Keypoint Signatures for Fast Learning and Recognition*

Michael Calonder Vincent Lepetit Pascal Fua

Computer Vision Laboratory, EPFL, Switzerland
Email: {michael.calonder,vincent.lepetit,pascal.fua}@epfl.ch

Abstract. Statistical learning techniques have been used to dramatically speed-up keypoint matching by training a classifier to recognize a specific set of keypoints. However, the training itself is usually relatively slow and performed offline. Although methods have recently been proposed to train the classifier online, they can only learn a very limited number of new keypoints. This represents a handicap for real-time applications, such as Simultaneous Localization and Mapping (SLAM), which require incremental addition of arbitrary numbers of keypoints as they become visible.
In this paper, we overcome this limitation and propose a descriptor that can be learned online fast enough to handle virtually unlimited numbers of keypoints. It relies on the fact that if we train a Randomized Tree classifier to recognize a number of keypoints extracted from an image database, all other keypoints can be characterized in terms of their response to these classification trees. This signature is fast to compute and has a discriminative power that is comparable to that of the much slower SIFT descriptor.

1 Introduction

Feature point recognition and matching are crucial for many vision problems, such as pose estimation or object detection. Most current approaches rely on local descriptors designed to be invariant, or at least robust, to affine deformations. Among these, the SIFT descriptor [1] has been shown to be one of the most effective [2] and its recognition performance can be further enhanced by affine rectifying the image patches surrounding the feature points [3]. However, such descriptors are relatively slow and cannot be used to handle large numbers of feature points in real-time.

Faster SIFT-like descriptors such as SURF [4] achieve 3 to 7-fold speed-ups by exploiting the properties of integral images. However, it has recently been shown that even shorter run-times can be obtained without loss in discriminative power by reformulating the matching problem as a classification problem. This approach relies on an offline training phase during which multiple views of the feature points to be matched are used to train a classifier to recognize them

* This work has been supported in part by the Swiss National Science Foundation.

based on a few pairwise intensity comparisons [5, 6]. The resulting run-time computational complexity is much lower than that of SIFT-like descriptors while preserving robustness to viewpoint and lighting changes. However, this comes at the cost of an offline training phase that is relatively slow and ill-adapted to real-time applications, such as Simultaneous Localization and Mapping (SLAM), which require learning the appearance of new feature points as they become visible. As a result, real-time algorithms that rely on this approach keep a very tight bound on the number of keypoints they work with [7].

In this paper, we remove this limitation and show that online learning of keypoint appearance can be made fast enough for a real-time system to handle a virtually unlimited number of keypoints. At the heart of our approach that we will refer to as a Generic Tree (GT) algorithm is the following observation: If we train a Randomized Tree classifier [5] to recognize a number of keypoints extracted from an image database, all other keypoints can be characterized in terms of their response to these classification trees, which we will refer to as their *signature*. Because the signature can be computed very fast, the learning becomes quasi-instantaneous and therefore practical for online applications. We attribute this desirable behavior to the fact that, assuming the initial set of keypoints is rich enough, the new keypoints will be similar to some of those initial points and the signature will summarize these similarities. In other words, we replace the hand-crafted SIFT descriptor by one that has been empirically learned from training data to be very selective. Remarkably, this can be done using a fairly limited number—300 in our experiments—of initial keypoints.

In the remainder of the paper, we first discuss related work. We then describe our method and compare it against state-of-the-art ones on standard benchmark images. Finally, we show that it can be integrated into a SLAM algorithm to achieve real-time performance.

2 Related Work

As discussed in the introduction, state-of-the-art approaches to feature point matching can be classified into two main classes.

Those in the first class rely on local descriptors designed to be invariant, or at least robust, to specific classes of deformations [8, 1]. They often require scale and rotation estimates provided by a keypoint detector. Among these, the SIFT descriptor [1], computed from local histograms of gradients, has been shown to work remarkably well, especially if one rectifies the image patches surrounding the feature points [2, 3]. We will therefore use it as a benchmark against which we will compare the performance of our approach. However, it must be noted that, because the SIFT descriptor is complex, it is also relatively slow to evaluate. On a modern PC, it takes approximately 1ms per feature point[1], which limits the number of feature points that can be handled simultaneously to less than 50 if

[1] Some commercial implementations of SIFT, such as the one by Evolution Robotics, can be up to 10 times faster by using careful coding and processor extensions, but those are not easily available. Furthermore, this would not represent a fair compari-

one requires frame-rate performance. SURF [4] is closely related to SIFT and achieves a 3 to 7-fold speed increase by efficiently using integral images and box filters to compute the descriptor, which means that from 150 to 350 keypoints could be handled. By contrast, our approach can compute several thousand signatures at frame-rate.

