RGB-D Camera-Based Parallel Tracking and Meshing

Sebastian Lieberknecht[*] metaio GmbH
Andrea Huber[†] metaio GmbH
Slobodan Ilic[‡] TUM
Selim Benhimane[§] metaio GmbH

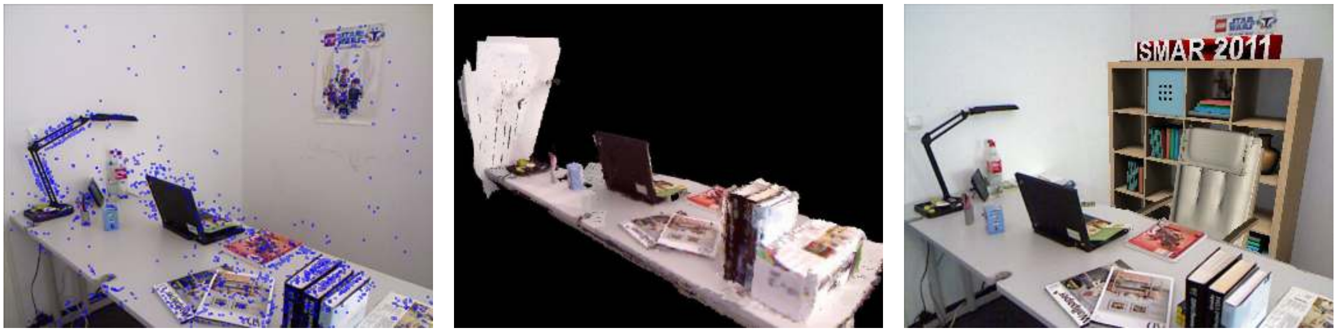


Figure 1: A real-time tracking method based on a sparse 3D map is estimating in real-time a consumer RGB-D camera's motion with respect to an unknown environment (left). At the same time, the method is reconstructing the environment as a dense textured mesh (center). The parallel tracking and meshing opens the way to unprecedented possibilities for several AR applications as the camera motion is accurately estimated and the augmentations are seamlessly integrated in the surrounding environment thanks to convincing occlusions (right).

Abstract

Compared to standard color cameras, RGB-D cameras are designed to additionally provide the depth of imaged pixels which in turn results in a dense colored 3D point cloud representing the environment from a certain viewpoint. We present a real-time tracking method that performs motion estimation of a consumer RGB-D camera with respect to an unknown environment while at the same time reconstructing this environment as a dense textured mesh.

Unlike parallel tracking and mapping performed with a standard color or grey scale camera, tracking with an RGB-D camera allows a correctly scaled camera motion estimation. Therefore, there is no need for measuring the environment by any additional tool or equipping the environment by placing objects in it with known sizes. The tracking can be directly started and does not require any preliminary known and/or constrained camera motion.

The colored point clouds obtained from every RGB-D image are used to create textured meshes representing the environment from a certain camera view and the real-time estimated camera motion is used to correctly align these meshes over time in order to combine them into a dense reconstruction of the environment.

We quantitatively evaluated the proposed method using real image sequences of a challenging scenario and their corresponding ground truth motion obtained with a mechanical measurement arm. We also compared it to a commonly used state-of-the-art method where only the color information is used. We show the superiority of the proposed tracking in terms of accuracy, robustness and usability. We also demonstrate its usage in several Augmented Reality scenarios where the tracking allows a reliable camera motion estimation and the meshing increases the realism of the augmentations by correctly handling their occlusions.

1 Introduction

For several vision-based Augmented Reality (AR) applications, determining the relative motion of the camera with respect to an unknown environment with end-user hardware was made possible thanks to approaches inspired from Davison's MonoSLAM [7]. This approach and its successors are performing real-time tracking of visual features extracted from the captured images. The features need to be seen in many images for which the camera has performed a motion that is sufficient enough for estimating the depth and consequently reconstructing the 3D coordinates of the features. This is generally based on the structure-from-motion principle. In order to get correctly scaled 3D coordinates of the reconstructed points and therefore a correctly scaled camera motion, these approaches usually require an explicit manual measurement of some parts of the environment or equipping it with known objects. Another possiblity to induce scale is to ask the user to perform a constrained camera motion – often the camera needs to move between two known frames such that its optical center position varies with a metrically known scaled translation.

We see here some limitations of the existing approaches. First, before reconstructing a point and adding it to the map, the point needs to be tracked over multiple frames that have an estimated camera pose. This delays the participation of a newly visible physical point in the estimation of the full camera motion. Second, either the environment needs to be partially measured or pre-equipped or the user needs to have some experience with the system in order to correctly perform constrained camera motion that allows a correct scale estimation. Third, since the existing approaches are mainly based on visual features (often extracted where some texture gradient is available), the online map that is obtained from the existing approaches is generally sparse and could not be used, even after post-processing and meshing, for occlusion handling or similar AR tasks that require a meshed version of the environment.

Camera systems that are designed to additionally provide a correctly scaled depth of an imaged pixel would solve the above problems. However, for several years, typical depth camera systems had low resolution, noisy measurements, restricted working area and/or high cost. Whether they are based on Time-Of-Flight technology

[*] e-mail:sebastian.lieberknecht@metaio.com
[†] e-mail:andrea.huber@metaio.com
[‡] e-mail:slobodan.ilic@cs.tum.edu
[§] e-mail:selim.benhimane@metaio.com

or on standard Digital Fringe Projection, these camera systems did not have the huge impact that the advent of the Microsoft Kinect has had in last couple of months. In fact, this end-user low cost and relatively high resolution RGB-D camera is based on RGB camera registered to a stereo system composed of an infra-red structured light projector combined with an infra-red camera which allows for pixel depth computation. Despite the relatively high 640×480 resolution of the produced depth maps, the Kinect does not provide depth for each pixel of the color image; furthermore there is both regular sensor noise and especially quantization noise present which significantly increases with distance. Originally targeted for indoor use and intended for gaming devices, the Microsoft Kinect got an even higher interest from the research community once PrimeSense, the creator of its reference design, released official drivers for this device.

1.1 Contributions

In this paper, we investigate how such device allows a huge step forward in the AR field. In fact, we propose a method based on a consumer RGB-D camera that estimates the camera motion with respect to an unknown environment and that builds in the same time a dense meshed and textured version of the surrounding environment.

