A Framework For Evaluating Visual SLAM

Jan Funke
Jan.Funke@inf.tu-dresden.de
Tobias Pietzsch
Tobias.Pietzsch@inf.tu-dresden.de

Fakultät Informatik
Technische Universität Dresden
01062 Dresden, Germany

Abstract

Performance analysis in the field of camera-based simultaneous localisation and mapping (Visual SLAM, VSLAM) is still an unsolved problem. For VSLAM systems, there is a lack of generally accepted performance measures, test frameworks, and benchmark problems. Most researchers test by visually inspecting their systems on recorded image sequences, or measuring accuracy on simulated data of simplified point-cloud-like environments. Both approaches have their disadvantages. Recorded sequences lack ground truth. Simulations tend to oversimplify low-level aspects of the problem. In this paper, we propose to evaluate VSLAM systems on rendered image sequences. The intention is to move simulations towards more realistic conditions while still having ground truth. For this purpose, we provide a complete and extensible framework which addresses all aspects, from rendering to ground truth generation and automated evaluation. To illustrate the usefulness of this framework, we provide experimental results assessing the benefit of feature normal estimation and subpixel accurate matching on sequences with and without motion blur.

1 Introduction

Building systems for camera-based simultaneous localisation and mapping (Visual SLAM, VSLAM) is a challenging task. A complete implementation of VSLAM has to deal with a wide range of issues ranging from low-level image processing, to high-level decision-making about when and where to remove old or select new map features. Real-time constraints and noisy input data raise further difficulties.

Given the complex nature of the implementations and the inherent dependency of the input data, the performance analysis of such systems is considerably hard as well. Some efforts have been made to analyse VSLAM systems using simulated environments. Most of these approaches abstract from the image processing layer which we belief has a great influence on the overall performance. In this paper, we propose a VSLAM evaluation framework which allows for performance tests on rendered image sequences. High-quality rendered images are close enough to real-world images to violate many of the assumptions usually made in the image processing part of VSLAM. This allows for a deeper study and evaluation of the whole VSLAM system from the lowest to the highest level.

We can broadly distinguish four kinds of approaches to evaluate VSLAM systems using experiments. In the first class of approaches, the SLAM system is run on recorded image sequences where ground truth is not available. Performance analysis is based on *visual*

inspection. Almost all recent work include an evaluation of this kind, e.g., [3, 5, 6, 8, 10, 13, 21]. Typically, these papers show screenshots of the system map/trajectory visualisation output, verbally explain what is going on at prototypical images in the sequence, and give interpretations of the visual results. The overall performance is usually described in broad categories such as whether a sequence is "well-tracked throughout" or "fails after x seconds", and whether loops are closed successfully. Using this approach, it is possible to illustrate strengths and weaknesses of a particular technique in the face of all the problems and artifacts occurring on real-world examples. However, it does not provide any hard quantitative results. Given the complex structure of the implementations, comparisons between different systems based on visual inspection alone are difficult. Usually, a lot of heuristics are involved in the implementation. It is often impossible to say what particular combination of components is responsible for, e.g., a successful or failed loop closure.

To obtain a quantitative measure of achieved reconstruction accuracy, some authors use *geometric constraints* in the scene. For instance, when the scene is known to be planar, the distance of estimated features to the plane (which is either known *a priori* [19] or an estimated best fit [10]) can be evaluated. Other constraints are right angles known *a priori* [19] and hand-measured distances in the scene [20]. In the latter case, inaccuracies cannot be avoided, of course. This second kind of approaches complements visual inspection by giving quantitative results at least for some restricted scene types.

Another solution is to record measurements made by the VSLAM system and compute a *maximum likelihood reconstruction* using bundle adjustment. The SLAM estimate can then be compared to this solution, e.g., [9]. This is well-suited for evaluating the performance of the filtering algorithm, for instance the influence of model linearisation. However, it abstracts from other influences such as feature matching accuracy, data association, and feature selection heuristics, that is, anything leading up to the 2D measurements.

