Automatic Relocalization and Loop Closing for Real-Time Monocular SLAM

Brian Williams, *Member*, *IEEE*, Georg Klein, *Member*, *IEEE*, and Ian Reid, *Member*, *IEEE*

Abstract—Monocular SLAM has the potential to turn inexpensive cameras into powerful pose sensors for applications such as robotics and augmented reality. We present a relocalization module for such systems which solves some of the problems encountered by previous monocular SLAM systems—tracking failure, map merging, and loop closure detection. This module extends recent advances in keypoint recognition to determine the camera pose relative to the landmarks within a single frame time of 33 ms. We first show how this module can be used to improve the robustness of these systems. Blur, sudden motion, and occlusion can all cause tracking to fail, leading to a corrupted map. Using the relocalization module, the system can automatically detect and recover from tracking failure while preserving map integrity. Extensive tests show that the system can then reliably generate maps for long sequences even in the presence of frequent tracking failure. We then show that the relocalization module can be used to recognize overlap in maps, i.e., when the camera has returned to a previously mapped area. Having established an overlap, we determine the relative pose of the maps using trajectory alignment so that independent maps can be merged and loop closure events can be recognized. The system combining all of these abilities is able to map larger environments and for significantly longer periods than previous systems.

Index Terms—Tracking, 3D/stereo scene analysis, autonomous vehicles.

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 preintegrated 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 Localization and Mapping (SLAM) and a real-time (30 fps) monocular SLAM implementation was first described by Davison (recently summarized in [1]).

Typically, tracking systems rely on a prior overcurrent pose and this prior is used to limit the search for visual feature correspondences. Constraining the search in this way both limits the chances of false correspondences and allows the system to run in real time. However, there are several situations which violate the assumptions in this prior and can cause these systems to fail.

- *B. Williams is with the Jet Propulsion Laboratory, M/S 198-233J, Oak Grove Drive, Pasadena, CA 91109. E-mail: Brian.P.Williams@jpl.nasa.gov.*
- *G. Klein is with Microsoft Corporation, 906 14th Ave E, Seattle, WA 98112. E-mail: gk@robots.ox.ac.uk.*
- *I. Reid is with the Department of Engineering Science, University of Oxford, Parks Rd, Oxford OX1 3PJ, UK. E-mail: ian@robots.ox.ac.uk.*

Manuscript received 28 May 2009; revised 9 June 2010; accepted 25 Nov. 2010; published online 24 Feb. 2011.
Recommended for acceptance by G.D. Hager.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-2009-05-0338.
Digital Object Identifier no. 10.1109/TPAMI.2011.41.

First, rapid camera motions, occlusion, and motion blur (phenomena which are common in all but the most constrained experimental settings) violate the assumptions which lead to failed observations followed by tracking failure. 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 second situation occurs when previously seen landmarks come back into view but the system, using the prior, is uncertain of where to search for them. This happens when the estimate of the relative position of these landmarks from the camera is very uncertain or unknown and occurs when the camera has traversed a loop in the environment or has had a period of failed tracking.

A solution for all of these problems is a method to determine the camera pose relative to the map without relying on the assumptions used in frame-to-frame tracking, i.e., to relocalize the camera. In this paper, we present such a relocalization module based on landmark recognition: While the SLAM system is estimating the geometry of the environment, a novel extension of Lepetit's image patch classifier [2] learns the appearance of the image patch at each map feature. These learned landmarks are used for relocalization when tracking fails or when the camera is reintroduced to the locality of an historically mapped area.

We analyze the performance of the module and show its operation in the context of a full working SLAM system. We show how the relocalization system, coupled with a trajectory-based map alignment method, can be used to merge multiple independently built maps of the same region of the world. This method is also used to increase the accuracy when mapping larger areas by detecting and closing loops. Without this ability to recognize previously

mapped areas, the error in the map can grow without bound. These improvements lead to a monocular SLAM system which can build large, accurate maps while robustly tracking the pose of the camera.

2 Related Work

The global relocalization problem has been much explored in robotics, where it is called the "kidnapped-robot" problem. The goal is to determine the location of the robot relative to the map after it has been placed within the mapped environment at an unknown location. Dellaert et al. [3] demonstrate global localization using a particle filter-based approach for a robot with sonar sensors. Neira et al. [4] also demonstrate localization in 2D, but their system uses a stochastic map using a robot equipped with a laser scanner. They determine the pose of the robot relative to a map of point features using the observations from a single timestep. Like our system, they use a combination of geometric compatibility and random sampling. Clemente et al. [5] used this algorithm in monocular SLAM to perform map-to-map matching to detect loop closure. Correspondences are found between landmarks common to the two submaps using both their visual appearance and their geometry. However, with the sparse maps typically used in monocular SLAM, common landmarks cannot be guaranteed. This makes the method unreliable [6].

Video-based systems have the advantage that they deliver appearance as well as geometry, and are therefore well suited to finding correspondences between the latest measurements (the current image) and a set of previously seen visual features or keyframes. For example, Reitmayr and Drummond [7] use edge-based tracking to track a handheld camera relative to a known model. When tracking fails, point feature correspondences are established between the latest image and a set of recent keyframes. From these matches, they solve for the relative rotational component of the pose (taking the translational component to be zero). Providing a keyframe exists in the vicinity, this usually gives a sufficiently accurate estimate of the pose to reinitiate tracking, which can then converge to the correct full pose. Klein and Murray [8] also relocalize their monocular SLAM system using a keyframe-based approach. Rather than searching for point correspondences though, the entire images are matched using sum-squared-difference after subsampling and applying a Gaussian blur.

Cummins and Newman [9] demonstrate a visual SLAM system without a metric map. Instead, a topological map is built in *appearance space*, with each location modeled as bag of visual words. Relocalization is then cast as an image retrieval problem. By learning a first-order generative model for the co-occurrence of the visual words, they achieve a measure of robustness to perceptual aliasing since the learning process discovers that visual words related to patterns like brickwork or bushes tend to co-occur in the world. However, if a keyframe-based approach is to be used in a system with a metric map, there remains the problem of localizing in the geometric space of the map, i.e., establishing keyframe to map correspondences. Most recently, Eade and Drummond [10] have done just this, combining bag-of-words SIFT with image-to-map correspondences. They use a global-to-local approach, first determining which submap is currently in view using the bag-of-words approach. Then, in a method similar to that presented here, matches are found to local landmarks in the image before using RANSAC and a three-point-pose algorithm to determine the camera pose relative to the map. In order to achieve real-time performance in the matching phase, they use a cheaper, but much abbreviated 16D SIFT descriptor. To compensate for the simpler descriptor, recall is improved by storing multiple descriptors for each landmark. We compare this approach to our own in Section 5.3.

The present paper builds on our previous work. The relocalization module was introduced in [11] and then made faster using landmark recognition in [12]. Here, we present a more detailed discussion of the performance of this system. The map merging and loop closure detection techniques have been discussed in [13] and compared to other methods in [6]. We present new experiments here demonstrating the performance of the technique.

3 SLAM Implementation

The underlying SLAM system we use is based on Davison's monocular SLAM implementation "SceneLib" [1] 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 [1], 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 relocalization. During normal tracking operation, we use *active search* to establish image-to-map correspondences: An association for a map landmark is only sought in the vicinity of where that landmark is predicted to lie. The region to search is given by the innovation covariance of that landmark, 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.

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 further mitigate against this, we employ the joint compatibility test proposed by Neira and Tardós [14]. Given a set of putative matches between image observations and landmarks 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 prewarping, active search and joint compatibility, and filter update—typically takes 5 ms (out of a total budget of 33 ms for frame-rate operation) on a Core 2 Duo 2.7 GHz machine for a map of 30 landmarks. Maps must be kept small as the EKF update is $O(N^2)$.

