SlamDunk: Affordable Real-Time RGB-D SLAM

Nicola Fioraio and Luigi Di Stefano

CVLab - Dept. of Computer Science and Engineering, University of Bologna
Viale Risorgimento, 2 - 40135 Bologna, Italy
{nicola.fioraio,luigi.distefano}@unibo.it

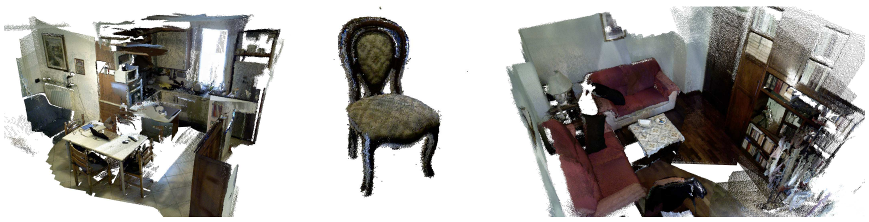

Fig. 1. SlamDunk creates in real-time high quality point clouds in diverse settings, such as *e.g.* object scanning and reconstruction of indoor environments.

Abstract. We propose an effective, real-time solution to the RGB-D SLAM problem dubbed SlamDunk. Our proposal features a multi-view camera tracking approach based on a dynamic local map of the workspace, enables metric loop closure seamlessly and preserves local consistency by means of relative bundle adjustment principles. SlamDunk requires a few threads, low memory consumption and runs at 30Hz on a standard desktop computer without hardware acceleration by a GPGPU card. As such, it renders real-time dense SLAM affordable on commodity hardware. SlamDunk permits highly responsive interactive operation in a variety of workspaces and scenarios, such as scanning small objects or densely reconstructing large-scale environments. We provide quantitative and qualitative experiments in diverse settings to demonstrate the accuracy and robustness of the proposed approach.

Keywords: RGBD SLAM, real time SLAM, relative bundle adjustment, camera tracking

1 Introduction

For many years, the Simultaneous Localization And Mapping (SLAM) problem has been addressed mainly by deploying either expensive and accurate 3D sensors, *e.g.* laser scanners, or monocular RGB cameras. Though impressive results dealing with laser measurements have been reported in literature [3, 17], size, cost and computational issues limit the range of addressable applications significantly. On the other hand, though monocular SLAM has reached a considerable maturity [6, 13], it still mandates dedicated hardware acceleration, *e.g.* by GPGPU processing, to attain dense 3D reconstruction in real-time [19].

Nowadays, consumer-grade RGB-D cameras capable of delivering color and depth in real-time, such as the Microsoft Kinect and Asus Xtion Pro Live, gain increasing interest as low-cost and low-power sensors suited to both robot platforms, *e.g.* in personal/service robotics, as well as user-operated systems. As for SLAM, real-time scanning based on GPGPU processing has been demonstrated by the well-known KinectFusion system [18], while most existing approaches deploying RGB-D cameras without any hardware acceleration run notably slower.

In this work we propose a novel SLAM system for RGB-D cameras designed to be accurate, robust and as efficient as to operate in real-time without any kind of hardware acceleration. As illustrated in Fig. 1, the system can scan small objects as well as reconstruct large scale environments like a room or an entire apartment. The proposed approach creates a local map of the workspace which enables multi-view feature tracking. Scalability and real-time operation are achieved by smart feature selection and by avoiding global optimization through relative bundle adjustment [22]. However, unlike previous proposals [14, 24, 10, 8], we never marginalize point matches so to take advantage of all the available information for pose refinement. The code of our system will be soon made available under an open-source license and, to render usage as easy as possible, endowed with a clean interface to adjust only a few key parameters. We believe that our system will allow any user, equipped with an RGB-D sensor and commodity hardware, to scan and reconstruct in real-time indoor environments both effortlessly and effectively. Hence, we dub it SlamDunk.

