Compact Signatures for High-speed Interest Point Description and Matching *

Michael Calonder, Vincent Lepetit, Pascal Fua
EPFL, Lausanne, Switzerland
{first.last}@epfl.ch

Kurt Konolige, James Bowman, Patrick Mihelich
Willow Garage, Menlo Park, CA, U.S.A.
{konolige,jamesb,mihelich}@willowgarage.com

Abstract

Prominent feature point descriptors such as SIFT and SURF allow reliable real-time matching but at a computational cost that limits the number of points that can be handled on PCs, and even more on less powerful mobile devices. A recently proposed technique that relies on statistical classification to compute signatures *has the potential to be much faster but at the cost of using very large amounts of memory, which makes it impractical for implementation on low-memory devices.*

In this paper, we show that we can exploit the sparseness of these signatures to compact them, speed up the computation, and drastically reduce memory usage. We base our approach on Compressive Sensing theory. We also highlight its effectiveness by incorporating it into two very different SLAM packages and demonstrating substantial performance increases.

1. Introduction

The ability to very quickly compute local descriptors on regular PCs or less powerful handheld devices has recently become a critical component of many applications, in particular for localization or object recognition purposes. A recent approach [6] introduced *signatures*, a local descriptor that can be computed much faster than SIFT [14] or SURF [5]: Given a Fern classifier [19] trained offline to recognize a number of keypoints extracted from an image database, the signature of a new keypoint is taken to be the response of the Fern and can be used to match it by performing a nearest neighbor search among signatures. Computing a Fern response is extremely fast because it only requires a small number of elementary operations.

However, even though the authors of [6] noted that the signature vectors are long but sparse, they did not exploit it. As a result, matching them still involves many more elementary operations than absolutely necessary. Moreover, evaluating the signatures requires storing many distributions of the same size as themselves and, therefore, large amounts of memory. Our own implementation of this method requires around 100 MB to store these distributions, before even starting to learn signatures for new keypoints. This makes it difficult to exploit its speed advantage on low-power low-memory devices. And even though this is not as much an issue on a standard PC, more efficient descriptor construction and matching can enable new applications, such as real-time place recognition, that are heavy consumers of descriptor matching.

In this paper, we show that compacting the signatures by multiplying them by random projection matrices solves this problem and leads to an implementation that is both faster and far more memory efficient. As recent research in Compressive Sensing [2, 4, 9] shows, because the signatures are sparse, these random projections entail almost no information loss and this is what gives our approach its power. Remarkably, as will be shown in the experimental section, using either a Random Ortho-Projection or a PCA projection yields virtually the same results. This sheds light on the inner workings of the many methods [16], which perform PCA dimensionality reduction of SIFT-style descriptors. It suggests that their success owes more to the underlying sparsity of these descriptors than to PCA itself.

More significantly, our approach is different from earlier ones that compute high-dimensional descriptors and then reduce their dimensionality. We can perform the projections offline: since we use Ferns to compute the signatures by averaging distributions and since the projection is linear, we can project these distributions offline and only compute the means online. As a result, we never explicitly have to handle high-dimensional vectors, which is key to reducing memory requirements by a factor of about 20 to 45 with respect to the original approach [6]. Testing on standard benchmarks shows a matching performance similar to that of the original approach and better than that of SURF [5], while being about 4 times faster than the former and 32 times faster than the latter when running on the same CPU.

Computing 1000 signatures takes 15 ms and 4.1 MB of RAM on a regular Intel 2.4 GHz PC. Since a SLAM algorithm such as [11, 10] typically needs to learn far fewer than

*This work has been supported in part by the PEGASE EC project.

1000 points per frame, we have been able to successfully integrate our algorithm into the publicly available PTAM code [11]. This yields automated and reliable reinitialization of the algorithm whenever needed, as shown in Fig. 5, without slowing down the tracking.

The real-time capability of our approach has also been demonstrated in an online robotics application [12], where it was used for geometric matching of images as part of a place recognition algorithm. Non-approximative feature matching is the time-critical component of the algorithm, and compact signatures enable continuous online relocalization with no false positives.

2. Related Work

State-of-the-art approaches to feature point matching can be partitioned into two main classes, those that rely on invariant descriptors and those that treat matching as a classification problem. For the purpose of this discussion, please bear in mind that our approach can handle several thousand signatures at frame-rate on a standard CPU without requiring GPU acceleration.

Invariant Descriptors. Methods in this class rely on local descriptors designed to be invariant, or at least robust, to specific image distortions [20, 14]. They often require scale and orientation estimates provided by a keypoint detector. Among these, the SIFT descriptor [14], computed from local gradient histograms, has been shown to work remarkably well, especially if one rectifies the image patches surrounding the feature points [15, 17]. However, because the SIFT descriptor is complex, it is relatively slow to evaluate. A standard implementation on a modern PC requires approximately 1 ms per feature point, which limits the number of points that can be handled simultaneously to less than 50 if one requires frame-rate performance and it takes a GPU implementation to achieve a 10-fold speed-up [21].