In Davison's system, the remaining CPU cycles were used to search for and initialize new landmarks. A window is placed in the image in a semirandom location (regions with few current landmarks and which are "coming into view" are favored). Corner locations are detected and the strongest in the window selected for addition to the map. We insert such landmarks directly into our map parameterized by the visual direction and inverse depth, similarly to [15], with a large uncertainty on the latter. However, we do not simply take the strongest landmark; rather, we take advantage of the discriminatory powers of the landmark recognition component to choose a landmark unlike existing ones in the map (see Section 4.2.3). Choosing more easily discriminated landmarks improves the performance of the relocalization module.

4 Relocalization Module

In this section, we present one of the key contributions of this paper: our relocalization module for the monocular SLAM system. This module can determine the pose of the camera relative to a 3D map of landmarks using the estimated geometry of the landmarks and the latest image from the camera. To do this, the landmarks must first be recognized in the image. Then, the pose of the camera can be determined. We begin with a brief description of how the camera pose is determined before presenting our method for landmark recognition.

4.1 Determining the Camera Pose

Our system determines the pose of the camera relative to the landmarks mapped by the SLAM system. Given a set of potential matches from image points to map landmarks, we apply the well-known method of Fischler and Bolles to relocalize the camera relative to the map using their three-point-pose algorithm and RANSAC [16]. Consensus for each calculated pose is evaluated by checking how well that pose predicts the location of the image projection for the other map landmarks. When a good pose is found, it is optimized using the consensus set.

Establishing a set of potential matches is a nontrivial problem. Though RANSAC can deal with some degree of mismatches, the ratio of inliers (true matches) to outliers must be sufficiently high for the algorithm to produce a result within a reasonable time—for our purposes, this is one video frame time, 30-40 ms. To this end, we develop a classification-based feature recognition/matching approach in the following section that is capable of rapidly producing a potential match set. In addition, we further improve the performance of RANSAC by guiding the selections of three matches to increase the chances obtaining a usable three inliers. Landmarks with reliable, unique classifications are likely to be correctly matched and so are chosen more often in the random sampling stage. Those with low classification scores or those which occur many times in images are penalized and consequently chosen at random less often. We further reject match triplets which are collinear or very close together in the image as these produce poor pose estimates. We further check match triplets against a covisibility 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 landmarks from distant parts of the map which are unlikely to all be correct. Together, these measures reduce the number of RANSAC trials required for successful pose determination.

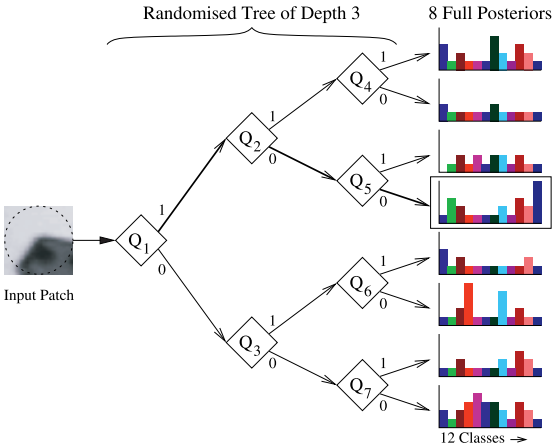


Fig. 1. **Randomized tree classifier of Lepetit and Fua.** This diagram illustrates the structure of their classifier using small (depth 3) tree. See Section 4.2.1 for a description of their algorithm.

4.2 Recognizing Landmarks through Classification

The underlying key to achieving rapid correspondence between the current image and the map is our landmark recognition algorithm, which supplies a set of putative matches, described in this section. We begin by reviewing the work of Lepetit and Fua [2], and then describe our modifications that yield a classifier better suited to our needs.

4.2.1 *The Classifier of Lepetit and Fua*

Lepetit and Fua [2] employ randomized trees of simple image tests rapidly to detect reoccurrences of previously trained image patches in a new input image. Their system can detect reoccurrences even in the presence of image noise, changes in scale, rotations, aspect ratio, and illumination changes. The novelty of their method is to treat real-time feature recognition as a classification problem, where each class is trained with (ideally) all of the different appearances a single feature can take due to the 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 method is a forest of N binary decision trees. A single simplified tree of depth $D = 3$ for recognizing $C = 12$ classes is illustrated in Fig. 1.[1] Each tree maps an input image patch to one of 2^D leaves of the tree. During supervised training, the relative frequencies of arriving at a leaf for each class are accumulated, yielding $P(L|c)$, i.e., the probability of arriving at leaf node L given that the class is c.

The mapping is the result of a series of simple binary tests Q which make the tree's decision nodes: Each test Q

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

compares the patch's Gaussian-smoothed intensity values $I_\sigma(\cdot)$ at two pixel locations $\vec{a}$ and $\vec{b}$ such that

$$Q_i = \begin{cases} 0, & I_\sigma(\vec{a}_i) - I_\sigma(\vec{b}_i) >= 0, \\ 1, & \text{otherwise.} \end{cases} \quad (1)$$

During recognition, the conditional probabilities yielded by each tree are summed to obtain the maximum likelihood class. While each individual tree may itself be a rather poor classifier, the combination of N trees' conditional probabilities yields good recognition results.

The training phase in which the conditional probabilities are learned requires hundreds or thousands of patches belonging to each known class to be "dropped down" each tree. Training patches are either synthetically generated by warping [2] or extracted from live video in which the camera undergoes carefully controlled motion, a process termed *harvesting* [17], with the aim of teaching the trees all possible views of a class. Monocular SLAM systems are particularly well suited to take advantage of training through harvesting as will be discussed in Section 5.3.

One of the key results in [2] concerns the selection of the locations $\{\vec{a}, \vec{b}\}$. During a training phase, these may be chosen from a small random set so as to maximize each test's information gain (in accordance with [18]); alternatively, they can be chosen entirely randomly, which results in a small drop in classification performance but massively reduces the time required for training.

4.2.2 Our Method: Randomized Lists

Here, we describe our approach to landmark recognition via classification, which results from various modifications to the method above, making it better suited to our application.

First, it is clear that in our monocular SLAM setting, in which the goal is to build a map and localize simultaneously, a separate training phase (such as is used in [2]) would be a self-defeating exercise. This means that the binary tests Q_i must be preselected 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 [19].) The results of these tests form a binary string which indexes into the array of 2^D class likelihoods. Using the list structure allows for significant improvements in runtime speed. (For example, we can replace patch rotation with a lookup table of prerotated tests.)

Second, besides this structural change, we also make changes to the operating characteristics of the classifier; whereas the method of Lepetit and Fua operates with many dozens or hundreds of keypoints visible per frame, our SLAM implementation uses a far sparser set of landmarks: At any given frame, only 10-20 might be visible. For this reason, it is important to tune the classifier toward 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 Fig. 2 and the algorithm is given explicitly in Algorithm 1. Note in particular that virtually the same code is used for both training and classification. We discuss the various changes in more detail below and consideration of the performance of the classifier is then described in Section 5.2.

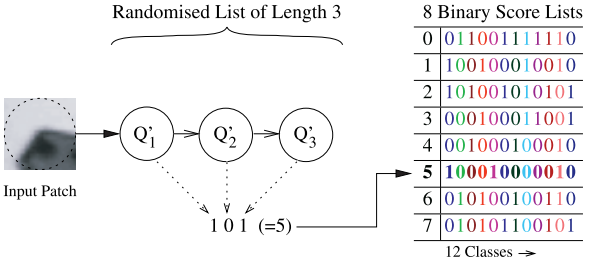


Fig. 2. **Our randomized lists classifier.** This diagram illustrates the structure of our classifier for comparison to the method of Lepetit and Fua [2] shown in Fig. 1. A description of the algorithm is given in Section 4.2.2.

