with zero rotation — its initial pose defines the world coordinate frame, and therefore we also assign zero uncertainty to initial orientation in the covariance matrix. The angular velocity estimate is also initialized at zero, but a high value is assigned to σ_Ω, in our case $\sqrt{2}\ \frac{\text{rad}}{\text{sec}}$, in order to deal with an initial unknown velocity. This is a remarkable system characteristic: the map can be initialized from a camera which is already rotating. In fact in the experiments, the camera was already rotating when tracking commenced.

Feature Initialization and Deletion When a new image is obtained, if the number of features predicted to be visible inside the image goes below a threshold, in our case around 15, a new feature is initialized. A rectangular area without features is selected randomly in the image, and searched for a single salient point by applying the Harris detector [12]. Figure 2 shows an initialization example.

This simple rule means that at the start of tracking the field of view is quickly populated with features which tend to be well-spaced and covering the image. As the camera rotates and some features go out of view, it will then demand that new features are initialized — but if regions of the viewsphere are revisited, new features will not be added to the map since old features (still held in the state vector) are simply re-observed.

When an initial measurement of a new feature is obtained, the state vector is expanded with the new feature estimate $\hat{\mathbf{y}}_j$. Defining $\mathbf{R}_j$ the image measurement noise covariance, the covariance matrix is expanded as follows:

$$\mathbf{P}_{new} = \mathbf{J} \begin{pmatrix} \mathbf{P} & 0 \\ 0 & \mathbf{R}_j \end{pmatrix} \mathbf{J}^\top$$

$$\mathbf{J} = \begin{pmatrix} I & 0 \\ \mathbf{J}_1 \end{pmatrix}, \quad \mathbf{J}_1 = \left(\frac{\partial \mathbf{h}_i^{-1}}{\partial \mathbf{x}_v}, 0, \ldots, \frac{\partial \mathbf{h}_i^{-1}}{\partial \mathbf{z}_i} \right)$$

Features with a low successes/attempts ratio in matching — in practice 0.5 — are deleted from the map if at least 10 matches have been attempted. This simple map maintenance mechanism allows deletion of non-trackable features — for example those detected on moving objects or people. Non persistent static scene features (for instance caused by reflections) are also removed.

5 Meshing and Mosaicing

At any step k we have available a map of infinite points:

$$\mathcal{Y}^k = \{\mathbf{y}_i\}, \quad i = 1 \ldots n_k \ . \tag{6}$$

After processing every image, the location estimate of *every* point in the map is updated and hence changed. Additionally, on each step some map points might be deleted or added as new.

Input data: $\mathcal{M}^{k-1}$, $\mathcal{Y}^k$, image at step k.

Output data: $\mathcal{M}^k$.

Algorithm:

1.- $\mathcal{D}^k = \{D_l^k\}$, Spherical Delaunay triangulation.

2.- $\mathcal{M}^k$ is a subset of $\mathcal{D}^k$. Every D_l^k is classified as:

Elastically updated triangle: D_l^k already in $\mathcal{M}^{k-1}$, but not in $\mathcal{M}^k$. If visible in current image, the tile texture can be updated.

New and visible: D_l^k not in $\mathcal{M}^{k-1}$ and the visible in image k. D_l^k is included in $\mathcal{M}^k$. Texture is gathered from image k.

New but not visible: D_l^k not in $\mathcal{M}^{k-1}$ but not fully visible in image k. It is not added to $\mathcal{M}^k$.

Figure 3: Triangular mosaic update algorithm.

The mosaics we build are made up of a set of textured elastic triangular tiles attached to the map of infinite points. A mesh triangle T_j is defined as:

$$T_j = \{j_1, j_2, j_3, \mathrm{TX}_j\} \ , \tag{7}$$

where $\{j_1, j_2, j_3, \}$ identify the map points to which the triangle is attached and TX_j defines the triangle texture. The triangle is termed as *elastic* because the location estimates of the points to which it is attached are updated at every step, and hence the triangle and texture are deformed accordingly.

The triangular mesh mosaic at step k is defined as:

$$\mathcal{M}^k = \{T_j^k\}, \quad j = 1 \dots m_k \ . \tag{8}$$

5.1 Updating the Mesh

As the map is updated at every step, the mosaic has to be updated sequentially as well. The mosaic update consists of updating the elastic triangles, and potential deletion and creation of triangles.

