Online Learning of Binary Feature Indexing for Real-time SLAM Relocalization

Youji Feng[1], Yihong Wu[1], Lixin Fan[2]

[1]Institute of Automation, Chinese Academy of Sciences
[2]Nokia Research Center, Tampere

Abstract. In this paper, we propose an indexing method for approximate nearest neighbor search of binary features. Being different from the popular Locality Sensitive Hashing (LSH), the proposed method construct the hash keys by an online learning process instead of pure randomness. In the learning process, the hash keys are constructed with the aim of obtaining uniform hash buckets and high collision rates, which makes the method more efficient on approximate nearest neighbor search than LSH. By distributing the online learning into the simultaneous localization and mapping (SLAM) process, we successfully apply the method to SLAM relocalization. Experiments show that camera poses can be successfully recovered in real time even there are tens of thousands of landmarks in the map.

1 Introduction

simultaneous localization and mapping (SLAM) has been extensively studied in both robotics and computer vision [1, 2]. One important module in SLAM is the relocalization, *i.e.* the recovery of the camera pose after tracking failure. Since the camera pose can be estimated using correspondences between 3D points in the map and 2D features in the image, the key problem in the relocalization is to obtain the 3D-2D correspondences.

The problem of obtaining 3D-2D correspondences is typically treated as the problem of feature matching: during the mapping process, each 3D point in the map is associated with some image features; in the relocalization process, the same type of features are extracted from the image and matched against those of the 3D points. Some existing works [3–5] on image-based localization employ robust image features such as SIFT [6] and Daisy [7] in their systems. They are able to localize an image accurately even the scene contains millions of 3D points. However, due to the expensive computation of these features, they are not well suited for SLAM relocalization in which real time performance is of critical importance. Williams *et al.* [8] propose a relocalization approach using the adapted random ferns [9] to collect 3D-2D correspondences. While being performed in real time, their approach can hardly be extended to large scenes due to the large memory footprint. Recently, Straub *et al.* [10] have designed a relocalization module based on binary features [11–13]. Their system is demonstrated on a scene containing tens of thousands 3D points and achieves near real

time performance due to two attractive characteristics of binary features: a)they are extracted faster than SIFT like features by two orders of magnitude; b)the distance of two binary features can be computed by using fast SSE instructions. To further speed up feature matching, Straub *et al.* also use Locality Sensitive Hashing [14] to perform approximate nearest neighbor(ANN) search.

In this paper, we demonstrate a relocalization module which employs binary features. Being different from [10], we propose for ANN search an indexing method that is more efficient than LSH. The higher efficiency is achieved through an online learning process. Initially, the hash keys are randomly generated as in LSH, and then they are adapted incrementally during the SLAM process with the objective to attain more uniform hash buckets and higher collision rates. Experiment results show that using the proposed indexing method takes less time than using the original LSH to reach the same search accuracy, or reaches higher accuracy by taking the same time.

The rest of this paper is organized as follows: Section 2 gives an overview of the entire SLAM system; Section 3 introduces the relocalization module and the proposed indexing method; Experiment results are shown in Section 4 and Conclusions are drawn in Section 5.

2 System Overview

Our SLAM implementation is adapted from PTAM [2]. It consists of two threads, *i.e.* the background mapping thread and the foreground tracking thread. The mapping thread collects keyframes from the video sequence and performs structure from motion to build a map of the environment. The process is done incrementally. When a new keyframe is inserted, FAST corners [15] are first extracted, and then some of the corners are identified as the observations of the old 3D points, while the remaining are matched against the corners in the nearest keyframes to triangulate new 3D points. The tracking thread estimates the camera pose of each frame by tracking the 3D points in the map. Under the assumption of continuous camera motion, the observations of the 3D points can be searched around their predicted locations. Once the observations are found, *i.e.* the 3D-2D correspondences are established, the camera pose is trivially estimated by using a non-linear optimization routine. During the tracking process, tracking failure may happen due to sudden changes of the illumination, full occlusions, or extreme motion blur. Then a relocalization module is implemented to re-estimate the camera pose and restart the tracking process.