Of course, SIFT and SURF are nevertheless effective for well-designed real-time applications. For example, it has been shown that feature points can be used as visual words [9] for fast image retrieval in very large image databases [10]. The feature points are labeled by hierarchical k-means clustering of their SIFT descriptors, which allows to use very many visual words. However, the performance is measured in terms of the number of correctly retrieved documents rather than the number of correctly classified feature points. For applications such as pose estimation or SLAM, the latter criterion is much more important.

A second class of approaches to feature point matching relies on statistical learning techniques to compute a probabilistic model of the patches surrounding them. The one-shot approach of [11] uses PCA and Gaussian Mixture Models but does not account for perspective distortion. Since the set of possible appearances of patches around an image feature, seen under changing perspective and lighting conditions, can be treated as a class, it was later shown that a classifier based on Randomized Trees [12] can be trained to recognize them [5] independently of pose. This is done using a database of patches that is obtained by warping keypoints of a reference image by randomly chosen homographies. The resulting algorithm has very fast run-time performance but requires a computationally training phase that precludes online learning of new feature points. This limitation has been partially lifted by optimizing the design of the classifier and exploiting the power of modern graphic cards [7], but still only allows for learning small numbers of new feature points. By contrast, the method we propose here can learn virtually unlimited numbers of new signatures at a very small computational cost.

3 Generic Trees

The idea behind our Generic Trees (GTs) method is to take advantage of a fast classifier to efficiently compute short description vectors, or signatures, for arbitrary keypoints.

We start from a relatively small set of keypoints that we extracted from images such as those of Fig. 1. We refer to this set as *base set* and train a Randomized Tree classifier to recognize the keypoints in the base set under arbitrary perspective, scale, and lighting conditions [5]. Given a new keypoint that is *not* in the base set, we show below that the classifier responds to it in a way that is also stable to changes in scale, perspective, and lighting. We therefore take this response to be the compact and fast-to-compute signature we are looking for.

son to our own implementation that does not include any such coding but could be similarly sped-up.

Fig. 1. 3 out of 7 landscape images from which the base set of keypoints were extracted.

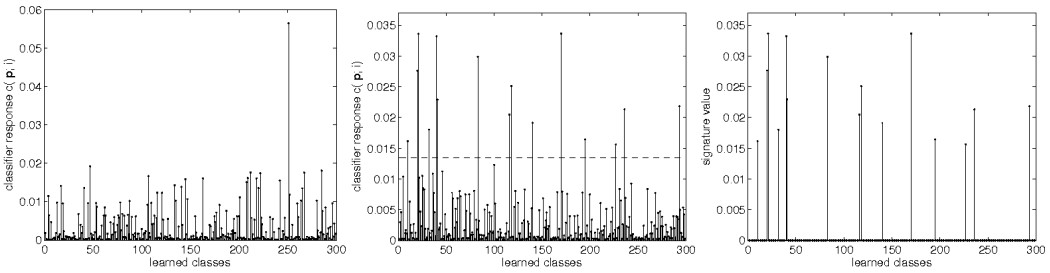

Fig. 2. LEFT: Response of trained Randomized Trees to a patch around a keypoint in the base set, $N = 300$. The response has typically only one spike. MIDDLE: Response to a patch of a keypoint that is *not* in the base set. The response has several peaks which are stable under viewpoint variation. RIGHT: A typical signature. The thresholding process of Eq. 2 takes the raw classifier response and sets all values below t to zero.

3.1 Stable Signatures

More formally, every keypoint $\mathbf{u}_i \in \mathbb{R}^2$ in the beforementioned base set is related to exactly one point $\mathbf{k}_i$ in 3D. Let us start with a set of N points $\mathbf{K} = \{\mathbf{k}_1, \ldots, \mathbf{k}_N\}$, $\mathbf{k}_i \in \mathbb{R}^3$, and refer to N as *base size*. We then build a classifier based on Randomized Trees that is able to recognize the $\mathbf{k}_i$ under varying viewing conditions [5]. Let $\mathbf{p}_i$ be the patch centered on $\mathbf{u}_i$. Then the classifier provides a function $\mathcal{C}(\mathbf{p}_i)$ mapping a patch $\mathbf{p}_i$ to a vector in $\mathbb{R}^N$. Using the notation $\mathcal{C}^{(j)}(\mathbf{p}_i)$ to refer to the j-th element of the vector $\mathcal{C}(\mathbf{p}_i)$, $1 \leq j \leq N$, we can state a special property of $\mathcal{C}$:

$$\mathcal{C}^{(j)}(\mathbf{p}_i) \quad \text{is} \quad \begin{cases} \text{large} & \text{if } j = i \\ \text{small} & \text{otherwise} \end{cases} .$$

This is shown in Fig. 2-LEFT for $i = 250$ and $N = 300$.