Figure 3 summarizes the algorithm to sequentially update the mosaic $\mathcal{M}^{k-1}$ into $\mathcal{M}^k$. After processing image k, the map $\mathcal{Y}^k$ is available. A spherical Delaunay triangulation $\mathcal{D}^k$ for the map $\mathcal{Y}^k$ points is computed using Renka's [21] algorithm:

$$\mathcal{D}^k = \{D_l^k\} \ , \tag{9}$$

where every triangle $D_l^k = \{l_1, l_2, l_3\}$ is defined by the corresponding 3 map features. The complexity of the triangulation is $O(n_k \log n_k)$ where n_k is the number of map features.

The triangles which will be included in $\mathcal{M}^k$ are a subset of the triangles in the full triangulation $\mathcal{D}^k$. Every triangle D_l^k in $\mathcal{D}^k$ is classified to determine its inclusion in $\mathcal{M}^k$ mosaic according to the algorithm described in Figure 3: triangles are either carried over from the previous mosaic or newly created if texture can be immediately captured from the current image. Notice that triangles in $\mathcal{M}^{k-1}$ but not in D_l^k (due to a change in mesh topology) are deleted. Figure 4 illustrates with an example the different cases in the triangular mesh mosaic update.

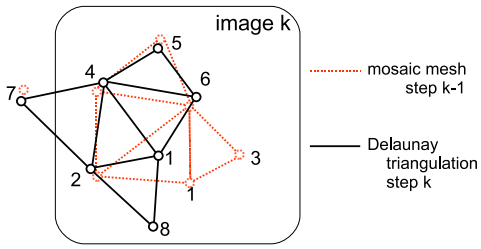

Figure 4: Mesh update example. Triangle {456} is elastically updated. {128} new and visible: created because of new map point 8. {247} new not visible; the triangle was not in $\mathcal{M}^{k-1}$, but it is not totally visible in image k. {136} is deleted as map point 3 is removed. Points 1, 2, 4 and 6 are kept in the map but a significant change in the estimated position of point 1 has caused the triangulation to 'flip', so, {126} , {246} are deleted, while {124} , {146} are new and visible.

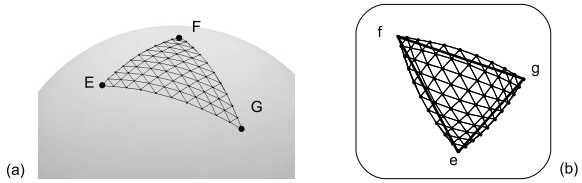

Figure 5: (a) Triangular tile defined by map points E,F,G, meshed as subtriangles represented over the unit sphere. (b) Shows the backprojection in the image; notice how the subtriangles also compensate the radial distortion.

5.2 Tile Texture Mapping

The three points defining a triangular tile are positions on a unit sphere. The texture to attach to each tile is taken from the region in the image between projections of these three vertices. In a simple approach to mosaic building, the triangles could be approximated as planar, and the mosaic surface a rough polyhedral. However better results can be achieved if the triangular tile is subdivided into smaller triangles that are backprojected over the spherical mosaic surface. Additionally,

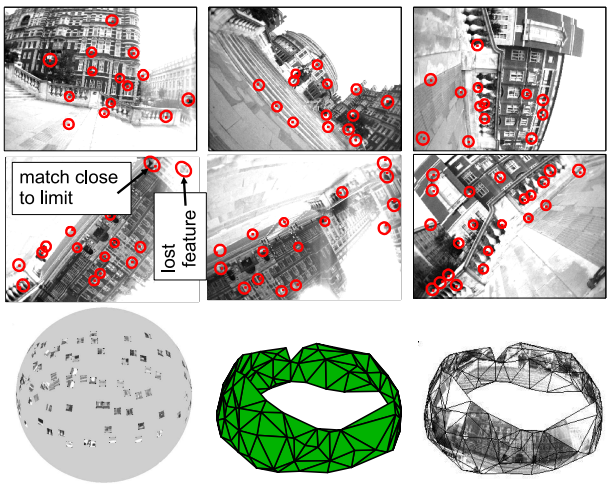

Figure 6: 360° pan and cyclotorsion. Top left: first image in sequence. The first image on the second row is at the loop closure point; notice the challenging illumination conditions. Third row: unit sphere with feature patches, triangular mesh and simple texture.