Algorithm 1. our_classifier(image I, corner position x)

```
I_σ ⇐ Gaussian_smoothing(I)
g ⇐ compute_dominant_gradient(I_σ, x);
for each list n = 1 to N do
    for each question i = 1 to D do
        [a_i, b_i] ⇐ lookup_Q'_i(n, i, g)
        if |I_σ(a_i) − I_σ(b_i) − z_i| < noise_threshold then
            leaf_address(bit i) ⇐ "don't care" state
        else if I_σ(a_i) − I_σ(b_i) >= z_i then
            leaf_address(bit i) ⇐ 0
        else
            leaf_address(bit i) ⇐ 1
        end if
    end for
    leaf_probability ⇐ lookup_probability(n, leaf_address)
    if classifying then
        for each class c = 1 to C do
            if leaf_probability(bit c) = 1 then
                increment score(c)
            end if
        end for
    else if training then
        leaf_probability(bit c) ⇐ 1
    end if
end for
if classifying then
    for each class c = 1 to C do
        if score(c) > threshold then
            add_to_match_hypotheses(c)
        end if
    end for
end if
```

Multiple hypotheses and class independence. For an extended exploration, it is unrealistic to assume that all map landmarks will appear completely unique. Due to perspective and lighting changes—and self-similarities common in man-made environments—some map landmarks will resemble one another. The method of Lepetit and Fua [2] penalizes such cases: Only the strongest match is returned, and since the likelihoods are normalized, the presence of multiple similar landmarks penalizes the score of all of them, sometimes to the extent that none is recognized. Here, we consider each class's 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 online.

Binary leaf scores. Instead of storing full normalized likelihoods over the classes at each leaf, we store a binary score string of one bit per class. A class's 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 effect at recognition time is equivalent to having a low, binary threshold on the likelihood values. A list "votes" for a class if a training patch of that class ever looks sufficiently like it to obtain identical decisions in the list. This is repeated for every list in the classifier. Every class which scores a minimum number of list votes is accepted as a valid class label hypothesis. While this clearly sacrifices some of the discriminatory power of the unthresholded likelihood, the benefit is a reduction in storage requirements ($2^D \times C \times N$ bits for all scores) which allows operation with higher values of D. We demonstrate the value of this in Section 5.2

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:

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

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 online learning (also referred to as "feature harvesting," see Sections 4.2.3 and 5.3).

Explicit noise handling during training. In [2], 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 the 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.

4.2.3 Training the Classifier

The classifier must be trained to recognize each new landmark that is added to the map during "steady-state" SLAM operation. When the landmark is first observed, we train the classifier with 400 synthetically warped versions of the landmark appearance so that it can immediately be recognized under a variety of views. We use a GPU to perform the warps. Even so, this initial training is costly at around 30 ms per new landmark; 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 leaf representation, there is no benefit in incrementing a class conditional likelihood more than once, so we can stop training once a large fraction (95 percent) of warped examples revisit previously seen leaf nodes.

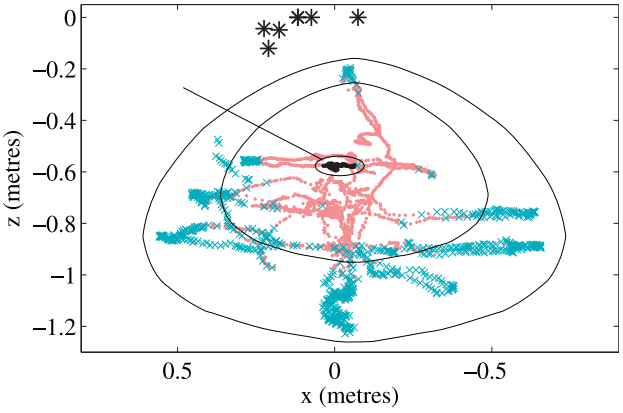


Fig. 3. **Relocalization volume.** This figure shows the typical volume over which the module can relocalize after having harvested in only the small central volume. The pose was tracked and the system attempted relocalization at every frame outside of this (• for success, × for fail). Some example frames from this experiment are shown in Fig. 4.

This relatively crude initial training stage is subsequently reinforced using real image data acquired or "harvested" during normal SLAM operation: When a map landmark is successfully observed by the SLAM system, the classifier is also trained with this new view of the landmark.

To improve the distinctiveness of the landmarks in the map, the classifier aids in the selection of new map landmarks. When the system is initializing a new map landmark, it is usually faced with a choice of potential landmarks in the image. Rather than simply taking the one with the strongest corner response, as in [1], each candidate corner is instead first classified with the current randomized lists classifier. The system then chooses to initialize the landmark that scores lowest against all other landmarks already in the map. This leads to a map of landmarks which are more easily discriminated.

5 Performance of the Relocalization Module

In this section, we discuss the performance of the relocalization module. We will show the general performance of the relocalization module before discussing the performance of our randomized lists classifier alone. Later, in Section 6, we will discuss how the relocalization module can be used to solve several problems encountered in monocular SLAM.

5.1 Relocalization Performance

First, we analyze the range of viewpoint over which relocalization is possible. It is important for the system to be able to relocalize even when the current viewpoint does not precisely match an earlier pose of the camera. In this experiment, a small map was generated and harvesting performed while moving the camera across a small range of viewpoints indicated by the top row of images in Fig. 4 and in the metric map in Fig. 3. Mapping and harvesting were then disabled, and the system attempted relocalization and tracking from a much larger set of viewpoints. Relocalization was successful from a range of novel views that have not been trained during harvesting. Some examples are given in the bottom row of Fig. 4.

Fig. 4. **Relocalization 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 relocalization from the views shown in the bottom row despite the difference in viewpoints.

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

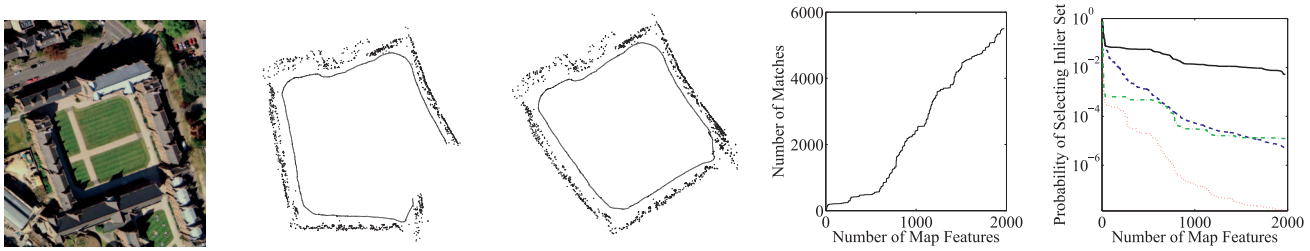
In a real-time system, the relocalization module must also be very fast. It must determine the pose quickly enough for that information to still be relevant, i.e., within the frame rate of the system. If the camera pose is not found within that time, the module begins again with the next camera image. Table 1 shows the time taken (on a Core 2 Duo 2.7 GHz processor) by the individual stages of a typical relocalization with 54 landmarks in the map. Here, the classifier has returned 42 matches, of which seven are inliers: RANSAC must therefore find one of the 35 inlier triplets out of 11,480 possible match triplets. In this run, 335 triplets were hypothesized but most of these could be rejected before even attempting three-point pose by testing for covisibility 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 19 ms.

TABLE 1
Typical Relocalization Timings (Map Size 54)

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**

We have also tested the scalability of the module when working in large maps with a much greater number of landmarks. The key to relocalizing quickly in these situations is the match scoring heuristics, which improve the probability of choosing an inlier set of landmark matches. Fig. 5 shows the effect of these heuristics on the probability of a successful relocalization as the number of landmarks grows. In this experiment, a large map of 1,975 landmarks was created in a university courtyard of $60\text{ m} \times 70\text{ m}$. The relocalization module was tested using a single image acquired within the space spanned by the map. Initially, only three truly visible landmarks are considered. For each of these landmarks, only the true correspondence to each is found in the image. With three correct matches found and no false positives, the probability of choosing an inlier set is 100 percent. As the set of landmarks considered grows to include all of the truly visible landmarks, some false correspondences are also found, so the probability of choosing an inlier set decreases slightly. The set of landmarks considered is then increased to include the landmarks that are not visible in the image. Any correspondences to these landmarks are false positives and will decrease the chances of an inlier set being chosen. With the entire map of landmarks considered, the large number of false matches leads to a tiny probability of randomly selecting a set of three correct matches needed for relocalization. However, by taking into account the simple heuristics outlined here, many of the incorrect combinations are unlikely to be chosen.