The tracking output is a correctly scaled camera motion with no need for measuring the environment or equipping it with known size objects and with no need for preliminary known and/or constrained camera motion. Thanks to the RGB-D camera, the sparse map used for the tracking is composed of back-projected feature points that are guaranteed to be on physical surfaces on which the structured light is projected.

When meshing the environment, for every captured camera frame, we mesh only the colored 3D points that are imaged for the first time and then align that mesh with respect to a common coordinate system using the estimated camera motion. In order to determine whether a 3D point has already been used for the meshing or not, we render in the background the complete meshed version of the environment and create masks based on the depth buffer which tells whether a currently observed 3D point is already represented in the meshed or should be potentially integrated.

For validating the tracking approach, we recorded a set of sequences with an RGB-D camera attached on the end-effector of a high precision mechanical measurement arm. We compared the proposed approach with the commonly used state-of-the-art method PTAM [13] where only the color information is used. The proposed parallel tracking and meshing system provides the camera motions for more images, is generally more accurate and precise, requires less input data and is correctly scaled. Thanks to the tracking performance and the high density of the meshed map, we are able to use it in several AR scenarios where the tracking allows a reliable camera motion estimation and the meshing increases the realism of the augmentations by correctly handling their occlusions.

The paper is structured as follows. We will first position our contribution with respect to the existing and related state-of-the-art. Then, we will describe the approach used for the real-time RGB-D camera motion estimation. We will later explain the meshing of the surrounding environment. For comparing the proposed approach with PTAM, we will first explain how the ground truth data used for the benchmarking was generated, then we will present the results of the evaluation. We will later illustrate the usage of the proposed method on different AR scenarios and conclude with summarizing the different contributions and possible future work.

2 Related Work

Visual real-time tracking with respect to known or unknown scenes is essential and an incontrovertible component of vision-based AR applications. There were numerous algorithmic contributions in the topics in the last few years. But, if Davison *et al.*'s seminal MonoSLAM [6, 7] showed that it is possible to perform Simultaneous Localization and Mapping (SLAM) using a single camera on end-user hardware in real-time, Klein and Murray with PTAM [13] showed that adapting and updating the algorithms used for estimating the camera motion in AR according to end-user available computational capabilities allows to get impressive tracking results in small AR workspaces. In fact, estimating the camera motion by tracking the environment and in parallel building a feature-based sparse map was made possible thanks to the generalization of multi-core processors on desktop computers and laptops.

Many extensions of the above approaches like [8, 9] or [4] showed that it is possible to scale the SLAM approaches to a larger environment e.g. by handling multiple local maps. Recently, Newcombe and Davison [16] showed that with a higher computational power where a single standard hand-held video camera is attached to a powerful PC and with the usage of the computational power of the Graphics Processing Unit (GPU), it is possible to get a dense representation of a desktop scale environment and highly textured scene while performing the tracking using PTAM. The density of the online created map was increased with stereo-dense matching and GPU-based implementations. Sánchez *et al.* [17] showed that the GPU could also be used for effectively replacing the (global) bundle adjustment component of optimization-based SLAM systems such as PTAM by an inherently parallelizable refinement step based on Monte Carlo simulations, thus freeing resources on the CPU for other tasks.

Also, recently, Castaneda *et al.* [3] replaced the generally used standard hand-held video camera with a combination of a Time of Flight (204×204) resolution camera and a (640×480) RGB camera and modified the measurement model and the innovation formulas of the Extended Kalman filter used by MonoSLAM to improve the tracking results. This work did not use a powerful PC but instead a typical expensive depth camera system.

The very recent release of the Xbox 360 Kinect as Microsoft's end-user device had a big impact in the consumer gaming hardware. It is a low cost and relatively high resolution RGB-D camera consisting of a stereo system composed of an infra-red structured light projector combined with an infra-red camera allowing pixel depth computation and to which an RGB camera is registered. This device has directly been used by [12] for surfel-based modeling of indoor environments. The system proposed by [12] does not run in real-time and works on recorded videos, it does not perform any real-time or inter-frame tracking.

In this paper, we propose to use such RGB-D camera in order to perform parallel tracking and meshing of the environment in real-time. The tracking method performs motion estimation of the camera with respect to an unknown environment and at the same time we reconstruct this environment as dense textured mesh online. We demonstrate the usage of this system in several challenging AR scenarios and applications.

3 Real-time Camera Motion Estimation

A typical SLAM system consists of three major parts: Map building, Inter-frame Tracking and Relocalization. The proposed system uses a map for tracking that consists of RGB-D keyframes with associated camera poses and sparse 3D points with descriptors originating from the keyframes. The keyframes are selected images which are captured at camera poses that are far from each other in terms of rotation and translation of the camera.

In the following, we will describe every stage and how they take the depth image into account.

3.1 Inter-frame tracking

The camera motion estimation is realized with sparse optical flow on projections of reconstructed points from the map from the last to

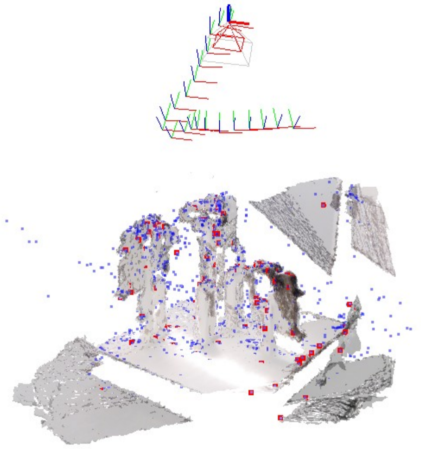


Figure 2: The sparse feature point cloud created online during the real-time tracking of a challenging industrial scene that has multiple self-occlusions and is mainly composed of metallic, reflective and poorly textured surfaces. The features from the closest keyframe are shown in red. The coordinate systems correspond to the keyframe positions. The poses of the current camera and of the closest keyframe are highlighted. The dense reconstructed mesh which is also created online during the real-time tracking is overlaid in order to better understand the localization of the features.

the current camera image. The updated 2D positions are then used to estimate the pose from 2D–3D correspondences.

In particular, we use the pyramidal implementation of Lucas-Kanade optical flow algorithm [15, 2]. Using the optical flow from the last to the current image is very robust against lighting changes and sudden motions, however prone to drift as small inaccuracies accumulate. To mitigate drift, we determine the closest keyframe in terms of rotation and translation of the camera and reproject features of the map into the current image. As the pose may already incorporate drift, we update the features' positions individually again with Lucas-Kanade.