3 Relocalization

The aim of the relocalization is to estimate the camera pose of an image after the tracking failure. Being the same as in the tracking process, the core problem in the relocalization is also to find the observations of the 3D points, except that the observations can no longer be predicted from the camera poses of previous

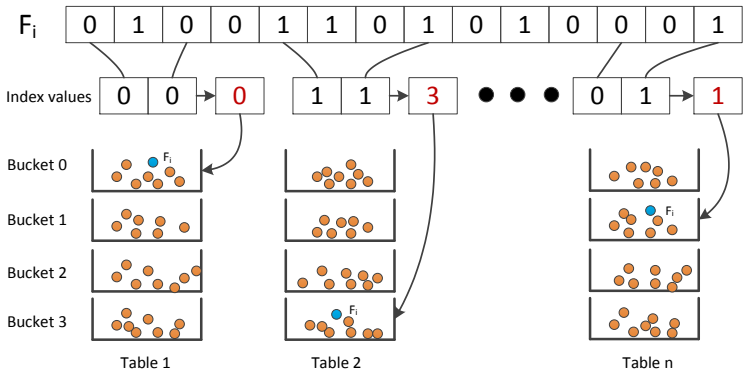

Fig. 1. An example of indexing a database feature. Each hash key consists of two randomly selected bits.

frames. Once the observations are found, the camera pose can be estimated by RANSAC [16] and Perspective-n-Points algorithms.

Obtaining the observations of the 3D points is treated as the problem of feature matching. In the mapping process, whenever the observation of a 3D point is found in a keyframe, a binary descriptor at this observation is extracted. Consequently, each 3D point corresponds to a set of binary descriptors which are referred to as database features. In an image being relocalized, Fast corners [15] along with their binary descriptors, referred to as query features, are first extracted. Then, the top two nearest neighbors of each query are searched among the database features. If the ratio test [6] is past, the nearest neighbor is deemed as the match. To be efficient, we use an indexing method to perform fast ANN search instead of brute-force search. The indexing method is more efficient than LSH due to an online learning process, which is presented below after a brief description on LSH.

3.1 LSH for ANN Search of Binary Features

LSH uses hash keys to index a feature. It assigns an index value to the feature under each key. When performing ANN search, database features are stored in multiple hash tables. Each hash table corresponds to a hash key. The designated table entry, *i.e.* hash bucket, in which a database feature should be stored is determined by the index value. Given a query feature, the index values are first assigned and the corresponding buckets are found in the same way as database features. Then a linear search is performed among the database features in these buckets to retrieve the approximate nearest neighbors. For binary features, the hash key is simply a set of bits randomly selected from the descriptor bits. Consequently, a binary code can be assigned to a feature by concatenating the values at these bits, and the index value of the feature is the integer converted by the binary code. Fig. 1 gives a simple example on how a database feature is indexed and stored in the hash tables.

3.2 The Proposed Indexing method

There are two factors closely related to the efficiency of the indexing scheme for ANN search: the distribution of the bucket sizes and the collision rate of matched feature pairs. The size of a bucket indicates the number of the database features stored in the bucket. Usually, the more uniform the bucket sizes are, the faster ANN search will be, which have been noticed by Rublee *et al.* [12]. In this paper (see analysis below), we illustrate that maintaining uniform bucket sizes is equivalent to minimizing the total ANN search time. The collision rate of matched feature pairs is the probability of that two matched features collide in the same bucket. Higher collision rate leads to higher search accuracy. With both factors being considered, our indexing method aims to attain uniform buckets and high collision rates, so that ANN search can be performed fast and accurately. We reach this aim through a learning process, in which the bits composing the hash keys are selected to minimize a corresponding cost function. In the remainder of the section, we present the learning process together with the derivation of the cost function.

To alleviate the latency in the system and handle the ever-changing set of database features, the bit selection is distributed into the SLAM process. Initially, the bits composing each hash key are selected randomly, as in LSH. Whenever a new keyframe is inserted and the set of the database features is updated, one bit of each hash key is re-selected to minimize the cost function. Then the database features are re-indexed using the new hash keys.

