Real-Time RGB-D Camera Relocalization

Ben Glocker* Shahram Izadi Jamie Shotton Antonio Criminisi

Microsoft Research, Cambridge, UK

ABSTRACT

We introduce an efficient camera relocalization approach which can be easily integrated into real-time 3D reconstruction methods, such as KinectFusion. Our approach makes use of compact encoding of whole image frames which enables both online harvesting of keyframes in tracking mode, and fast retrieval of pose proposals when tracking is lost. The encoding scheme is based on randomized ferns and simple binary feature tests. Each fern generates a small block code, and the concatenation of codes yields a compact representation of each camera frame. Based on those representations we introduce an efficient frame dissimilarity measure which is defined via the block-wise hamming distance (BlockHD). We illustrate how BlockHDs between a query frame and a large set of keyframes can be simultaneously evaluated by traversing the nodes of the ferns and counting image co-occurrences in corresponding code tables. In tracking mode, this mechanism allows us to consider every frame/pose pair as a potential keyframe. A new keyframe is added only if it is sufficiently dissimilar from all previously stored keyframes. For tracking recovery, camera poses are retrieved that correspond to the keyframes with smallest BlockHDs. The pose proposals are then used to reinitialize the tracking algorithm.

Harvesting of keyframes and pose retrieval are computationally efficient with only small impact on the run-time performance of the 3D reconstruction. Integrating our relocalization method into KinectFusion allows seamless continuation of mapping even when tracking is frequently lost. Additionally, we demonstrate how marker-free augmented reality, in particular, can benefit from this integration by enabling a smoother and continuous AR experience.

1 INTRODUCTION

Recent advances in dense tracking and mapping [14] have enabled systems for accurate, real-time 3D reconstruction of the physical world using low cost commodity hardware [13] such as Microsoft's Kinect sensor. Real-time reconstruction enables exciting applications in augmented reality (AR) [9, 12]. Knowing the actual 3D geometry of objects in a scene combined with the ability to sense the depth at frame rate overcomes several challenges in AR such as occlusion handling and fusion of real and virtual objects.

The underlying processing pipeline of different reconstruction systems based on depth sensors is similar. The camera motion is tracked frame-to-frame where the pose update is determined by computing a relative transformation between a (partially) reconstructed world (*i.e.* the map) and a 3D point cloud obtained from the live depth measurements given by the sensor. This transformation can be for instance computed by employing a robust version of the iterative closest point (ICP) algorithm as it is implemented in the KinectFusion approach [13]. Given the estimated camera pose, new 3D measurements are integrated into the map yielding an updated and refined reconstruction of the scene. Different variants of this 3D reconstruction pipeline exist which mainly differ in the details how tracking [12, 23, 24] and mapping [8, 20, 25] are implemented.

*e-mail: glocker@tum.de

1.1 Importance of Relocalization

In order to acquire an accurate map of the scene, these reconstruction pipelines rely on a steady stream of successfully tracked frames. Tracking failure can have severe consequences. Integrating measurements with incorrect poses yields implausible, invalid geometry and might destroy already reconstructed areas in the map. Indeed, if tracking failure can be detected, at least map corruption can be prevented. However, in augmented reality applications in which the pose of the camera is required to correctly overlay virtual objects onto the real world, tracking failure leads to an abrupt and unpleasant end of the user's experience.

The causes for tracking failure are versatile. Rapid camera motion and sudden change of viewpoint are probably the predominant ones where frame-to-frame tracking fails. In addition, and particularly relevant to systems where the map is spatially restricted to a limited area of the world, tracking which relies on the reconstructed map fails every time the camera points outside the restricted domain. This can frequently happen in AR scenarios in which the user is in full control of a hand-held or head-mounted camera.

To this end, it is of great practical importance to integrate a camera relocalization module which allows instant recovery from tracking failure. Incorporated into the 3D reconstruction pipeline, such a module allows seamless continuation of mapping even when camera tracking is frequently lost. This avoids the frustrating task of restarting an entire scan from scratch due to tracking failure. Also, this makes AR applications more robust. Augmentations can be visualized with the correct transformation immediately after successful recovery of the camera pose.

In this paper we propose a relocalization module which can be easily integrated into existing reconstruction pipelines. Our approach is inspired by different components of previous work, resulting in an efficient and scalable algorithm that provides a solution to the main causes of tracking failure in systems such as KinectFusion. We will demonstrate that both continuation of mapping and robust AR are enabled by our algorithm.

1.2 Related Work

Camera relocalization has been studied in the context of simultaneous tracking and mapping (SLAM) systems. While approaches exist that require an offline training phase (*e.g.* [22]), we focus below on approaches which are capable of *online* real-time performance. One can roughly categorize existing approaches into two categories, though hybrid [18] and more exotic variants exist [2, 15].

The first category are landmark-based approaches (LbAs) [3, 26]. During successful tracking, fiducial landmarks, also called *keypoints*, are extracted from the camera images, encoded by a descriptor, and stored in a database together with their 3D locations. When tracking is lost, landmark candidates are detected in the incoming frame and based on descriptor similarity putative matches are established between those candidates and stored keypoints. The combination of the three-point-pose algorithm and RANSAC [5] is then commonly employed to determine the pose of the camera.