The neighborhood of the updated position is checked for photo-consistency using the Sum-of-Squared Differences of a window around the 2D position of the features. However, we assume that the tracker did not drift arbitrarily, thus we enforce the displacement of the 2D feature to be less than a given threshold. This threshold is used to avoid arbitrary movements of features on near-uniform surfaces.

The pose is computed using non-linear robust pose estimation, parameterized with exponential maps. The pose of the last camera frame serves as initial guess for the estimation when available. To limit the influence of outliers, we use a non-linear robust pose estimation based on Tukey M-Estimator [18]. The features which were classified as outliers by the robust pose estimation (final weight equal to zero) and the features with high reprojecton error are discarded from further tracking.

The Lucas-Kanade tracking is the main computational task of the tracking component. Due to real-time constraints, we limit the number of the tracked features to around 300, and all lost features are replaced by reprojected features from the closest keyframe in order to maintain the maximum number of tracked features.

3.2 Sparse feature mapping

The system starts with an empty map. When adding keyframes, we run a scale-invariant feature extractor and descriptor based on the principle of the method described in [1] on the captured camera images and make sure that the keyframes contain a sufficient amount of features (we used 40 in the evaluation) for which there are also readings from the depth sensor. The 3D points are projected using the camera intrinsic parameters and the measured depth value. We chose to define the global coordinate system at the camera pose when the first keyframe was captured.

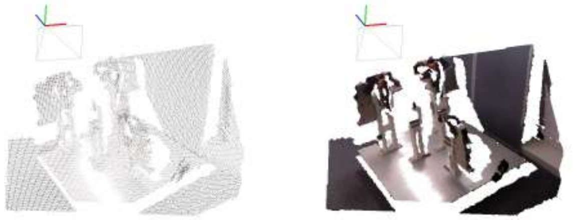


Figure 3: Input 3D point cloud captured from one camera viewpoint (left) and its corresponding textured mesh (right)

When the first keyframe is taken, the inter-frame tracking is started and the pose is computed using the 2D–3D correspondences. At every new frame, the 2D position of the features is updated using the optical flow tracking algorithm. In case the current translation or the current rotation is far enough from all existing keyframes, the current frame is considered as a keyframe candidate. For every keyframe candidate, we extract feature points and match their descriptors against the map. The matches are validated using the 3D distance of their corresponding 3D coordinates transformed into the common coordinate system using the inter-frame tracking estimated pose. Features which do not survive these tests are discarded from the keyframe as they would lead to increased mismatching.

To reduce the drift of the tracking, we estimate the keyframe poses using the matching based on the descriptors. The map grows by adding new (non-matched) features extracted from new keyframes. The 2D–3D correspondences of the matched features are stored to be able to refine the 3D point position of the mapped feature at a later point in time, e.g. by Kalman Filtering. We currently do not filter the depth maps from the Kinect but instead we rely on a pose estimation based on M-Estimators [18] which works well in the general case where we have a couple of hundreds correspondences.

3.3 Relocalization

In case the tracking is lost, the relocalization is done by extracting a given maximal number of features from the current camera image and matching them against the map. Severe mismatches are removed via RANSAC [11] on the reprojection error of the points. The initial pose is then refined using the non-linear robust pose estimation as the one used for the inter-frame tracking. If the pose estimation is successful, the features of the closest keyframe are projected into the current camera image and added to the correctly matched features (as detailed above) and the inter-frame tracking is restarted.

4 Online Environment Mesh Creation

In this section, we explain the approach we use to reconstruct the environment as a textured dense mesh. The reconstruction process is done in parallel to the tracking and consists of different tasks: the selection of the RGB-D frames for updating the meshed model, the meshing of the selected point cloud set corresponding to the RGB-D frame and the alignment of the newly created mesh. In general, we favoured integration speed of new data over global accuracy as the system should be used online with no special hardware besides the RGB-D sensor.

4.1 From RGB-D values to 3D colored point cloud

In comparison to a standard color camera, an RGB-D camera can additionally provide the depth value of pixels. Once the RGB-D camera is intrinsically calibrated, we create 3D point clouds using the depth image as follows. For every a homogeneous 2D image point $\mathbf{p}_i$ with a measured depth value z_i, we build the corresponding inhomogeneous 3D point $\mathbf{x}_i$ as:

$$\mathbf{x}_i = z_i \mathbf{K}^{-1} \mathbf{p}_i$$

where $\mathbf{K}$ is the (3×3) upper triangular matrix of the camera intrinsic parameters. Given the registered RGB camera image, we associate a color to every 3D point as shown in Fig. 3. We eventually filter the 3D points such that we only keep points that are not farther than a certain distance from the depth sensor (we used two meter for the Microsoft Kinect). This helps to improve the quality of the meshed point cloud, as the uncertainty of the depth measurements from the Kinect increases significantly with the depth.

4.2 Creating local 3D textured meshes

Creating a model from range data in the simplest case consists in the registration of each new depth map to a model and concatenation of them. However, essentially keeping all data quickly occupies large quantities of memory while there may be only limited new information at all. A proper integration into a global model is beneficial, and in general, there are two approaches which are used: Volumetric or implicit modelling on the one hand [5] and surface-based or explicit modelling on the other hand [19]. While the former can also deal with more complex shapes, it in general needs an extra step to generate an explicit representation (e.g. used for visualization), is usually tied to a pre-specified volume and computationally more expensive than explicit modelling.

As the proposed system should run at least near real-time on standard hardware, the meshing is done using the method of Turk and Levoy [19] with the adjustment of using a fixed threshold for the maximal edge length per vertex-pair instead of using a flexible treshold as proposed by the authors since we work on a fixed range volume. Turk and Levoy convert the depth map into a mesh by moving a window over it and analyzing the 3D distances of the vertices corresponding to the corners of the window. The vertices are linked if their distance is below a given threshold; in order to preserve details, the smaller diagonal is preferred to form triangles. In case of missing data or a depth value bigger than our aforementioned empirical threshold of 2 m, we do not create triangles containing this vertex. The texturing of the meshes is done by associating the color of the corresponding pixel in the RGB image to every vertex. Despite technically not fully correct, we use 'texturing' as synonym for 'vertex-coloring' througout the paper instead of the usually associated 'texture mapping'. Figure 3 shows the meshing and texturing results of a 3D point cloud of an object captured from a certain camera viewpoint.