These heuristics allow the system to relocalize in maps of several thousand landmarks. At this point, the limiting factor then becomes the memory required to store the classifier. The randomized lists classifier trades memory for speed and, at 1.25 MB per class, this limits the number of



(a) (b) (c) (d) (e)

Fig. 5. **Scalability of the relocalization module.** A large map of a university courtyard was created to test the scalability of the relocalization module. The map built contains 1,975 landmarks over an area of $60\text{ m} \times 70\text{ m}$. (a) Aerial photo of the courtyard. (b) The map showing the features and trajectory viewed from directly above. (c) The map after loop closure using the method described in Section 6.3. (d) As the number of landmarks considered grows, so too does the number of potential matches to these landmarks in an image. (e) The probability of selecting a set of three correct matches plotted against the number of landmarks considered is increased. When landmarks which are not present in the chosen frame are considered, any matches found are false and will decrease the chances of selecting an inlier set. Without match scoring heuristics, the probability of selecting a set of three true correspondences in a large map would be very low. The graph on the right shows the probability of choosing a set of three inliers randomly without guidance ($\cdots$), if multiple matches to the same map landmark are penalized (- - -), if covisibility information is used ($\cdot - \cdot -$), and if both of these checks are used (—).

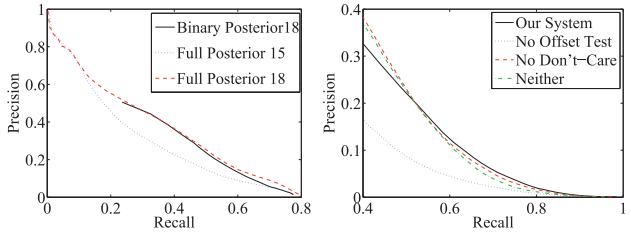


Fig. 6. Performance of the randomized tree classifier.

landmarks that can be stored in RAM. Though this makes the randomized lists classifier unsuitable for arbitrarily large numbers of landmarks, we note that in practice, the actual limiting factor in our system is the size of the maps the EKF can handle. To deal with this, we make use of submaps, and each submap has its own classifier [5], [6].

5.2 The Effects of Classifier Design Choices

To test the performance impact of the landmark classifier design choices described in Section 4.2.2, two tests were performed on a controlled 2,300-frame SLAM run during which a map of 70 landmarks was built without tracking failure. The tracked landmark 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 percent of the known landmarks 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.) Full precision recall curves are given in Fig. 6.

Table 2 demonstrates the effect of using a binary leaf score list instead of a full class conditional likelihood (with 8 bits per class in our implementation). As may be expected, simplifying to a binary score reduces classification performance for equal tree depths; however, the binary score outperforms the full likelihood 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).

Table 3 illustrates the effects of the other classifier adaptations on classification rates. The use of intensity offsets, together with the explicit training of all image noise permutations, produces 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.

TABLE 2
Binary versus Full Classifier Performance

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

TABLE 3
Performance Impact of Classifier Modifications

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

It should be noted that the low precision scores in the above tables are obtained without any landmark appearance harvesting. If this is enabled, precision increases from 0.09 to 0.2, and the actual value encountered during real-world relocalization likely lies between these two. Finally, the heuristic guided sampling criteria described in Section 4.1 help boost inlier rates to a level normally manageable by RANSAC.

5.3 The Benefit of Harvesting

In many applications, keypoint recognition is forced to find correspondences using only a single example view of the keypoint. This has led to the development of rich feature descriptors like SIFT [20] with inbuilt invariance to many transformations. However, rather than attempting to build some invariance into the descriptor, an appealing alternative is to learn the appearance of the keypoint over the complete range of conditions and viewpoints from real-world examples. In practice, this is usually not possible since labeled training examples for each keypoint are not available. However, in our application, the SLAM system can provide reliable real-world training examples at a rate of 30 Hz while a landmark is visible, and so this online harvesting provides a way to take advantage of this wealth of appearance information. Each time a landmark is successfully tracked by the SLAM system, its current appearance is used to update the learned likelihoods by "dropping it down" the lists.

An experiment was performed to assess the benefit of harvesting on landmark recognition rates. The randomized list classifier (RLC) was also compared to the popular SIFT descriptor. For the experiment, 50 landmarks were initialized and tracked by the SLAM system during a 3-minute sequence. Harvesting (online appearance learning) was performed during the first 2 minutes of the sequence. The classifier was then frozen and the final minute of the sequence was used to test the recognition rate. Keypoints were detected in each frame of the test sequence using the FAST corner detector [21]. Each of these corners was then classified. Ground truth for landmark locations was given by the SLAM system. Multiple hypotheses were allowed since the goal is recall rather than precision. To achieve scale invariance for SIFT matching, a scale pyramid was built. Descriptors were then created for the FAST corners for all scales since FAST corners do not have an intrinsic scale unlike the DOG extrema typically used in SIFT. The results of this experiment can be seen in Fig. 7.

With only synthetic training performed using the first observation of each landmark, the randomized list classifier performs poorly compared to a SIFT descriptor created using only the first observation. However, once the

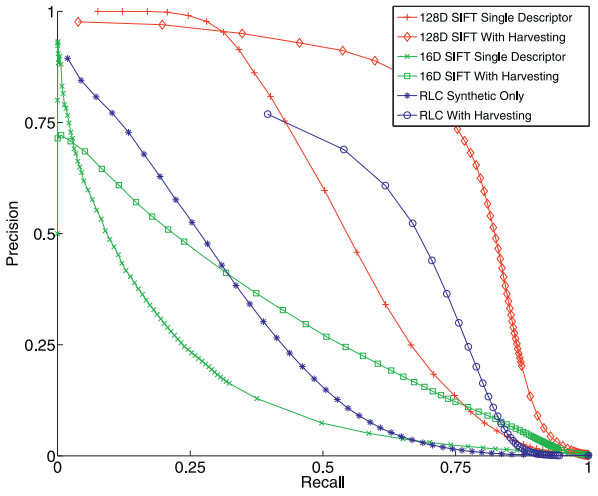


Fig. 7. **The benefit of harvesting.** Classification performance was tested for 50 landmarks during the final minute of a 3-minute sequence. The first 2 minutes of the sequence were available for harvesting. A Randomized List Classifier trained with only synthetic warps of a single patch performs poorly compared to a 128D SIFT descriptor made from a single patch. However, when the RLC is also trained with 2 minutes of harvesting from a range of viewpoints, the recall is higher than for 128D SIFT. To achieve real-time performance using SIFT, Eade and Drummond used a small 16D descriptor but boosted the recall with harvesting when the appearance changed significantly. Here, harvesting was performed on every frame of the training sequence. The computational cost required to compare the huge number of SIFT descriptors is prohibitive in a real-time system, whereas the classification time for the RLC does not grow.

randomized list classifier is trained using harvesting, its recall greatly increases.

For comparison, a fourth test was performed combining harvesting with SIFT. In this test, a SIFT descriptor is stored for every observation in the training sequence. Matching is then performed during the test sequence against this much larger descriptor set. This greatly improves the recall, but this method would be not be feasible in practice due to the prohibitive cost in matching against the huge (and growing) number of descriptors. In contrast, the recognition time for the randomized list classifier does not increase as more examples are trained. Eade and Drummond were able to use SIFT with harvesting and achieve real-time performance by using short 16D SIFT descriptors and only harvesting when appearance changed enough to significantly alter the SIFT descriptor [10]. The results given in Fig. 7 would have higher recall since harvesting is performed on every training frame.