Furthermore, let $\mathcal{T}(\mathbf{p}, \boldsymbol{\Theta})$ be a transformation of an image patch $\mathbf{p}$ under viewing condition change $\boldsymbol{\Theta}$. $\boldsymbol{\Theta}$ typically encodes changes in illumination, viewpoint, or scale. If the classifier has been trained well, we can assume that

$$\forall \boldsymbol{\Theta} \quad : \quad \mathcal{C}(\mathbf{p}) \approx \mathcal{C}\big(\mathcal{T}(\mathbf{p}, \boldsymbol{\Theta})\big) . \tag{1}$$

When we consider a new 3D-point κ that does *not* belong to $\mathbf{K}$ and center a patch $\mathbf{q}$ on the keypoint corresponding to κ, we can define the signature of the patch $\mathbf{q}$ simply as

$$\mathsf{signature}(\mathbf{q}) = \mathcal{C}(\mathbf{q}).$$

A patch $\mathbf{q}'$ centered on the keypoint of κ in another image can be written as $\mathcal{T}(\mathbf{q}, \boldsymbol{\Theta})$, for some $\boldsymbol{\Theta}$. Under the assumption of Eq. 1, the signature of $\mathbf{q}'$ is equal to the signature of $\mathbf{q}$ because

$$\mathsf{signature}(\mathbf{q}') = \mathcal{C}(\mathbf{q}') = \mathcal{C}\big(\mathcal{T}(\mathbf{q}, \boldsymbol{\Theta})\big) = \mathcal{C}(\mathbf{q}) = \mathsf{signature}(\mathbf{q}).$$

In other words, the signature is stable under changes in viewing conditions.

3.2 Sparse Signatures

Because κ is not a member of $\mathbf{K}$, the response of the Trees to the corresponding patch $\mathbf{q}$, $\mathcal{C}(\mathbf{q})$, has typically more than one peak. Such a response is shown in Fig. 2-Middle. In practice, only few of the values in $\mathcal{C}(\mathbf{q})$ are large. We therefore replace the signature above by one which is much sparser:

$$\mathsf{signature}(\mathbf{q}) = \mathrm{th}(\mathcal{C}(\mathbf{q})) = \left[\mathrm{th}(\mathcal{C}^{(1)}(\mathbf{q})), \ldots, \mathrm{th}(\mathcal{C}^{(N)}(\mathbf{q}))\right]^\top, \tag{2}$$

where $\mathrm{th}(\cdot)$ is a thresholding function:

$$\mathrm{th}(x) = \begin{cases} x \text{ if } x \geq t \\ 0 \text{ if } x < t \end{cases}. \tag{3}$$

As shown in Fig. 2-RIGHT, most of the values in the signatures are null and matching reduces to computing the Euclidean distance between two (sparse) signatures. To this end, efficient approaches exist and we use the best-bin-first (BBF) algorithm [13] to match signatures.

The threshold t is a free parameter the user has to provide and it depends on the base size N, because the response of the classifier is normalized. Its impact on the accuracy and the speed of the classifier will become clear in Section 4.

3.3 Base Set and Trees

As discussed above, we extract the base set from landscape images such as those of Fig. 1. It is worth noting that we have tried extracting them from other kinds of images, such as pictures of indoor scenes or animals, which has not resulted in any appreciable change in the performance of our algorithm. In other words, our experiments show that we can extract the base set from almost any kind of image as long as it exhibits enough structure and variety.

In practice, we use the DOG/SIFT feature detector, which typically finds several thousands of keypoints in each image. To create a base set of size N, we *randomly* select N of these keypoints, the only constraint being that they should be at least 5 pixels away from each other.

We then train a Randomized Trees classifier to recognize the resulting keypoints. Fig. 3 illustrates the response of our GTs to arbitrary patches, given this classifier. Roughly speaking, computing their signature amounts to finding the subset of the base set they most resemble to.

Fig. 3. The leftmost image of each row represents a patch from a test image. The remaining images in the same row represent the patches surrounding the 10 keypoints the test patch looks most similar to according to our Randomized Tree classifier, in decreasing similarity order.

Fig. 4. Images for the Wall (LEFT) and Light (RIGHT) datasets.

4 Results

We first use three publicly available datasets to characterize the behavior of our GTs and then to compare their performance to SIFT, which is widely acknowledged as one of the most effective descriptor in terms of recognition rate. We then demonstrate that the GTs can be effectively integrated into a SLAM algorithm. The resulting system has the ability to learn the signatures of a virtually unlimited number of keypoints and therefore to detect them at every time step without having to depend on knowing the previous camera pose, thus giving it great robustness to occlusions and abrupt camera motions.