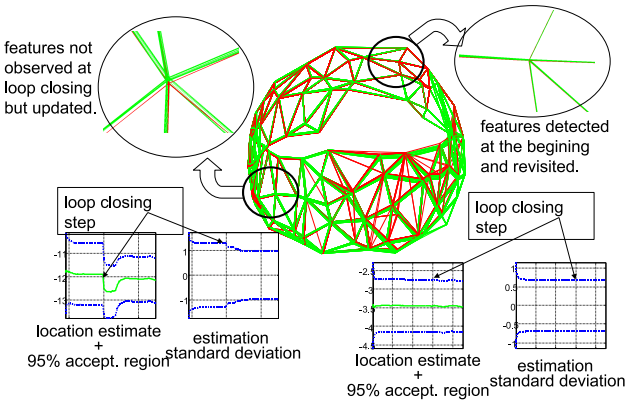

Figure 7: Superimposed meshes for 90 frames before loop closure (red) and 90 frames after loop closure (green). The part of the mesh opposite the loop closure is magnified at the left of the figure. The first features detected in the map are magnified at the right of the figure. We also show the camera elevation estimation history along with its standard deviation; notice the uncertainty reduction at loop closure.

the camera radial distortion is better compensated by the subtriangles. Figure 5 illustrates the improvement due to the division of a triangular tile into subtriangles.

6 Experimental Results

We present experiments demonstrating sequential mosaic building using images acquired with a low cost Unibrain IEEE1394 camera with a 90° field of view and 320×240 monochrome resolution at 30 fps. The first experiment shows the result from a sequence taken from a hand-held camera performing a 360° pan and cyclotorsion rotation — this early experiment performed offline in a Matlab implementation. The second experiment shows results obtained in real-time in a full C++ implementation with the more sophisticated sub-triangle mosaicing method. Both sequences were challenging because there were pedestrians walking around, and the camera's automatic exposure control introduced a great deal of change in the image contrast and brightness in response to the natural illumination conditions.

6.1 360° Pan and Cyclotorsion

The hand-held camera was turned right around about 1.5 times about a vertical axis, so that the originally-viewed part of the scene came back into view in 'loop closure'. Care was taken to ensure small translation, but relatively large angular accelerations were permitted and the camera rotated significantly about both pan and cyclotorsion axes.

Figure 6 shows selected frames showing the search region for every predicted feature and the matched observations. The frames we show are the at the beginning of the sequence, at loop closure and during the camera's second lap. At the loop closure, we can see that of the first two re-visited features, one was not matched immediately and the other was detected very close to the limit of its search region. However, in the following frames most of the re-visited features were correctly matched despite the challenging illumination changes. It should be noticed that the loop is closed seamlessly by the normal sequential prediction-match-update process, without need any additional steps — no back-tracking or optimization. The mosaic after processing all the images is also shown.

We believe that the seamless way that loop closing is achieved is in itself a very strong indication of the achieved angular estimation accuracy with this algorithm. The predictions of feature positions just before redetection at loop closure only differ from their true values by around 6 pixels (corresponding to less than 2 degrees) when the camera has moved through a full 360°. After loop closing and the correction this forces on the map, this error is significantly improved. On the second lap, where the rotation of the camera takes it past parts of the scene already mapped, previously observed features are effortlessly matched in their predicted positions, the map having settled into a satisfyingly consistent state.

It is worth noting the effect that loop closing has on the mesh, and hence on the mosaic. Figure 7 displays superimposed all of the meshes for the 90 steps before the loop closure (we plot one in every five steps in red), along with meshes for the 90 steps after the

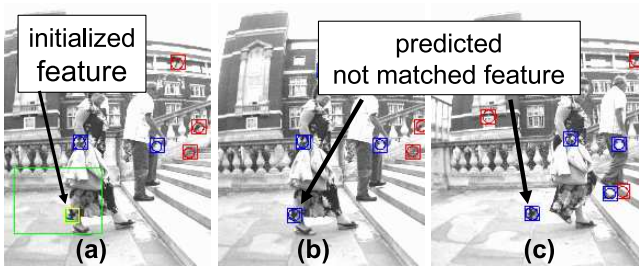

Figure 9: (a) Feature initalized on a moving object. (b) after 1 and (c) 10 frames, the feature is no longer matched because it is outside its acceptance region. Eventually this non matching feature is deleted from the map.