Timing alone also makes a good case for the randomized lists classifier for real-time systems. The classifier takes an average of 0.12 ms to train each new appearance of a landmark and then 0.04 ms to classify each detected corner. 128D SIFT takes 0.29 ms to create each new descriptor; however, more preprocessing is also required. A Gaussian pyramid is needed for scale invariance, and gradient images are required for descriptor creation. Combined, these take 440 ms, already beyond the frame rate of the system. Faster implementations of SIFT do exist, but the computation of a rich 128D descriptor cannot be made faster than the simple pixel intensity comparisons required for classification using randomized lists.

Calonder et al. [22] have proposed a randomized tree classifier-based method that does not involve harvesting. Instead, a randomized tree classifier is created offline and trained to recognize a few hundred distinct generic features. During online operation, features are then recognized using their "signature." A signature is a descriptor for a patch generated from the response of the classifier over all the generic classes. These signatures achieve performance similar to SIFT descriptors but can be calculated much faster. Their system also does not suffer from the growing memory footprint required to train new landmarks when using a nongeneric classifier. The price paid is a lower recall than what can be achieved through harvesting in situations where many examples are available for training.

6 Applications of Reliable Relocalization

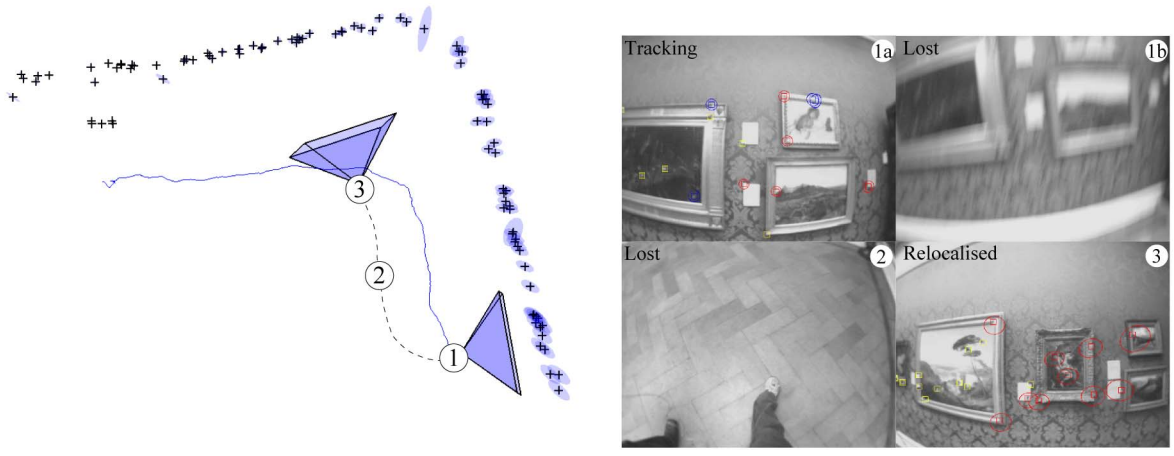
In this section, we show three different applications of our relocalization module for a monocular SLAM system. All three of these techniques have been incorporated into a full working system that can reliably map much larger areas (using submaps) over much longer durations than, for example, [1]. Though a detailed description of the full system is beyond the scope of the present paper, a summary of the interoperation of the modules including the role of the relocalization is given in Algorithms 2 and 3.

Algorithm 2. Main Loop

```
loop
  image ⇐ wait_for_next_frame();
  if relocalisation_needed then
    if time_since_failure < 1 second then
      attempt_relocalisation(image, current_submap);
    else
      stop_tracking(current_submap);
      start_independent_submap();
    end if
  else
    SLAM_update(image, current_submap);
    if is_tracking_well(current_submap) then
      try_overlap_detection_and_alignment();
      if too_many_landmarks(current_submap) then
        stop_tracking(current_submap);
        start_neighbouring_submap();
      end if
    else
      relocalisation_needed ⇐ true;
      cancel_alignment_in_progress();
    end if
  end if
end loop
```

Algorithm 3. try_overlap_detection_and_alignment()

```
submap_A ⇐ current_submap;
if no_alignment_in_progress then
  submap_B ⇐ select_another_submap();
  attempt_relocalisation(image, submap_B);
  if relocalisation_was_successful(submap_B) then
    alignment_in_progress ⇐ true
    retain_current_camera_pose(submap_A);
    retain_current_camera_pose(submap_B);
  end if
```



(a) (b)

Fig. 8. **Recovery from tracking failure.** While building a map along two walls of an art gallery, the camera is suddenly pointed at the floor and moved to a previously mapped area. The system quickly relocalizes and continues tracking during after the six tracking failures during the sequence. (a) Overhead view of the final map (3σ ellipses) and trajectory estimate. The first tracking failure (1b) and subsequent relocalization (3) are numbered in the map and in the camera frames (b).

```
else
  track_camera_in_frozen_map(image, submap_B);
  if is_tracking_well(submap_B) then
    if length(common_trajectory) > thresh then
      if submaps_are_independent(submap_A, submap_B)
      then
        submap_A ⇐ merge_maps(submap_A, submap_B);
        delete(submap_B);
      else
        determine_relative_alignment(submap_A,
          submap_B);
        add_new_edge(submap_A, submap_B);
      end if
    end if
  else
    alignment_in_progress ⇐ false
    remove_retained_pose(submap_A);
    remove_retained_pose(submap_B);
  end if
end if
```

6.1 Recovery from Tracking Failure

To be useful in real-world conditions, a monocular SLAM system must be robust to tracking failure. 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 landmarks. At this point, normal SLAM operation is halted to prevent corruption of the map estimate. The system then begins to use the relocalization module to recover from this tracking failure.

The system attempts relocalization on each new frame from the camera. When tracking conditions improve, this relocalization attempt is successful and the camera pose is determined. The EKF is then reinitialized. First, the new camera pose is put in the state vector with an inflated uncertainty. Then, the image positions of landmark consensus set are used as observations to estimate the true camera pose uncertainty and its correlation with the map. 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.

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 known position and that it could be oriented in any direction. Landmark correspondences returned by the classifier are filtered by potential visibility: Landmarks which could not be visible from any point in the sphere are given zero weight. Thus, our system switches seamlessly from an efficient local relocalization immediately after tracking is lost to a global kidnapped-robot relocalization after a few seconds have elapsed.

By recovering after tracking failure, our monocular SLAM system is able to complete sequences which would be very challenging 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 relocalizes again using previously mapped landmarks.

To demonstrate our system's mapping and relocalization operation, we tested the system on 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 was previously mapped, whereupon relocalization occurs within one to two frames. Fig. 8 shows the map created and the trajectory for the run. The first relocalization is annotated in the map, numbered to match the images on the right.

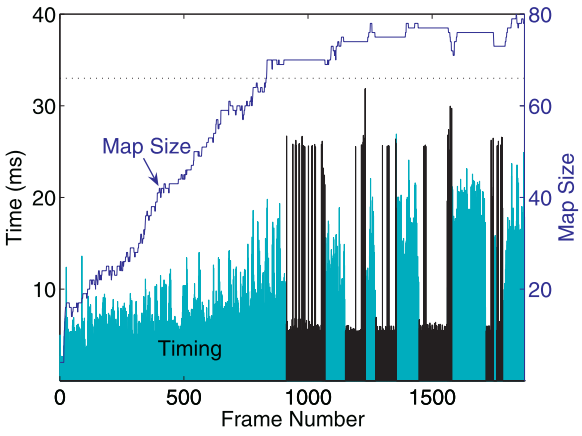


Fig. 9. **Processing time.** While performing SLAM on the sequence shown in Fig. 8, processing is performed in real time. Frames where the system is lost and relocalization is attempted are shown in black.

