Real-Time SLAM Relocalisation

Brian Williams, Georg Klein and Ian Reid
Department of Engineering Science, University of Oxford, UK
{bpw,gk,ian}@robots.ox.ac.uk

Abstract

Monocular SLAM has the potential to turn inexpensive cameras into powerful pose sensors for applications such as robotics and augmented reality. However, current implementations lack the robustness required to be useful outside laboratory conditions: blur, sudden motion and occlusion all cause tracking to fail and corrupt the map. Here we present a system which automatically detects and recovers from tracking failure while preserving map integrity. By extending recent advances in keypoint recognition the system can quickly resume tracking – i.e. within a single frame time of 33ms – using any of the features previously stored in the map. Extensive tests show that the system can reliably generate maps for long sequences even in the presence of frequent tracking failure.

1. Introduction

Real-time visual tracking can be used to estimate the 6-DOF pose of a camera relative to its surroundings. This is attractive for applications such as mobile robotics and Augmented Reality (AR) because cameras are small and self-contained and therefore easy to attach to autonomous robots or AR displays. Further, they are cheap, and are now often pre-integrated into mobile computing devices such as PDAs, phones and laptops.

Real-time camera pose tracking works best when some form of map or model of the environment to be tracked is already available. However, it is also possible to generate this map on the fly: in this context, the problem is known as Simultaneous Localisation and Mapping (SLAM) and a real-time (30fps) monocular SLAM implementation was first described by Davison (recently summarised in [3]).

The main problem with most existing monocular SLAM implementations is a lack of robustness. Typically tracking systems rely on a prior over current pose and this prior is used to limit the search for visual feature correspondences, yielding very rapid frame-to-frame localisation. However, rapid camera motions, occlusion, and motion blur (phenomena which are common in all but the most constrained experimental settings) violate the assumptions in the prior and therefore can often cause tracking to fail. While this is inconvenient with any tracking system, tracking failure is particularly problematic for SLAM systems: not only is camera pose lost, but the estimated map could become corrupted as well.

The alternative to "tracking" is repeated, data-driven detection of pose, potentially requiring no prior on the pose. This of course assumes that the map (or target structure) has already been acquired. [10] demonstrated real-time repeated detection of pose via a forest of fast classifiers used to establish correspondence between image and target features. Their tracker is robust to many of the failure modes of monoSLAM since pose is re-estimated from scratch at every frame. Their system requires the structure of the target and its appearance (encoded as point features with associated image patches) to be trained before localisation can occur. This training process is automatic, nevertheless their work falls short of being truly a *simultaneous* localisation and mapping system. Furthermore the smoothness and accuracy of tracking do not approach those of the state-of-the-art visual SLAM systems.

Our work combines these two ideas, using a motion prior when this is reliable, yielding smooth and very fast mapping, but resorting to data-driven localisation when tracking is detected to have failed. The SLAM system learns the structure of the environment while a novel extension of Lepetit's image patch classifier learns the appearance of image patches, used for re-localisation when tracking fails.

Our system is characterised by:

- real-time, high-accuracy localisation and mapping during tracking
- real-time (re-)localisation when when tracking fails
- on-line learning of image patch appearance so that no prior training or map structure is required and features are added and removed *during* operation.

resulting in a state-of-the-art SLAM system whose robustness greatly exceeds that of others (eg, [3]). We demonstrate this by operating the system reliably for significantly longer

periods and in larger environments than any previous work in real-time monoSLAM.

In order to achieve these gains we do not simply reproduce [10] in conjunction with SLAM (though the difficulty in doing so should not be underestimated). Rather we describe novel extensions to their approach to feature appearance learning that are crucial in reducing the learning and classification times, and the memory requirements without compromising the frame-rate operation of the underlying SLAM system or the ability of the classifier to differentiate between features. Further we show the classifier more closely integrated into the process of map-building, by using classification results to aid in the selection of new points to add to the map. We provide extensive results demonstrating the efficacy and reliability of the new system.

2. Related Work

Recovery from tracking failure is important not only for SLAM systems, and a number of different attempts at improving robustness have recently been presented. These range from very local methods which attempt to bridge tracking failure during a few frames to methods which attempt a global recovery.