The remainder of this paper is organized as follows. Next section reviews related publications and highlights similarities and differences with respect to our proposal. Sec. 3 presents the notation used throughout the paper. The proposed method is described in Sec. 4 and the experimental findings reported in Sec. 5. Finally, in Sec. 6 we outline some concluding remarks and potential research directions.

2 Related Work

Besides GPU-based approaches [18, 26, 5], a few other methods try to solve the SLAM problem for the specific case of RGB-D cameras. Two relevant approaches are RGB-D Mapping, introduced by Henry *et al.* [10], and the very similar RGB-D SLAM system, proposed by Endres *et al.* [8]. In both, visual features are extracted and matched to find correspondences between the current and previous frame (or previous keyframe). Then, 2D feature points are projected into the 3D space thanks to the available depth measurements and camera pose is estimated robustly by RANSAC-based Absolute Orientation [1]. We adopt the visual features approach alike, but within a multi-view matching scheme which provides more robust tracking especially when the previous frame is not informative enough. In RGB-D Mapping [10] camera pose estimation is further refined by ICP-like alignment, which yields improvements mainly in rather dark or texture-less environments. Loop closure is explicitly addressed by separately matching a subset of previous frames (or keyframes) to the current one. Instead, our multi-view feature pool is built dynamically along with camera movement, thereby also handling metric loop closure inherently during camera tracking. Both the above mentioned systems, and ours alike, refine camera trajectory through pose graph relaxation [15]. However, inspired by [9], we consider all the relevant feature matches instead of marginalizing to pose-pose constraints. Finally, RGB-D Mapping as well as RGB-D SLAM achieve real time operation only when using binary features [20] or GPU acceleration [27].

A different research line deals with dense estimation of the visual odometry between two consecutive RGB-D frames, as proposed by Steinbruecker *et al.* [23] and Kerl *et al.* [12]. However, their work is more focused on low-drift pairwise tracking rather than on development of a full-fledged SLAM system. Accordingly, they simply create a new keyframe every time the current frame can no longer be matched to the last keyframe, thus likely wasting memory in presence of a loopy trajectory. Furthermore, detection of a keyframe is followed by a loop closure detection which considers as candidates all camera poses within a certain translation distance. As already mentioned, loop closure is instead handled inherently by our tracking proposal. Finally, [23, 12] perform a pose-pose graph optimization similarly to [10, 8].

More recently, two other real-time RGB-D SLAM systems running on a CPU have been proposed. Scherer and Zell [21] combine the well-known PTAM [13] with the relative bundle adjustment approach [22]. They select the best keyframe for tracking by feature re-projection, then a pairwise alignment based on sparse optical flow and RANSAC-based pose estimation is deployed. Instead, we propose a multi-view tracking engine, so to attain higher accuracy on the same benchmark dataset.

Dryanovski *et al.* [7] have proposed a real-time visual odometry system based on an ICP-like registration between keypoints on the current frame and a sparse 3D landmark map. However, real-time operation is achieved with QVGA image resolution, while in this work we deploy full VGA images. Also, they update landmarks' positions using a Kalman filter, while we adopt local camera trajectory optimization.

It is worth pointing out that one of the main contributions of SlamDunk consists in the use of an active window of keyframes (see Sec. 4.1), which is both novel and key to our multi-view tracking approach. Indeed, other methods based on feature tracking, *e.g.* PTAM [13], cannot handle large workspaces due to the exceedingly large number of features that would be considered as candidates for tracking. On the other hand, proposals conceived to deal with long mapping paths, such as [24, 22], find the active region following the existing connections between keyframes along the camera path; instead, by relying on the current camera position only, we foster the creation of links and, possibly, close short and medium size loops seamlessly.