Recalling the two factors that affect the efficiency, the cost function should contain terms about the uniformity of the bucket sizes and the collision rate:

The uniformity of the bucket sizes. Let the normalized bucket sizes of a hash table be $\{s_n, n = 1, 2, ..., N\}$ subject to $\sum s_n = 1$, where N is the total number of buckets, then the extent to which the bucket sizes are uniform can be expressed by

$$u = \sum_{n=1}^{N} (s_n - \frac{1}{N})^2 = \sum_{n=1}^{N} s_n^2 - \frac{1}{N}. \tag{1}$$

The smaller u is, the more uniform the bucket sizes are. This expression is well compatible with the relation between the uniformity of the bucket sizes and the speed of ANN search. Since the time cost of the linear search in the buckets dominates the ANN search process, and under the assumption that the query features have similar distribution to the database features, the time cost is proportional to $\sum_{n=1}^{N} s_n^2$. As only one bit at a time is re-selected, the uniformity can also be expressed by the form

$$u' = \frac{\sum_{n=1}^{N} s_n^2}{\sum_{m=1}^{N/2} \tilde{s}_m^2}, \tag{2}$$

where $\{\tilde{s}_m, m = 1, 2, ..., \frac{N}{2}\}$ are constant, representing the normalized bucket sizes of the hash table which is obtained by indexing the database features with a key consisting of the unchanged bits. It can be verified that the value of u'

belongs to [0.5 1]. $u' = 0.5$ indicates that with the newly selected bit in the hash key, the time cost of the linear search can be reduced by a half compared to with only the unchanged bits in the hash key. While $u' = 1$ indicates that the newly selected bit does not bring any time saving. In the cost function, the term about the uniformity is set as $\frac{1}{1-u'}$ to encourage a small u' and to avoid the situation that $u' = 1$.

The collision rate. The collision rate of matched feature pairs is actually the probability that two matched features coincide with each other at all bits of the hash key. Refer to the probability that two matched features coincide at a certain bit as the stability p_c of the bit, then according to the principle of greedy algorithms, the most stable one is preferred when selecting a bit for the hash key. Thus the cost term about the collision rate is simply set as $1 - p_c$. Since the features of a 3D point are actually the matches of each other, matched feature pairs can be easily obtained in the database and p_c can be estimated by making statistics on these pairs.

Given the above two terms about the collision rate and the uniformity, the cost function C reads as:

$$C = \lambda(1 - p_c) + \frac{1}{1 - u'}, \tag{3}$$

where λ is a preset weight.

Algorithm 1 Bit Selection

Input: $\mathbb{H}_K = \{b_{s_1}, ..., b_{s_k}, ..., b_{s_K}\}$, with b_{s_k} being re-selected; $\mathbb{F}$, all the database features.
begin
  Remove b_{s_k} from $\mathbb{H}_K$ to get $\tilde{\mathbb{H}}_K = \{b_{s_1}, ..., b_{s_K}\}$;
  Index $\mathbb{F}$ using $\tilde{\mathbb{H}}_K$;
  Count the bucket sizes $\{\tilde{s}_m, m = 1, 2, ..., \frac{N}{2}\}$;
  Generate 40 random numbers$\{r_1, r_2, ..., r_{40}\}$,
  $r_i \in [1\ D]$;
  Compute the stability of each b_{r_i};

  for i=1; i≤40; i++; **do**
    Replace b_{s_k} in $\mathbb{H}_K$ with b_{r_i} to get $\mathbb{H}'_K$;
    Index $\mathbb{F}$ using $\mathbb{H}'_K$;
    Compute the cost function C;
  end
  Select $r*$ from $\{r_i\}$ so that C is minimized;
  Replace b_{s_k} with b_{r^*} in $\mathbb{H}_K$;
end
Output: $\mathbb{H}_K = \{b_{s_1}, ..., b_{r^*}, ..., b_{s_K}\}$.

For a better understanding of the proposed method, the process of selecting one bit for a hash key is summarized in Algorithm 1. The hash key is denoted

as $\mathbb{H}_K = \{b_{s_1}, b_{s_2}, ..., b_{s_K}\}$, where b_{s_k} is the $s_k th$ bit of the descriptor, $s_k \in [1\ D]$ and D the length of the descriptor.