4.3 Aligning the meshes in a global coordinate system

The meshing of the point cloud described above consists of defining triangles with the 3D colored point cloud. As we currently join meshes by concatenating them, the actual joining consists of simply transforming the 3D points with the inverse of the camera pose associated to the RGB-D frame. In Figure 4, we show the alignment of two local meshes that results into a combined mesh in a global coordinate system.

When performing the real-time tracking, the camera has typically small inter-frame movements. This means that there is a large overlap between two meshes created of two consecutive frames. For example, one can see that there are some overlapping regions between the two meshes of Figure 4. The overlap needs to be taken into account since otherwise, i.e. when every arriving RGB-D image would be integrated into the global mesh, the capacity of the main memory would be exceeded after a couple of minutes because of the potentially massively redundant data.

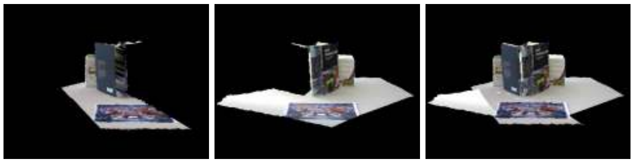


Figure 4: Two local meshes (left and middle) aligned in a global coordinate system (right)

We initially tried to only consider frames with associated camera poses that have a translation and/or rotation further than a given threshold from the already meshed keyframes. This works well in many cases. But, choosing the threshold needs a compromise: if too high, it makes it harder to mesh some unmeshed parts. And if too low, the overlap between the regions becomes too large.

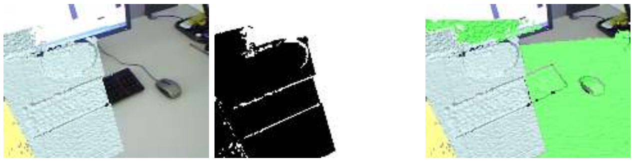
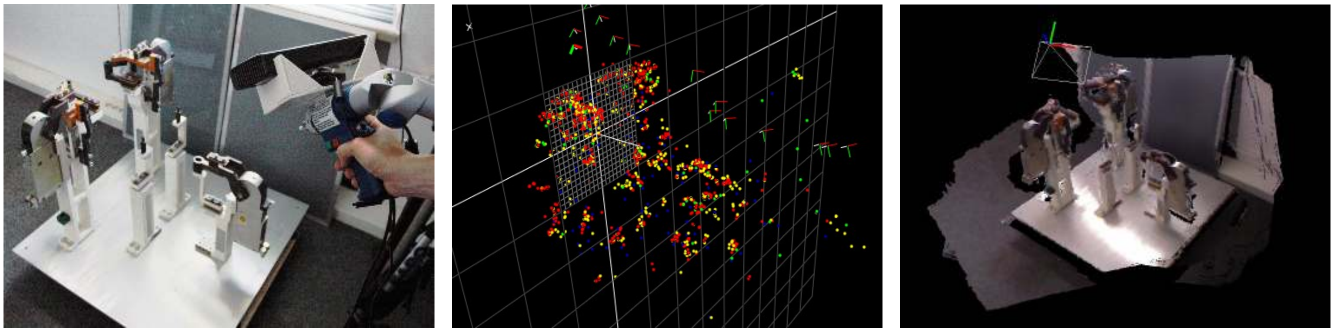


Figure 5: Determining the contribution of a potential new keyframe. The reconstrution is rendered from the current camera viewpoint (left), creating a binary mask (center) set at rendered pixels. New geometry from the potential keyframe is only added to the reconstruction when there either was no geometry at all (right) or in case the rendered geometry is further than a specified distance away.

Instead of only checking the position and the viewing angle of the camera, our approach makes sure that only "new" points are used for the meshing. Therefore, we additionally filter the 3D points before meshing. As illustrated in Figure 5, a binary mask and depth buffer is created by rendering the reconstructed scene from the current camera view point. Unmasked 3D points are directly considered for updating the mesh, i.e. 3D points for which no geometry was rendered at their reprojected 2D position. Only relying on the binary mask would prohibit adding meshes of objects that are first observed in front of already reconstructed geometry, which e.g. happens when moving camera around a fixed object in an attempt to scan its geometry.

To also add close objects, we additionally check for masked pixels whether the depth stored in the rendered depth buffer is greater than the value of the depth map by at least some threshold. Finally, in order to close small gaps which especially occur on the boundaries of registered depth maps, as can also be observed by close inspection of Figure 5, we additionally erode the binary mask such that new geometry may also be added on the boundaries despite already existing geometry closer than the threshold.

Even though we avoid the creation of (massively) redundant meshes, still the decision of which frame should be integrated into the global reconstruction is of importance. One option is to process the frames continuously. The advantage of this method would be that the 3D model is created fluently and fast since the updates are small. Another possibility is, as discussed earlier, to decide on the basis of the current camera pose if a frame should be processed or not. This possiblity is very fast to evaluate. Since the meshing is done in a separate thread, we finally chose to process every incoming camera image, but only add those local meshes that contributed at least a certain number of triangles to the global mesh as this visually provided the best result.



(a) Hardware setup used for evaluation.

(b) Online reconstruction by PTAM [13]

(c) Online reconstruction of proposed method

Figure 6: Ground Truth data generation setup and and reconstructions of the environment from similar viewpoints using PTAM [13] based on RGB images and the proposed method based on RGB-D images.