3 Preliminaries

Throughout the paper we will assume to receive RGB-D frames from a camera, such as the Microsoft Kinect or the Asus Xtion Pro Live, so that at each time stamp i there exist a color image, C_i, and a depth map, D_i. Camera intrinsic parameters are known and the depth map is warped so to match the color image pixel-by-pixel. Thus, given the RGB

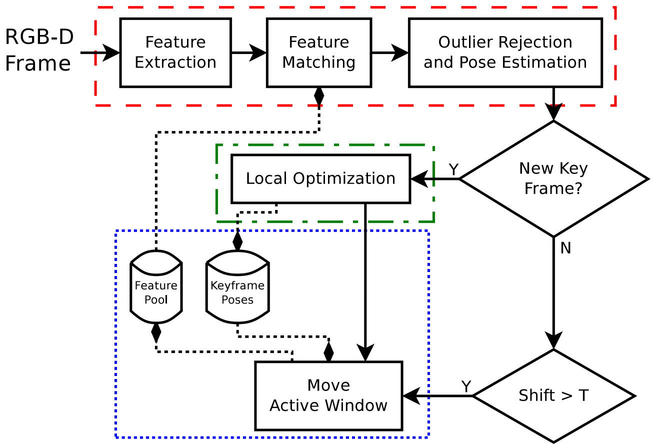

Fig. 2. The SlamDunk pipeline comprises three main modules: Local Mapping (blue dotted line), Camera Tracking (red dashed line) and Local Optimization (green dash-dot line).

camera matrix

$$\mathrm{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

a pixel $\mathbf{m} = (u, v)$ in C_i can be back-projected into the 3D space by

$$\mathbf{p} = \mathrm{K}^{-1} \begin{bmatrix} u \cdot \mathrm{D}_i(\mathbf{m}) \\ v \cdot \mathrm{D}_i(\mathbf{m}) \\ \mathrm{D}_i(\mathbf{m}) \end{bmatrix} . \tag{2}$$

Then, the global coordinates of $\mathbf{p}$ are obtained by rotation and translation according to the estimated camera pose, generally written as a 3×3 rotation matrix R and a translation vector $\mathrm{t} \in \mathbb{R}^3$. Hereinafter, we will denote the camera pose at time i as

$$\mathrm{T}_i = \begin{pmatrix} \mathrm{R}_i & \mathrm{t}_i \\ \bar{\mathbf{0}} & 1 \end{pmatrix} \tag{3}$$

and the corresponding global coordinates of $\mathbf{p}_i$ as $\mathrm{T}_i[\mathbf{p}]$.

4 The SlamDunk Algorithm

As depicted in Fig. 2, the proposed SLAM algorithm can be decoupled into three main modules, namely Local Mapping (Sec. 4.1), Camera Tracking (Sec. 4.2) and Local Optimization (Sec. 4.3).

As for the first module, we model the camera path as a collection of keyframes and store their poses within a quadtree data structure. However, as suggested by [24, 22], to permit exploration of large workspaces we do not consider the whole path for tracking and mapping but, instead, dynamically select a subset of neighboring keyframes, referred to as *active keyframe window*, which creates a local map of the workspace that gets updated smoothly along with camera motion.

Following the pipeline in Fig. 2, each incoming RGB-D frame, made out of a color and depth image, is fed to the Camera Tracking module, which estimates the current camera pose. Purposely, visual features, such as SIFT [16] or SURF [2], are extracted from the color image and matched against the local map of the environment. More precisely, the features found in the current frame are matched into an indexing structure, referred to as *Feature Pool* in Fig. 2, which stores all the features extracted from the color images of the frames belonging to the active keyframe window. Then, incorrect matches are filtered by means of a few RANSAC iterations and camera pose estimated from the inlier set.

Joint optimization of camera poses helps counteracting tracking errors and minimizing drift. As highlighted in Fig. 2, for the sake of efficiency SlamDunk optimizes poses *locally* and upon detection of a new keyframe rather than after tracking each incoming frame. Keyframe detection is based on the overlap between the current frame and the local map, thus effectively selecting those frames which contribute to better cover the explored environment. As executing a costly global optimization across all keyframes would hinder exploration of large workspaces, we rely on an efficient Local Optimization module which enforces consistency between a subset of keyframes linked to the newly spawned one. Our approach is particularly effective with very loopy trajectories, as when the camera comes back frequently to previously explored locations, we do not repeatedly spawn keyframes, as, *e.g.*, [8, 23] would, but instead hook up to the known map. Moreover, as discussed in Sec. 4.4, by combining information from camera tracking and the local map, SlamDunk can infer loop closure and trigger a local optimization accordingly.