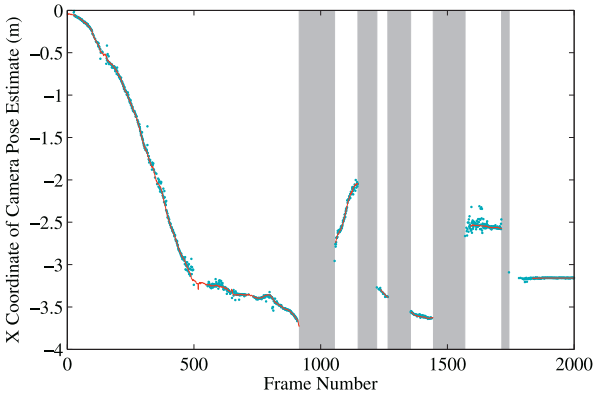


Fig. 10. **Relocalization versus tracking.** For the sequence in Fig. 8, the x-coordinate of the pose estimated by frame-to-frame tracking (solid line) is compared to the raw output of the relocalization module run at every frame. The trajectory estimated via tracking is smoother. The gray regions indicate untrackable sections when the camera is pointed at the floor or the images are heavily blurred.

System timings are shown in Fig. 9; this plots the processing time required per frame during the art gallery sequence. Each frame (including those during which recovery is attempted) is processed in under 33 ms; this is possible for maps up to 80-90 landmarks; beyond this submapping must be used (Section 6.3). Black timing values indicated frames during which the system was lost; of these, those frames with low runtimes are those in which insufficient landmarks were detected to attempt relocalization.

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 landmarks. Despite erratic camera motion and hundreds of tracking failures, the map was not corrupted. Relocalization was consistently successful until the arrival of coworkers changed the environment (our current map management policies do not scale well to sequences of this length).

Since the relocalization module 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 landmarks, active search typically takes only 0.5 ms to establish correspondences, while full relocalization requires 19 ms. Accuracy is a second reason: In experiments comparing the position estimates of localization and tracking for every frame of a sequence (Fig. 10), we consistently find SLAM poses to be smoother and more accurate.

Finally, we continually update the classifier with new images of each patch being tracked. Note that the classifier is used for feature recognition when lost, not for tracking. In our SLAM system, when the camera is not lost, landmark matching is performed using normalized cross correlation with an image patch which is never modified. This avoids drift in the SLAM system and allows it to act as a supervisor to train the classifier.

6.2 Independent Map Joining

Here, we show how the relocalization module is used to initiate the merger of two maps depicting overlapping regions of the world into a single map. While map merging can be used to build a single large map using several different sequences taken in the same environment (perhaps offline), its principal use in our system is after a tracking failure. After the failure, if an immediate relocalization is not possible (because, for instance, there are no known landmarks in the field of view), the system begins a new, *independent map*. Later, when the camera returns to the previously mapped region and this is detected by the relocalization algorithm, the new map can be merged with the old one.

To merge two independent maps, the relative rotation, translation, and scale between the maps must be determined. In [5], this transformation was determined using a set of common landmarks between the two maps. However, since the maps were built independently, there is no guarantee that the system would have initialized the same landmarks in both maps in the overlapping region. It was shown in [6] that relying on common landmarks in monocular SLAM leads to unreliable detections of overlap.

Instead, the trajectory of the camera relative to each map during a short sequence is used. This common trajectory provides the relative transformation between the two maps without relying on common landmarks. At each new frame, while tracking in map A the system attempts relocalization in the map with potential overlap, map B. If this relocalization is successful, then a possible overlap between map A and map B is indicated. While the SLAM process continues map A, the pose is tracked in map B (but the map is not updated to avoid overcounting evidence if maps A and B are merged) from the initial relocalized position. If the camera pose is tracked successfully for several seconds in map B, then a true overlap between the two maps is deemed to exist. A summary is given in Algorithm 3.

To align the two maps, a single pose correspondence, $\hat{\mathbf{x}}^A_{cam} \rightarrow \hat{\mathbf{x}}^B_{cam}$, from the common trajectory is sufficient to give the relative rotation and translation but not the scale difference. To determine the scale difference, the distance between this first pose, $\hat{\mathbf{x}}_{cam}$, and a second pose, $\hat{\mathbf{x}}_{cam^*}$, from the end of the trajectory is used. These two pose estimates are maintained in the EKF state vector, $\hat{\mathbf{x}}$, so they become properly correlated with the landmarks through the covariance matrix, $\mathtt{P}$.

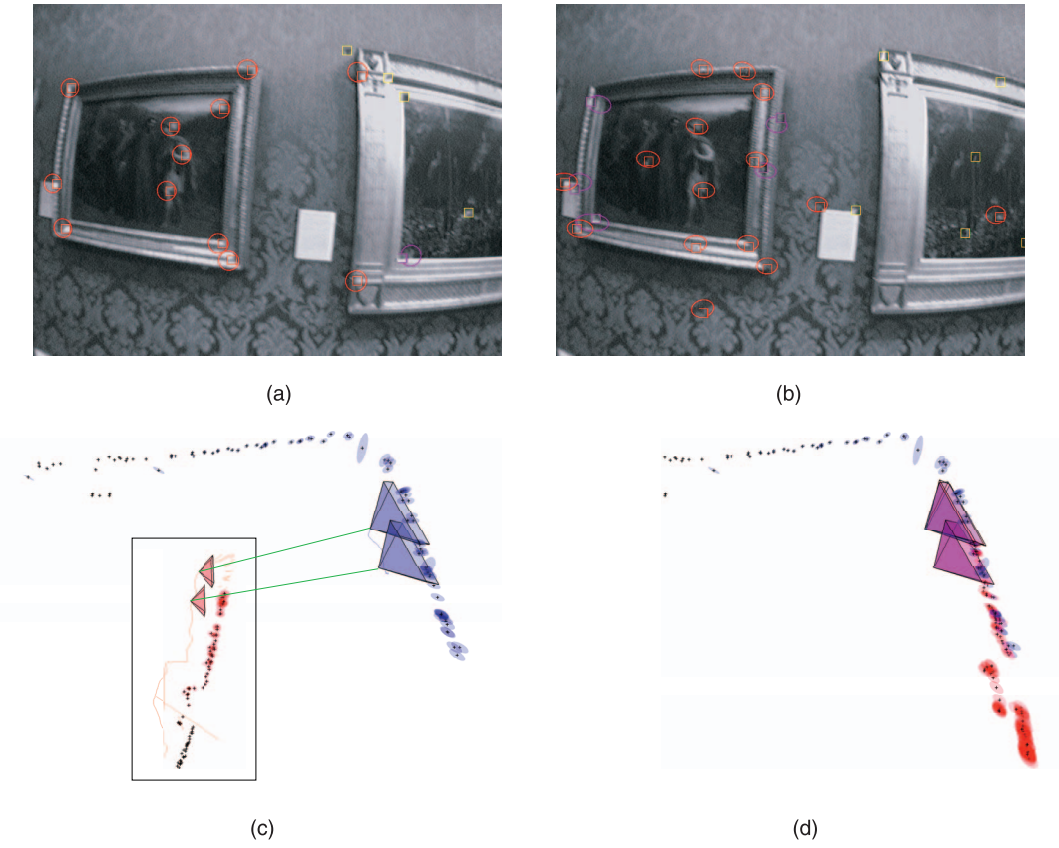


Fig. 11. **Independent map merging.** The system tracks the camera during a second sequence in the same art gallery used in Fig. 8. The camera eventually moves into a region already mapped in the first sequence. The module relocalizes in the first map (blue), and tracks a common trajectory relative to each map (c). The landmarks being tracked in the same image frame for the first and second map are shown in (a) and (b), respectively. Note that though the maps overlap in the world, they do not necessarily share landmarks; instead, two poses from the common trajectory are used to determine the alignment and the two maps can then be merged into one, as shown in (d).