4.1 Performance Evaluation

The three image datasets we used for our experiments –Wall[2], Light[2], and Fountain[3]—are depicted by Figs. 4 and 5. The Wall and Light scenes are planar and the relationship between two images in the database can be expressed by a homography. By constrast the Fountain scene is fully three-dimensional and we have access to an accurate laser-scan that can be used to establish explicit one-to-one correspondences at arbitrary locations.

[2] Available at `http://www.robots.ox.ac.uk/~vgg/research/affine/`.
[3] Available at `http://cvlab.epfl.ch/~strecha/multiview/`.

Fig. 5. Two 1440 × 960 images from the Fountain dataset.

Given several images from the same database, we take $m \leftrightarrow n$ to indicate that we use image m as a reference image from which we extract keypoints that we try to match in image n. For the Wall dataset, we tested $1 \leftrightarrow 2$ and $1 \leftrightarrow 3$, corresponding to a change in camera orientation of 20 and 40 degrees, respectively. From the Light dataset, we tested $1 \leftrightarrow 2$, $1 \leftrightarrow 3$, and $1 \leftrightarrow 4$. From the Fountain dataset, we used $1 \leftrightarrow 2$ and $1 \leftrightarrow 3$.

We define the *recognition rate*, the main performance criterion, as the ratio of the number of correct matches to the total number of interest-points in the reference image. For all tests described in this section, we proceed as follows. We extract a number of DOG/SIFT keypoints from the reference image and compute the coordinates of their corresponding points in the test image using the known geometric relationship between the two. We then compute the SIFT descriptors and GT signatures for both sets of test points and store them in two separate test image databases. Matching a point in the reference image then simply amounts to finding the most similar point in terms of either its descriptor or its signature in the test image database. Note that not detecting interest-points in the test image but using geometry instead prevents repeatability problems of the keypoint detector from influencing our results. Furthermore, since we apply the same procedure for SIFT and for GTs, we do not favor either technique over the other.

Signature Length and Base Size There are only two parameters in GTs: The threshold t and the base size N. t implicitely determines the *signature length*, that is, the number of non-zero entries in the signature. This length, in turn, has a direct impact on the recognition rate. However, it does *not* impact the time it takes to compute the descriptor, it only affects the time it takes to match: The sparser the signature, the less computation is required to compare a given number of signatures. Hence, t controls the trade-off between the recognition rate and the computational effort one is willing to make. By contrast, increasing N slows down both signature computation and matching.

Fig. 6-LEFT shows the distribution of the signature length computed using 1000 keypoints on the Fountain dataset $1 \leftrightarrow 2$ for $t = 0.01$. This yields a mean

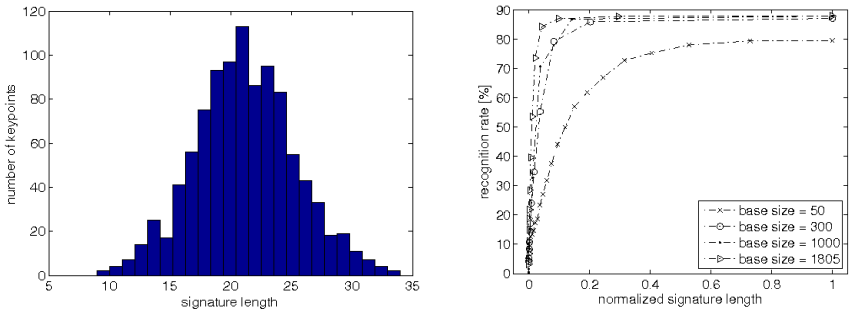

Fig. 6. LEFT: Typical distribution of the GT signature length. The signatures were computed for 1000 keypoints from the Fountain image pair $1 \leftrightarrow 2$. The mean length is 21.2 with a standard deviation of 3.9. RIGHT: Recognition rate as a function of normalized signature length for base sizes ranging from 50 to 1805. The *normalized* signature length is taken to be the ratio of the number of non-zero entries in the signature and the base size N.

signature length of 21.2 with standard deviation 3.9. 95% of the signatures have a length in the range $\{13, ..., 29\}$ which is a very compact representation. The recognition rate for this case amounts to 75.4%.

In Fig. 6-RIGHT, we plot the recognition rate as a function of the signature length for different values of N, using the Fountain dataset $1 \leftrightarrow 2$. We see that $N = 50$ is too small to achieve the best possible performance of 87.9%, but that going beyond $N = 300$ does not bring any significant improvement. In the remainder of the paper, we will therefore use $N = 300$ for our experiments.