Chekhlov *et al.* have built two systems which can bridge short tracking failures in a monocular SLAM context. This was first done by exploring multiple hypotheses with a particle filter [11] and later by using a rich feature descriptor to provide robust data association [2]. Both of these methods are local in that they rely on the point of recovery being close to the point of failure.

In a known-model-tracking context, Reitmayr [13] is able to recover from tracking failure using keyframes gathered previously during tracking. After failure, the current camera view is compared to a store of previously gathered key-frames and a close match provides an initial pose estimate from which tracking can recommence. This approach can recover from the proximity of a finite section of the previously traversed path, but also does not offer global relocalisation; by contrast, Rahimi [12] uses a larger number of key frames to close loops and prevent drift in structure-from-motion, but this technique is costly as the pose of each key-frame must be maintained in the estimation.

We desire a more global approach to relocalisation using the map created rather than just key frames. The closest approach to ours is [15] who achieves global relocalisation for a robot moving in 2D. They find matches to image features with map features using SIFT and find the pose using RANSAC or a hough transform.

This paper builds on our previos work [16]. The main improvement here is the feature recognition which is now faster, more invariant and supports on-line learning.

3. SLAM Implementation

The underlying SLAM system we use is based on Davison's monocular SLAM implementation "SceneLib" [3] with a number of improvements that make it close to the state-of-the-art for such systems. At the core is a probabilistic representation of the world map and the pose of the camera maintained by an Extended Kalman Filter (the distributions are thus assumed to be Gaussian). Like [3] we incorporate rotational (and a degree of perspective) invariance via local patch warping, but we make a more naive assumption that the patch is fronto-parallel when first seen. This gives sufficient invariance to all but extreme distortions.

It is vitally important to maintain the integrity of the map if the system is to use it for tracking and relocalisation. During normal tracking operation we use *active search* to establish image-to-map correspondences: an association for a map feature is only sought in the vicinity of where that feature is predicted to lie. The region to search is given by the innovation covariance of that feature; i.e. it takes into account the uncertainties in the map, the camera pose and the expected measurement uncertainty. Since our distributions are assumed Gaussian, a typical search window at 3σ will be elliptical in the image. Like [4], we confine the active search in the ellipses to the neighbourhood of corners produced by the "FAST" corner detector [14].

Though the search regions are in fact strongly correlated (through the camera pose uncertainty), each correspondence is treated independently. This can lead to mutually incompatible matches being incorporated by the filter, and consequent corruption of the map. To mitigate further against this we employ the joint compatibility test proposed by Neira and Tardós [8]. Given a set of putative matches between image observations and features in a correlated map, this test determines the maximal set of pairwise compatible matches using an interpretation tree search, thus allowing straightforward exclusion of incorrect matches.

The whole procedure – image acquisition, filter prediction, patch pre-warping, active search and joint compatibility, and filter update – typically takes 5ms (out of a total budget of 33ms for frame-rate operation) on a Core 2 Duo 2.7GHz machine for a map of 30 features.

In Davison's system, the remaining CPU cycles were used to search for and initialise new features. A window is placed in the image in a semi-random location (regions with few current features and which are "coming into view" are favoured). Corner locations are detected and the strongest in the window selected for addition to the map. We insert such features directly into our map parametrised by the visual direction and inverse depth, similar to [7] with a large uncertainty on the latter. However we do not simply take the strongest feature; rather we take advantage of the discriminatory powers of the randomised lists classifier to choose a feature unlike existing ones in the map (see Section 4.3).

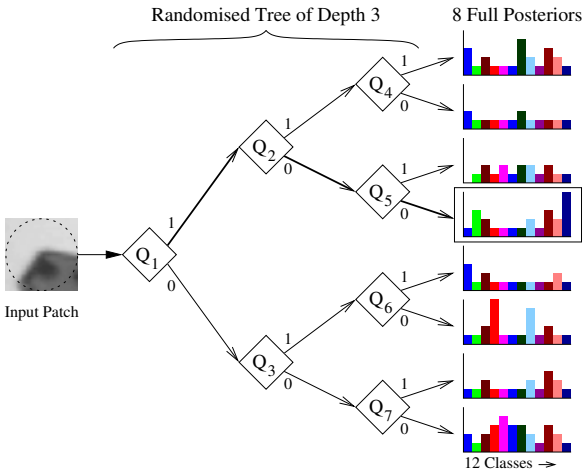

Figure 1: A Randomised Tree as described in [6]

The active search process is a major strength of this system, enabling real-time mapping and localisation within a probabilistic framework. However it is also a weakness, since tracking will fail as soon as the map features are not detected within the search regions. This regularly happens when there are sudden motions, or if motion blur means the image is temporarily unusable for point feature detection. Indeed it is important to detect such moments so that incorrect data associations are not inadvertently accepted and filtered, thus corrupting the map. In our system, tracking is deemed to have failed when all of the attempted observations in a frame have been unsuccessful, the camera pose uncertainty has grown too large, or if the predicted camera view does not contain any mapped features.