We denote the second category as image-based approaches (IbAs) [6, 10]. Indeed, all methods discussed here rely on image information, however, the main difference to the first category is that IbAs make use of global image matching and do not require explicit landmark detection. During successful tracking, compact

representations of whole images are generated and stored together with the corresponding camera poses. Those frame/pose pairs are commonly referred to as *keyframes*. When tracking is lost, the compact representation of the incoming frame is compared to the ones of all keyframes. The poses of the most similar keyframes are retrieved and then used to directly reinitialize the tracking algorithm.

Both approaches have their advantages and drawbacks. Arguably, the main advantage of LbAs is their ability to recover the pose for frames from novel views. As long as a sufficient number of keypoints can be recognized, the camera pose can be determined. Depending on the visual characteristics of the scene and possible artifacts such as motion blur, it might not always be possible to obtain sufficiently many matches. Also, the construction of the keypoint database in real-time settings can be challenging. Often a costly online training phase is required which demands additional resources such as a background thread or extra GPU computations [26]. Another limitation of LbAs lies in their inherent sparse representation of the scene. Some approaches are limited to store only a few thousand unique points, as discussed in [3, 10]. The optimal choices for a suitable pipeline of robust detection [19], description [1, 27] and matching [16] is certainly a challenge on its own.

Instead of map locations, in IbAs a set of densely distributed keyframes represents the reconstructed scene. This allows direct retrieval of pose proposals for tracking initialization and does not require the pose estimation step of LbAs. The main challenges are related to the online determination of keyframes and the definition of efficient frame similarity measures. For the first part, often heuristics are employed such as thresholds on distances in pose space. For instance, a tracked frame is added to the set of keyframes only if the camera translation and orientation are sufficiently different from the previous keyframe [6]. This heuristic might yield non optimal scene coverage. The second issue regarding efficient similarity evaluation is commonly tackled by using compact representations such as heavily downsampled images and normalized intensity differences [6, 10]. With increasing number of keyframes, the time needed for the search of the most similar ones can be a limiting factor of IbAs in real-time settings. The fact that tracking can only be recovered from views which have been approximately visited before has been recently approached by utilizing synthesized views [6]. However, rendering such views can be costly and defining an optimal sampling strategy in pose space is non trivial.

In light of prior work, we propose a simple yet powerful relocalization approach which falls into the IbA category. Our method makes use of randomized ferns which have been previously used in the context of keypoint-based relocalization [26]. The way in which we make use of ferns is quite different and will be described in Section 2. Our relocalization approach comes with the following contributions:

1. Efficient frame encoding scheme allows compact representation of keyframes, and fast retrieval of pose proposals for tracking recovery;

2. Automatic discovery and sampling of keyframes by exploring the space of appearance during tracking and avoiding spatial sampling heuristics;

3. Scalability to large scenes through a small memory footprint, large model capacity, and minimal computational demands.

These properties are essential for the integration of our method into 3D reconstruction pipelines. After discussing the technical details, we investigate the relocalization performance in an experimental evaluation in Section 3. We then discuss in Section 4 a practical application of marker-free AR realized with the KinectFusion system equipped with our relocalization module. In Section 5, we discuss current limitations and future work.

2 Relocalization via Randomized Ferns

The underlying concept of our relocalization approach is based on the ability of compact code generation for camera frames and efficient evaluation of code similarities between different frames. We start by introducing the idea of using randomized ferns for generating compact codes.

2.1 Frame Encoding

Given an RGB-D image frame $I : \Omega \subset \mathbb{R}^2 \mapsto \mathbb{R}^4$, we define a function to read an individual pixel value $I_c(\mathbf{x}) \in \mathbb{R}$ at location $\mathbf{x} \in \Omega$ in channel $c \in \{R, G, B, D\}$. For convenience we introduce the notation $I(\theta) = I_c(\mathbf{x})$ with $\theta = (c, \mathbf{x})$.

In order to be able to generate compact code representations for RGB-D image frames, we first define a fern $F = \{f_i\}_{i=1}^n$ as a set of n consecutive nodes f_i where each node represents a binary test parametrized by a pair (θ_i, τ_i). Each test can be evaluated on the image data as

$$f(I, \theta, \tau) = \begin{cases} 1 & \text{if } I(\theta) \geq \tau \\ 0 & \text{if } I(\theta) < \tau \end{cases} . \tag{1}$$

Here, τ is a threshold on the image pixel value $I(\theta)$. Evaluating all f_i of a fern F in consecutive order generates a binary code block $b_F = f_1 \dots f_n \in \mathbb{B}^n$. Further, if we are given a conservatory $C = \{F_k\}_{k=1}^m$ of m diverse ferns, we can concatenate their individual code blocks into a single code $b_C = b_{F_1} \dots b_{F_m} \in \mathbb{B}^{mn}$. This mechanism allows us to generate (non unique) codes for any RGB-D image frame. A particular binary image code b_C^I depends on the total number of ferns m, the number of nodes n in each fern, the binary test parametrization (θ_i, τ_i) of each node, and of course on the visual content of the image frame I.

The idea of using ferns for generating codes for image patches has been introduced in [17]. This has also been applied to the task of camera relocalization [26]. In both works, the idea is to learn compact codes which allow efficient keypoint recognition instead of encoding whole image frames. At test time, the conservatory of ferns is utilized as a classifier in order to find putative matches between incoming frames and a learned keypoint database.

Although inspired by the great performance of those fern-based approaches, we use ferns in a way which is quite different from previous works. We employ whole frame encoding for our keyframe-based relocalization method for which we propose a test procedure which allows us to simultaneously compute frame dissimilarities between a new frame and all previously encoded frames. Let us first define how we determine the dissimilarity between frames.