Finally, VSLAM systems can be run on *simulated data*, and evaluated against ground truth (which is of course available in simulations). Abstraction from systematic influences at the image processing level can be a problem here, too. Often simulated scenes are modelled as point clouds and measurements obtained as projections of these points with additive Gaussian noise, e.g., [2, 4, 5]. Besides making idealised assumptions about feature matching, another drawback in [4] is that map features are predefined, i.e., the VSLAM system has no control of when and where to initialise features.

In this paper, we propose to use rendered image sequences to improve upon point-cloud simulation. Many effects occurring in real sequences can be emulated in rendered images, like non-Lambertian surfaces, changing illumination, motion blur, shading, and sensor noise. In this way, simulation conditions can be moved much closer to the realism of recorded real imagery. In other areas of computer vision (e.g., optical flow estimation) rendered images are part of the standard benchmarks [1].

In two recent papers, VSLAM results were evaluated on rendered images. Chekhlov *et al*. [3] use a camera trajectory in front of a texture-mapped plane to analyse the performance of their exemplar based feature descriptor. Klein and Murray [12] test their system on a synthetic scene produced in a 3D rendering package. They compare their results and those of an EKF-based system with respect to ground truth. However, this comparison is limited to the estimated trajectories. The estimated maps are only compared by visual inspection.

In contrast to these authors, we generate ground truth feature positions for the rendered sequences. From this ground truth, we can also compute the ideal measurements, a reference against which the results of feature matching may be compared. Additionally, we offer support for arbitrarily complex scenes and trajectories, different camera and image degra-

dation models, automated rendering and evaluation – all in one integrated and extensible framework. A main goal in designing our framework was to make it easy to add new scenes and evaluation code, such that it can be readily used and extended by others for their own research. It is publicly available [11], and completely build on free software.

The framework may be used to obtain answers about single components of a VSLAM system, e.g., "What is the influence of different feature removal heuristics to trajectory and map accuracy?", "Which feature matching method minimizes the measurement errors?". In addition, the availability of the ground truth may be used for quick hypothesis checking, e.g., "What would be the benefit of accurate estimation of feature normals?"

After describing the framework in detail in Section 2, we exemplify the latter two of these use cases are Section 3. A discussion of our results and an outline of future work concludes the paper in Section 4.

2 VSLAM Evaluation Framework

The motivation for building the framework presented in this section was the possibility to easily set up and perform VSLAM experiments on rendered image sequences. As we are convinced that such a framework is of interest to other researchers as well, we focused on the following requirements. The framework should be freely available, that means, built on free software only. It should be easy to use, that is, its adaptation to existing VSLAM implementations should be straightforward. Moreover, the framework should be extensible to encourage contributions from the community.

For image generation, we use the POV-Ray command line ray tracer [17], which is free software and allows rendering of realistic images. More importantly, POV-Ray allows to calculate the intersection point of a ray with the whole scene. This feature is used for ground truth generation, as described in Section 2.4.1. Moreover, a range of high detail scene descriptions is freely available for this renderer.

A large part of our framework deals with automatizing the processes of creating image sequences and evaluating experiment results. For the interface between the evaluation framework and the VSLAM system, we provide a lightweight API in C++ that allows access to the created sequence images and straightforward data logging of the VSLAM process.

For system integration and easy extendability, all relevant data throughout the framework is stored in an XML format. This includes camera and trajectory definitions, log data, ground truth, and intermediate evaluation results. The log data is processed using an extensible set of Python scripts.

2.1 Overview

Our evaluation framework consists of four components: a *repository*, a *rendering system*, an *interface* for VSLAM systems and an *evaluation system* (see Figure 1). The *repository* is used to store common items such as the descriptions of 3D scenes, camera trajectories and camera definitions which are needed by both the *rendering system* and the *evaluation system*. The *rendering system* is used to create image sequences from a specific setup of *repository* items. Using the *interface* (the lightweight C++-API), each of these sequences can be used instead of a regular camera (mono or stereo) to perform experiments on the VSLAM system under consideration. This *interface* can also be used to easily create structured log data during the experiment which are required for the *evaluation system* afterwards. For example,