SURF [5] is closely related to SIFT and achieves a 3 to 7-fold speed increase by using integral images and box filters to compute the descriptor, which means that from 150 to 350 keypoints can be handled while still achieving high-quality matching. Even though matching SURF descriptors can also be sped up using a GPU [7], such a GPU implementation is still about 7% slower than matching signatures on an ordinary CPU, not to mention the fact that computing the descriptors in the first place is about 32 times slower. Nevertheless, to the best of our knowledge, SURF currently represents one of the best compromises between speed and reliability and we use it to provide the baseline against which to compare our approach.

Of course, both SIFT and SURF already are unquestionably effective for well-designed real-time applications. For example, natural feature tracking at frame rates of up to 20 Hz has recently been demonstrated on cell phones [24], which is very impressive but requires a highly sophisticated approach to combining detection and tracking to avoid computing and matching too many descriptors. Similarly, feature points have been used as visual words [22] for fast image retrieval in very large image databases [18]. The feature points are labeled by hierarchical k-means clustering of their SIFT descriptors, which allows the use of very many visual words. However, it is worth noting that 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.

Matching as Classification. 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 [8] 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 [1] can be trained to recognize them independently of pose [13]. 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 intensive 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 [25], but still only allows for incremental learning of relatively few feature points.

It was recently shown that the slow training phase could be eliminated by describing a new keypoint in terms of its signature, taken to be the set of responses of a Fern classifier trained offline to recognize a number of keypoints extracted from an image database [6]. In practice, this signature is a long sparse vector that effectively characterizes the point. Its computation is very fast but requires RAM storage of many large distributions to evaluate the Fern's response. In this paper, we explicitly exploit the sparsity of the signatures to achieve an implementation that is both fast and memory efficient.

3. Method

We first briefly summarize the approach to keypoint description set forth in [6] and formalize its memory requirements. We then show how we take advantage of Compressive Sensing [2, 4, 9] insights to turn the sparse signature vectors it produces into compact ones saving memory while speeding up the computation.

3.1. Sparse Signatures

The signature vectors introduced in [6] describe the appearance of an image patch $\mathbf{p}$ in terms of the responses of a Fern classifier [19] trained offline to recognize a predefined set B of N reference keypoints. Let this *base classifier* be composed of J Fern units $\{F_i\}_{i=1}^{J}$, which are binary structures such as those depicted in Fig. 1. Each F_i partitions patches into 2^d leaves by performing d binary comparisons between pixel intensities. Each leaf contains an N-dimensional vector $\mathbf{t}_i(\mathbf{p})$ storing the probabilities that $\mathbf{p}$ is one of the N reference keypoints given $\mathbf{p}$ reached that leaf. The full response vector $\mathbf{r}(\mathbf{p})$ for all J Ferns is taken to be

$$\mathbf{r}(\mathbf{p}) = \frac{1}{Z} \sum_{1 \leq i \leq J} \mathbf{t}_i(\mathbf{p}) \quad , \tag{1}$$

where Z a normalizer s.t. its elements sum to one[1]. In practice, when $\mathbf{p}$ truly corresponds to one of the reference keypoints, $\mathbf{r}(\mathbf{p})$ contains one element that is close to one where all others are close to zero. Otherwise, it contains a few relatively large values that correspond to reference keypoints that are similar in appearance and small values elsewhere. Furthermore, this distribution of small and large values is highly invariant to changes in viewpoint and lighting. By definition, the *sparse signature* is given by

$$\mathbf{s}(\mathbf{p}) = \mathbf{\Theta}(\mathbf{r}(\mathbf{p}), \theta) \quad , \tag{2}$$

where $\mathbf{\Theta}(\cdot, \theta)$ represents element-wise thresholding with threshold θ. It is an N-dimensional vector with only a few non-zero elements that is mostly invariant to different imaging conditions and therefore presents a useful descriptor for matching purposes. The whole process is depicted by Fig. 1 TOP.

Good parameter choices to achieve good matching performance are $J = 50$, $d = 10$, and $N = 500$ and therein lies the drawback of this approach. To perform the computation described above, we need for each of the 2^d leaves in each of the J Ferns an N-dimensional vector of floats. This means that the total memory requirement to store the Fern classifier is

$$\mu = b \; J \; 2^d \; N \quad \text{bytes} \quad , \tag{3}$$

where b is the number of bytes required to store a float, usually 4 or 8. As discussed earlier, this works out to more than 100 MB that are needed even before starting to learn descriptors for any new keypoint.