It is the recovery from such situations that is the main focus of this paper. The system aims to relocalise relative to the map it has created, but first it must detect some of these mapped features in the image. Active search can no longer be used since the camera pose is unknown. Instead, we search the entire image, seeking matches between features detected in the image and those stored in the map. To make this process fast and effective we use the randomised lists key-point recognition algorithm.

4. Feature Recognition with Randomised Lists

We now describe the algorithm for map feature recognition that is key to achieving rapid correspondence between the current image and the map. We begin by reviewing the work of [6], and then describe a number of modifications that yield a classifier better suited to our needs.

4.1. The implementation of Lepetit & Fua

Lepetit and Fua employ randomised trees of simple image tests rapidly to detect re-occurrences of previously trained image patches in a new input image. The system

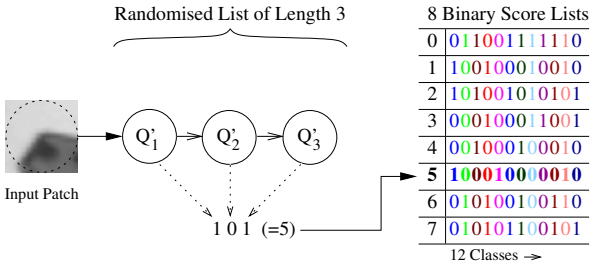

Figure 2: The simplified classifier structure used here

can detect re-occurrences even in the presence of image noise, changes in scale, rotations, aspect ratio and illumination changes. The novelty of the system is to treat real-time feature recognition as a classification problem, where each class is trained with (ideally) all the different appearances a single feature can take due to aforementioned changes. The classification approach allows the use of a simple classifier which can be evaluated quickly; a disadvantage is the requirement to train many instances of every class.

At the center of the system is a forest of N decision trees. A single simplified tree of depth D=3 is illustrated in Figure 1. Each tree maps an input image patch to one of 2^D posterior distributions (the leaves of the tree) which describe the probability of the patch belonging to one of C=12 previously trained feature classes.[1] The mapping is the result of a series of simple binary tests Q which make the tree's nodes: each test Q simply compares the patch's Gaussian-smoothed intensity values $\mathrm{I}_\sigma(\cdot)$ at two pixel locations $\boldsymbol{a}$ and $\boldsymbol{b}$, such that

$$Q_i = \begin{cases} 0 & \mathrm{I}_\sigma(\boldsymbol{a}_i) - \mathrm{I}_\sigma(\boldsymbol{b}_i) >= 0 \\ 1 & \text{otherwise} \end{cases} . \qquad (1)$$

Once the patch's posterior probability for every tree has been estimated, the class whose sum of posteriors is greatest forms the classification result (subject to a threshold.) While each individual tree may itself be a rather poor classifier, the combination of N trees' posteriors yields good recognition results. The posterior distributions themselves are built during a training phase; hundreds or thousands of patches belonging to each known class are simply dropped down each tree and the associated posterior is incremented. Training patches are either synthetically generated by warping ([6]) or harvested from a carefully controlled training video sequence ([10]) with the aim of teaching the trees all possible views of a class.

One of the key results in [6] concerns the selection of the locations $\{\boldsymbol{a}, \boldsymbol{b}\}$. During a training phase, these may be chosen from a small random set so as to maximise each test's information gain (in accordance with [1]); alternatively they can be chosen entirely randomly, which results in a small

[1] In an actual implementation, reasonable values for these parameters might be N=40, D=15 and C=100-1000.

drop in classification performance but massively reduces the time required for training.

4.2. Randomised Lists for SLAM recovery

Lepetit and Fua originally applied the classifier described above to the recognition and localisation of known textured objects (e.g., a book) for which the classifier could be trained once with hundreds of optimally chosen classes and then re-used many times. In our monocular SLAM setting, in which the goal to the build a map and localise simultaneously, a separate training phase would be a self-defeating exercise. This means that the binary tests Q_i must be pre-selected at random.