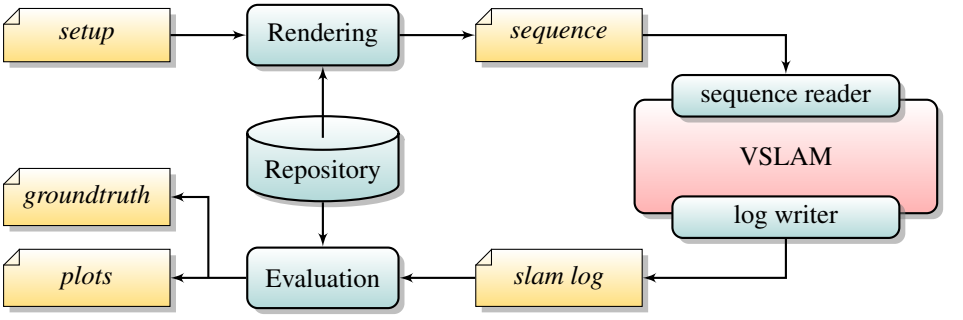

Figure 1: Overview of the evaluation framework. An image sequence is rendered and provided as input to the VSLAM system under evaluation. A log of the systems operation is created and processed afterwards by the *evaluation system*. Together with the exactly known scene structure and trajectory, the log is used to generate ground truth for feature positions and measurements. The final result is a number of error plots comparing logged estimated values and ground truth. Interface classes *sequence reader* and *log writer* are provided to allow easy adaption to existing systems.

the log data includes estimated location of camera and/or map features, 2D image measurements, and feature initialisation events. The *evaluation system* consists of an extensible set of Python scripts that perform various evaluation tasks, e.g., the computation of ground truth and the creation of data plots for the experiments.

2.2 Rendering System

The input to the rendering system is a *setup* description. Every *setup* is used to generate one *sequence*.

A *setup* selects a combination of the following repository items: a *scene*, a *camera configuration*, a camera *trajectory*, and *rendering options*.

The *scene* defines the 3D environment in which the camera will be placed. We use the POV-Ray [7] raytracer for image generation, and so the *scene* description is in the native POV-Ray format. All other repository items are simple XML files.

The *camera configuration* consists of camera type (mono/stereo), internal calibration parameters, image resolution, shutter speed, and frame rate. The *trajectory* describes camera motion as a list of keyframes. Every keyframe comprises a timestamp and a camera pose (rotation and translation). The *rendering options* influence the quality of the raytraced images. For instance, there are options to control the quality of anti-aliasing, lighting, and motion blur calculation.

Using the setup information, the camera is moved along the trajectory and a image sequence is rendered. Begin and end of the sequence are defined by the timestamps of the first and last trajectory keyframes. The number of frames to be rendered in-between depends on the framerate of the camera. In general, trajectory keyframes need not match the framerate of the camera or even be equidistant in time. Camera poses for frames between keyframes are interpolated between keyframe poses. Camera positions are linearly interpolated, spherical linear interpolation is used for the camera rotation. Being an advantage of this approach, the

trajectories can be obtained from a variety of sources, e.g., hand-crafted using some modeling program, recorded by an IMU, and so on. However, keyframes should not be too far apart in time, because otherwise the linear interpolation may cause discontinuities. Currently, the framework includes some synthetic trajectories and trajectories estimated from real image sequences using a VSLAM system [18].

After completing the setup step, the camera is placed at each interpolated frame pose, internal camera parameters are set as specified in the *camera configuration*, and an image is rendered using POV-Ray (two images in the stereo case). Motion blur effects are simulated by rendering and averaging multiple images from nearby poses. The amount of blur is controlled by the shutter speed, which defines the range of poses around the timestamp of the current frame. In addition to motion blur, the only image degradation effect currently available is additive Gaussian noise. We have decided to add the noise on-the-fly in the *sequence reader* of the VSLAM API (see Section 2.3) because this allows to test the same setup with different noise levels without re-rendering the sequence again and again.