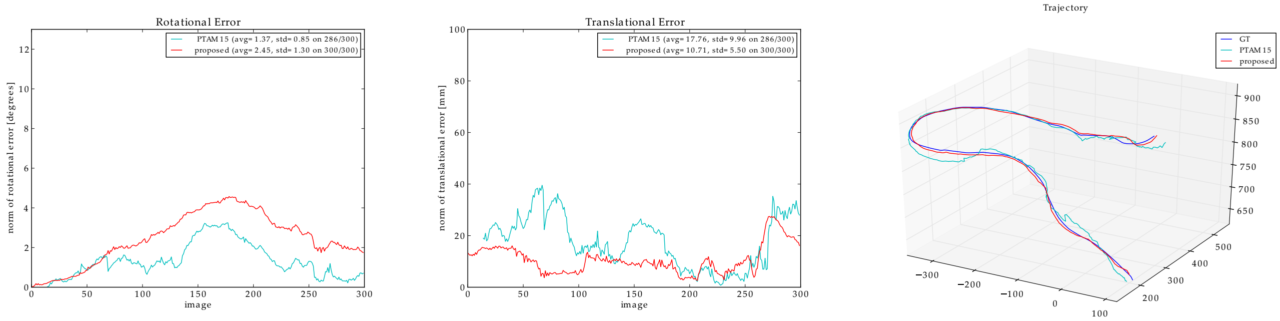


Figure 7: The proposed method and PTAM are evaluated against the mechanical ground truth (GT), the results of the first sequence are shown by error plots of the rotation (left) and translation (center) as well as a trajectory of the camera centers (right). PTAM is given frame 0+15 for initialization which provided the best results for this sequence. Despite a good rotation estimation from PTAM, the proposed method estimated the translation with significantly higher accuracy and thus estimates the camera trajectory much closer to the GT trajectory.

5 Ground Truth-based Evaluation of the Tracking

We evaluated the accuracy and precision of the camera motion estimation and the inherent scale of the proposed method using real data. This was done using the ground truth motion of a high precision mechanical measurement arm similar to [14]. We recorded four sequences of a challenging industrial object, shown in Figure 6. For comparison, we also evaluated PTAM [13] on the same sequences. In the following, first the generation of the ground truth sequences is presented, followed by the quantitative and qualitative evaluation.

5.1 Creation of the Ground Truth data

When using synthetic data, besides the unavoidable planning of the general scene and motion of the (virtual) camera, decisions have to be taken on how to degrade the virtual data to make it resemble more the data which will inevitably be used as input of the system at hand. However, modeling the imperfections of real-world cameras is a task on its own, especially when it comes to modelling the sensor noise or the camera colors or lens.

In order to side-step this issue, we opted for creating ground truth (GT) sequences using images from a real sensor. For this, we used a mechanical measurement arm from Faro[10] which has seven axes and provides the pose of the tip of its end effector relative to the base with an accuracy better than 0.013 mm within its working volume of 1.2 m from its base. Onto the end effector, we rigidly attached the Microsoft Kinect as can be seen in Figure 6(a). We pointed the mounted camera towards an industrial object and moved it around the object. To avoid frame dropping, we preallocated memory for both the VGA depth and color images and later saved them uncompressed on the hard-drive. The poses from the measurement arm were available at 250 Hz and also buffered in the main memory before saving them on the hard-drive at once.

For capturing the images, we used the official drivers and the OpenNI framework from PrimeSense. The depth and color images are captured using a rolling shutter, they are updated at 30 Hz independently, without hardware synchronization. We chose to push new color and depth images into the system whenever both images were updated, we assign a timestamp at the time when we have access to both image buffers. The poses of the measurement arm are also timestamped as soon as we are able to read them.

The last task remaining is to synchronize the poses of the FaroArm to the images. In contrast to our previous work [14], we chose not to alter the environment by introducing accurately positioned fiducials for this task. Instead, we directly align the estimated camera trajectories of the evaluated method to the trajectory of the measurement arm. We obtain an initial estimation of the scale and Euclidean transformation by assuming accurate timestamps and creating 3D-3D correspondences of the camera centers to the tips of the end effector using the method of Umeyama [20]. The poses of the measurement arm were interpolated on the $\mathbb{SE}(3)$ manifold to match the timestamps of the images.

Umeyama's method was designed to align point clouds, it is optimal when the assumption holds that the correspondences contain exclusively errors belonging to a normal distribution – but as we are using it to align trajectories where one should be evaluated to the other, and additionally we also have to still synchronize the measurements, we use the result only as first step in the alignment process.

In a second step, we take both the possibility of outliers better into account and additionally also search for the offset of the timestamps. We use a Nelder-Mead simplex to minimize the distance of corresponding 3D points, parameterized by the 6-DOF Euclidean transformation (using exponential maps for the rotation), the scale of the trajectory and the offset of the timestamps. Noise and outliers are handled by re-weighting the error with the Tukey M-Estimator function.

Thus, the GT can be generated for every sequence individually. We observed that the offset of the timestamp was for all sequences in the order of ±50 microseconds after alignment and thus seemed to have no substantial influence in the optimization. Therefore it can be justified that we used the GT sequences aligned to one test run also to evaluate other test runs of the same sequences in order to be able to directly compare the results.

5.2 Quantitative evaluation of the tracking

We then used the GT sequences to evaluate the accuracy and precision of the estimated camera motion of the proposed method and compared it with results from PTAM [13]. Our method is always initialized on the first frame of the sequence and is able to track the full sequences. To initialize PTAM it is necessary to carefully move the camera a certain distance to establish an initial stereo configuration. The baseline of these frames affect the scale of the map that PTAM builds (and the scale of the trajectory that PTAM estimate). When evaluating PTAM, for all sequences, we used the first frame and varied the second image of the initial stereo setup from frame 1 to frame 50. For some image pairs, the initialization of PTAM did not succeed.

In contrast, thanks to the usage of metric depth maps, the proposed method estimated an identical scene scale for all four sequences as can be seen in Figure 8. As it needs only a single frame for the initialization, we tested how choosing this frame from the first 50 frames of each sequence would affect the scale factor needed for metric alignment of the trajectory. It turned out that the scale factor value is relatively stable around 1 and with a low variance.

Table 9 presents the results of the evaluation of the estimated camera pose all sequences for the proposed method as well as for several PTAM initializations. The evaluation shows that PTAM's accuracy and precision depends on which image pairs are used for the initialization. There was no clear rule which frame the user should use in order to always get the best rotation and translation estimation. With some image pairs, it was even not able to initialize despite a large baseline between the frames. This is one of the disadvantage of the method. We show in Figure 7 the detailed results for the first sequence. Using the proposed method initialized on frame 0 and the best result that PTAM could achieve on this sequence (using frame 0 and frame 15), despite a good rotation estimation from PTAM, we are still getting much better translation estimation and our estimation of the camera trajectory with the proposed method is much closer to the GT trajectory.

5.3 Quantitative evaluation of the reconstruction

As initially stated, we do not post-process the range images obtained from the Kinect other than neglecting samples further than 2 m and currently enlarge the reconstruction by adding new triangles to the mesh which otherwise stays unchanged. We evaluated the quality of the reconstruction process by comparing the mesh of the industrial object to known ground truth geometry.