If the tests are random, it is not necessary for the tests at any one level of a tree to be different. Instead of a tree with $2^D - 1$ potential tests, a list of D sequential tests is sufficient, and produces identical classification performance (Indeed, Ozuysal *et al*. have independently developed this approach and further improvements in their most recent work [9]). The results of these tests form a binary string which indexes into the array of 2^D posteriors. Using the list structure allows for significant improvements in run-time speed (for example, we can replace patch rotation with a look-up-table of pre-rotated tests.)

Besides this structural change, we also make changes to the operating characteristics of the classifier; whereas the systems of Lepetit and Fua operate with many dozens or hundreds of keypoints visible per frame, our SLAM implementation uses a far sparser set of features: At any given frame, only $10 - 20$ might be visible. For this reason, it is important to tune the classifier towards recall rather than precision, and this motivates further changes. We also propose some tweaks to boost efficiency and classification rates. The modified classifier is illustrated in Figure 2 and the changes are described below.

Multiple Hypotheses and Class Independence - For an extended exploration, it is unrealistic to assume that all map features will appear completely unique. Due to perspective and lighting changes - and self-similarities common in man-made environments - some map features will resemble one another. The method of Lepetit and Fua penalises such cases: only the strongest match is returned, and since posteriors are normalised, the presence of multiple similar features penalises the score of all of them, sometimes to the extent that none is recognised. Here, we consider each class' score independently and the classifier returns all classes scoring higher than a threshold. The independent treatment of classes has the further benefit that the classification rate of any one class is not affected by the later addition of other classes, which is very important for a system which is continually trained on-line.

Binary Leaf Scores - Instead of storing full normalised posterior distributions over the classes at each leaf, we store a binary score string of one bit per class. A class' score at a leaf is either zero (if no training example from that class ever found the leaf) or one (if at least one training example did.) The benefit of this approach is a reduction in storage requirements ($2^D \times C \times N$ bits for all scores) which allows operation with higher values of D.

Intensity Offset Tests - In man-made environments, image areas of uniform intensity are common. For regions of uniform color, tests of the form "is this pixel brighter than the other" only measure image noise. We replace tests Q_i with modified tests Q'_i:

$$Q'_i = \begin{cases} 0 & \mathrm{I}_\sigma(\boldsymbol{a}_i) - \mathrm{I}_\sigma(\boldsymbol{b}_i) >= z_i \\ 1 & \text{otherwise} \end{cases} \quad (2)$$

where z_i is a randomly chosen intensity offset in the range 0-20. These tests increase repeatability in areas of the patch with uniform intensity. The slight reduction in illumination invariance is in practice mitigated by feature harvesting.

Explicit noise handling during training - In [6], random noise is added to training examples to make the classifier noise-resistant. This is not done here; instead, during training, we explicitly check if each training test Q'_i is close to a noise threshold. If this is a case, the ith bit of the binary string being formed is set to a "don't care" state. When the full string is formed, the scores for *all* possible values of the string are updated for the class (this is made possible by using lists instead of trees.) This reduces the number of training patches required to achieve noise tolerance, which is important for real-time operation.

4.3. Training the Classifier

The classifier must be trained to recognise each new feature that is added to the map during "steady-state" SLAM operation. When the feature is first observed we train the classifier with 400 synthetically warped versions of the feature so that it can immediately be recognised under a variety of views. We use a GPU to perform the warps. Even so this initial training is costly at around 30ms per new feature; so that this does not adversely affect the real-time performance of the SLAM system, it is performed in a background thread. Further, we can in some cases reduce the number of warps performed: with our binary posterior representation, there is no benefit in incrementing a class posterior more than once, so we can stop training once a large fraction (95%) of warped examples re-visit previously seen posterior nodes.

This relatively crude initial training stage is subsequently reinforced using real data harvested [10] during normal SLAM operation: when a map feature is successfully observed by the SLAM system, the classifier is also trained with this new view of the feature.

To improve the distinctiveness of the features in the map, the classifier aids in the selection of new map features.