3.2. Compact Signatures

The Compressive Sensing literature [2, 4, 9] shows that high-dimensional sparse vectors can be reconstructed from their linear projections into much lower-dimensional spaces. Many kinds of matrices can be used for this purpose. As discussed in the Appendix, Random Ortho-Projection (ROP) matrices are a good choice and can be easily constructed by applying a Gram-Schmidt orthonormalization process to a random matrix [23].

[1]Note that this response is computed as the average response as in [1] and not as a Naive Bayesian score as in [19].

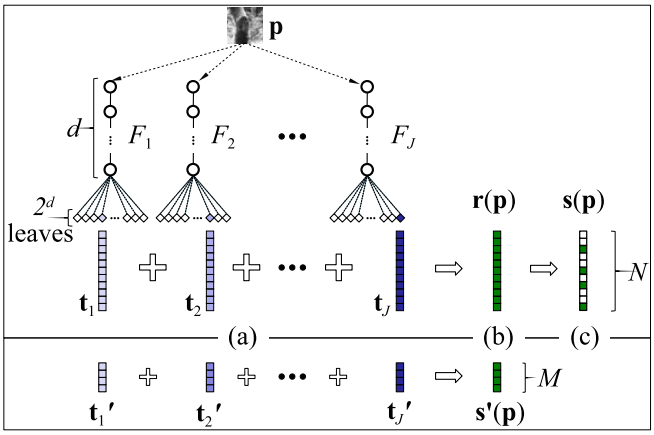

Figure 1. Illustration of the signature creation process for a new keypoint surrounding patch $\mathbf{p}$. TOP Creating a sparse signature. (a) $\mathbf{p}$ is dropped through all ferns F_i, yielding J $\mathbf{t}_i$ vectors. (b) All $\mathbf{t}_i$ are summed up to compute $\mathbf{r}(\mathbf{p})$ of Eq. 1. (c) Only in the case of sparse signatures, the $\mathbf{r}(\mathbf{p})$ are thresholded yielding a sparse signal. BOTTOM For a compact signature, instead of summing the $\mathbf{t}_i$, we sum the much shorter $\mathbf{t}_i'$ and skip the thresholding step.

This has provided us with the key insight of this paper, which is that the sparse signature vectors of Section 3.1 can be turned into much-lower dimensional ones using the same kind of projection matrix. This reduces the memory requirements at no loss in matching performance.

More formally, let $\Phi \in \mathbb{R}^{M \times N}$ with $M \ll N$ be a ROP matrix. Given the Fern's response $\mathbf{r}(\mathbf{p})$ of Eq. 1, we take the *compacted signature* to be the M-dimensional vector $\mathbf{s}'(\mathbf{p}) = \Phi \, \mathbf{r}(\mathbf{p})$. The key to the effectiveness of our approach is that $\mathbf{s}'(\mathbf{p})$ can be evaluated without explicitly performing the matrix multiplication or even calculating $\mathbf{r}(\mathbf{p})$. Since

$$\mathbf{s}'(\mathbf{p}) = \Phi \left[\frac{1}{Z} \sum_{1 \leq i \leq J} \mathbf{t}_i(\mathbf{p}) \right] = \sum_{1 \leq i \leq J} \frac{1}{Z} \, \Phi \, \mathbf{t}_i(\mathbf{p}) \quad , \tag{4}$$

$\mathbf{s}'(\mathbf{p})$ can be computed by simply summing the appropriate *compressed leaf vectors* $\mathbf{t}_i' := \frac{1}{Z} \, \Phi \, \mathbf{t}_i(\cdot) \in \mathbb{R}^M$, all of which can be computed *offline* once the base Fern classifier has been trained. See Fig. 1 BOTTOM for an illustration.

In Eq. 3, this replaces N by $M \ll N$ and divides the memory requirements by N/M. Furthermore, evaluating $\sum_{i=1}^{J} \mathbf{t}_i'$ requires again N/M times fewer elementary operations than Eq. 1 would. This is central as it is done every time a descriptor is computed.

Note that when computing $\mathbf{s}'$, we used the Fern's re-

sponse vector of Eq. 1 instead of the thresholded signature of Eq. 2. This is justified by Compressive Sensing theory that only requires the to-be-compressed signal to be inherently sparse without making the sparsity explicit.

3.3. Quantization

The computation can be further streamlined by quantizing the compressed leaf vectors $\mathbf{t}'_i$ and representing them using one byte per element instead of the 4 bytes required by floats, which reduces memory requirements by an additional factor 4.