Finally, as with RGB-D cameras every color image is paired with a corresponding depth map, a complete, dense reconstruction of the environment is achieved by back projection and, possibly, raycasting into a 3D occupancy grid [11]. Accordingly, SlamDunk provides dense 3D reconstruction in real-time without any post-processing step.

4.1 Local Mapping

Unlike [24, 22], to select the subset of neighboring keyframes determining the local map of the workspace, we neither measure the distance between keyframes according to the estimated camera path nor we count the number of tracked features. Indeed, as illustrated in Fig. 3, we index two coordinates of the 3D translation vectors estimated for all keyframes into a quadtree approximately parallel to the ground floor. Then, to obtain the active keyframe window, *i.e.* the local map of the workspace, we just query the quadtree for a squared window of size W around a given camera pose. The choice of a quadtree data structure enables faster insertion and retrieval than more complex indexing approaches such as octrees or KD-Trees. The query to define a new active keyframe window is performed whenever a new keyframe is spawned and, more precisely, after

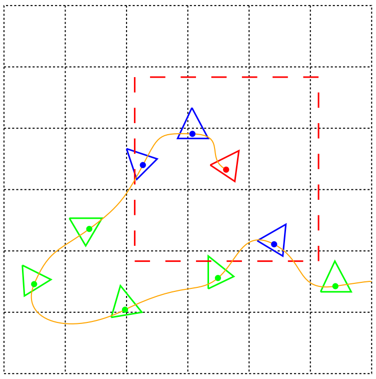

Fig. 3. Keyframe poses are indexed and retrieved within a quadtree structure. A squared window (red square) centered at a given pose (red triangle) is built and queried to get the active keyframes (blue triangles).

the related Local Optimization (Sec. 4.3), so that the query is centered at the optimized pose of the newly created keyframe.

The visual features associated with active keyframes are gathered and indexed into a single KD-Tree structure, the *Feature Pool* in Fig. 2. Moreover, the number of indexed features is further reduced by considering the frustum of the camera pose placed at the center of the active window, so we accept a visual feature *iff* the corresponding 3D point lies inside that volume, *i.e.* it is actually visible from that viewpoint.

Finally, even though the current frame is not spawned as a keyframe, we might still need to move the active keyframe window. Indeed, if the camera moves back to an already well mapped location, there would be no need to create new keyframes; however, as the camera moves the overlap with the local map inevitably decreases so that, although useless and costly, a new keyframe is eventually created. This issue can be dealt with by having the active window follow camera movements. Though, in principle, the active window may be moved after every frame, such an approach usually implies a high computational effort and often small or no improvements at all. Therefore, a more efficient strategy consists in moving the active window only upon a significant camera shift with respect to the center of the local map. Accordingly, we compare the relative translation between the currently estimated pose and the active window center (referred to as “Shift” in Fig. 2): if this is found above a threshold T, we move the active window and recompute the feature pool.

4.2 Camera Tracking

Camera pose estimation is achieved through feature tracking across multiple views. Visual features [16, 2] are extracted from the current RGB image and matched to the *Feature Pool*, *i.e.* the dynamic KD-Tree which stores the features associated with active keyframes (Sec. 4.1). We perform an approximate search with high confidence and a few threads (*e.g.* 4, as reported in Tab. 3) in order to reduce the overall complexity and thus search time. The KD-Tree search is the sole parallelized block of the entire pipeline.

The matching scheme is inspired by the *ratio* criterion described in [16]: given a threshold $\theta \in [0,1]$, we accept a match with Euclidean distance d_0 *iff* the next-closest feature provides a distance d_1 such that $\theta' = d_0/d_1 < \theta$. However, if a feature from the current frame has been also extracted from two different active keyframes, a correct matching step would return a ratio almost equal to 1, making it impossible to detect meaningful matches. Therefore, we compute such ratio looking for the next-closest feature *belonging to the same keyframe* as the closest one. Accordingly, we aim at the best matches from the most similar views.