loop closure (plotted in green). Every sequential step improves our estimates of the locations of all the map features, but in a loop closure step the improvement is much greater. As the initial camera orientation is chosen to be the base reference, the first features in the map are very accurately located (with respect to the first camera frame). These features' locations are hardly modified by the loop closure, but all other parts of the mesh are updated significantly. We particularly draw attention to an area of the mesh 180° opposite the loop closure point, where the feature estimates and therefore the mesh are noticeably modified in the update corresponding to loop closure thanks to the correlation information held in covariance matrix, despite the fact that these features are not observed in any nearby time-step.

Figure 7 also shows graphs of the estimate history for the two magnified features, along with the standard deviation in their elevation angles. The feature far from the loop close area clearly shows a loop closing correction and covariance reduction. The feature observed at the start of the sequence shows almost no loop closing effect because it was observed when the camera location had low uncertainty.

6.2 Real-Time Mosaic Building

A version of the system has been implemented in C++ achieving real time performance at 320×240 pixels resolution, 30 frames/second. We show results from a 360° pan rotation. The system initializes a new feature when less than 15 map features are visible in the image to keep the map size around a hundred features.

Figure 8 shows the evolution of the mosaic. We focus on the texture alignment at loop closure. Figures 8(g) and 8(i) display two magnified mosaic views close to the loop closure. In each magnified view, the left-hand part of the mosaic seen got its texture from a frame at the beginning of the sequence while the right area got texture from a frame after the loop closure (frames nearly a thousand images apart). The excellent texture alignment achieved is an indicator of the advantages of our sequential SLAM approach to mosaic building. No blending technique has been applied to reduce the seam effects.

A movie showing the sequential mosaicing is available at: `http://webdiis.unizar.es/~josemari/ijcv.avi`. The video shows how the system robustly deals with moving people and cars. Figure 9 shows an example of robustness with respect to moving objects. A feature was intialized on the skirt of a walking pedestrian. As the feature corresponds to a moving object, after some time it is no longer matched *inside* the acceptance region, and finally it is deleted from the map.

6.3 Processing Time

Real-time experiments were run on a 1.8 GHz Pentium M laptop with OpenGL accelerated graphics card. In a typical run, we might have: a) 70 map features, implying a state vector dimension of $7 + 70 \times 2 = 147$. b) 15 features measured per frame, implying a measurement vector dimension of $15 \times 2 = 30$. c) 30 fps, so 33.3 ms available for processing. d) Process noise with standard deviation $4 rads^{-2}$ modeling expected angular accelerations.

Under these conditions, an approximate breakdown of typical processing time 21.5ms per frame is as follows: a)Image acquisition 1 ms. b) EKF prediction 3 ms. c) Image matching 2 ms. d) EKF update 10 ms. e), f) Delaunay triangulation 0.5 ms. g) Mosaic update 5ms.

The remaining time processing is used for the graphics functions, scheduled at low priority, so a graphics refresh might take place every two or three processed images.

It should be noticed that the computational complexity of the EKF updates is of order $O(N^2)$, where N is the number of map features. The Delaunay triangulation step has complexity of order $O(N \log N)$, while updating the triangles of the mosaic currently has order $O(N^2)$ (since each triangle is compared with every other one, though it should be possible to improve this). In our algorithm the EKF update dominates the computational cost. We have shown that within the bounds of a spherical mosaicing problem (where the whole viewsphere can be mapped with on the order of 100 features) the complexity is well within practical limits.

7 Conclusions

A mosaic built from an elastic triangular mesh sequentially built over a EKF SLAM map of points at infinity inherits the advantages of the sequential SLAM