As the result of the steps described above, we obtain a set of images along the trajectory. In the case of a stereo camera, the left and right image of every stereo image pair are merged into a single output image. Finally, a *sequence* description file is created. A *sequence* consist of a list of timestamped images and a set of parameters that should be available to the VSLAM system (e.g., camera calibration settings). The images and parameters are accessible through the interface described in the next section.

2.3 VSLAM Interface

Interfacing the evaluation framework with existing VSLAM systems is made easy using a lightweight API in C++ consisting of two parts.

2.3.1 Sequence Reader

The *sequence reader* provides camera parameters and images of a *sequence*. It is supposed to replace the camera driver of the VSLAM system. The *sequence reader* is provided with an *experiment* description in XML, containing the name of the sequence to process.

Additionally, an *experiment* may contain arbitrary parameter settings which can be queried as key-value pairs. This should allow for a large number of experiments to be run in a batch (possibly in parallel). One possible scenario is to run a VSLAM system on a number of sequences with varying parameter settings and observing the influence on the overall result. Another attractive option is to compare the results of different systems on a benchmark set of sequences, identifying the advantages and drawbacks of the different approaches.

2.3.2 Log Writer

The *log writer* collects data from the VSLAM run and writes them into a set of *slam log* files. In particular, the *slam log* consists of the following data items (for every frame of the sequence):

- the estimated camera pose,
- the estimated map, i.e., all 3D feature positions,
- the measured 2D feature positions, and

- the 2D positions of newly initialised features.

Please note that the evaluation methods and measures currently implemented are targeted at point-feature-based systems, i.e., systems that maintain a representation of the map as a set of 3D point features. Every feature point is identified by unique feature-IDs, which appears in every measurement and position estimate.

Logging the initial image locations of new features is important for two reasons: Firstly, this data is used to obtain ground truth 3D positions for all features. Secondly, this data can be also be fed back into the *sequence reader* in another experiment. This way, we can force the VSLAM system to initialise the same features in every run, alleviating the comparison of the resulting maps.

Currently, the log does not include uncertainty information, e.g., covariance matrices, (although it is quite easy to extend the log formats to do so.) Covariance information might not be available directly (for instance in a system using sub-maps, or a particle filter). If it is available, it might not be in the right parameter space (for instance in systems using inverse depth parameterisations). However, to compare different systems, uncertainty information should be represented in a unified way. This necessitates covariance projection and/or approximation prior to logging. Addressing the effects of such approximations correctly in evaluation measures is not straightforward. A general solution seems rather difficult here.

2.4 Evaluation System

The *evaluation system* consists of an extensible collection of *evaluation scripts*. Some scripts compute the ground truth required for the evaluation. Other scripts are used to analyse the outcome of the the experiments, as described below.

In the given framework, both types of scripts are handled in the same way: Each script may depend on certain files, e.g., *slam log* files, and produce one ore more files, e.g., plot data or ground truth. The output of one script may be the input to another, which results in a dependency graph for running these scripts. The framework takes care of resolving these dependencies, which allows easy integration of new scripts by stating their requirements and outcomes.

2.4.1 Calculating Ground Truth

Ground truth is determined as an intermediate step in the *evaluation system* by dedicated *evaluation scripts*. Ground truth is computed for each set of *slam log* files. It consists of the real trajectory, the map of real feature positions, and the set of ideal measurements.

The real trajectory is already given (it was used to render the sequence). From the *slam log*, the exact frame where each feature was initialised is known. Using this information, the camera pose for each feature initialisation can be determined straightforwardly. The ray through the image plane on which the feature was initialised is intersected with the (known) scene. This intersection yields the ground truth feature position.

Given the ground truth feature positions and camera pose, the projection of every feature point in every frame can eventually be computed. These projections are the *ideal measurements*, i.e., the best image measurements that feature matching could provide.

2.4.2 Evaluation Procedures

The other type of scripts use ground truth and estimated data to compute quantitative error measures. Currently, we provide basic evaluation measures for trajectory error, map error, and measurement error.

For the trajectory error, the estimated and ground truth camera poses are evaluated at every frame with respect to the Euclidean distance between camera centers, and the Frobenius norm of the difference of camera rotations. The result is plot data of translation error over time, and rotation error over time, respectively.