Comparing Recognition Rates Fig. 7 illustrates our results on the Wall database. On the left, we plot the relationship between the signature length and $\frac{1}{\sqrt{t}}$, which is roughly linear. On the right, we plot the performance of GTs as a function of $\frac{1}{\sqrt{t}}$ for two image pairs and we represent the corresponding performance of SIFT by a horizontal line. In Fig. 8, we plot similar graphs for the Fountain and Light datasets. Note that in all cases, GTs rapidly reach their peak performance as $\frac{1}{\sqrt{t}}$ increases, that is, as t decreases. In practice, this means that as long as t is not taken to be too large, its exact value has little influence on the algorithm's recognition performance.

GTs perform a bit better than SIFT on the Light dataset, slightly worse on the Wall dataset, and almost identically on the Fountain dataset. In other words, in terms of recognition performance, GTs and SIFT are almost equivalent.

Comparing Computation Times Performing a completely fair comparison between the SIFT and GTs is non-trivial. SIFT re-uses intermediate data from the keypoint extraction to compute canonic scale and orientations and the descriptors, while Randomized Trees only require keypoint locations and can therefore work with arbitrary detectors. On the other hand, the distributed SIFT C

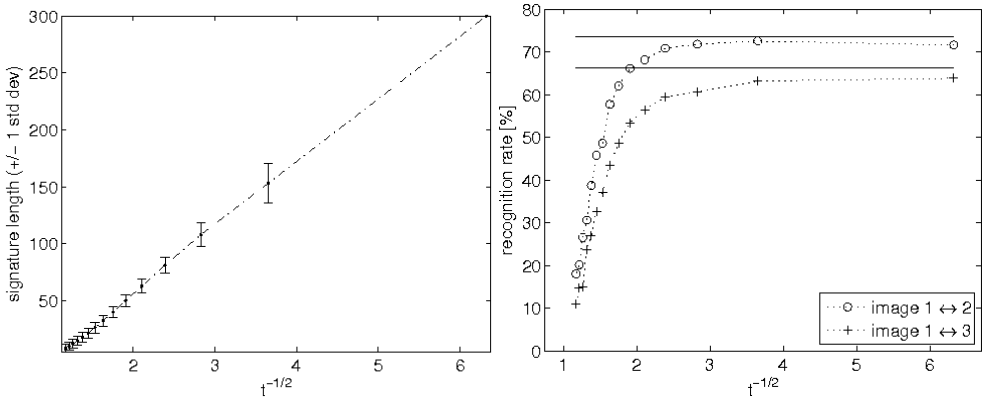

Fig. 7. Wall dataset results. LEFT: Signature length as a function of $\frac{1}{\sqrt{t}}$. The relationship is roughly linear. RIGHT: Recognition rates as a function of $\frac{1}{\sqrt{t}}$. The top curve corresponds to image pair $1 \leftrightarrow 2$, the lower one to $1 \leftrightarrow 3$. The horizontal lines denote the corresponding SIFT results.

code is not optimized for the very last cycle[4]. However, the level of optimization of our GT code being roughly comparable, we regard this as a fair comparison. In [6], the authors detail what makes the original Trees much faster at classification in comparison to SIFT. The arguments they put forth carry directly over to the GTs.

Here we compare the CPU time SIFT and GTs require to compute the descriptors.[5] Postprocessing steps are excluded, in particular matching, it can be done efficiently for both SIFT descriptors and GT signatures as discussed in Section 3. The times are given in Fig. 9 for 1000 SIFT keypoints on the Fountain dataset. Overall, we obtain a 35-fold speedup for the GTs with respect to SIFT. Note that it would *not* make much sense to compare only the time for feature vector computation of SIFT against the total time of GTs since SIFT cannot forgo the preprocessing stage that computes the orientation and scale of the features. However, to ensure fairness, we added a "virtual" amount of time on the GT side to account for the time spent by an efficient detection algorithm such as FAST [14] to detect 1000 interest-points.

As discussed earlier, faster SIFT-like detectors such as SURF [4] have been proposed and yield a 3 to 7-fold speed increase, which still leaves GTs with a 5 to 11-fold speed advantage.