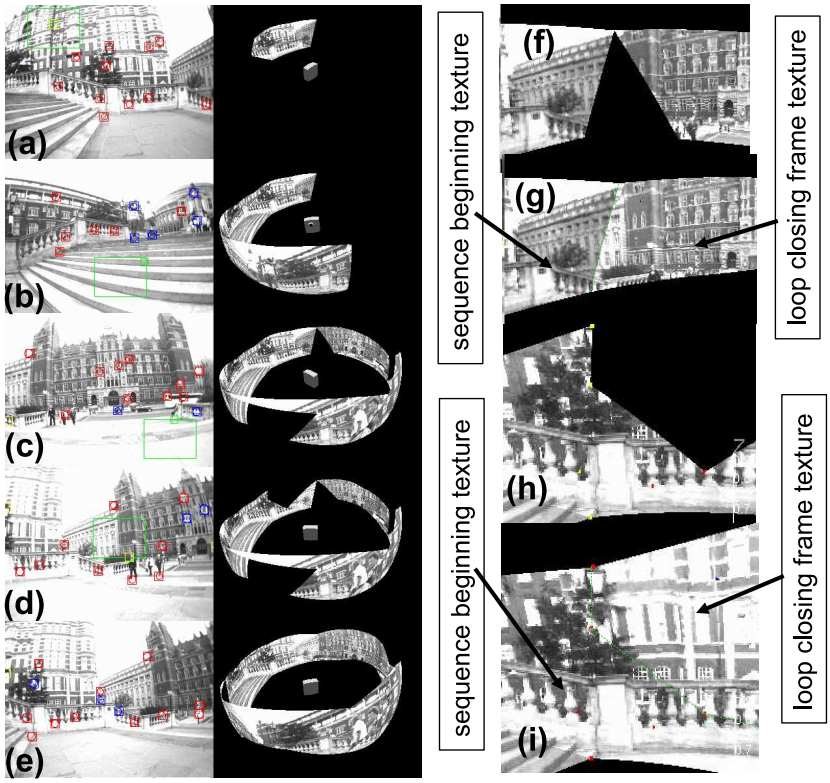

Figure 8: Real-time mosaicing with loop closure. (a), (b) sequence start; (c) the frame after the first loop closing match; (d) a few frames after loop closing (e) the mosaic after almost two full laps. (f) (magnification of (c)) and (g) show two consecutive steps after loop closing; (g) is a close-up view of two adjacent mosaic areas. In (g) the left-most mosaic tiles got their texture early in the sequence, while those on the right obtained texture after loop closing. Notice the high accuracy in texture alignment; the line highlights the seam. (h) (magnification of (d)) and (i) show two consecutive steps after several loop closing matches. (i) is a close-up view of two adjacent mosaic areas with textures taken from early and loop closing frames; again notice the alignment quality. A line highlights the seam —otherwise difficult to observe.

approach: probabilistic prior knowledge management through the sequence, sequential updating, real-time performance and loop closing.

The experimental results presented using real images from a low cost camera display the validity of the approach in a challenging real scene with jittery hand-held camera movement, moving people and changing illumination conditions. Real-time seamless loop closing is demonstrated, removing all the drift from rotation estimation and allowing arbitrarily long sequences of rotations to be stitched into mosaics: the camera could rotate all day and estimates would not drift from the original coordinate frame as long as the high-quality features of the persistent map could still be observed. We think that as well as its direct application to mosaicing, this work shows in general the power of the SLAM framework for processing image sequences and its ability, when compared with off-line methods, to efficiently extract important information: pure sequential processing, long track matches, and loop closing matches.

Our paper concerns mosaicing, and is intended to offer a contribution when compared with other mosaicing papers. Full 3D mosaicing in real-time is still some way off, so we have focused on building 2D mosaics from sequences with homography geometry — in our paper, specifically from a purely rotating camera, though we believe that our method could be straightforwardly modified to also cope with mosaicing a plane observed by a rotating and translating camera.

We believe that there are many applications which open up with real-time mosaicing — in any situation where the goal is to build a living mosaic which is built up instantly in reaction to camera motion our approach will be useful. This mosaic can find application especially in augmented reality because it provides a real-time link between the camera images and real scene points. Another interesting possibility real-time mosaicing gives is that a user could control the rotation of the camera while looking at the growing mosaic in order to extend and improve it actively. In future work, we intend to look at such issues as real-time super-resolution,

where we envisage a mosaic sharpening before a user's eyes thanks to the quality of repeatable rotation registration.

Acknowledgements

Research supported by Spanish CICYT. DPI2006-13578, EPSRC GR/T24685, an EPSRC Advanced Research Fellowship to AJD, and a Royal Society International Joint Project between the University of Oxford, University of Zaragoza and Imperial College London.

We are very grateful to David Murray, Ian Reid and other members of Oxford's Active Vision Laboratory for discussions and software collaboration.