Notes on the implementation. As can be seen from Algorithm 1, the time cost of the bit selection process is linear with the number of the database features, namely the number of the elements in $\mathbb{F}$. To make Algorithm 1 be scalable, we actually use only a subset, which is randomly composed and has up to 80,000 elements, of $\mathbb{F}$ in Algorithm 1. We find this strategy brings about a constant time cost for the bit selection process while retaining the ability to discover 'good bits', *i.e.* the bits producing uniform bucket sizes and high collision rates.

To further reduce the training time, we perform the bit selection process on not all but a half of the hash keys when a new keyframe is inserted, and the two halves are selected in turn. This strategy would slow down the discovering of good bits but effectively reduces the system latency.

4 Experiments

In this section, we first evaluate the efficiency of the proposed indexing method on ANN search, and then show the comprehensive performance of the relocalization module. The entire system is implemented in C++ and runs on a laptop with an Intel Core i3-2310 2.1GHz CPU. The binary feature employed in the system is BRISK [13].

4.1 The ANN Search Efficiency of The Indexing Method

Dataset We construct a dataset to evaluate the ANN search efficiency of the proposed indexing method. We first take a long video containing 12,821 frames around a building, and then run the SLAM system described in Section 2 on this video, from which 293 keyframes are extracted and 37,641 3D points along with 175,207 BRISK descriptors are obtained. These descriptors will serve as the database features in the experiment. To obtain query features with known ground truth matches, we select 500 well tracked frames from the video and reproject 400 visible 3D points on each of them. The binary descriptors at these reprojections are extracted as query features. Consequently, in each of the 500 frames we have 400 query features of which the corresponding 3D points are known.

Evaluation criterion The ANN search efficiency is demonstrated by the search accuracy and the search time. For a query feature, if the obtained approximate nearest neighbor corresponds to the same 3D point as itself, the ANN search is deemed as successful. The search accuracy is then defined as the ratio of successful ANN searches. The search time is simply the time cost of the whole ANN search process.

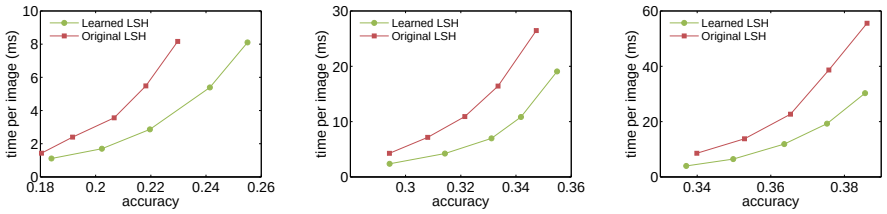

Fig. 2. The ANN search results of the original LSH and the the learned LSH. (a) The results obtained by using 2 hash tables. (b) The results obtained by using 6 hash tables. (c) The results obtained by using 10 hash tables.

Setups To demonstrate the advantage of the proposed indexing method, namely the learned LSH, we compare the ANN search performance of the method to that of the original LSH and the setups are listed below.

- **Original LSH**. Three configurations of the number of hash tables, *i.e.* 2,6 and 10 respectively, are used in the experiment. In each configuration the length of the hash key ranges from 17 to 13, resulting in different search accuracies and timings.
- **Learned LSH**. The configurations of the number of hash tables and the length of the hash key are identical to the original LSH. Each hash key in the learned LSH is the same as that in the Original LSH initially, and is changed during the SLAM process. The final hash keys used for the ANN search are the ones learned from all the 293 keyframes.

Results Fig. 2(a) to Fig. 2(c) demonstrate the ANN search results in terms of search accuracies and search timings obtained by using 2, 6, 10 hash tables respectively. It can be seen that under these configurations, the ANN search performances of the learned LSH are consistently improved over the original LSH. Using the learned LSH takes less time than using the original LSH to reach the same search accuracy. For example, when 2 hash table is used, the time cost can be reduced by about a half. These results indicate that the bit selection process is effective. Rather than being randomly generated, better hash keys can be learned by exploiting the features collected in the SLAM process. To be noted that, since only real time performance is of interest, the comparisons are performed in relatively low accuracy regions.