2.2 Frame Dissimilarity via Hamming Distance

Given compact representations b_C^I and b_C^J for two camera frames I and J, we can define their frame dissimilarity in terms of the block-wise hamming distance (BlockHD) as

$$\text{BlockHD}(b_C^I, b_C^J) = \frac{1}{m} \sum_{k=1}^{m} b_{F_k}^I \equiv b_{F_k}^J , \tag{2}$$

where the equivalent operator $\equiv$ returns 0 if two code blocks are identical and 1 if there is at least one bit difference. Thus, the BlockHD is simply counting the number of differing code blocks. The normalization with respect to m maps the distance to the $[0, 1]$ interval which is a nice property for parameter selection of an algorithm. In contrast to the well-known bit-wise hamming distance (HD) which counts the number of differing bits of two codes, the block-wise version has a property which is important for our task. Varying the parameter n which impacts the length of the blocks, also directly impacts the precision/recall characteristics of BlockHD when utilizing it for the search for similar frames. Intuitively, the probability that code blocks b_F^I and b_F^J are equivalent

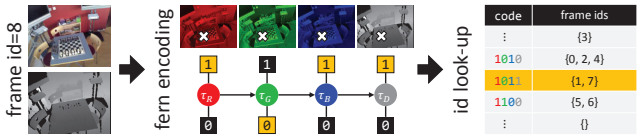

Figure 1: **Frame encoding:** Fern-based frame encoding takes an input RGB-D image and generates small code blocks for each fern based on simple binary tests evaluated at random image locations. A code block points to a row of a code table associated to a fern storing the *id*s of keyframes with equivalent codes. In harvest mode, the *id* of the incoming frame is added to the row if the minimum dissimilarity κ_I is above a threshold. For tracking recovery, poses are retrieved from a hash table corresponding to keyframes that are most similar. Similarity depends on the counts of co-occurrence of the incoming *id* and the ones stored in different code tables.

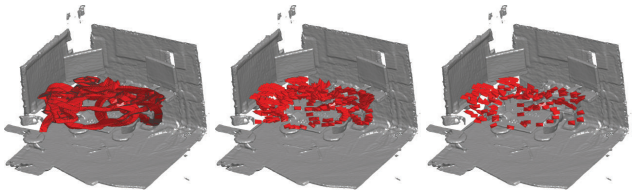

Figure 2: **Harvesting keyframes:** The left image shows a reconstructed scene with 4000 tracked frames. Each camera pose is shown as a red frustum. When simulating harvesting from this stream of frames, 310 are accepted as keyframes for $\kappa_I > 0.2$ (middle). Increasing the threshold to $\kappa_I > 0.3$ yields only 128 frames being accepted (right). We found that the coverage in the middle is favorable over the one on the right, while an even denser coverage does not improve relocalization performance (see Fig. 5).

decreases with increasing bit length due the increasing number of binary tests for which the image data would need to yield the same responses. This would increase precision but might decrease recall, because two frames with low BlockHD are likely to be very similar if the block is long. Though we might miss other similar frames due to image noise or just by chance because of the hard thresholding in Eq. (1). In contrast, for shorter codes the probability is higher that two frames with different visual content yield equivalent responses. The recall intuitively increases in this case, though the precision might be rather low. We might observe many different frames with very similar codes. In case of 1-bit blocks which corresponds to ferns with only one node, the BlockHD is equivalent to the bit-wise HD. We will now describe how we can efficiently compute the BlockHD.

2.3 Harvesting Keyframes

The main difference to previous fern-based approaches is the way we utilize the output of the fern. Remember a fern with n nodes can generate 2^n unique codes. We associate each fern F with a code table T_F with 2^n rows. Each row stores a set of frame identifiers (*id*s)[1] and all sets are initially empty. In addition to the sets of *id*s which are unique for each fern, we also define one global hash table P taking *id*/pose pairs as input elements. This set is also initially empty and will be used to store the camera poses of keyframes.

Let us now assume a steady stream of tracked camera frames with pairs $(I,H)_{id}$ where $H \in SE(3)$ is the camera pose composed of rotation and translation. Here, the *id* is assumed to be unique for a frame/pose pair. For each incoming frame we can generate the code b_C^I and add its *id* to the m sets in the corresponding rows which are associated with the individual code blocks. Additionally, we would add the pose H with key *id* to the global hash table P.

Assuming a couple of tracked frames have been already encoded and stored using this strategy. Now, every time we are about to add a new *id* to a set in a row of a code table T_F, we can also immediately read out the previously stored identifiers (*cf.* Fig. 1). We know that those must correspond to frames which have an equivalent code block b_F. In fact, if we simply count those co-occurrences of previously stored frames along the m rows where we are about to add the new *id*, we can *simultaneously* compute the dissimilarities between the new frame and all stored frames. Assuming the count of co-occurrence for two frames I and J using the above procedure is denoted as q_{IJ}, then we can equivalently to Eq. (2) compute their dissimilarity by

$$\text{BlockHD}(b_C^I, b_C^J) = \frac{m - q_{IJ}}{m} \quad . \tag{3}$$

[1]This is different to ferns used for classification [17, 26] where empirical distributions over keypoints are stored in the rows of the code tables.