Let $\{t_i^j\}_{j=1}^M$ denote the elements of $\mathbf{t}'_i$. Then we define the elements $\{\bar{t}_i^j\}_{j=1}^M$ of the quantized version $\bar{\mathbf{t}}_i$ of $\mathbf{t}'_i$ as

$$\bar{t}_i^j = \left\lfloor \frac{\min(t_i^j, p_{95}) - p_0}{p_{95} - p_0} \left(2^q - 1\right) \right\rfloor \quad , \tag{5}$$

where q the number of bits required to code individual elements of the signature and p_0 denotes the minimum value occurring over all leaves in the current Fern and p_{95} is the corresponding 95% percentile. We noticed empirically that using this percentile actually increases stability. Since we use bytes to represent the signatures, we could use $q = 8$. However, taking $q = 4$ does not affect matching performance but allows us to more effectively take advantage of hardware acceleration when computing the full signature $\sum_{i=1}^J \bar{\mathbf{t}}'_i$.

There are two appealing ramifactions to the quantization. First, experiments showed that after the quantization we can reduce the depth d of the Ferns from 10 to 9 without any loss in accuracy, and hence reduce the memory requirements by another factor of 2. Second, we observe a speed-up in creating the *compact signature*[2]

$$\bar{\mathbf{s}}(\mathbf{p}) = \Big(\sum_{1 \leq i \leq J} \bar{\mathbf{t}}_i(\mathbf{p}) \Big) \rhd \left(\lceil \log_2 J \rceil + q - 8 \right), \tag{6}$$

where $\mathbf{a} \rhd b$ denotes a bit-wise right shift of each of the elements of $\mathbf{a}$ by b bits. Storing the sum of J leaf values, each with a maximum value $2^q - 1$, requires $\lceil \log_2 J \rceil + q$ bits. However, experiments showed that the least significant bits of those values do not carry substantial information. Hence, we can fit them into one byte of the final signature by a right-shift by $\lceil \log_2 J \rceil + q - 8$ bits. In our implementation, $J = 48$ and $q = 4$, which means that we shift by 2.

The speed-up in signature creation arises from the fact that the summations, now of bytes and not of floats anymore, can be carried out faster. This applies to native C code of course, but if supported by the hardware, also allows for vectorization using SIMD[3] instructions. In addition, the same speed-up argument applies to nearest neighbor search as well, especially as we found the L_1 norm resulting in the same ordering of distances among signatures as the L_2 norm. For this reason we always use the L_1 norm to match compact signatures, as it is computed more efficiently.

[2]Before, we referred to the $\mathbf{s}'$ as compact**ed** signatures. For the remainder of the paper, when we mention compact signatures we always refer to the quantized, short vectors $\bar{\mathbf{s}}$.

[3]Single Instruction Multiple Data, in our case SSE2.

4. Results

In this section we first compare, both in terms of matching performance and speed, our compact signatures against the original sparse signatures [6] that inspired this work and against SURF-64 [5], which is a broadly-used approach for real-time applications. We then discuss the integration of our signatures into two SLAM software packages [12, 25] for reinitialization and place recognition purposes, which leads to a substantial performance increase in both cases.

4.1. Comparing with Sparse Signatures and SURF

To assess matching performance and speed, we use the four publicly available datasets: Wall[4], Light[4], Jpg[4] and Fountain[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. The Jpg dataset was generated by simply saving a reference image at various levels of compression, keeping all other parameters constant.

4.1.1 Matching Performance

The Wall and Fountain datasets test for robustness to viewpoint changes, the Light one for changes lighting conditions, and the Jpg one for the influence of JPG compression artifacts. All three kinds of robustness are important in practice.

Given several images from the same database, we take $m|n$ 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 and Light datasets, we tested $\{1|2, \ldots, 1|6\}$, corresponding to substantial changes in viewpoint and lighting, respectively. From the Jpg dataset, we tested $\{1|2, \ldots, 1|5\}$, and from the Fountain dataset only $1|2$ and $1|3$, as for this dataset the number of matches quickly decays due to occlusion.

We define the *recognition rate* as the ratio of the number of correct matches to the total number of interest-points in the reference image. To compute it, we first extract a number of SURF keypoints from the reference image and compute the coordinates of their corresponding points in