The map error for every frame is computed as the mean Euclidean distance of estimated and ground truth map features. The result is plot data of map error over time. Prior to evaluating map error, the estimated map can be translated, rotated, and scaled such that map error is minimized. The latter is especially interesting for monocular SLAM, as it does not penalise unobservable scale.

For each measurement, the image distance between the ideal and actually measured image projection is computed. The result can be used to generate scatter plots of measurements over all or individual features.

All these scripts store their results in plain GNUplot data files. The framework provides rudimentary support for generating various plots from these data.

3 Experimental Results

This section illustrates the evaluation of a complete VSLAM implementation on a usual test scenario.

The framework is used to generate two test sequences, using the same scene, camera, and trajectory but different motion blur settings.

The scene consists of the inside of a cube of dimension $6 \times 6 \times 6$ meters. The faces of the cube are texture-mapped with photographs of a cluttered indoor environment. We use an approximately circular loop trajectory with the camera facing roughly in the direction of motion. The trajectory was obtained by recording a real image sequence using a handheld stereo camera and reconstructing the camera path using the VSLAM implementation described in [18]. The camera is modeled after a Point Grey Bumblebee® stereo camera with 640×480 resolution and 65° horizontal field of view.

This setup is used with two different settings for motion blur. The first variant uses no motion blur at all. For the second variant, the shutter time is 0.03 seconds, resulting in quite severe blurring artifacts. Motion blur is generated using 10 camera pose samples per rendered image.

To collect experimental data, the VSLAM implementation described in [18] is used together with the C++ API of the framework. The approach of [18] is a fairly standard point-feature based SLAM system: The joint camera and map state is represented as multivariate Gaussian distribution, which is updated using an Extended Kalman Filter (EKF). The dynamics of the camera are modelled as a constant velocity motion, similar to [7]. The implementation makes use of up-to-date techniques such as a variant of inverse-depth feature parametrisation [15] and Joint Compatibility [16] testing for data association.

In the first experiment, we simply run the VSLAM system on both sequences (with and without motion blur). To get comparable data, we tweaked some parameters of the system such that the loop is closed in both cases and no gross errors occur. Results of this experiment

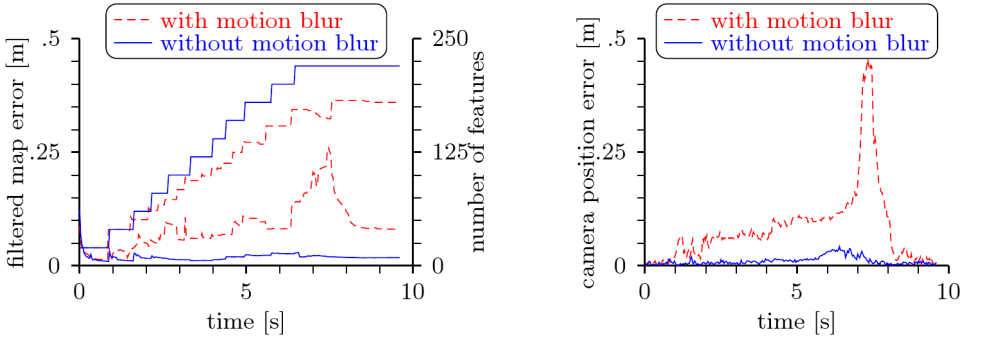

Figure 2: The two curves at the bottom of the plot on the left show the mean map feature error over time. Initialisation of feature depth from stereo matching can cause extreme outliers to enter the map. Usually, no further measurement attempts are made for these features and consequently they are never removed from the map. This behaviour is a weakness of the initialisation in [13]. Features with an individual error of more than 10 times the median error (at the current frame) are filtered to remove these outliers from the evaluation. The upper two curves show map size over time. The plot on the right shows the evolution of camera translation error over time.

are shown in Figure 2. The plots illustrate some of the information that is produced by the *evaluation system*, namely the evolution of map size, mean map error, and camera translation error. The corrupting influence of motion blur on the system performance is clearly visible. In the motion blurred sequence, it is pure luck indeed that the system achieves loop closure at all. Nevertheless, it is interesting to observe the huge reduction in translation error on loop closure.