It is worth pointing out that our multi-view matching scheme enables seamless recovery from tracking failures as long as the camera does not exit the current local map. Tracking failures may occur due to the camera being directed towards a featureless area of the scene or moved while temporarily occluded. In more general terms, a SLAM systems based on pairwise feature matching [10, 8, 23] is inherently prone to failure in case of no or low overlap between the current and previous frame (or previous keyframe). On the other hand, our multi-view approach may keep tracking even under these circumstances, as shown in the supplementary material.

As correspondences are sometimes incorrect, an outlier rejection strategy is employed after the feature matching phase. For each feature pair, we back project in 3D the associated pixels, thus obtaining 3D point correspondences between the current frame and the active window. Then, a standard RANSAC-based Absolute Orientation procedure [1] removes false correspondences and allows robust estimation of the current camera pose.

Once the pose of the current frame has been estimated, we decide upon keyframe spawning. This is a crucial step in every keyframe-based SLAM framework [24, 13]: a fine sampling strategy would inevitably waste memory, while keeping too few frames may lead to a tracking failure. Our approach is to rely on the amount of overlap between the frame and the current local map. Accordingly, we coarsely divide the frame into $N \times M$ cells and count how many contain more than F features successfully matched into the *Feature Pool*: if the count is smaller than a threshold τ, a new keyframe is created. Though very simple, we found this approach to provide good tracking quality under a variety of diverse circumstances.

4.3 Local Optimization

Keyframe detection brings in a vast amount of fresh information about the environment, sometimes even a loop closure. Hence, we refine our estimates and reduce relative mapping errors by means of camera pose optimization. Following Kümmerle *et al.* [15], we map the problem onto a general (hyper) graph, where vertexes represent the unknown 6DOF camera poses and edges encapsulate constraints to be satisfied. In particular, we minimize the following cost function

$$\mathbb{F}(\mathrm{T}_0, \ldots, \mathrm{T}_{n-1}) = \sum_{(\mathbf{m}_i, \mathbf{m}_j) \in \mathscr{C}} w_{ij} \left\| \mathbf{e}_{ij} \right\|^2 \tag{4}$$

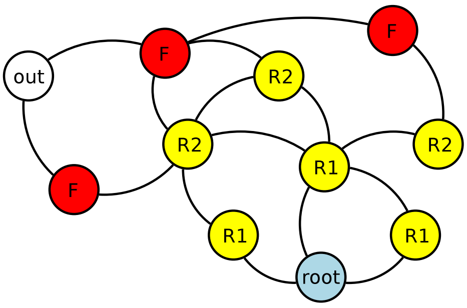

Fig. 4. Local mapping ($R = 3$). Starting from the selected root ("root" - blue), a breadth-first search reveals the first ("R1" - yellow), the second ("R2" - yellow) and the third ("F" - red) ring. The vertexes in the red ring are kept fixed. The white vertex ("out") does not contribute at all to the minimization problem.

where $\{\mathrm{T}_0, \ldots, \mathrm{T}_{n-1}\}$ are the unknown poses, $\mathscr{C}$ is the set of feature matches between keyframes and $w_{ij} \left\|\mathbf{e}_{ij}\right\|^2$ is the cost associated to a feature match, with $w_{ij} \in [0,1]$ representing a weight related to the degree of confidence of the match (*i.e.* $1 - \theta'_{ij}$, see Sec. 4.2). Exploiting the relationship between the RGB and Depth images, constraints between pixels, *i.e.* feature matches, are turned into constraints between 3D points (see Sec. 3). Thus, the $\mathbf{e}_{ij}$ term in Eq. (4) can be written as

$$\mathbf{e}_{ij} = \mathbf{p}_i - \mathrm{T}_i^{-1}\mathrm{T}_j\left[\mathbf{p}_j\right] \tag{5}$$

where $(\mathbf{p}_i, \mathbf{p}_j)$ is the 3D point match associated to the feature match $(\mathbf{m}_i, \mathbf{m}_j) \in \mathscr{C}$ and $(\mathrm{T}_i, \mathrm{T}_j)$ their respective camera poses.