In addition, for every incoming new frame I we can determine the minimum BlockHD with respect to previously stored frames as

$$\kappa_I = \min_{\forall J} \text{BlockHD}(b_C^I, b_C^J) = \min_{\forall J} \left(\frac{m - q_{IJ}}{m} \right) \quad . \tag{4}$$

The value κ_I provides useful information about how well the new frame is represented by the set of stored keyframes. A low value reflects that a very similar frame is already present, while a high value indicates a novel view from a pose which should probably be stored as a keyframe. Based on this observation, we propose a strategy for online harvesting of tracked frames and automatic determination of keyframes. Based on the value κ_I and a predefined threshold t, we decide whether an incoming frame is added or discarded. It should become clear that such a threshold influences how densely the reconstructed scene is covered by keyframes (see Fig. 2). Intuitively, a compromise is desired which avoids redundant information to be added to the scene representation while the coverage of the scene should be sufficiently dense. Note, that our harvesting strategy is completely driven by the visual content of the camera frames and spatial sampling heuristics based on pose offsets [6] are avoided.

2.4 Tracking Recovery

Once online harvesting of tracked frames is in place and a few keyframes have been collected, tracking recovery can be achieved through fast retrieval of pose proposals. Assuming camera tracking has failed for an incoming frame. We can still perform exactly the same encoding procedure as in harvest mode and efficiently compute the dissimilarities to all stored keyframes. Now, instead of determining the value of the minimum distance, we directly determine the *id*s of the k nearest keyframes. This allows us to retrieve their stored camera poses from the hash table P. Depending on the underlying tracking and mapping system, we can use those k poses to reinitialize the tracking algorithm. It is also possible to employ a weighted averaging scheme to regress a pose proposal using the retrieved poses and their distances similar to the idea proposed in [6]. If reinitialization is unsuccessful for all proposed poses, we repeat the same procedure for the next incoming camera frame until tracking is recovered. In the meantime, the mapping process is paused and automatically resumed after recovery. Remember we assume a user controlled camera, and it seems likely that at some point the user will move the camera to a similar pose which has been successfully tracked previously. One could also imagine a visual guidance which helps the user to move the camera into areas of keyframe coverage. We found that in systems such as KinectFusion displaying the currently reconstructed scene usually gives sufficient guidance.

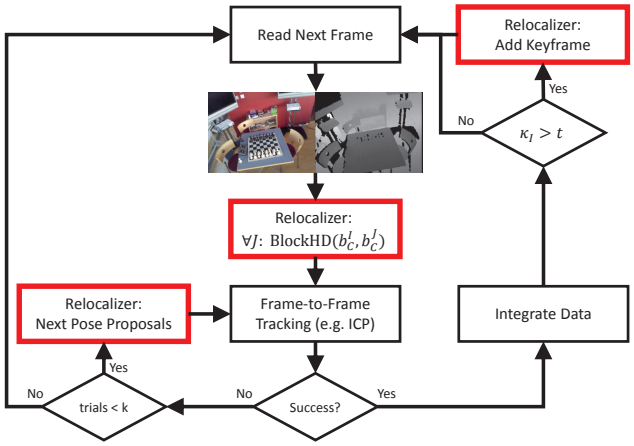

Figure 3: **Pipeline integration:** The diagram illustrates the integration of our relocalization module into a standard 3D reconstruction pipeline such as KinectFusion. Every incoming frame is pushed through our encoding and frame dissimilarity mechanism. Depending on whether frame-to-frame tracking is successful, we can forward tracked frames to the map integration and keyframe extraction procedures, or we initiate pose proposal retrieval for tracking recovery.

2.5 Fern Construction

We have not discussed the details for the fern parameters introduced in Section 2.1. In particular, the binary test parameters (θ_i, τ_i) seem crucial for obtaining useful compact frame representations. However, and maybe not too surprising, we found that randomness injected into the construction of the ferns yields overall best performance. Similar findings are reported in related approaches [11, 17, 26]. Our default construction strategy for the entire conservatory of ferns is explained in the following. The impact of parameters such as the number of ferns m and the keyframe acceptance threshold t are investigated in our experiments.

Each fern consists of $n = 4$ nodes with one node per RGB-D channel, yielding $F = \{f_R, f_G, f_B, f_D\}$. Here, this also predefines the parameters c_i within the set $\{\theta_R, \theta_G, \theta_B, \theta_D\}$. We randomly sample the image locations $\mathbf{x}_i$ at which the binary tests are applied from a uniform distribution over the image domain Ω. Here, we sample one location per fern, such that $\mathbf{x}_R = \mathbf{x}_G = \mathbf{x}_B = \mathbf{x}_D$. This has the effect that a code block b_F corresponds to feature responses from all image channels obtained at the same location. The feature test thresholds τ_i are uniformly sampled such that $\tau_R, \tau_G, \tau_B \in [0, 255]$ for RGB and $\tau_D \in [800, 4000]$mm for depth.

3 Experimental Evaluation

We created the '7-scenes' dataset[2] to evaluate relocalization performance. All scenes were recorded from a handheld Kinect at 640×480 resolution. We used KinectFusion [13] (with care to avoid loss of track) to obtain the 'ground truth' camera poses. For each scene, several sequences were captured by different users, and split into two distinct evaluation sets. One set is used as a steady stream of tracked frames for simulating harvesting of keyframes, the other set is used for error calculation. The frames exhibit ambiguities (*e.g.* repeated steps in 'Stairs'), specularities (*e.g.* reflections in 'RedKitchen'), motion blur, lighting conditions, flat surfaces, and sensor noise. An overview of the dataset is shown in Fig. 4 and details are given in Table 1. The varying difficulties of the scenes are reflected in the errors, consistently across different approaches. In the following, our relocalization module operates on downsampled, smoothed images with a resolution of 40×30 and Gaussian blur of $\sigma = 2.5$.