The reconstruction was obtained from the first sequence, using every 4th depthmap pixel in each direction and a maximal allowed edge length of 50 mm. An update to the model was done only when the new depth image provided more than 2000 new triangles, the final model of the object contained around 14k vertices and 19k faces after removing the background. The recovered model was aligned manually to the known geometry, then we computed the discrepancy based on the point-to-plane/face distances of the reconstructed vertices to the known faces, visualized in figure 10. The median error was 9 mm, lower and upper quartiles 3 mm and 17 mm respectively. In the absence of any filtering mechanisms these should be regarded as upper bounds for Kinect-based reconstruction, although the level of error is already acceptable in typical AR maintenance scenarios for mid-sized objects as the one used in the experiments.

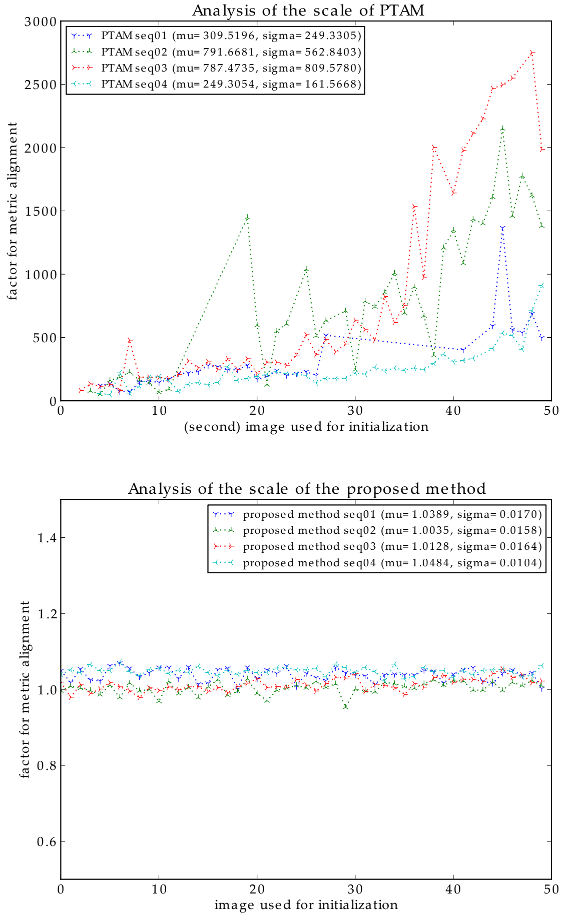


Figure 8: Comparison of the scale needed for robustly aligning the trajectories to the metric GT. Note that the scale of PTAM varies strongly depending on the image pair chosen for initialization (we used frame 0 and a variable second frame (x-axis of the plot)). We do not display the image pairs for which the initialization of PTAM was not successful. In contrast, our method is able to initialize from every frame successfully and the estimated scale is stable around 1 for all tested sequences independently of the image used for the initialization.

6 Application on Different AR Scenarios

We demonstrate the proposed method on different AR scenes and scenarios. It has been tested on scenes similar to the one illustrated in the supplementary material and in Figure 11. The initial map of the scene is built from the first image and extended afterwards. The tracking handles moderate and fast camera motion. The runtime of the tracking part of the system highly depends on the convergence of inter-frame feature tracking both from the last camera frame as well as from the closest keyframe to the current camera frame. On the desktop computer used for testing (Intel i7

		rotation [deg]		translation [mm]		images tracked
		μ	σ	μ	σ	
seq01	proposed	2.45	1.30	**10.71**	**5.50**	**300/300**
	PTAM 05	7.04	3.72	94.55	42.35	282/300
	PTAM 10	2.53	1.25	15.95	8.73	291/300
	PTAM 15	**1.37**	**0.85**	17.77	9.97	286/300
	PTAM 20	3.75	2.01	28.36	17.18	281/300
	PTAM 25	1.85	1.02	26.69	11.44	276/300
seq02	proposed	4.03	2.00	**6.85**	**2.82**	**300/300**
	PTAM 05	15.53	10.50	44.02	46.31	120/300
	PTAM 10	3.05	2.56	39.62	41.50	175/300
	PTAM 15	–	–	–	–	0/300
	PTAM 20	**1.88**	**0.83**	11.18	6.58	281/300
	PTAM 25	13.78	5.06	162.95	78.85	243/300
seq03	proposed	**5.76**	3.36	21.49	8.93	**300/300**
	PTAM 05	7.17	5.28	38.60	15.99	58/300
	PTAM 10	19.36	12.09	18.69	**8.05**	69/300
	PTAM 15	6.59	**3.24**	**17.31**	8.11	225/300
	PTAM 20	6.86	3.33	18.08	9.52	212/300
	PTAM 25	7.95	3.84	18.77	8.45	276/300
seq04	proposed	2.62	1.47	**13.18**	**6.37**	**300/300**
	PTAM 05	–	–	–	–	0/300
	PTAM 10	13.62	9.43	97.83	58.16	142/300
	PTAM 15	**2.05**	1.19	35.50	31.09	162/300
	PTAM 20	2.45	1.22	17.36	13.55	150/300
	PTAM 25	2.43	**1.02**	18.81	12.73	276/300

Figure 9: Mean and variance of the error in rotation and translation of the proposed method and PTAM with different initializations. We used frames 0+5 for PTAM 05, 0+10 for PTAM 10 *etc*. The best results per sequence are highlighted in bold.

2.8 GHz CPU), the inter-frame feature tracking takes around 23 ms on average for 200 features. During fast motion, many features can be lost due to motion blur and the time per frame may reach 50-70 ms. The relocalization consists of matching the feature descriptors extracted from the current image to those of the map. This is currently done exhaustively, i.e. the runtime scales with the product of the number of features in the map and camera image. For the sequence, the relocalization together with reprojection from the closest keyframe takes between 25 ms and 230 ms for a map with around 15000 mapped features from 63 keyframes.