When the system is initialising a new map feature it is usually faced with a choice of potential features in the image. Rather than simply take the one with the strongest corner response as in [3], each candidate corner is instead first classified with the current randomised lists classifier. The system then chooses to initialise the feature that scores lowest against all other features already in the map. This leads to a map of features which are more easily discriminated.

5. Relocalisation Using the Randomised Lists Classifier

When the SLAM system has become lost, the randomised lists classifier is used on each new frame to generate a list of feature correspondence hypotheses detected in the image. We then apply the well-known method of Fischler and Bolles to relocalise the camera relative to the map using three feature correspondences and their three-point-pose algorithm [5]. Consensus for each calculated pose is evaluated by checking how well that pose predicts the location of the image projection for the other map features. When a good pose is found, it is optimized using the consensus set before the SLAM system is reinitialised with it. The consensus set is also used to derive an initial (artificially inflated) camera pose uncertainty. If the pose estimate is indeed close enough to the true estimate then one or two iterations of the EKF (with the map fixed to avoid corruption if the pose is not correct) are sufficient to refine the pose and to reduce the uncertainty dramatically.

To increase RANSAC performance, we make some effort to select sets of three matches in a way which increases the chances obtaining three inliers. This is done by assigning a probability to each feature correspondence returned by the classifier: these weights are determined by a combination of a motion model and a set of match scoring heuristics.

The motion model considered when the camera is lost is different from that used when tracking: instead of a constant velocity model with white noise acceleration, we consider a random walk with a maximal linear velocity of 1.5 m/s (walking speed) and an arbitrarily high rotational velocity. This predicts that the camera is within a steadily expanding sphere centered about its last know position and that it could be oriented in any direction. Feature correspondences returned by the classifier are filtered by potential visibility: features which could not be visible from any point in the sphere are given zero weight. Thus, our system switches seamlessly from an efficient local relocalisation immediately after tracking is lost, to a global kidnapped-robot relocalisation after a few seconds have elapsed.

We further apply a set of match scoring heuristics: matches with low classification scores are penalised, as are classes which occur many times in the image. We further reject match triplets which are colinear or very close together in the image as these produce poor pose estimates. We also check match triplets against a co-visibility database maintained by the SLAM system: sets of matches which have never been observed together in a single image are culled. This check prevents attempting to calculate a pose using three features from distant parts of the map which are unlikely all be correct. Together, these measures help to reduce the number of RANSAC trials required for successful re-initialisation.

6. Results

6.1. Classifier Implementation

This section describes the performance impact of the classifier design choices described in Section 4.2. Two tests were performed on a controlled 2300-frame SLAM run during which a map of 70 features was built without tracking failure. The tracked feature positions are considered ground truth for classifier evaluations at every frame. Classes were trained using synthetic warps only; no harvesting was performed since this would inflate the classifiers' performance under the slow-moving conditions tested. We set the operating point of the classifier to 0.65 recall (65% of the known features visible in the image are detected) and test how our changes affect the classifier's precision (the fraction of returned matches which are actually inliers.)

Leaf node type	Depth	Memory Use	Precision
Full Posterior	18	1321 MB	.12
Full Posterior	15	164 MB	.07
Binary Score	18	164 MB	.09

Table 1: Binary vs. full posterior classifier performance

Table 1 demonstrates the effect of using a binary leaf score list instead of a full posterior (with 8 bits per class in our implementation.) As may be expected, simplyfing the posterior to a binary score reduces classification performance for equal tree depths; however the binary score outperforms the full posterior when using the same memory footprint, since a larger depth can be used. Using the binary score further reduces classification times and eliminates an otherwise problematic tuning constant (the uniform prior distribution density.)

Intensity Offset Test	Don't-care-bit	Precision
No	No	.07
No	Yes	.03
Yes	No	.08
Yes	Yes	.09

Table 2: Performance impact of classifier modifications

Table 2 illustrates the effect of the other classifier adaptations on classification rates. The use of intensity offsets to-

gether with the explicit training of all image noise permutations produce a noticeable increase in performance at a negligible increase in classification time. It is interesting to note that the noise-aware training by itself (without the intensity-offset tests) *reduces* classification performance: this is due to the frequent occurrence of areas of uniform intensity in many of the patches used; the use of don't-care bits in these areas causes them to match *any* input image patch.

It should be noted that the low precision scores in above tables are are obtained without any feature harvesting. If this is enabled, precision increases from 0.09 to 0.2, and the actual value encountered during real-world relocalisation likely lies between these two. Finally, the heuristic guided sampling criteria described in Section 5 help boost inlier rates to a level normally manageable by RANSAC.