[2] http://research.microsoft.com/7-scenes/

3.1 Overall System

We integrated our relocalization module into the KinectFusion pipeline (see Fig. 3) which relies on model-based ICP frame-to-frame tracking. In order to detect ICP tracking failure (and success), we employ a plausibility check on the magnitude of camera motion and on the ICP residual error. If the relative motion or residual is too large, ICP reports tracking loss and relocalization is initiated. A short demonstration of this system is shown in the supplementary video which highlights the importance of the relocalization module. In particular, the system is able to immediately recover tracking when the camera frequently leaves and re-enters the limited volume of the world that is being reconstructed.

3.2 Pose Proposal Strategies

There are several ways of using a pose retrieval approach such as ours for relocalization. One way is to simply initialize the tracking algorithm with the nearest neighbor (NN) pose, *i.e.* the one of the keyframe with smallest BlockHD. It is also possible to retrieve a set of kNN proposals and initialize the tracking with each of those. Besides proposing directly the poses of keyframes, we can also interpolate a proposal via weighted averaging over the kNN poses [6].

In the following, we will compare relocalization performance for different strategies, namely NN, kNN, and weighted average pose (WAP). In contrast to NN and WAP, kNN corresponds to a multiple proposal approach where the poses of the k closest keyframes plus their WAP are used for initializing ICP. If ICP reports success for several of those poses, the one with lowest residual error is selected for continuing normal frame-to-frame tracking and mapping.

3.3 Tiny Image Baseline

We compare our method to another keyframe approach denoted as 'tiny image baseline'. This baseline represents whole image matching approaches [6, 10]. Keyframes are stored with their camera poses, after downsampling to 40×30 pixels and applying a Gaussian blur of $\sigma = 2.5$ pixels [10]. For pose retrieval with best possible accuracy (at the expense of speed), we use brute-force matching against *all* available stored frames using the normalized distance over RGB-D as defined in [6]. It should be noted that this exhaustive search is impractical for real-time systems. To this end, keyframe sampling heuristics are usually employed to keep the number of keyframes at a reasonable level [6]. Such heuristics, however, do not guarantee sufficient coverage of a scene. In order to eliminate the factor of insufficient keyframe density, we opt for the brute-force search strategy which should yield best possible performance for the baseline. We use the same pose proposal strategies as for our method, *i.e.* NN, kNN, and WAP.

3.4 Error Metric

Our main metric for comparing different settings and approaches is the percentage of frames for which the pose was successfully recovered. Here, we define recovery to be successful if the final estimated pose of frame-to-frame tracking after ICP is within 2 cm translational error and 2 degrees angular error compared to ground truth.

3.5 Results

In the following we investigate different properties of our relocalization. Besides quantifying the actual performance for pose recovery in comparison to a baseline, we also explore the impact of different parameters and the scalability of our method to larger scenes. We also report detailed timings and computational impacts of the individual components of our method.

Effectiveness: Our main quantitative results are summarized in Table 1. Both, our approach and the baseline perform best in terms of successfully recovered frames when using the kNN proposal strategy. Overall, our approach is able to recover from significantly more test frames compared to the baseline. In all the exper-

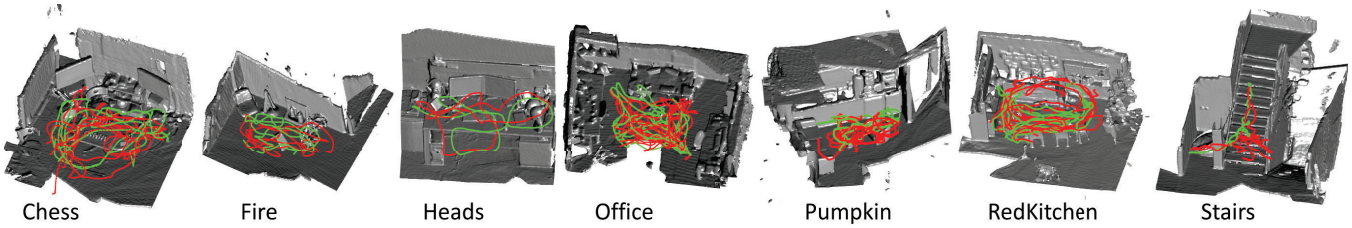

Figure 4: **RGB-D dataset:** For each of the seven scenes, we have recorded several sequences of tracked RGB-D camera frames. The frame trajectories used for simulating the online harvesting are shown in red. The frames used for evaluating tracking recovery are shown green.

Table 1: **Main results:** Summary of the relocalization performance evaluated on the RGB-D dataset '7-scenes'. Percentages correspond to the number of successfully recovered frames (within 2cm translational error and 2° angular error). The best performance is obtained with our relocalization approach and the *k*NN pose proposal strategy. Remarkably, the scalability test on the 'All-in-One' representation indicates similarly good performance compared to scene-specific representations, indicating promising scalability properties for our compact encoding scheme.