Inspired by [22, 24], instead of a time consuming global optimization, we devise a local optimization approach. As illustrated in Fig. 4, we pick the last detected keyframe as root for a breadth-first search into the optimization graph. Then, all the vertexes found in the first R rings from the root are included into the optimization problem. However, those belonging to the outermost ring only provide constraints (edges) to the problem but are not subject to optimization, *i.e.* they are treated as constants rather than unknowns. This way, we ensure that the local pose configuration does not move w.r.t. the rest of the pose graph. If we find no camera poses in the R^{th} ring, then we fix the root in order to avoid gauge freedom. Thus, we dynamically adapt the optimization procedure to the local configuration, with just a small overhead as the map size increases and always preserving local consistency.

4.4 Metric Loop Closure

SlamDunk handles loop closure both implicitly as well as explicitly. As for the former modality, it is worth pointing out that the keyframes comprising the active window in Fig. 3 are spatially adjacent but not necessarily temporally close one to another along the camera path. This means that spatially adjacent keyframes gathered at distant times can contribute to the local map against which the camera is tracked: as such, tracking seamlessly deploys metric loops although no explicit detection is carried out. However, such implicit handling of loop closures would not trigger any pose optimization unless a new keyframe linking the two opposite endpoints of the loop is spawned, this event depending on the degree of overlap between the current frame and the local map, which is a criterion unrelated to loop closure detection. Therefore, we also explicitly detect metric loops by analysis of the pose graph. More precisely, upon successful tracking, if the overlap between the current frame and the local feature map does not legitimize the creation of a new keyframe, we still check whether at least a pair of keyframes matching the tracked frame would exhibit a shorter mutual distance on the pose graph because of the new links thus created. In such a case, we promote the tracked frame as a new keyframe and trigger a local optimization. In particular, given the list of matched keyframes, we measure the mutual shortest paths in the pose graph. If there exists a pair of such keyframes at a distance greater than a threshold, we insert the current frame in the map as a new keyframe, thus shortening their distance. We found a proper value for the threshold to coincide with the number of rings, R, used for the Relative Bundle Adjustment step of Sec. 4.3.

5 Experimental Results

In this section we evaluate our proposal both quantitatively and qualitatively. As for quantitative experiments, we use several sequences from the publicly available RGB-D benchmark dataset [25], which includes color and depth streams from a Microsoft Kinect and Asus Xtion Pro Live cameras, as well as ground truth data obtained by a motion capture system which tracks camera movements. Due to the complexity of the testing environment, the estimated ground truth is sometimes not very accurate; nevertheless quantitative results are still meaningful and useful to compare different SLAM systems. In particular, Tab. 1 reports the results obtained by SlamDunk (SD) on four Kinect sequences together with those published by the authors of a similar system based on visual features, *i.e*. RGB-D SLAM [8]. As for SlamDunk, we test here three different variants, each relying on a different state-of-the-art feature detector and descriptor, namely SIFT (SD-SIFT) [16], SURF (SD-SURF64) and extended SURF (SD-SURF128) [2], from the OpenCV implementation[4]. Throughout all the quantitative and qualitative experiments described in this Section, the other parameters of our method are set to the values reported in Tab. 3. The results reported in Tab. 1 vouch that SlamDunk is typically more accurate that RGB-D SLAM regardless the adopted visual features. Moreover, SD-SIFT runs at 6/8 FPS, SD-SURF128 at 20/25 FPS (with a maximum of 30 FPS) and SD-SURF64 at 35/40 FPS. Conversely, RGB-D SLAM achieves real-time operation only when using ORB features [20] or GPU acceleration. On average, the feature matching and outlier rejection steps take about 2 ms, while the feature pool update, which scales with the size of the active window, usually requires from 4.8 to 14.6 ms. The local optimization step could be the most time consuming, since it strongly depends on the number of active camera poses and their corresponding matches. In our experiments it required from 3.6 to 37.3 ms, though, being the optimization run only upon keyframe creation, its impact on the overall performance is limited.