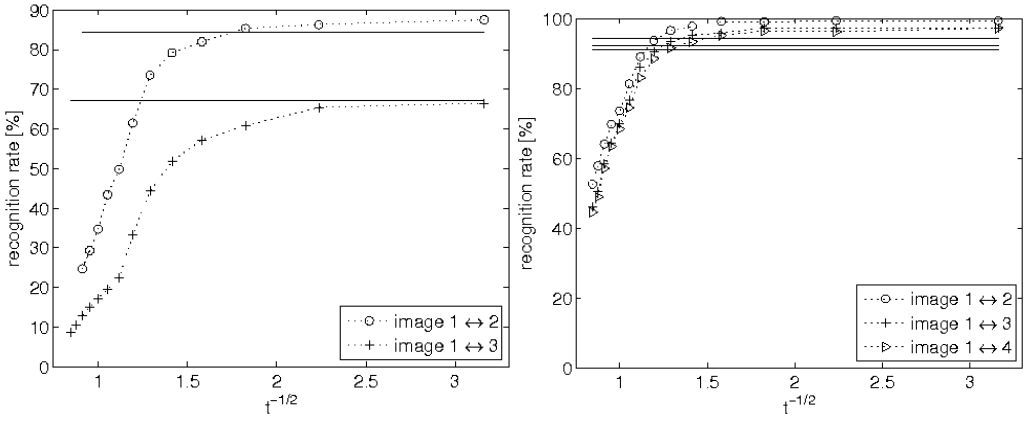

Fig. 8. LEFT: Recognition rates as a function of $\frac{1}{\sqrt{t}}$ for the Fountain dataset. The top curve corresponds to image pair $1 \leftrightarrow 2$ and the lower one to $1 \leftrightarrow 3$. RIGHT: Recognition rates as a function of $\frac{1}{\sqrt{t}}$ for the Light dataset. The three curves correspond to image pairs $1 \leftrightarrow 2$, $1 \leftrightarrow 3$, and $1 \leftrightarrow 4$. The horizontal lines denote the corresponding SIFT results.

4.2 SLAM using Generic Trees

In this section we demonstrate that GTs can increase the robustness of a Visual SLAM system. Their role is twofold. We first use them to bootstrap the system by localizing the camera with respect to a known planar pattern in the scene. Second, we incrementally train a second set of GTs to recognize new landmarks reconstructed by the system. They make the system robust against severe disturbances such as complete occlusion or strong shaking of the camera, as evidenced by the smoothness of the recovered camera trajectory in Fig. 10.

For simplicity, we use a FastSLAM [15, 16] approach with a single particle to model the distribution over the camera trajectory. This is therefore a simplified version of FastSLAM, but our GTs are sufficiently powerful to make it robust.

As discussed above, we use two different sets of GTs. We will refer to the first set as "Offline GTs", that we trained offline to recognize keypoints on the planar pattern. It is visible in the first frame of the sequence to bootstrap the system and replaces the four fiducials used by many other systems. We show the initialization process in Fig. 10 (a). This increases the flexibility of our system since we can use any pattern we want provided that it is textured enough. The second set of GTs, the "Online GTs", are incrementally trained[6] to recognize the 3D landmarks the SLAM system discovers and reconstructs.

[4] In the author's words: "[SIFT was] implemented as efficiently as possible while still maintaining intuitive code".

[5] All experiments were run on a Single-Threaded 2 GHz Intel Xeon machine.

[6] In this context, *to train* simply means *computing the signature and adding it to the database*.

Task	Time
Gaussian pyramid	1434 ms
DOG pyramid	277.4 ms
Feature scales	0.2362 ms
Feature orientations	91.34 ms
Assemble final descriptor	339.2 ms
Total time	2142 ms

(a) SIFT

Task	Time
Keypoint detection (est'd)	5 ms
Compute base distribution	33.21 ms
Thresholding	1.217 ms
Total time	39.43 ms

(b) GTs

Fig. 9. Total time and time required by substeps in SIFT and GTs. The entry *Keypoint detection* in (b) has been added to make the comparison more fair. The value was estimated from [14].

Our complete algorithm goes through the following steps:

1) Initialize the camera pose and some landmarks by detecting a known pattern using the Offline GTs.
2) Detect keypoints and match them using the Offline and Online GTs against the known landmarks.
3) Estimate the camera pose from these correspondences using a P3P algorithm and RANSAC [17]. The estimated pose is refined via a non-linear optimization.
4) Refine the location estimates of the inlier landmarks using an Extended Kalman filter.
5) Create new landmarks. Choose a number of detected keypoints that *do not* belong to any landmark in the map and initialize the new landmarks with a large uncertainty along the line of sight and a much smaller uncertainty in the camera's lateral directions.
6) Retrain GTs with good matches from 2) and the new landmarks.
7) Loop to step 2.

With this system we demonstrate that both smooth tracking and recovery from complete failure can be naturally integrated by employing GTs for the matching task.

The reconstructed trajectory in Fig. 10 (f) shows only tiny jags at the order of a few millimeters and appears as smooth as a trajectory that was estimated in a filtered approach to SLAM, e.g. MonoSLAM [18, 19]. This is especially noteworthy as the camera's state is re-estimated from scratch *in every frame* and there is no such thing as a motion model.[7] At the same time, this is a strong indication for an overall correct operation, since an incorrect map induces an unstable state estimation and vice versa. In total the system mapped 724

[7] Other systems commonly use a motion model to predict the feature location in the next frame and accordingly restrict the search area for template matching.