[4]avail. `http://www.robots.ox.ac.uk/~vgg/research/affine`

[5]avail. `http://cvlab.epfl.ch/~strecha/multiview`

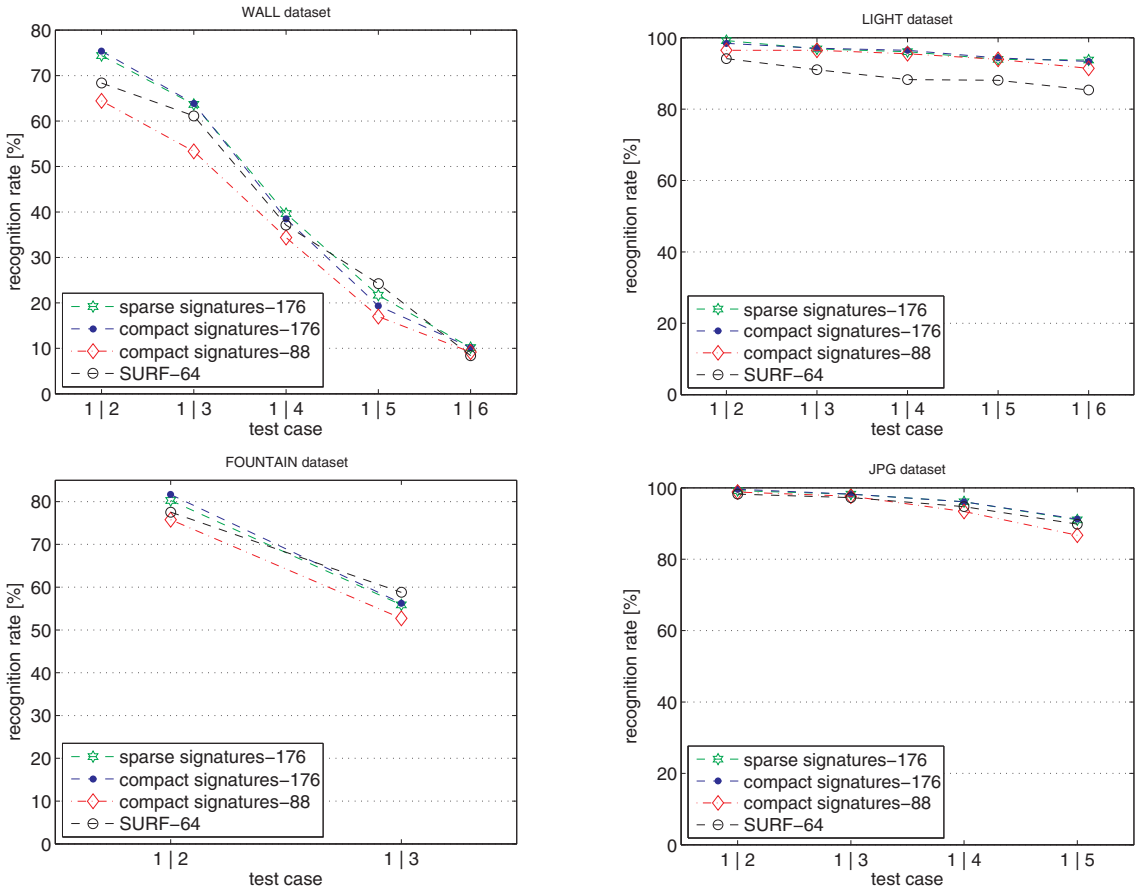

Figure 2. Recognition rate on four datasets, testing practically most relevant invariance requirements of keypoint descriptors. The number following the method name indicates the descriptor length. Note that SURF uses floats whereas compact signatures consist of bytes.

the test image using the known geometric relationship between the two. We then evaluate the SURF descriptors and the sparse and compact signatures on the reference and test points which yields $3 \times 2 = 6$ sets of descriptors that we store in a database. Matching a point in a reference image then simply amounts to finding the nearest neighbor to its descriptor in the appropriate 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 SURF and for compact signatures, we do not favor either technique over the other. In the experiments depicted by Figure 2, we use sparse signatures of size $N = 500$ and compact signatures of size M either 88 or 176, the latter being divisible by 16, which allows for code optimization. The corresponding memory usage and number of bytes per descriptor are given in Table 1. These experiments show that:

- Provided that we use the $M = 176$ compact signatures, the dimensionality reduction does not impact performance, as predicted by the Compressive Sensing theory.
- In 14 out of a total of 16 test cases the signature-based methods are slightly more accurate than SURF.
- The performance drops slightly if we use $M = 88$, which might be warranted if we need to run on a very low-memory device.

	Classifier	Descriptor
Sparse Sig.	93.75 MB	$\approx 175B$
Compact Sig.-176	4.13 MB	176 B
Compact Sig.-88	2.06 MB	88 B
SURF-64	n/a	256 B

Table 1. Memory usage statistics. Using the compact signatures drastically reduces the memory requirements. Furthermore, even though the $M = 176$ compact descriptor is longer than the SURF descriptor, it requires less storage because it is made of bytes instead of floats.

In other words, M is a parameter that lets us control the trade off between memory requirements and matching reliability and there is no reason to ever go beyond $M = 176$, as further illustrated by Figure 3. As argued in the intro-

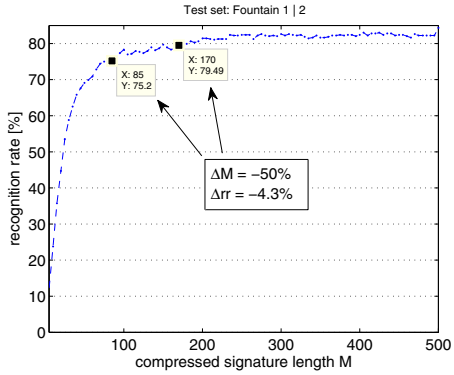

Figure 3. Recognition rate as a function of the compact signature length M. It stops increasing significantly beyond $M = 170$.