	Spatial	**#Frames**		**Tiny Image Baseline**			**Our Results (m=500, t=0.2)**				**All-in-One**
Scene	**Extent**	Harvest	Recovery	NN	WAP	*k*NN	NN	WAP	*k*NN	Keyframes	*k*NN
Chess	3m^3	4k	2k	70.8%	72.4%	80.6%	68.1%	76.8%	**85.8%**	310	82.0%
Fire	4m^3	2k	2k	56.6%	56.9%	60.9%	55.4%	52.1%	**71.2%**	136	66.0%
Heads	2m^3	1k	1k	49.0%	50.2%	61.0%	52.3%	42.1%	**81.1%**	93	76.1%
Office	5.5m^3	6k	4k	60.8%	61.3%	65.8%	60.9%	51.1%	**76.0%**	666	72.7%
Pumpkin	6m^3	4k	2k	54.6%	56.1%	59.7%	48.2%	51.9%	**63.9%**	297	62.5%
RedKitchen	6m^3	7k	5k	46.1%	46.8%	49.3%	44.7%	41.6%	**54.8%**	574	52.7%
Stairs	5m^3	2k	1k	25.2%	27.3%	29.9%	18.3%	25.0%	**37.0%**	72	32.7%
Average				51.9%	53.0%	58.2%	49.7%	48.6%	**67.1%**		63.5%

iments, we set $k=5$ such that in total 6 poses (including the WAP) are used for initializing ICP. We have also tested *k*NN without including the interpolated pose and got consistently worse results for both methods (not reported here). Using WAP alone as a single proposal does not perform too well for both our method and the baseline (see Table 1). However, adding it as an additional pose proposal in the *k*NN approach seems beneficial.

The given percentage of successful frames is always with respect to the total number of frames given in the column 'Recovery'. The column 'Harvest' indicates the number of frames used for simulating keyframe harvesting. The number of accepted keyframes for our approach is given in the column 'Keyframes'. Note that these quantitative results compare all approaches in an offline setting where all Harvest frames are observed before the Recovery frames are tested. For examples of the online relocalization system of Fig. 3, please see the supplementary video.

Parameters: The main results are obtained with our default setting of $m=500$ ferns and a keyframe acceptance threshold of $t=0.2$. In Fig. 5 we compare the performance of our method using the *k*NN strategy with varying parameters m and t. We observe that initially adding more ferns improves relocalization while further improvement beyond 500 ferns is marginal. Regarding keyframe acceptance, we find that a lower threshold ($t=0.1$) yielding denser scene coverage does not necessarily improve relocalization. Setting the threshold too high ($t=0.3$) yields quite sparse sampling of keyframes and relocalization performance decreases. A visual example for the coverage of the 'Chess' scene for $t=0.2$ and $t=0.3$ is shown in Fig. 2. The number of accepted keyframes for the three different thresholds is for our largest scene 'RedKitchen' 1388, 574, and 277, and for the smallest scene 'Heads' 222, 93, and 43.

Scalability: An important aspect of keyframe-based approaches is scalability to large scenes. In particular, this is a challenge for whole image matching approaches such as our baseline where the search for closest keyframes can become impractical when thousands of keyframes are stored. In order to evaluate the scalability of our approach, we performed the following experiment. We constructed a single conservatory of ferns with our default parameters $m=500$ and $t=0.2$. We then used all 26,000 frames from all seven scenes for keyframe harvesting. In total, 2091 frames are accepted as keyframes, which is slightly less than the sum over all keyframes from individual scenes, *i.e.* 2148. In the most right column in Table 1 we report the relocalization performance when using this 'All-in-One' keyframe representation. The overall performance is only slightly worse compared to the scene-specific constructions. We believe this indicates promising scalability properties for our encoding scheme and a model capacity which is sufficient for representing large scenes of more than 30m^3 and thousands of frames.

Timings: Real-time performance of relocalization is essential when being integrated into a tracking and mapping pipeline. In the following we report average timings for individual components of our method. All timings have been acquired while running KinectFusion on the 'RedKitchen' scene where a relocalization module based on the default setup and 574 stored frames is assumed to be in place. This allows us to measure the expected impact of keyframe harvesting and recovery under realistic conditions. While the KinectFusion pipeline itself is mostly running on GPU (Nvidia Geforce GTX 580), our relocalization runs currently entirely on the CPU using a single core (Intel Xeon 2.27 GHz).

The key intervention to the existing pipeline is that every incoming frame is pushed through the frame encoding and dissimilarity computation mechanism. So, even in normal tracking mode our method has an impact on the overall tracking and mapping performance. We found this impact to be very small, with only 3ms for frame encoding including computation of κ_I. A KinectFusion system running at 30 FPS will continue to run at about 27 FPS when keyframe harvesting is running in the same computation thread. Of course, the relocalization module could also run in a parallel, dedi-

Figure 5: **Parameter evaluation:** Relocalization performance when varying the number of ferns m (top graph) and changing the keyframe acceptance threshold t (bottom graph). Initially increasing the number of ferns improves performance, which levels off after $m=500$. The threshold t influences the number (in white) and density of keyframes. Optimal coverage is obtained with $t=0.2$ (see Fig. 2 for an example).

cated thread. When tracking is lost, we measure 160ms for camera recovery with kNN including 6 runs of ICP (for the k+1 proposals). The total impact of relocalization during tracking recovery is about 165ms, which keeps the system reasonably responsive. For comparison, the timings increase slightly for the 'All-in-One' representation with 2091 keyframes where harvesting takes about 7ms.