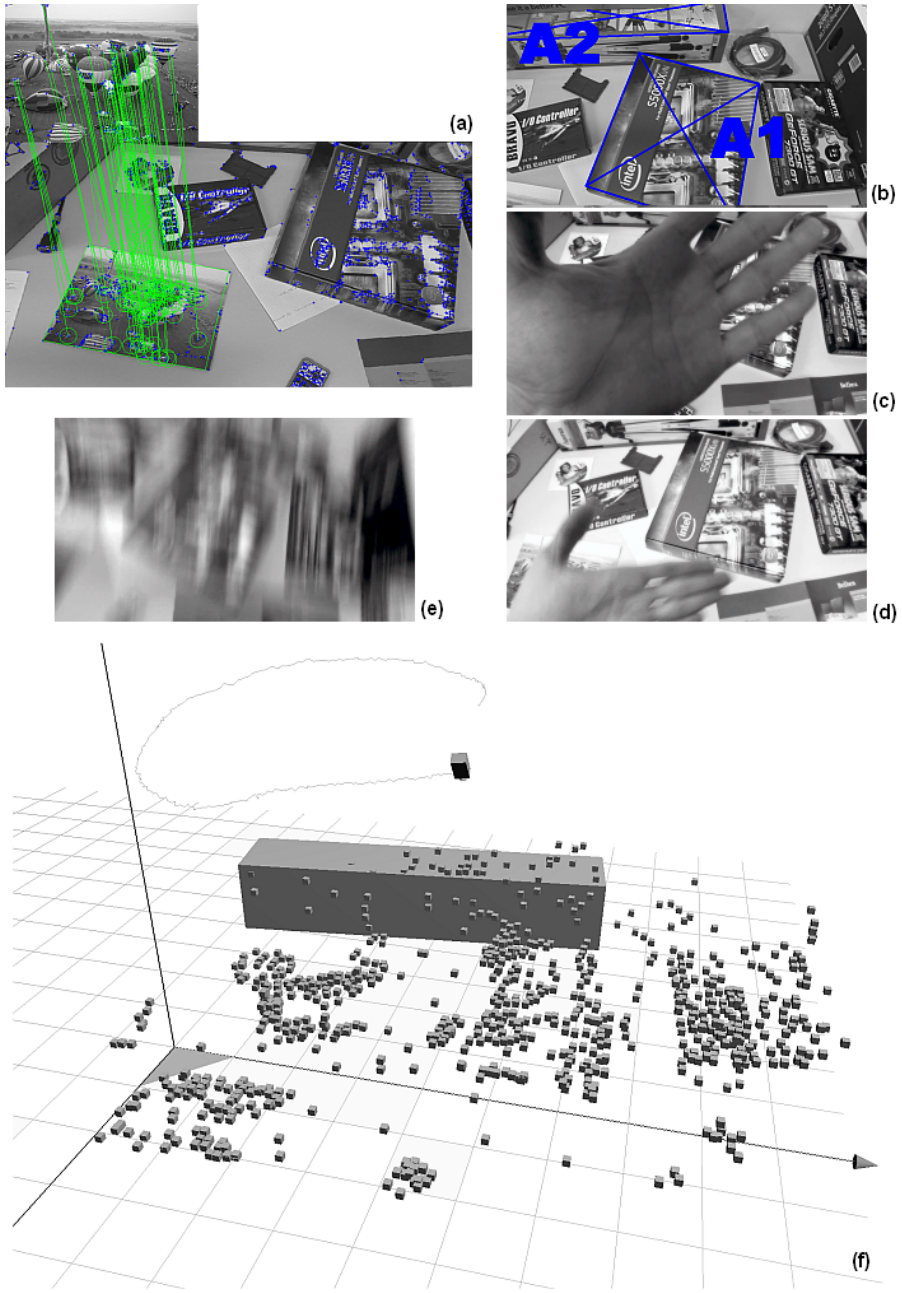

Fig. 10. GTs applied to SLAM. (a) Initialization: The pattern at the top is detected in the image yielding initial landmarks and pose. (b-e) Four images from the sequence, which features both partial/total occlusions, strong camera shaking, and motion blur. (f) The reconstructed trajectory and landmarks. The latter are depicted as cubes centered around the current state of the corresponding EKF. Note how smooth the trajectory is, given the difficulties the algorithm had to face. The rectangular box corresponds to a true object in the scene and was inserted manually in order to support the visual inspection of the scene.

landmarks and ran stable over all 2554 frames of the sequence. A few frames are shown in Fig. 10 (b–e).

Recently, [7] presented a system that is also capable of recovering from complete failure. They achieved robustness with a hybrid combination between template matching and a modified version of Randomized Trees. However, their map typically contains one order of magnitude fewer landmarks and there has been no indication that the modified Trees will still be capable of handling a larger number of interest-points.