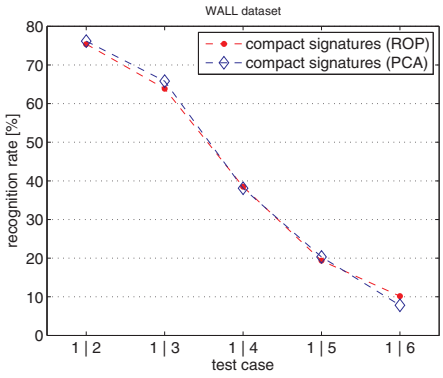

Figure 4. Recognition rate on the Wall dataset using compact signatures. The plot compares two different methods for dimensionality reduction: ROP and PCA.

duction, the effectiveness of the presented approach arises from the intrinsic sparseness of the signatures, rather than from the method employed for dimensionality reduction. To demonstrate this, we compare the performance of compact signatures based on a Random Ortho-Projection (ROP) to one based on principal component analysis (PCA). To assess the effect of PCA, we simply substitute the $M \times N$ ROP projection matrix Φ in Eq. 4 by the PCA matrix. The M rows of the PCA matrix contain by definition the eigenvectors of the covariance matrix of the Fern's leaf distributions, which can be computed once the training of the Fern finished. Hence using PCA instead of a ROP does not induce any structural change of the presented method. Interestingly, as shown in Figure 4, it does not result in any significant change in recognition rate. We believe this has potential implications for all methods that rely on PCA for dimensionality reduction of image descriptors. Their discriminative power arises from the underlying sparsity of the representation and the correct theoretical framework to understand their behavior might be that of Compressive Sensing theory.

4.1.2 CPU Time and Memory Consumption

In Table 2 we summarize the time spent on a 2.4 GHz machine both to compute the descriptor and to perform an exhaustive nearest-neightbor search (n^2) for matching purposes. In this case, 512 keypoints were used. The values for SURF are given in [5, 7] and were slightly rescaled in order to make all values comparable on a 2.4 GHz CPU. The GPU implementation [7] is a general one for matching floating point vectors and thus also applicable to SURF-64.

	Description (512 kpts)	n^2-Matching (512×512 kpts)
Sparse Signatures	31.3 ms	27.7 ms
Compact Signatures-176	**7.9 ms**	**6.3 ms**
SURF-64	255 ms	200 ms
SURF-64 + ANN	–	91 ms
SURF-64 on GPU	–	6.8 ms

Table 2. Timings. All values were measured on the CPU, except for the last row. ANN: Approximative Nearest Neighbor.

4.2. Applications

SLAM Relocalization. Thanks to their high efficiency, compact signatures are well suited to be employed in ressource-demanding real-time applications. To demonstrate this, we integrated them in a recent SLAM system called Parallel Tracking and Mapping [11, 10]. PTAM runs in parallel in two threads, one estimating the pose from the map and one constantly bundle-adjuststing the map and keyframes, which are special frames that have the full 3D pose attached and are collected from time to time. Their purpose is to allow for automated re-initialization when the system gets lost, for example due to total occlusion or image blur from too fast movement. In the publicly availabe code, relocalization is achieved by creating a small blurry version of the actual image from the camera and matching this via a sum of squared differences-approach with all the small blurry images of the keyframes. Taking the best match, the two blurry images are aligned in a least square sense, thus yielding a 2D rotation plus translation that allows–together with the 3D information of the best keyframe–to estimate the 3D position of the frame in question.

The blurry image idea works well but is not suited for large viewpoint changes. This restriction can be removed by employing compact signatures to match frames based on keypoints. For every new keyframe, the signatures of the 100 strongest keypoints are computed and stored along with the keyframe. When relocalizing, the 100 strongest keypoints' signatures of the new frame are matched against those of every keyframe, which yields the best match along with a score that can be thresholded to decide if the matched frame is the correct one. No geometric consistency check is applied. In the typically small environment PTAM is de-

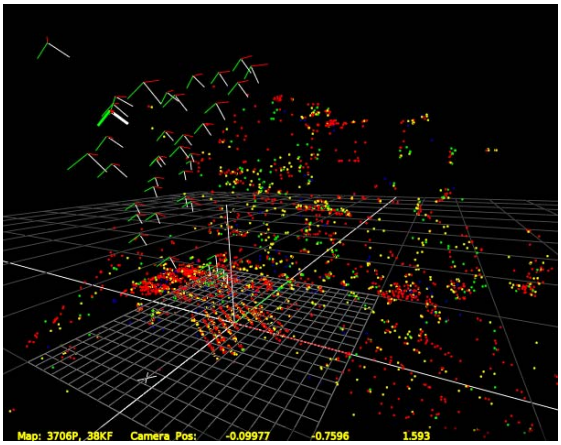

Figure 5. Screenshot from the PTAM system running in a desktop environment.

signed for, the number of keyframes hardly exceeds a few tens making exhaustive search among the keyframes feasible while still operating in real-time. Figure 5 gives an idea from the PTAM system in action. The tripods in the top left quadrant are the poses where keyframes were taken. When lost in some reasonable distance from a keyframe, the compact signatures allow for a quick and accurate relocalization. The map contains about 3700 points and 38 keyframes.