6.2. SLAM with Recovery

By recovering after tracking failure, our monoSLAM system is able to complete sequences which would be very challenging (or probably even impossible) for other similar systems. The system is easily able to detect when tracking fails and then stop the SLAM system to preserve map integrity. Tracking and mapping are only resumed when the system relocalises again using previously mapped features.

Given the ability to re-initialise rapidly and often, we can afford to tune our system to be much more cautious than previous monoSLAM systems. Thus there are times when tracking is stopped due to failed observations when dead reckoning according to the motion model might otherwise actually carry it through. However, the predicted camera poses in such cases are not likely to be accurate and as likely error grows, so do linearisation errors and the the possibility of poor data association. Our system simply restarts tracking and avoids these potential sources of map corruption.

To demonstrate our system's mapping and relocalisation operation we include a video sequence tracked on a laptop in an art gallery. The camera moves along two walls of paintings. Several times during this run, tracking is lost when the camera is suddenly pointed at the floor. The system is able to detect all of these occasions, correctly stops mapping, and does not attempt to resume mapping until the camera once again points at one of the walls which were previously mapped, whereupon relocalisation occurs within 1-2 frames. Figure 3 shows the map of 80 features and trajectory for the run. The first relocalisation is annotated in the map, numbered to match the images on the right.

To evaluate the reliability of the system, it was run for over an hour (>100,000 frames) in an open-plan office environment. The map generated contained 70 features. Despite erratic camera motion and hundreds of tracking failures, the map was not corrupted and was still consistent enough 30 minutes into the sequence to enable a panning loop closure. Although the volume of space traversed by the camera was small compared to the size of the environment, this is purely due to unresolved limitations in the underlying SLAM system (no support for occlusions and N^2 complexity in map size for the EKF.) Relocalisation was consistently successful until the arrival of co-workers changed the environment (our current map management policies do not scale well to sequences of this length.)

Corner detection (found 145 corners)	2 ms
Classification (found 42 matches)	12 ms
Selection of Match Triplets (chose 335)	0.3 ms
Evaluation of Poses (tried 14)	0.7 ms
Final Pose Optimisation	4 ms
Total	**19 ms**

Table 3: Typical relocalisation timings (map size 54)

Table 3 shows the time taken (on a Core 2 Duo 2.7GHz processor) by the individual stages of a typical relocalisation with 54 features in the map. Here the classifier has returned 42 matches of which 7 are inliers: RANSAC must therefore find one of the 35 inlier triplets out of 11480 possible match triplets. In this run, 335 triplets were hypothesised but most of these could be rejected before even attempting three-point pose by testing for co-visibility and image geometry; the three-point-pose algorithm was only run on 14 hypotheses. In this case, a correct pose was produced in a total of 19ms.

Further system timings are shown in Figure 4; this plots the processing time required per frame for another run of the art gallery sequence. Each frame (including those during which recovery is attempted) is processed in under 33ms; this is possible for maps up to $80-90$ features, beyond this the N^2 EKF updates cause skipped frames. Black timing values indicated frames during which the system was lost; of these, those frames with low run-times are those in which insufficient features were detected to attempt relocalisation.

Figure 5 investigates the volume of space in which relocalisation is possible. A small map was generated by moving the camera in a straight line away from an initialisation sheet (top row of images). Mapping and harvesting were then disabled, and the remaining images shows a range of novel views from which the system can correctly relocalise.

Even with a significant range over which recovery should be possible, there are still cases in which it can fail in practice. In order for the system to relocalise several conditions must be met: Mapped features must be visible in the image; the relative feature positions in 3D must be accurate; the corner detector must detect a corner at the feature location; finally, at least five correct matches to features must be found by the classifier (three for the pose and two for consensus). In practice, when recovery from a particular pose is not possible, moving the camera a small distance in any direction will often allow the system to succeed.