The evolution of the search efficiency. Since hash keys of the learned LSH are changed incrementally during the SLAM process, we may want to know how the search efficiency of the learned LSH evolves. To figure out this, some experiments are carried out as follows. We stop the bit selection process after k keyframes are inserted and then use the resulted hash keys to perform ANN search for query features in the above dataset. The results are shown in Fig. 3(a), from which we can see that the search efficiency roughly increases as k increases.

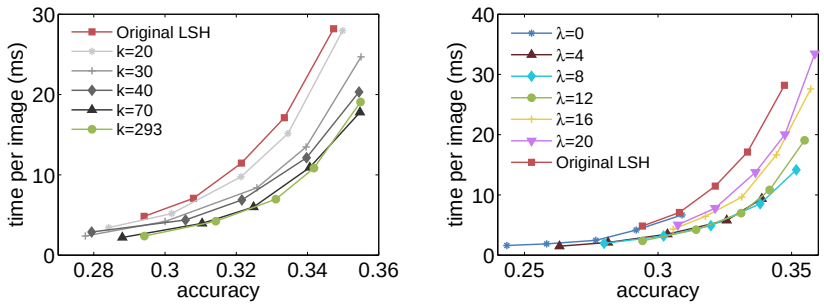

Fig. 3. (a) The ANN search results of the learned LSH with hash keys learned from the top k keyframes. 6 hash tables are used. (b) the ANN search results of the learned LSH with different values of λ. 6 hash tables are used.

This is reasonable since as k increases, more bit selection processes are performed to increase the chance of find good bits, and wider range of training data,*i.e.* larger number of database features, is also available. It can also be seen from Fig. 3(a) that as k increase to some extent, *e.g.* 70, the performance nolonger improves. This result suggests that it is not necessary for endless learning and we may stop the learning after enough bit selection processes are performed.

The choice of λ. There is a parameter λ in the cost function (Eq. 3) needed to be determined. Fig. 3(b) shows its impact on the efficiency of the indexing method. It can be found that either a too small value, *e.g.* 0, or a too big value *e.g.* 20, degrades the search efficiency, and a moderate value is preferred. So in this paper, we empirically set λ to be 12.

The timing of learning. The learning time cost is dominated by the bit selection process. Benefited from the strategy described in Section 3.2, the time cost of the bit selection process is constant. We use 10 hash tables with key length of 14 and find that selecting one bit for a hash key takes about 53 ms even the number of database features increases to 175207. Since we perform bit selection on a half of the hash keys at a time, the learning time cost at a time is about 260 ms. This time cost is favorable since the learning process runs on the background thread and 260 ms is much less than the time interval, usually several seconds, between two keyframes.

4.2 The Relocalization Performance

To evaluate the performance of the proposed relocalization module, we run this module on two video segments and try to relocalize each frame of them. The first video segment (V1) contains 8000 frames and is clipped from the video which has been used in Section 4.1 to evaluate the efficiency of the indexing method. The scene map involved in this segment contains 37,641 3D points and 175,207 database features. The second video segment (V2) is taken by moving

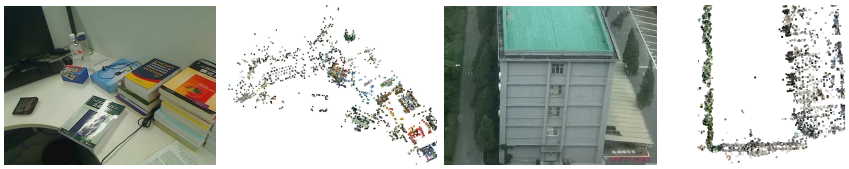

Fig. 4. Left, the scene in which the second video segment is taken. Right, the scene in which the first video segment is taken.

the camera above an office desk and contains 1063 frames. The map involved in the second video segment is built by running SLAM on another video, from which 31 keyframes are extracted and 4173 3D points along with 20,846 database features are obtained. Fig. 4 gives a shot on the scenes and maps involved in the two videos.