Place Recognition. We also incorporated our compact signatures in a view-based stereo Visual SLAM system [12] for online place recognition purposes. It relies on a view-based map that consists of individual views and metric 6DOF links between them formed by geometric matching. The map is optimized online to minimize the total error of the links among the views.

Fig. 6 LEFT depicts the map obtained for a large indoor environment from six different runs using only information from stereo images. Place recognition (PR) was successfully used to patch together the different runs, and to close large and small loops within a run and among the different runs.

To close loops and recover from localization failures, we employ an online PR system that matches the current reference view against the whole map. It involves a vocabulary tree prefilter [18] that proposes candidate matches followed by a robust geometric SfM consistency check to confirm valid candidates.

For the geometric check, we need to generate n^2 matches from the reference view features to the candidate view features. Typically there are 300 features per view, and we check the top 15 candidates. Using compact signatures for the features has several advantages: They are robust to lighting and viewpoint changes, they are cheap to compute, and accurate n^2 matching is very fast. Fig. 6 RIGHT shows the timing for PR and map optimization over the course of a run with 1200 keyframes. Note that the geometric check has a constant average time of around 70 ms, about half of which is from n^2 matching. Using SURF-64, the best available alternative, would slow the matching step to over 500 ms, dominating the time budget for place recognition.

5. Conclusion and Future Work

We have proposed a highly efficient feature point descriptor with a matching performance comparable to some of the best state-of-the-art methods while being either much faster or much less greedy in terms of memory. The strength of our *compact signatures* comes from explicitly exploiting the sparsity of the signatures produced by Generic Trees [6] and their theoretical underpinning comes from Compressive Sensing theory.

In practice, compact signatures require almost 50 times less memory than sparse ones and can be computed and matched several times faster. This makes them ideally suited for implementation on low-memory low-power devices, which is something we will demonstrate in future work by implementing place recognition algorithms on such devices.

Appendix

In this appendix, we link our approach to the Compressive Sensing theory [2, 4, 9] which provides the theoretical underpinning for using the ROP matrices of Section 3.2.

It has been shown that the dimensionality of sparse vectors $\mathbf{x}_i$ can be reduced by applying the linear transformation

$$\mathcal{T} \;:\; \mathbb{R}^N \rightarrow \mathbb{R}^M, \quad \mathbf{y}_i = \Phi\,\mathbf{x}_i, \quad \Phi \in \mathbb{R}^{M\times N} \tag{7}$$

where $M \ll N$, without information loss for an appropriately chosen matrix Φ. This means that the transformation exhibits the following two properties:

1. It preserves the metric structure of the original space, which means that $\forall \mathbf{x}_i \in \mathbb{R}^N$, $\forall \mathbf{y}_i \in \mathbb{R}^M$

$$\|\mathbf{x}_1 - \mathbf{x}_2\| < \|\mathbf{x}_1 - \mathbf{x}_3\| \rightarrow \|\mathbf{y}_1 - \mathbf{y}_2\| < \|\mathbf{y}_1 - \mathbf{y}_3\|\,,$$

   where $\|\cdot\|$ is the Euclidean 2-norm.

2. Any $\mathbf{x}$ can be uniquely recovered from $\mathbf{y}$.

These two properties are provably fulfilled if Φ has the k-Restricted Isometry Property (k-RIP) w.r.t. all k-sparse vectors $\mathbf{x}$. This requires that there exists a sufficiently small constant $\delta_k > 0$ such that

$$(1-\delta_k)\,\|\mathbf{x}\|_2^2 \leq \|\Phi\,\mathbf{x}\|_2^2 \leq (1+\delta_k)\,\|\mathbf{x}\|_2^2 \quad . \tag{8}$$

δ_k measures the degree to which Φ deviates from the perfect RIP where $\delta_k = 0$. In other words, the k-RIP ensures that