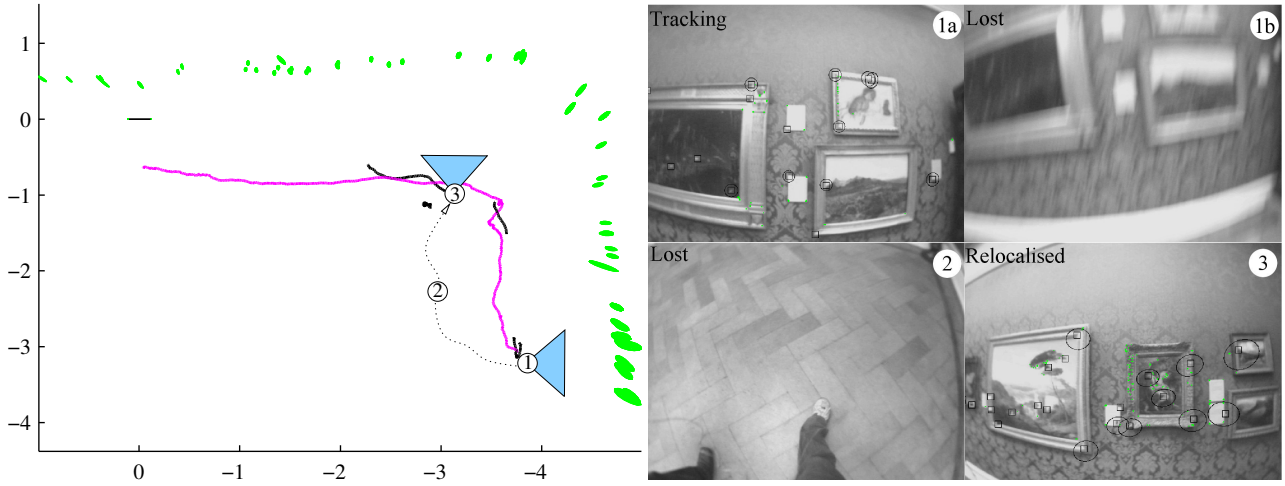

Figure 3: The system relocalises when it is moved to another region of the map. On the left is an overhead view of the final map (green 3σ ellipses) and trajectory estimate. Distances are in metres and the initialisation sheet is at the origin. The first tracking failure (1b) and subsequent relocalisation (3) are numbered in the map and in the camera frames(right).

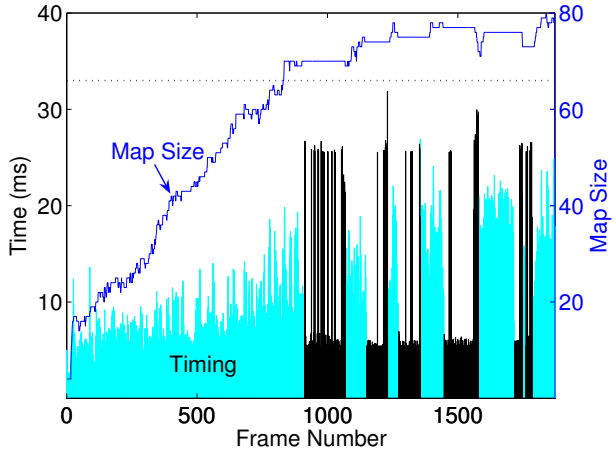

Figure 4: Processing time for the art gallery sequence. The system runs in real time as the map size increases to 80 features. Frames where the system is lost are shown in black.

Figure 5: Relocalisation from novel views. Here tracking (and thus harvesting) was only performed along a short linear motion illustrated by the top row of images. This is sufficient to allow relocalisation from the views shown in the bottom row despite the difference in viewpoints.

7. Discussion

7.1. Scalability

We have demonstrated relocalisation working well on map sizes up to 80 features. Beyond this, the EKF update time becomes the limiting factor in our SLAM implementation, both for tracking and for relocalisation. Although there are good reasons for favouring EKF SLAM in many applications, our relocalisation method is not tied to this approach and could be applied to other estimation algorithms (e.g. FastSLAM 2.0) which can demonstrably support larger map sizes in real time [4].

A larger number of map features inevitably results in a larger number of candidate correspondences - and hence a smaller inlier fraction - for the relocaliser. Intelligent match selection and fast rejection of sets of matches then become increasingly important if the correct pose is to be found in a reasonable time. To investigate how global relocalisation scales with map size, we have run the system offline on a pre-recorded outdoor sequence to generate a map with up to 200 features. Initial results indicate that although the number of hypothesised match triplets rejected by the early tests (co-visibility, co-linearity) grows rapidly with map size, the number of three-point-poses which need to be evaluated grows only very slowly, and the mean time required for a correct pose to be generated shows a linear growth with the number of classifier matches. A more thorough evaluation of the system's scalability will become possible when we can track large maps in real time.