We follow the pipeline described in Section 3 to relocalize each frame of V1 and V2. Since the camera intrinsic parameters are fixed and have been calibrated in advance, we use RANSAC and EPnP[17] followed by a non-linear optimization to estimate the camera poses from 3D-2D correspondences. The relocalization is deemed as successful if more than 12 inlier correspondences are found in the RANSAC process. The learned LSH is used for ANN search in the process of establishing 3D-2D correspondences. For comparison, the original LSH is also used. In V1, both the original LSH and the learned LSH has 10 hash tables with key length of 14, and the hash keys in the learned LSH are learned from all the keyframes. In V2, the original LSH has 10 hash tables with key length of 14 and the learned LSH has 10 hash tables with key length of 13. Again the hash keys in the learned LSH are learned from all the keyframes.

Table 1. The relocalization results.

Video segments	V1		V2	
# query frames	8000		1063	
# 3D points	37,641		4173	
# database features	175,207		20,846	
Indexing method	Original LSH	Learned LSH	Original LSH	Learned LSH
# relocalized frames	*7832.0*	*7859.0*	*927.8*	*953.2*
Timings (ms)				
Corner detection	8.3	8.5	7.8	7.8
Descriptor extraction	8.3	8.3	10.03	10.03
ANN search	*30.4*	*17.5*	*3.7*	*3.5*
RANSAC	0.5	0.5	1.2	1.2
Total	48.3	35.9	24.8	24.6

Comprehensive Results. Table 1 summarizes the relocalizaiton results which are averaged over 5 repetitions to alleviate the effect of the randomness in RANSAC. It is shown that in V1, 7859 frames out of 8000 frames,*i.e.* 98.2% are successfully relocalized at a speed of the frame rate, which demonstrates the good scalability of the relocalization module for large maps. Besides, due to the higher efficiency of the learned LSH compared to the original LSH, the time cost of ANN search can be reduced by 42%, and a slightly more,*i.e.* 0.3%, frames can be successfully relocalized. In V2, the view point differences between the query frames and the keyframes are larger than that in V1, thus a lower registration rate, *i.e.* 89.6%, is obtained. Since the map in V2 is relatively small, the time cost saving by using the learned LSH is not as prominent as that in V1. However the improvement of the registration rate, 2.4%, is now more noticeable. This improvement should also be ascribed to the higher efficiency of the learned LSH: by taking the same time, using the learned LSH produces higher search accuracy than using the original LSH.

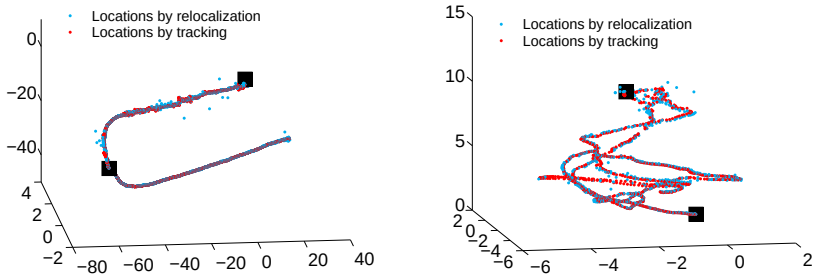

Fig. 5. The camera locations estimated by the relocalization module (the blue dots) and the tracking module (the red dots). (a)Camera locations of the first video segment. (b) Camera locations of the second video segment.

Relocalization accuracy. The aim of the relocalization is to restart the tracking. If the camera pose of a frame obtained by the relocalization module is close enough to the pose that should have been obtained by the tracking module, the tracking usually restarts successfully. So the accuracy of the relocalization should be measured by the differences between poses from the relocalization and poses from the tracking. In real situations, the relocalization is performed on frames suffering from tracking failure, and thus poses from the tracking can not be obtained. To evaluate the relocalization accuracy, we instead run the relocalization module on frames that are tracked without failure. Again, we use V1 and V2 in which all frames can actually be well tracked. Fig. 5 depicts the camera locations estimated by both modules. The black squares represent two special camera locations from the tracking module, one of them is the initial location and the other is the location farthest from the initial one. After dividing

the average distance between the tracked locations and the relocalized locations by the distance of the two special locations, we get a relative location error of 0.05% in v1 and 0.7% in v2. Defining the rotation error between two rotation matrices R_1 and R_2 as the angle of the rotation $R_1 R_2^T$, as in[4], we also get a average rotation error of 0.25 degrees in v1 and 0.4 degrees in v2. These results indicate that the relocalization is relatively accurate.