4 Marker-Free Augmented Reality

Real-time 3D reconstruction provided by RGB-D systems such as KinectFusion enables exciting applications. Here, we present a prototype of an AR system with potential use in medical and industrial environments where digital 3D models or scans of physical objects are available. In medical settings, anatomical scans of patients are frequently acquired with computed tomography (CT) or magnetic resonance imaging (MRI) systems for the task of diagnosis, interventional planning and surgical navigation. For example, in keyhole surgery the doctor uses the anatomical scan to plan the port placement of instruments to have optimal access to the region of interest [4]. Industrial AR systems for support and training of complex maintenance tasks benefit from overlay of 3D geometries and other information extracted from available CAD or mesh models [12, 21]. In contrast to systems based on optical tracking and RGB only cameras, an additional depth sensor can overcome challenges such as occlusion handling and fusion of real and virtual objects.

Let us assume we are given a surface mesh of a real world object consisting of a set of n 3D vertices $V = \{\mathbf{v}_i\}_{i=1}^{n}$. For example, the mesh could have been extracted from an available 3D scan and could represent the skin surface of a patient's head. In order to be able to correctly overlay object-specific visual information on top of the camera image, we need to find a transformation $T : \mathbb{R}^3 \mapsto \mathbb{R}^3$ which registers the mesh and the online 3D reconstruction. We assume the latter is represented as a truncated signed distance function (TSDF) [13] denoted as $D : \mathbb{R}^3 \mapsto \mathbb{R}$ where zero-crossings correspond to object surfaces. Finding the optimal transformation $\hat{T}$ can be formulated as an optimization problem as follows

$$\hat{T} = \arg\min_{T} \sum_{i}^{n} \min\left[\left|D(T(\mathbf{v}_i))\right|, \lambda\right] \quad . \tag{5}$$

Here, the value λ truncates the cost function which makes it robust to outliers and missing data in partially reconstructed objects. The minimum of the cost function corresponds to the transformation where a large number of mesh vertices is located at object surfaces, *i.e.* at zero-crossings in the TSDF volume. The advantage of this direct mesh-to-volume registration is that no explicit correspondences are required between the 3D model and the reconstruction. However, a sufficiently good initialization is important.

As mentioned earlier, we assume a user controlled camera which allows us to employ a simple manual initialization mechanism which works well in practice. Although, an automatic approach can be envisioned [7]. When a reasonable part of the object to be augmented has been reconstructed, we display the 3D mesh model with a fixed offset in front of the camera. The user's task is then to navigate the mesh model by moving the camera to the proximity of the reconstructed object. Enabling z-buffer based rendering gives additional guidance to the user when the mesh and the reconstruction are sufficiently close. The actual registration can then be performed automatically. We employ the downhill simplex algorithm as an iterative optimization method which requires less than 5 seconds for the registration. Once the transformation according to Eq. (5) is determined, any information contained in the object's 3D model or scan can be correctly overlaid on top of the tracked camera images. This procedure enables compelling, marker-free AR which for example allows to peek inside the human body and visualize anatomical details of clinically relevant structures as illustrated for a head phantom in Fig. 6. The supplementary video contains further examples.

5 Conclusion

The goal of this work was to develop an effective relocalization module which provides a solution to the main causes of tracking failure in systems such as KinectFusion. We demonstrated the performance of our module in a set of experiments and showed that continuation of mapping and marker-free AR can greatly benefit from our approach. A limitation of keyframe approaches in general is that for tracking recovery the camera view should not be substantially different from views represented by the keyframes. This is less of a problem in systems where the user is in full control of the camera, as it is intuitive to navigate the camera to an area that has been previously visited. In addition, in [6], a solution to this problem is proposed by employing synthetic view sampling. We believe a similar idea can be implemented here where our compact frame encoding is a key advantage. Instead of synthesizing whole frames, which can be costly, we only need to synthesize the compact codes for novel views which could dramatically reduce computational costs. This would potentially allow us to generate many views on-the-fly and allow a dense coverage of the reconstructed scene. One could think of employing a scene density estimator via the minimum BlockHD value κ_I as defined in Eq. (4) which could drive the view sampling towards undiscovered areas. This would overcome the need for sampling heuristics.

It would be interesting to investigate how loop-closure detection could be realized with our approach. Similar to [3], we could build a scene graph where stored keyframes correspond to nodes. As pointed out in [3], loop-closure and tracking recovery are similar events of edge creation between active and existing graph nodes.

In a similar context, one could explore the use of continuous pose retrieval for detecting tracking drift. Assuming a steady stream of tracked frames obtained from frame-to-frame tracking, the estimated poses could be compared on-the-fly to poses retrieved from the keyframe representation. A deviation in pose could indicate a drift and this information could potentially be used for online correction of the reconstructed map.