7.2. Detection vs Tracking

Since the relocalisation method is able to recover the camera pose at frame-rate with no prior information, one may ask why one would bother with tracking. Timing alone makes a persuasive case: for a map of 50 features, active search typically takes only 0.5ms to establish correspondences, while full relocalisation requires 19ms. Accuracy is a second reason: in experiments comparing the position estimates of localisation and tracking for every frame of a sequence, we have consistently found SLAM poses to be smoother and more accurate. To make the comparison fairer, some form of pose refinement and filtering could be introduced to smooth the results of the relocalisation algorithm: but this is pretty much what the SLAM system does anyway. Finally, we continually update the classifier with new images of each patch being tracked; using this classifier to track the patch in the next frame is the classic recipe for feature drift! Here, the image patches used for tracking are never modified and so feature drift is avoided, even with continual training of the classifier.

8. Conclusion

This paper has presented a real-time monocular SLAM system which can recover from the frame-to-frame tracking failures which are inevitable in real-world operation. Instead of trying to avoid tracking failure altogether, the system presented here automatically detects the failure, halts the SLAM system, and begins relocalising instead. Mapping is only resumed when the camera pose has been redetermined, thus preserving map integrity.

Relocalisation is performed by first using a randomised lists classifier to establish feature correspondences in the image and then RANSAC to determine the pose robustly from these correspondences. We have shown this method to be reliable and fast for map sizes of up to 80 features, an upper limit set by the SLAM system rather than the relocalisation algorithm. We have shown results from the system running reliably in various scenarios, including for over an hour during which time it has created a good map despite frequent camera shake, sudden motion and occlusion.

While we would not claim that real-time monocular SLAM is now ready for use outside the lab, the ability to carry on mapping after a tracking glitch has dramatically increased the usability of our experimental system: tracking failures for the most part just do not matter anymore and the consequent ability to track long live sequences routinely will greatly facilitate investigations into a variety of other SLAM problems.

9. Acknowledgements

This work was supported by the EPSRC through grants GR/T24685, GR/S97774 and EP/D037077 and a studentship to BW. We are grateful for discussions with Andrew Davison, David Murray and Tom Drummond, and to the Ashmolean Museum for granting filming permission.

References

[1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7):1545–1588, 1997.

[2] D. Chekhlov, M. Pupilli, W. Mayol-Cuevas, and A. Calway. Real-time and robust monocular SLAM using predictive multi-resolution descriptors. In *Proc. 2nd International Symposium on Visual Computing*, November 2006.

[3] A. Davison, I. Reid, N. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[4] E. Eade and T. Drummond. Scalable monocular SLAM. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 469–476, 2006.

[5] M. A. Fischler and R. C. Bolles. RANdom SAmple Consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[6] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1465–1479, 2006.

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

[8] J. Neira and J. D. Tardós. Data association in stochastic mapping using the joint compatibility test. In *IEEE Transactions on Robotics and Automation*, pages 890–897, 2001.

[9] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *Proc. IEEE Conference on Computing Vision and Pattern Recognition*, 2007.

[10] M. Ozuysal, V. Lepetit, F. Fleuret, and P. Fua. Feature harvesting for tracking-by-detection. In *Proc. European Conference on Computer Vision*, pages 592–605, 2006.

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

[12] A. Rahimi, L. Morency, and T. Darrell. Reducing drift in parametric motion tracking. In *Proc. IEEE Conference on Computer Vision*, volume 1, pages 315–322, 2001.

[13] G. Reitmayr and T. Drummond. Going out: Robust model-based tracking for outdoor augmented reality. In *Proc. IEEE International Symposium on Mixed and Augmented Reality*, pages 109–118, 2006.

[14] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Proc. IEEE International Conference on Computer Vision*, pages 1508–1511, 2005.

[15] S. Se, D. Lowe, and J. Little. Vision-based global localization and mapping for mobile robots. *IEEE Transactions on Robotics*, 21(3):364–375, 2005.

[16] B. Williams, P. Smith, and I. Reid. Automatic relocalisation for a single-camera simultaneous localisation and mapping system. In *Proc. International Conference on Robotics and Automation*, 2007.