References

[1] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association*, volume 179 of *Mathematics in Science and Engineering*. Academic Press, INC., San Diego, 1988. 5

[2] M. Brown and D. Lowe. Recognising panoramas. In *ICCV*, pages 1218–1225, Nice, 2003. 2

[3] D. Capel and A. Zisserman. Automated mosaicing with super-resolution zoom. In *CVPR*, pages 885–891, Jun 1998. 2

[4] J. A. Castellanos, J. M. M. Montiel, J. Neira, and J. D. Tardós. Sensor influence in the performance of simultaneous mobile robot localization and map building. In *LNCS. Vol 250*, pages 287 – 296. 1994. 3

[5] J. A. Castellanos and J. D. Tardós. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*. Kluwer Academic Publishers, Boston. USA, 1999. 3

[6] J. Civera, A. J. Davison, and J. M. M. Montiel. Inverse depth parametrization for monocular SLAM. *IEEE Trans. on Robotics and Automation*, Accepted. 3

[7] A. J. Davison. Real-time simultaneous localization and mapping with a single camera. In *ICCV*, 2003. 3

[8] A. J. Davison and D. W. Murray. Mobile robot localization using active vision. In *ICCV*, pages 809–825, Freiburg, Germany, 1998. 3

[9] L. de Agapito, E. Hayman, and I. A. Reid. Self-calibration of rotating and zooming cameras. *IJCV*, 45(2):107–127, Nov 2001. 2

[10] R. M. Eustice, H. Singh, J. J. Leonard, M. Walter, and R. Ballard. Visually navigating the RMS titanic with SLAM information filters. In *RSS*, 2005. 3

[11] H. Feder, J. Leonard, and C. Smith. Adaptive mobile robot navigation and mapping. *Int. Journal of Robotics Research*, 18(7):650–668, 1999. 3

[12] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, 1988. 5

[13] I. Jung and S. Lacroix. High resolution terrain mapping using low altitude aerial stereo imagery. In *ICCV, Nice*, 2003. 3

[14] D. Kim and K. Hong. Real-time mosaic using sequential graph. *Journal of Electronic Imaging*, 15(2):47–63, Apr-Jun 2006. 3

[15] J. H. Kim and S. Sukkarieh. Airborne simultaneous localisation and map building. In *ICRA*, pages 406–411, 2003. 3

[16] R. Marks, S. Rock, and M. Lee. Real-time video mosaicking of the ocean floor. *IEEE Journal of Oceanic Engineering*, 20(3):229–241, Jul 1995. 3

[17] E. M. Mikhail, J. S. Bethel, and J. C. McGlone. *Introduction to Modern Photogrammetry*. John Wiley & Sons, 2001. 4

[18] J. M. M. Montiel and A. J. Davison. A visual compass based on SLAM. In *ICRA*, pages 1917–1922, 2006. 1

[19] C. Morimoto and R. Chellappa. Fast 3D stabilization and mosaic construction. In *Proc. CVPR*, pages 660–665, 1997. 3

[20] D. Ortín, J. M. M. Montiel, and A. Zisserman. Automated multisensor polyhedral model acquisition. In *ICRA*, 2003. 3

[21] R. J. Renka. Algorithm 772: Stripack: Delaunay triangulation and voronoi diagram on the surface of a sphere. *ACM Transactions on Mathematical Software*, 23(3):416–434, 1997. 6

[22] H. Sawhney, S. Hsu, and R. Kumar. Robust video mosaicing through topology inference and local to global alignment. In *ECCV*, 1998. 2

[23] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *Int. J. Robotics Research*, 5(4):56–68, 1986. 3

[24] D. Steedly, C. Pal, and R. Szeliski. Efficiently stitching large panoramas from video. In *ICCV*, 2005. 2

[25] R. Szeliski and H. Shum. Creating full view panoramic image mosaics and enviromental maps. In *Proc SIGGRAPH*, pages 251–258, 1997. 2

[26] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment – A modern synthesis. In W. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, LNCS, pages 298–375. Springer Verlag, 2000. 4

[27] Z. Zhu, G. Xu, E. M. Riseman, and A. R. Hanson. Fast construction of dynamic and Multi-Resolution 360° panoramas from video sequences. *Image and Vision Computing*, 24(1):13–26, Jan 2006. 3