The system successfully passed a rough but quantitative validation step. First, we checked the relative accuracy for reconstructed pairs of 3D points and we found an error between 3.5 to 8.8% on their Euclidean distances. Second, the absolute accuracy was assessed by choosing two world planes A_1 and A_2 parallel to the ground plane. They are shown in Fig. 10 (a). We then measured their z-coordinates z_k^*, $k = \{1, 2\}$, by hand and computed for each of them the RMS error

$$e_k = \left(\frac{1}{|\mathbf{R}|} \sum_{\mathbf{R}} (z_i - z_k^*)^2 \right)^{\frac{1}{2}}$$
$$\text{with } \mathbf{R} = \{(x_i, y_i, z_i) \in \mathbf{L} \mid x_i \in [x_{\min}, x_{\max}], y_i \in [y_{\min}, y_{\max}]\}.$$

$\{x, y\}_{\min,\max}$ are the plane bounds and $\mathbf{L}$ is the set of all landmarks. We found $e_1 = 7$ mm and $e_2 = 10$ mm. Given that the camera is at roughly 0.6 to 1.4 m from the points under consideration, this represents a good accuracy.

5 Conclusion and Future Work

We have proposed a method that combines the strengths of two fundamentally different approaches to patch recognition. On one hand, the SIFT descriptor [1] established a good reputation regarding accuracy at the expense on computation time. On the other hand, statistical learning based approaches were found to be very fast at runtime but need an offline training phase.

The Generic Trees presented in this work achieve both speed and robustness to perspective and illumination changes by computing a signature based on the response of a statistical classifier trained using a small set of keypoints. In our current implementation, this set has neither been engineered nor selected to achieve maximal performance. We therefore see a large potential for improvement by seeking to optimize our choice of base keypoints.

References

1. Lowe, D.: Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision **20** (2004) 91–110
2. Mikolajczyk, K., Schmid, C.: A Performance Evaluation of Local Descriptors. In: Conference on Computer Vision and Pattern Recognition. (2003) 257–263
3. Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T., Van Gool, L.: A comparison of affine region detectors. International Journal of Computer Vision **65** (2005) 43–72

4. Bay, H., Tuytelaars, T., Van Gool, L.: SURF: Speeded up robust features. In: European Conference on Computer Vision. (2006)
5. Lepetit, V., Fua, P.: Keypoint recognition using randomized trees. IEEE Transactions on Pattern Analysis and Machine Intelligence **28** (2006) 1465–1479
6. Ozuysal, M., Fua, P., Lepetit, V.: Fast Keypoint Recognition in Ten Lines of Code. In: Conference on Computer Vision and Pattern Recognition, Minneapolis, MI (2007)
7. Williams, B., Klein, G., Reid, I.: Real-time slam relocalisation. In: International Conference on Computer Vision. (2007)
8. Schmid, C., Mohr, R.: Local Grayvalue Invariants for Image Retrieval. IEEE Transactions on Pattern Analysis and Machine Intelligence **19** (1997) 530–534
9. Sivic, J., Zisserman, A.: Video Google: Efficient visual search of videos. In: Toward Category-Level Object Recognition. Volume 4170 of LNCS. Springer (2006) 127–144
10. Nister, D., Stewenius, H.: Scalable Recognition with a Vocabulary Tree. In: Conference on Computer Vision and Pattern Recognition. (2006)
11. Fei-Fei, L., Fergus, R., Perona, P.: One-shot learning of object categories. IEEE Transactions on Pattern Analysis and Machine Intelligence **28** (2006) 594–611
12. Amit, Y., Geman, D.: Shape Quantization and Recognition with Randomized Trees. Neural Computation **9** (1997) 1545–1588
13. Beis, J., Lowe, D.: Shape Indexing using Approximate Nearest-Neighbour Search in High-Dimensional Spaces. In: Conference on Computer Vision and Pattern Recognition, Puerto Rico (1997) 1000–1006
14. Rosten, E., Drummond, T.: Machine learning for high-speed corner detection. In: European Conference on Computer Vision. (2006)
15. Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B.: FastSLAM: A factored solution to the simultaneous localization and mapping problem. In: Proceedings of the AAAI National Conference on Artificial Intelligence, Edmonton, Canada, AAAI (2002)
16. Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B.: FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI), Acapulco, Mexico, IJCAI (2003)
17. Fischler, M., Bolles, R.: Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Communications ACM **24** (1981) 381–395
18. Davison, A.J.: Real-Time Simultaneous Localisation and Mapping with a Single Camera. ICCV **02** (2003) 1403
19. Davison, A.J., Reid, I.D., Molton, N.D., Stasse, O.: Monoslam: Real-time single camera slam. IEEE Transactions on Pattern Analysis and Machine Intelligence **29** (2007) 1052–1067