6.1 AR-based virtual furniture trial

This scenario is meant to help the typical user who needs to virtually try a new furniture (e.g. a closet) in the room before buying it. The user would not only check the color and the model of the furniture but also its size. This requires a correctly scaled camera pose estimation. Thanks to the proposed approach, the furniture can now be placed at the desired position with the correct scale, without modifying the environment. Furthermore, due to the reconstruction of the environment the user gets a more realistic impression of the possible future look. Figure 12 shows a correctly scaled shelf and chair augmented both without and with occlusion from real objects. We can clearly see the advantage of the proposed real-time parallel tracking and meshing of the environment. To further assist the user, one could use the dense reconstruction also to restrict the movement of the virtual furniture such that e.g. it cannot be accidentally pushed ”through” a wall or in case there are moving parts like doors or drawers, it could be automatically checked whether they can be operated using their full designed range of motion.

6.2 Visual discrepancy check

Discrepancy check is of great use in an industrial application like prototyping. It is often required to visually compare a prototype with a produced model. Using AR allows to reduce the costs of construction since there is no need for manual as-is analysis by a construction engineer.

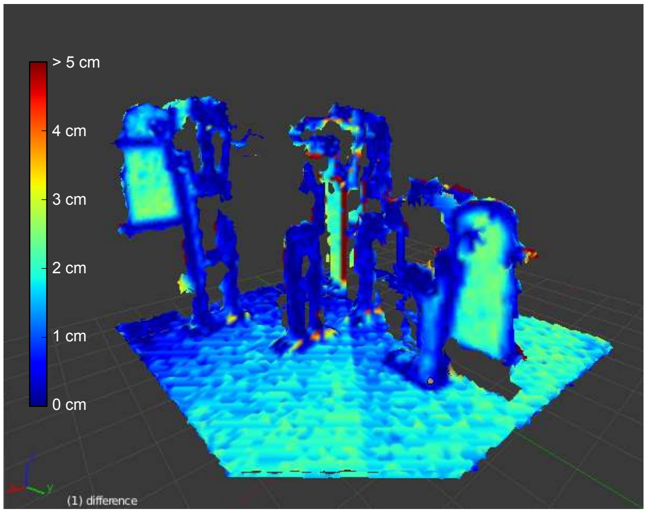


Figure 10: Evaluation of the reconstruction against the known geometry, using the industrial object of figure 6. The median error was 9 mm, 1st and 3rd quartiles were at 3 mm and 17 mm respectively.

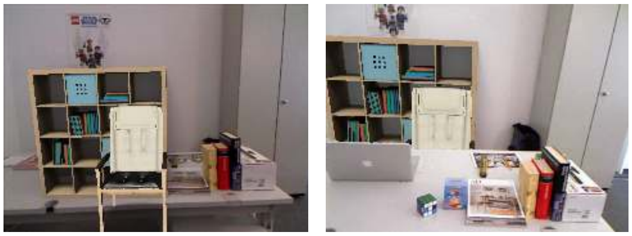


Figure 12: AR-based virtual furniture trial: correctly handling the occlusion thanks to the proposed parallel tracking and meshing of the environment makes the AR visualization much more realistic. While the camera moves, the camera motion is estimated and the environment model is updated (see supplementary material video).

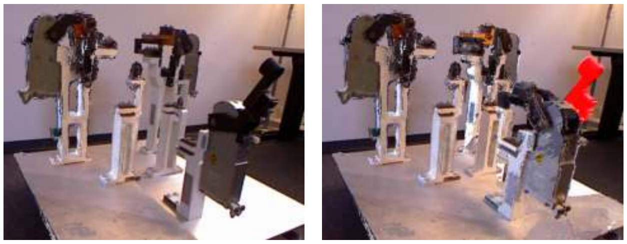


Figure 13: Live meshing allows to quickly create a textured meshed model of an object for visual discrepancy check (see supplementary material video).

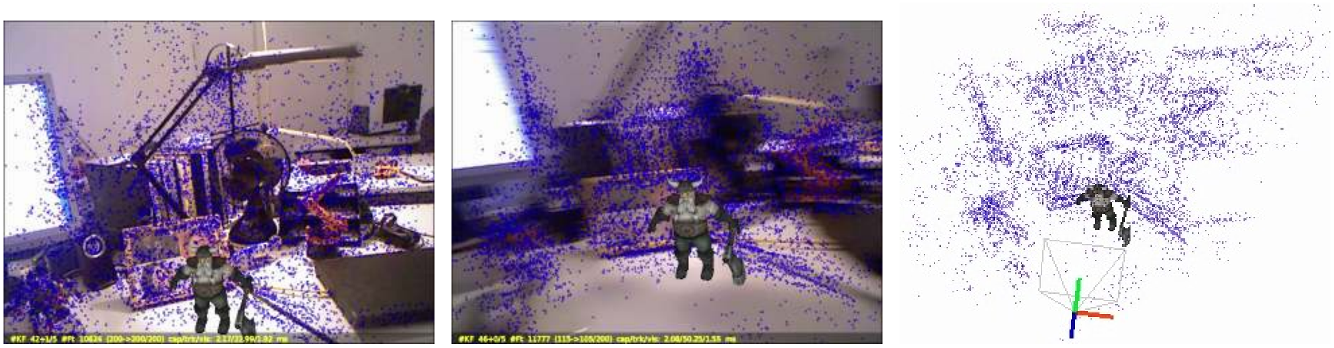

#KF 42+1/5 #Ft 10626 (200->300/200) cap/trk/vis: 2.17/22.99/1.92 ms
#KF 46+0/5 #Ft 11777 (115->105/200) cap/trk/vis: 2.08/50.25/1.55 ms

Figure 11: Evaluation using a live camera stream. *From left:* Tracking a desktop scenario with moderate and fast camera motion. The map reconstructed online consists of 63 keyframes and 14775 reconstructed features.

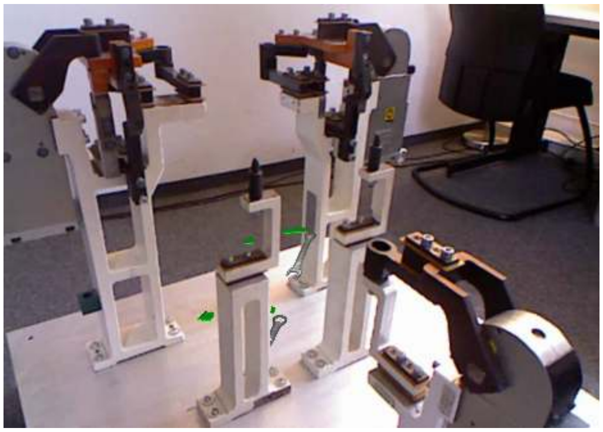


Figure 14: Maintenance instructions can be better understood when their occlusion are handled correctly as done for the lower of the two wrenches (see supplementary material video).