One advantage of the proposed approach is that it provides access to the *ideal* measurements, i.e., the projections of feature centers in all images. This allows to analyse the distribution of overall or per-feature measurement errors. The distribution of measurement error over all features and all frames is compared for the blurred and non-blurred sequence in Figure 3(a) and 3(b). As we would expect, the measurement errors for the blurred sequence are larger. The distribution is elongated in the x direction, which is the main blur direction

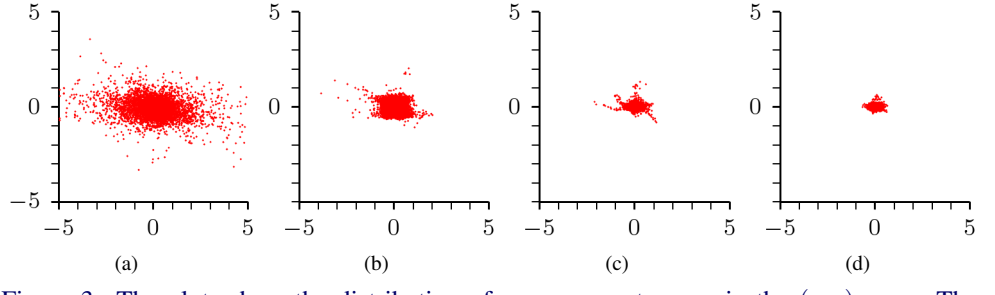

Figure 3: The plots show the distribution of measurement errors in the (u, v) axes. The motion blurred sequence was used for (a), all other plots were created using the non-blurred sequence. (a) and (b) used the pixel accurate measurement method, whereas (c) and (d) used sub-pixel accuracy. Additionally, for (d) perfectly known feature normals were assumed.

induced by the camera trajectory. Interestingly, in Figure 3(b) the error distribution is almost square-shaped. This can be attributed to the fact that measurements are only obtained with pixel accuracy.

For the third experiment, we have modified the measurement method of [18] to give subpixel-accurate feature matches. Feature measurements in [18] are obtained by finding the best match within an elliptic search region to a warped feature template. The warped template is obtained by applying a homography transformation to the initial feature patch. The homography is determined by the estimates of the camera pose, the camera pose at feature initialisation, and the normal vector of the feature patch (assumed to be facing the camera on feature initialisation). The best match is the pixel where the normalised sum of squared differences (NSSD) between the image and template is minimised. Because a stereo camera is used, we have to consider both images of the stereo pair, and the NSSD minimum is searched for in (u,v,d) space.

To refine the minimum localisation to subpixel accuracy, we fit a quadratic function to the NSSD errors in a $3\times3\times3$ neighbourhood around the best matching pixel. The minimum of the fitted function gives the subpixel accurate measurement. The resulting improvement of measurement accuracy is illustrated in Figure 3(c). For the subpixel experiment, the exact same features were used as in the previous experiment. This is achieved by feeding the log of initial 2D feature positions back into the system.

The final experiment is an example of quick hypothesis checking. Feature measurement error is in part also caused by badly warped templates due to errors in the assumed normal vectors of feature patches. We investigate how accurate normal estimation (as attempted by [14], for example) would improve measurement accuracy. To achieve this, we add an *evaluation script* which traces the ground truth feature normals. This information is fed back into the VSLAM system together with the initial 2D feature positions. Figure 3(d) illustrates the combined benefit of subpixel accurate measurements and known feature normals.

4 Conclusion

The presented approach to VSLAM evaluation falls within the category of simulation, with the obvious benefit of known ground truth. In contrast to point-cloud simulation, there are two advantages. Firstly, the VSLAM system is not restricted in its choice of map features because ground truth generation adapts to the systems decisions in this respect. This means that the effects of initialisation and map management heuristics are not ignored. Secondly, the VSLAM system operates on images instead of abstract 2D measurements. As we have shown in the previous section, this allows to analyse the effects of phenomena of the imaging process (such as motion blur) and compare implementation choices on the image processing level.