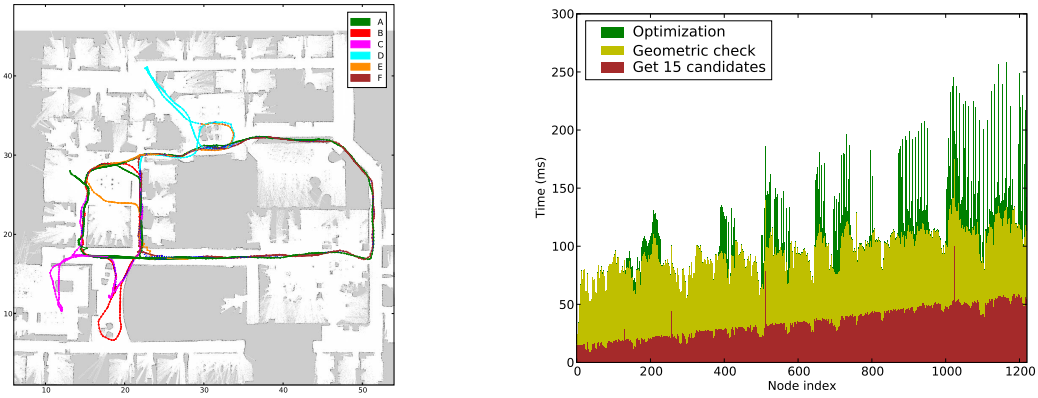

Figure 6. LEFT Map created online solely from stereo views during 6 robot runs in a large indoor environment. Place recognition is used to stitch them together and close loops. A laser-based map is shown in light gray underlining that the trajectories of the robot are reasonable. The map has 1228 views and 3826 connecting links. Distances are in meters. RIGHT Timing for view integration of keyframes into a view-based map.

all submatrices of $\Phi \in \mathbb{R}^{M \times k}$ are close to being isometries and hence distance preserving.

Even though designing such a Φ is in general NP-complete, both i.i.d. matrices and ROP ones have the k-RIP with probability almost 1, provided that $M \geq c\,k\,\log(N/k)$ with c being small constant [3].

In our experiments, we found that ROPs actually perform about 10% better than purely random ones, which is why we use them. A $M \times N$, $M \ll N$, ROP matrix can be constructed via Gram-Schmidt orthonormalization.

References

[1] Y. Amit and D. Geman. Shape Quantization and Recognition with Randomized Trees. *Neural Computation*, 1997.

[2] R. Baraniuk. Compressive sensing. *Signal Processing*, 24(4):118–120, 2007.

[3] R. Baraniuk, M. Davenport, R. Devore, and M. Wakin. A simple proof of the restricted isometry property for random matrices. *Constructive Approximation*, 2007.

[4] R. G. Baraniuk, M. Davenport, R. A. DeVore, and M. Wakin. A simple proof of the restricted isometry property for random matrices. *Construct. Approximation*, 2008.

[5] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. *CVIU*, 10(3):346–359, 2008.

[6] M. Calonder, V. Lepetit, and P. Fua. Keypoint signatures for fast learning and recognition. In *ECCV'08*.

[7] A. Chariot and R. Keriven. Gpu-boosted online image matching. In *ICPR'08*.

[8] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *PAMI*, 28(4):594–611, 2006.

[9] C. Hegde, M. Wakin, and R. Baraniuk. Random projections for manifold learning. In *NIPS'08*.

[10] G. Klein and D. Murray. Improving the agility of keyframe-based SLAM. In *ECCV'08*.

[11] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *ISMAR'07*.

[12] K. Konolige, J. Bowman, J. D. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua. View-based maps. In *RSS'09*, 2009.

[13] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *PAMI*, 28(9):1465–1479, 2006.

[14] D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *IJCV*, 20(2):91–110, 2004.

[15] K. Mikolajczyk and C. Schmid. A Performance Evaluation of Local Descriptors. In *CVPR'03*.

[16] K. Mikolajczyk and C. Schmid. A Performance Evaluation of Local Descriptors. *PAMI*, 27(10):1615–1630, 2004.

[17] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *IJCV*, 65:43–72, 2005.

[18] D. Nister and H. Stewenius. Scalable Recognition with a Vocabulary Tree. In *CVPR'06*.

[19] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast Keypoint Recognition using Random Ferns. *PAMI*, 2009. Accepted for Publication.

[20] C. Schmid and R. Mohr. Local Grayvalue Invariants for Image Retrieval. *PAMI*, 19(5):530–534, 1997.

[21] S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc. GPU-based video feature tracking and matching. In *Workshop on Edge Comp. Using New Commodity Architectures*, 2006.

[22] J. Sivic and A. Zisserman. Video Google: Efficient visual search of videos. In *Toward Category-Level Object Recognition*, volume 4170 of *LNCS*. 2006.

[23] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997.

[24] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose Tracking from Natural Features on Mobile Phones. In *ISMAR'08*.

[25] B. Williams, G. Klein, and I. Reid. Real-time slam relocalisation. In *ICCV'07*.