5 Conclusion

In this paper, we have presented a binary feature indexing method for real-time SLAM relocalization. The core of the indexing method lies in that hash keys are learned online with the aim of attaining uniform hash buckets and high collision rates. By implementing the learning process on the background mapping thread and activating it only when the map is updated, little latency is brought about to the system. Experiments show that the proposed indexing method is more efficient than LSH and the relocalization module is able to handle large maps with tens of thousands of landmarks.

Acknowledgement. This work was supported by the National Natural Science Foundation of China under Grant No. 61421004, the National Basic Research Program of China under grant No. 2012CB316302 and Nokia Research Grant No. LF14011659182.

References

1. Davison, A., Reid, I., Molton, N., Stasse, O.: Monoslam: Real-time single camera slam. IEEE Trans. Pattern Anal. Mach. Intell. **29** (2007) 1052–1067
2. Klein, G., Murray, D.: Parallel tracking and mapping for small ar workspaces. In: Proc. 6th IEEE and ACM International Symposium on Mixed and Augmented Reality. (2007) 225–234
3. Sattler, T., Leibe, B., Kobbelt, L.: Fast image-based localization using direct 2d-to-3d matching. In: Proc. IEEE Int. Conf. Computer Vision. (2011) 667–674
4. Lim, H., Sinha, S., Cohen, M., Uyttendaele, M.: Real-time image-based 6-dof localization in large-scale environments. In: Proc. IEEE Conf. Computer Vision and Pattern Recognition. (2012) 1043–1050
5. Yunpeng, L., Snavely, N., Huttenlocher, D., Fua, P.: Worldwide pose estimation using 3d point clouds. In: Proc. Eur. Conf. Computer Vision. (2012) 15–29
6. Lowe, D.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. **60** (2004) 91–110
7. Tola, E., Lepetit, V., Fua, P.: Daisy: An efficient dense descriptor applied to wide-baseline stereo. IEEE Trans. Pattern Anal. Mach. Intell. **32** (2010) 815–830
8. Williams, B., Klein, G., Reid, I.: Automatic relocalization and loop closing for real-time monocular slam. IEEE Trans. Pattern Anal. Mach. Intell. **33** (2011) 1699–1712
9. Ozuysal, M., Calonder, M., Lepetit, V., Fua, P.: Fast keypoint recognition using random ferns. IEEE Trans. Pattern Anal. Mach. Intell. **32** (2010) 448–461

10. Straub, J., Hilsenbeck, S., Schroth, G., Huitl, R., Moller, A., Steinbach, E.: Fast relocalization for visual odometry using binary features. In: Proc. IEEE Int. Conf. Image Processing. (2013) 2548–2552
11. Calonder, M., Lepetit, V., Strecha, C., Fua, P.: Brief: Binary robust independent elementary features. In: Proc. Eur. Conf. Computer Vision. (2010) 778–792
12. Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: Orb: An efficient alternative to sift or surf. In: Proc. IEEE Int. Conf. Computer Vision. (2011) 2564–2571
13. Leutenegger, S., Chli, M., Siegwart, R.: Brisk: Binary robust invariant scalable keypoints. In: Proc. IEEE Int. Conf. Computer Vision. (2011) 2548–2555
14. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proc. 25th Int. Conf. Very Large Data Bases. (1999) 518–529
15. Rosten, E., Drummond, T.: Machine learning for high-speed corner detection. In: Proc. Eur. Conf. Computer Vision. (2006) 430–443
16. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and auto cartography. Commun. ACM **24** (1981) 381–395
17. Lepetit, V., Moreno-Noguer, F., Fua, P.: Epnp: An accurate o(n) solution to the pnp problem. Int. J. Comput. Vis. **81** (2009) 155–166