The capability of analysing individual parts of a VSLAM system within this framework was illustrated. To achieve this, however, it is our experience that careful design of experiments is crucial and in no way trivial.

A long-term goal of this work is to create standardised benchmarks upon which different systems can be compared. Ideally, the evaluation should boil down to a single measure of *map quality*. To achieve this, one important question is how to incorporate estimates of map uncertainty. This is a difficult problem, especially because maps differ in number and types of features.

Besides tackling this issue, there are a lot of technical improvements possible. On the

one hand, this involves adding more realism to the sequence generation, e.g., better image degradation models like vignetting, shading, Bayer-filter simulation, radial distortion, auto-focus simulation, *etc*. On the other hand, ground truth generation capabilities should be extended, e.g., for other feature types like planar or line features, and for dynamic scenes.

References

[1] Middlebury computer vision pages. URL http://vision.middlebury.edu/.

[2] K. E. Bekris, M. Glick, and L. E Kavraki. Evaluation of algorithms for bearing-only slam. In *ICRA 2006*, May 2006.

[3] Denis Chekhlov, Mark Pupilli, Walterio Mayol-Cuevas, and Andrew Calway. Robust Real-Time Visual SLAM Using Scale Prediction and Exemplar Based Feature Description. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2007.

[4] A. Chiuso, P. Favaro, H. Jin, and S. Soatto. Structure from Motion Causally Integrated Over Time. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4): 523–535, April 2002.

[5] Javier Civera, Andrew J. Davison, and J. M. M. Montiel. Inverse Depth to Depth Conversion for Monocular SLAM. In *ICRA 2007*, 2007.

[6] Laura A. Clemente, Andrew J. Davison, Ian Reid, José Neira, and Juan D. Tardós. Mapping Large Loops with a Single Hand-Held Camera. In *RSS*, 2007.

[7] A. J. Davison. Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *International Conference on Computer Vision*, 2003.

[8] E. Eade and T. Drummond. Scalable Monocular SLAM. In *IEEE Computer Vision and Pattern Recognition*, 2006.

[9] E. Eade and T. Drummond. Monocular SLAM as a Graph of Coalesced Observations. In *ICCV 2007*, October 2007.

[10] E. Eade and T. Drummond. Unified Loop Closing and Recovery for Real Time Monocular SLAM. In *BMVC 2008*, 2008.

[11] Jan Funke and Tobias Pietzsch. VSLAM evaluation framework. URL http://vslam.inf.tu-dresden.de.

[12] Georg Klein and David Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *ISMAR'07*, 2007.

[13] Georg Klein and David Murray. Improving the agility of keyframe-based slam. In *ECCV 2008*, volume 5303, pages 802–815, 2008. Springer.

[14] N. D. Molton, A. J. Davison, and I. D. Reid. Locally Planar Patch Features for Real-Time Structure from Motion. In *British Machine Vision Conference*, 2004.

[15] J.M.M. Montiel, J. Civera, and A. J. Davison. Unified inverse depth parameterization for monocular SLAM. In *Robotics: Science and Systems (RSS)*, 2006.

[16] J. Neira and J. D. Tardos. Data Association in Stochastic Mapping Using the Joint Compatibility Test. *IEEE TRA*, 17:890–897, 2001.

[17] Persistence of Vision Pty. Ltd. (2004). Persistence of Vision (TM) Raytracer. URL http://www.povray.org/.

[18] Tobias Pietzsch. Efficient Feature Parameterisation for Visual SLAM Using Inverse Depth Bundles. In *Proc. BMVC'08*, 2008.

[19] Paul Smith, Ian Reid, and Andrew Davison. Real-Time Monocular SLAM with Straight Lines. In *BMVC*, 2006.

[20] Joan Solà, André Monin, and Michel Devy. BiCamSLAM: Two times mono is more than stereo. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4795–4800. IEEE, 2007.

[21] Brian Williams, Georg Klein, and Ian Reid. Real-time SLAM Relocalisation. In *Proceedings of the International Conference on Computer Vision*, 2007.