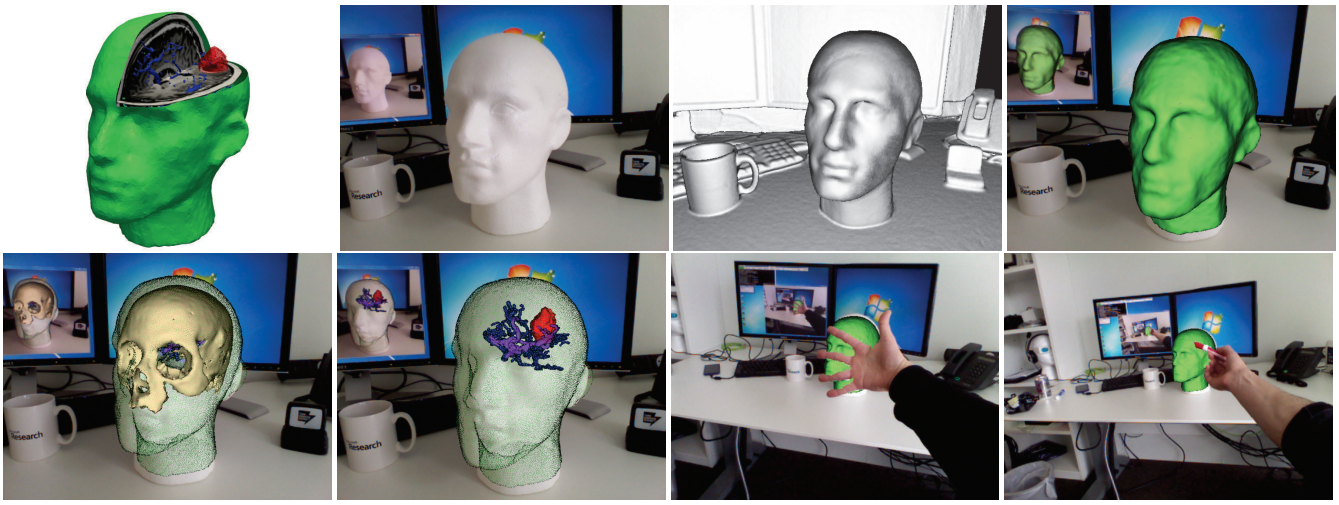

Figure 6: **Marker-free augmented reality:** Illustrated is a potential medical AR application. The top left image shows a skin surface model of a head phantom fused with a real MRI scan of a patient with a brain tumor. Relevant clinical structures such as the tumor (red) and important blood vessels (blue) are highlighted. The segmentation is commonly done during interventional planning. The second image shows our polystyrene head phantom. Reconstructing the same outer surface with a real-time system such as KinectFusion (third image, top row) allows a mesh-to-volume registration between the surface model and the TSDF of the reconstruction (top right image). The bottom row shows in-situ AR visualizations of anatomical structures overlaid on the RGB camera view. The live depth measurements also enable realistic occlusion handling.

REFERENCES

[1] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary robust independent elementary features. In *European Conference on Computer Vision (ECCV)*, 2010.

[2] M. Cummins and P. Newman. Appearance-only SLAM at large scale with FAB-MAP 2.0. *International Journal of Robotics Research (IJRR)*, 30(9):1100–1123, 2011.

[3] E. Eade and T. Drummond. Unified loop closing and recovery for real time monocular SLAM. In *British Machine Vision Conference (BMVC)*, 2008.

[4] M. Feuerstein, T. Mussack, S. M. Heining, and N. Navab. Intraoperative Laparoscope Augmentation for Port Placement and Resection Planning in Minimally Invasive Liver Resection. *IEEE Tans. on Medical Imaging (TMI)*, 27(3):355–369, 2008.

[5] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[6] A. P. Gee and W. Mayol-Cuevas. 6D Relocalisation for RGBD Cameras Using Synthetic View Regression. In *British Machine Vision Conference (BMVC)*, 2012.

[7] N. Gelfand, N. J. Mitra, L. J. Guibas, and H. Pottmann. Robust Global Registration. In *Eurographics Symp. on Geometry Processing*, 2005.

[8] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. *International Journal of Robotics Research (IJRR)*, 31(5):647–663, 2012.

[9] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *ACM Symposium on User Interface Software and Technology (UIST)*, 2011.

[10] G. Klein and D. Murray. Improving the agility of keyframe-based SLAM. In *European Conference on Computer Vision (ECCV)*, 2008.

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

[12] S. Lieberknecht, A. Huber, S. Ilic, and S. Benhimane. RGB-D camera-based parallel tracking and meshing. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[13] R. Newcombe, A. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[14] R. Newcombe, S. Lovegrove, and A. Davison. DTAM: Dense tracking and mapping in real-time. In *International Conference on Computer Vision (ICCV)*, 2011.

[15] K. Ni, A. Kannan, A. Criminisi, and J. Winn. Epitomic location recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[16] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[17] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE Tans. on Pattern Analysis and Machine Intelligence (PAMI)*, 32(3):448–461, 2010.

[18] G. Reitmayr and T. W. Drummond. Going out: robust model-based tracking for outdoor augmented reality. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2006.

[19] E. Rosten, R. Porter, and T. Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Tans. on Pattern Analysis and Machine Intelligence (PAMI)*, 32(1):105–119, 2010.

[20] H. Roth and M. Vona. Moving Volume KinectFusion. In *British Machine Vision Conference (BMVC)*, 2012.

[21] B. Schwald and B. De Laval. An augmented reality system for training and assistance to maintenance in the industrial context. *Journal of WSCG*, 11(1), 2003.

[22] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon. Scene coordinate regression forests for camera relocalization in RGB-D images. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[23] F. Steinbrucker, J. Sturm, and D. Cremers. Real-time visual odometry from dense RGB-D images. In *ICCV Workshop on Live Dense Reconstruction with Moving Cameras*, 2011.

[24] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald. Robust Tracking for Real-Time Dense RGB-D Mapping with Kintinuous. Technical Report 031, MIT-CSAIL, 2012.

[25] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially Extended KinectFusion. Technical Report 020, MIT-CSAIL, 2012.

[26] B. Williams, G. Klein, and I. Reid. Automatic Relocalization and Loop Closing for Real-Time Monocular SLAM. *IEEE Tans. on Pattern Analysis and Machine Intelligence (PAMI)*, 33(9):1699–1712, 2011.

[27] S. A. Winder and M. Brown. Learning local image descriptors. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.