After transforming the maps into a common coordinate frame using the transformation determined from the common poses, the result is two aligned but uncorrelated maps.

$$\hat{\mathbf{x}}_{aligned} = \begin{pmatrix} \hat{\mathbf{x}}^A_{cam^*} \\ \hat{\mathbf{x}}^A_{cam} \\ \hat{\mathbf{x}}^A_{map} \\ \hat{\mathbf{x}}^B_{cam^*} \\ \hat{\mathbf{x}}^B_{cam} \\ \hat{\mathbf{x}}^B_{map} \end{pmatrix} \quad \mathrm{P}_{aligned} = \begin{pmatrix} \mathrm{P}_A & 0 \\ 0 & \mathrm{P}_B \end{pmatrix}. \tag{3}$$

To calculate the correlation between the two maps, the constraint that the corresponding poses are identical is imposed. This can be done using an EKF update where the "observation," z, is the difference between the corresponding poses.

$$z = \begin{pmatrix} \hat{\mathbf{x}}^A_{cam} - \hat{\mathbf{x}}^B_{cam} \\ \hat{\mathbf{x}}^A_{cam^*} - \hat{\mathbf{x}}^B_{cam^*} \end{pmatrix}. \tag{4}$$

An EKF update is performed using the prediction that this observation should equal zero. Afterward, the combined map is correlated and the extra camera poses can be removed from the state.

This process is demonstrated in Fig. 11. A new map was begun in a different part of the art gallery used in Fig. 8. When landmarks from the old map come into view, relocalization occurs in the old map, and a common trajectory is estimated, as shown in Fig. 11c. Two poses from this common trajectory are then used to merge the two maps into the single correlated map shown in Fig. 11d.

6.3 Loop Closure Detection

Another application for the relocalization module is loop closure detection. Loops in a map are created when the camera returns to a previously mapped part of the world via a novel route. It is important for the SLAM system to recognize that it has returned and that old landmarks are back in view or it will not estimate the camera position correctly. This process of loop closure detection is far from trivial though and has led to significant research in the mobile robotics community. The difficulty lies in the accumulation of uncertainty in the state estimate as the camera traverses a large loop and in potential inconsistency of the EKF because of linearization errors.

Instead of using active search, we make use of relocalization as a separate loop closure detection module, which can determine the pose of the camera in these situations of high uncertainty.

Loop closure becomes more important when maps are built of larger environments. Since the EKF scales quadratically with the number of landmarks, it becomes necessary to use more sophisticated mapping strategies when dealing with large maps. We use a submapping technique developed for monocular SLAM by Clemente et al. [5]. The environment is broken into a series of smaller submaps whose relative

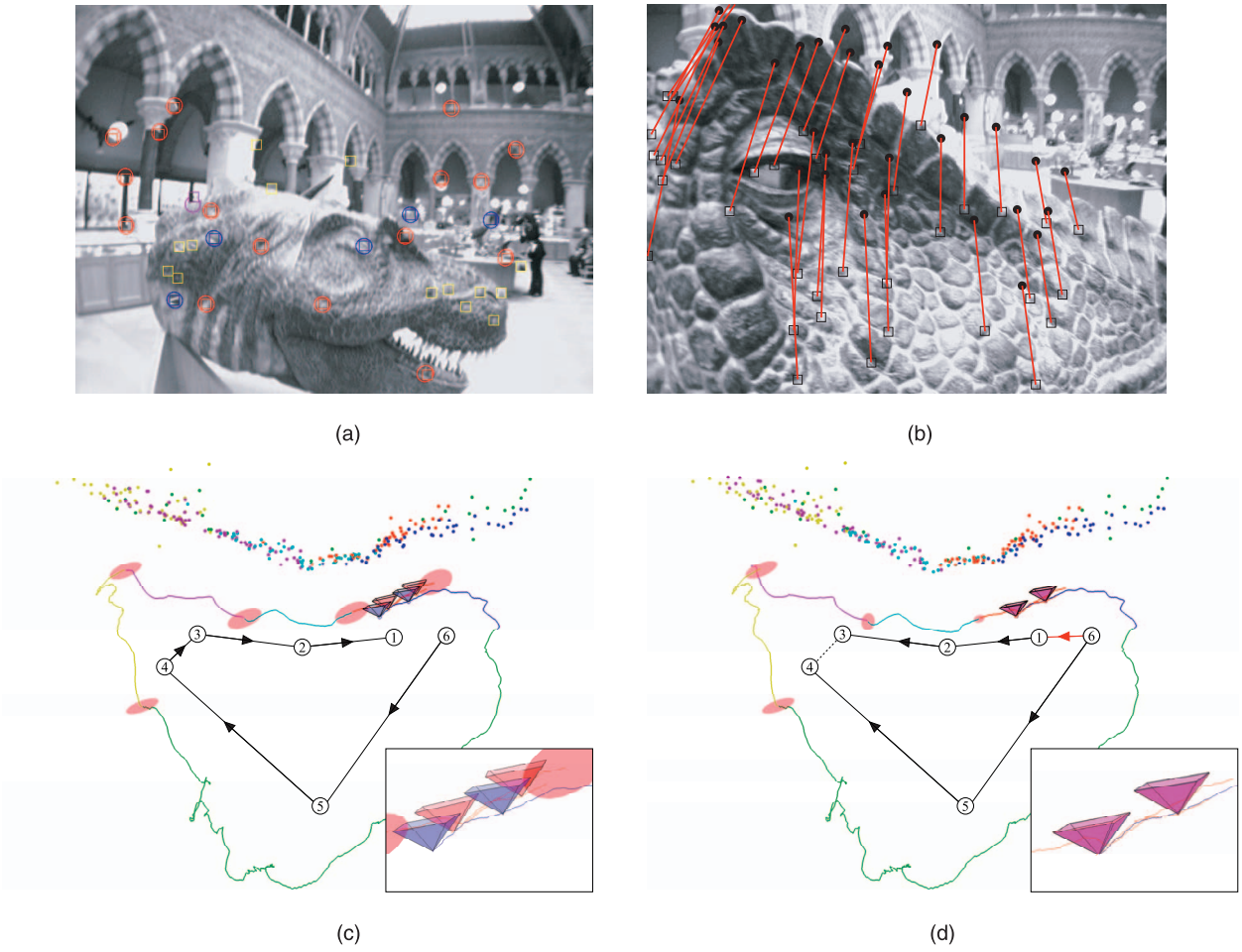


Fig. 12. **Loop closure detection.** The system tracks the camera as it moves around a loop. (a) A frame halfway through the sequence. Ellipses (mostly circles) indicate the 3σ "active search" regions for successful (red) and unsuccessful (blue) observations. The remaining landmarks in this submap (yellow) were not measured in this frame. (b) The discrepancy between the predicted (•) and true image location (□) of the landmarks in the first submap at the moment of loop closure, caused by accumulation of errors in the pose as the camera traveled around the loop. (c) Overhead view of the map estimated by the SLAM system before the loop closure has been completed. More distant landmarks than those on the dinosaur head are outside the borders of the figure. The inset shows how the two sets of poses from the common trajectory would overlap if the state estimate were perfect. (d) Global map after the loop closure has been taken into account and a new edge has been added to the submap graph. Further details appear in the main text.

position is determined by common landmarks between neighboring submaps. On the global level, the map is represented by a graph with a submap at each node and edges representing the transformations between the submaps. Closing a loop involves inserting a new edge into this graph between two submaps which have been found to overlap. More detail on this technique can be found in [6], where we have extended the technique to multiple loops and the retraversal of old submaps.

The relocalization module is used to detect loop closure events irrespective of which is the current submap. While tracking in the current submap, the system attempts in turn to relocalize in each of the previous submaps. When one of these relocalization attempts is successful, the system tracks the camera in this old submap while continuing SLAM in the current submap, analogous to the case of map merging. Likewise, after a few seconds of successful tracking indicating a true overlap, the poses from the two trajectories are used to determine the relative transformation between the submaps, and a new edge is added to the graph for the global map.