The presented example assumes a high precision of the tracking for which currently a mechanical measurement system like FaroArm is used best. However, for coarser discrepancy checks like e.g. the repositioned part shown in Figure 13, the dense mesh created online by the proposed method is sufficient. Once the current desired geometry is registered to the currently observed state, its potential differences can be easily highlighted (e.g. marked red as in Figure 13). Even simpler and also working in case there is no depth information of the current state of an object, one can use a virtual clipping plane to perform a visual discrepancy check.

6.3 Maintenance scenario

AR maintenance can be used to guide e.g. a technician during a repair or unmount an industrial machine as virtual replacement for the repair manual. For this task, the single steps could be displayed e.g. as illustrated in Figure 14, which shows the specific screws that should be loosened next. The realism of the augmentation can be again improved by using the meshed reconstruction of the environment as occlusion model. As a next step, one could think of combining this scenario with the discrepancy check, e.g. only proceeding to the next step of a repair manual when the correct execution of the current step is validated using the current depth images of the RGB-D camera.

7 Conclusion

We presented a real-time method based on a consumer RGB-D camera that estimates the camera motion with respect to an unknown environment while at the same time reconstructing a dense textured mesh of it. The system is initialized using a single frame, the reconstruction does not need constrained camera motion and especially pure rotational movement of the camera is handled transparently.

The scale of the reconstructed model is fixed and, in case of the specific Microsoft Kinect used in our experiments, seems to be at most 5-6% off from metric scale when using the proposed system for tracking as shown in the evaluation. We adopted the meshing algorithm of Turk and Levoy [19] for live incremental meshing the environment and showed AR scenarios that directly benefit from this mesh. We created ground truth sequences using a high precision mechanical measurement arm and evaluated the proposed RGB-D tracking and PTAM [13]. We obtain in general more accurate and precise results while tracking a higher number of images. The proposed method works already very well and makes it possible to get very satisfactory results despite the efficiency of the computations involved. The results could be further improved thanks to the following suggestions.

As from the current evaluation it is not clear whether the rather small offset from metric scale is coming directly from the device or the usage of the data by proposed method, a part of the future work may consist in an in-depth analysis of only the sensor readings.

Currently, the proposed system determines the 3D position of a feature used for tracking or a vertex used for meshing based on the aligned keyframe that first observed it. There is no refinement done afterwards, i.e. when a wrong measurement enters the mesh/map, it stays and possibly affects all subsequent integrations. In future work, refinement steps of the mesh could be integrated; there is a whole body of literature on this topic, including the early works of Turk and Levoy [19] and Curless and Levoy [5]. RGB-D images can be aligned also using a modified Iterative Closest Point (ICP) algorithm as *e.g.* proposed by Henry *et al.* [12] who also then used these alignments to create a graph-based mapping approach which later can be optimized to further improve the consistency of the map, for instance after a loop closure event. An ICP could also be used to align new keyframes inside the tracking component only, which should further increase the precision of the pose estimation. To keep the integration speed of new data on a high level, the new refinement strategies ideally should be run in the background while new RGB-D images arrive and are tenatively integrated.

In the current system, we do not distinguish between features used for relocalization and features used for tracking. We rely on sparse optical flow for inter-frame tracking and do not use an explicit 3D motion model. The speed of the tracking is primarily depending on the convergence properties of the optical flow, which on the one hand allows more flexible camera motions, but on the other hand is computationally more expensive than template matching with a 3D motion model. As the amount of features tracked seems to also have the biggest influence on the precision of the pose, one direction of future work consists in analyzing how tracking more features based on template matching instead of optical flow could affect the accuracy and speed of the system.

Another very promising direction of future work is centered around the idea to use also the camera's current depth map when not taking keyframes for mapping or meshing. This could potentially mitigate the effects of noise and especially the errors coming from the non-uniform discretization of the depth which occur when using the Microsoft Kinect.

Acknowledgements

This work was supported in part by BMBF grant Avilus / 01 IM08001 P.

References

[1] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. *Computer Vision and Image Understanding*, 110:346–359, 2008.

[2] J.-Y. Bouguet. Pyramidal implementation of the lucas-kanade feature tracker. OpenCV Documentation, 1999.

[3] V. Castaneda, D. Mateus, and N. Navab. SLAM combining ToF and high-resolution cameras. In *IEEE Workshop on Motion and Video Computing*, 2011.

[4] R. Castle, G. Klein, and D. Murray. Video-rate localization in multiple maps for wearable augmented reality. In *IEEE Int. Symp. on Wearable Computers*, 2008.

[5] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, 1996.

[6] A. Davison, W. Mayol, and D. Murray. Real-time localisation and mapping with wearable active vision. In *ISMAR*, 2003.

[7] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *ICCV*, 2003.

[8] E. Eade and T. Drummond. Scalable monocular slam. In *CVPR*, 2006.

[9] E. Eade and T. Drummond. Unified loop closing and recovery for real time monocular slam. In *BMVC*, 2008.

[10] FARO Europe GmbH & Co. KG. http://faro.com/.

[11] M. A. Fischler and R. C. 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.

[12] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D Mapping: Using depth cameras for dense 3d modeling of indoor environments. In *Int. Symposium on Experimental Robotics*, 2010.

[13] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *ISMAR*, 2007.

[14] S. Lieberknecht, S. Benhimane, P. Meier, and N. Navab. Benchmarking template-based tracking algorithms. *International Journal of Virtual Reality, Special Issue on Augmented Reality*, 15:99–108, 2011.

[15] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision (ijcai). In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, 1981.

[16] R. A. Newcombe and A. J. Davison. Live dense reconstruction with a single moving camera. In *CVPR*, 2010.

[17] J. R. Sánchez, H. Álvarez, and D. Borro. Towards real time 3d tracking and reconstruction on a GPU using monte carlo simulations. In *ISMAR*, 2010.

[18] C. V. Stewart. Robust parameter estimation in computer vision. *SIAM Review*, 41(3):513–537, 1999.

[19] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Computer Graphics Proceedings, Annual Conference Series*, 1994.

[20] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *PAMI*, 13:376–380, 1991.