This method was used to close the loop shown in Fig. 12. The sequence begins with the camera close to the dinosaur's eye. As it then moves along the head, it builds a series of submaps labeled 1-6 in the graph-based representation drawn alongside the global metric map. The global map is drawn relative to submap 6 and the arrows along the edges indicate the graph traversal used to draw the global map. Ellipses indicate the 3σ position uncertainty associated with the origin of each submap.

After reaching the end of the dinosaur head, the camera moves backward to see the whole room, as shown in Fig. 12a. The original landmarks cannot be observed due to the very large scale difference. In submap 6, the camera again approaches the front of the head and translates back up to the eye. The original landmarks from submap 1 come into view again, but due to inaccuracies in the camera estimate, the predicted image position for the landmarks is quite far from the true position, as shown in Fig. 12b. Perhaps, the system could successfully observe the landmarks in this case using large search ellipses and Joint Compatibility to determine data association. However, the inaccuracies grow

with the size of the loop and these methods will eventually fail. A loop closure detection algorithm must be able to determine the pose with no initial prior.

After relocalization in the first submap, the system tracks in both submaps and keeps two poses from this common trajectory in the state vector. Note that the two sets of poses in Fig. 12c would overlap if the estimated camera pose and global map were correct. The system uses this constraint to determine the transformation between the two submaps and adds a new edge to the global submap graph to close the loop. The global map can be redrawn with the loop in the map closed as shown in Fig. 12d. A comparison of this loop closure method to other techniques is given in [6] which also includes examples of mapping in larger multiloop environments.

7 Discussion

In this paper, we have described a relocalization module for monocular SLAM systems. Relocalization is performed by first using a randomized lists classifier to establish landmark correspondences in the image and then RANSAC to determine the pose robustly from these correspondences. The classifier is ideally suited to this scenario since it can achieve high recall by taking advantage of online learning of appearance and it provides the fast recognition required for a real-time system.

This module allows several improvements to be made to the monocular SLAM system. The architecture of the complete system is outlined in Algorithms 2 and 3. The system monitors tracking performance and initiates recovery if tracking fails due to sudden motion, occlusion, or any other difficulty. If the camera is not pointed back at a mapped region after such a failure, then recovery is not possible. Instead, an independent map is started which can later be connected to the original map if an overlap is detected. Similarly, the system searches for overlaps within the same map to detect loop closures and so improves the global map structure. The full system, with the improvements made possible by the relocalization module, is able to perform more reliably and accurately both for longer times and in larger environments.

The main limitation of the proposed system is the large memory requirement of the randomized lists classifier. More sophisticated storage methods could be used to compress the classification data for a smaller memory footprint. This remains for future work.

Acknowledgments

This work was supported by the EPSRC through grants GR/T24685, GR/S97774, and EP/D037077 and a studentship to BW. The authors are grateful for discussions with Andrew Davison, Juan Tardós, José Neira, David Murray, and Tom Drummond, and to the Ashmolean and Natural History Museums for granting filming permission.

References

[1] A. Davison, I. Reid, N. Molton, and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 29, no. 6, pp. 1052-1067, June 2007.

[2] V. Lepetit and P. Fua, "Keypoint Recognition Using Randomized Trees," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 28, no. 9, pp. 1465-1479, Sept. 2006.

[3] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo Localization for Mobile Robots," *Proc. Int'l Conf. Robotics and Automation,* 1999.

[4] J. Neira, J.D. Tardós, and J.A. Castellanos, "Linear Time Vehicle Relocation in SLAM," *Proc. Int'l Conf. Robotics and Automation,* 2003.

[5] L. Clemente, A. Davison, I. Reid, J. Neira, and J.D. Tardós, "Mapping Large Loops with a Single Hand-Held Camera," *Proc. Robotics: Science and Systems Conf.,* 2007.

[6] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid, and J. Tardós, "A Comparison of Loop Closing Techniques in Monocular SLAM," *Robotics and Autonomous Systems,* vol. 57, no. 12, pp. 1188-1197, 2009.

[7] G. Reitmayr and T. Drummond, "Going Out: Robust Model-Based Tracking for Outdoor Augmented Reality," *Proc. IEEE Int'l Symp. Mixed and Augmented Reality,* 2006.

[8] G. Klein and D. Murray, "Improving the Agility of Keyframe-Based SLAM," *Proc. 10th European Conf. Computer Vision,* pp. 802-815, 2008.

[9] M. Cummins and P. Newman, "FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance," *Int'l J. Robotics Research,* vol. 27, no. 6, pp. 647-665, 2008.

[10] E. Eade and T. Drummond, "Unified Loop Closing and Recovery for Real Time Monocular Slam," *Proc. British Machine Vision Conf.,* pp. 53-62, 2007.

[11] B. Williams, P. Smith, and I. Reid, "Automatic Relocalisation for a Single-Camera Simultaneous Localisation and Mapping System," *Proc. Int'l Conf. Robotics and Automation,* 2007.

[12] B. Williams, G. Klein, and I. Reid, "Real-Time SLAM Relocalisation," *Proc. IEEE Int'l Conf. Computer Vision,* 2007.

[13] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid, and J.D. Tardós, "An Image-to-Map Loop Closing Method for Monocular SLAM," *Proc. IEEE Int'l Conf. Intelligent Robots and Systems,* 2008.

[14] J. Neira and J.D. Tardós, "Data Association in Stochastic Mapping Using the Joint Compatibility Test," *IEEE Trans. Robotics and Automation,* vol. 17, no. 6, pp. 890-897, Dec. 2001.

[15] J.M.M. Montiel, J. Civera, and A.J. Davison, "Unified Inverse Depth Parametrization for Monocular SLAM," *Proc. Robotics: Science and Systems,* 2006.

[16] M.A. Fischler and R.C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Comm. ACM,* vol. 24, no. 6, pp. 381-395, 1981.

[17] M. Ozuysal, V. Lepetit, F. Fleuret, and P. Fua, "Feature Harvesting for Tracking-by-Detection," *Proc. European Conf. Computer Vision,* pp. 592-605, 2006.

[18] Y. Amit and D. Geman, "Shape Quantization and Recognition with Randomized Trees," *Neural Computation,* vol. 9, no. 7, pp. 1545-1588, 1997.

[19] M. Ozuysal, P. Fua, and V. Lepetit, "Fast Keypoint Recognition in Ten Lines of Code," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2007.

[20] D.G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int'l J. Computer Vision,* vol. 60, no. 2, pp. 91-110, 2004.

[21] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," *Proc. European Conf. Computer Vision,* pp. 430-443, 2006.

[22] M. Calonder, V. Lepetit, and P. Fua, "Keypoint Signatures for Fast Learning and Recognition," *Proc. European Conf. Computer Vision,* 2008.

Brian Williams received the PhD degree in engineering science from the University of Oxford in 2009. His thesis covered monocular SLAM systems for robotics and augmented reality. After leaving Oxford, he worked at the University of Cambridge on visual navigation and reconstruction for an robot airship. He is now a NASA postdoctoral research fellow at the Jet Propulsion Laboratory working on visual navigation for the proposed aerobot mission to Titan. He is a member of the IEEE.

Georg Klein graduated from the University of Cambridge in 2006 and was a postdoctoral research assistant in Oxford's Active Vision Laboratory until 2009. He is a scientist at Microsoft Corporation. His research interest is mainly the development and application of computer vision techniques for Augmented Reality. He is a member of the IEEE.

Ian Reid is a reader in engineering science and fellow of Exeter College at the University of Oxford, where he jointly heads the Active Vision Group. His research has touched on many aspects of computer vision, concentrating on algorithms for visual tracking, control of active head/eye robotic platforms (for surveillance and navigation), SLAM, visual geometry, novel view synthesis, and human motion capture. He serves on the editorial boards of the *Image and Vision Computing Journal* and the *IPSJ Transactions on Computer Vision Applications*. He is a member of the IEEE.