Table 1. RMSE of absolute trajectory error (meters) on four sequences from the freiburg1 RGB-D SLAM benchmark dataset (Microsoft Kinect camera).

Sequence	SD-SIFT	SD-SURF64	SD-SURF128	RGB-D SLAM
fr1/xyz	0.017	**0.016**	**0.016**	0.021
fr1/360	0.111	0.101	**0.084**	0.103
fr1/desk	**0.022**	0.027	0.025	0.049
fr1/floor	0.044	0.052	**0.042**	0.055
AVERAGE	0.048	0.049	**0.042**	0.057

Table 2. RMSE and median of absolute trajectory error (meters) on three sequences from the freiburg3 RGB-D SLAM benchmark dataset (Asus Xtion Pro Live camera).

Sequence		SD-SIFT	SD-SURF64	SD-SURF128
fr3/long office household	RMSE	**0.023**	0.026	0.027
	median	**0.021**	0.024	0.022
fr3/structure texture near	RMSE	**0.012**	0.015	0.016
	median	**0.007**	0.011	0.012
fr3/structure texture far	RMSE	**0.024**	**0.024**	0.025
	median	0.022	0.019	**0.015**
fr3/teddy	RMSE	**0.069**	0.095	0.089
	median	**0.033**	0.056	0.067
AVERAGE	RMSE	**0.032**	0.040	0.039
	median	**0.021**	0.028	0.029

Table 3. SlamDunk parameters used throughout the experiments

Number of Threads for KD-Tree Search	4
W	5m
T	$5\%W = 0.25$m
θ	0.8
N, M	4
F	1
τ	80%
R	3

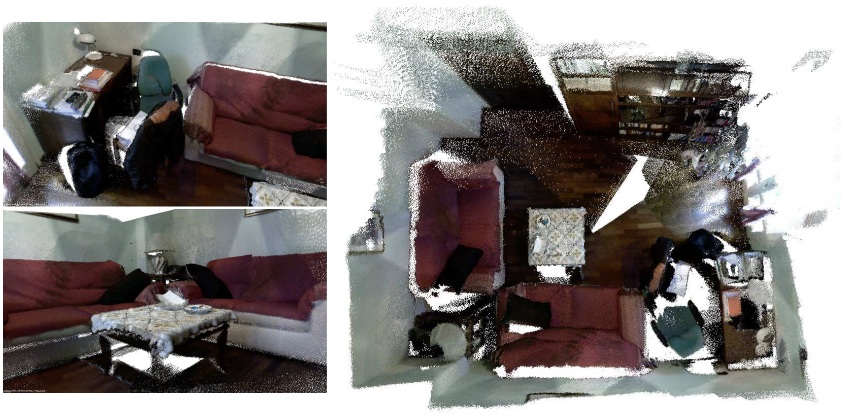

Fig. 5. Details of the room shown on the right image of Fig. 1.

Tab. 2 provides additional quantitative results on four sequences of the RGB-D benchmark dataset captured by an Asus Xtion Pro Live. To the best of our knowledge, the results achieved by RGB-D SLAM on these sequences are not available in any publication. Nonetheless, we report ours in Tab. 2, so to allow other researchers to compare quantitatively their methods to SlamDunk. Basically, these additional results on a different pool of sequences taken with a different camera confirm the good accuracy of our method. Between the three variants, SD-SIFT behaves consistently slightly better on the Asus Xtion sequences. However, the use of SURF features in SlamDunk provides overall the best trade-off between speed and accuracy. The Table reports also the median value of the absolute trajectory error, so to highlight how, in some cases, the median is significantly smaller than the RMSE, which is a symptom of a few larger errors within a trend of more accurate pose estimations than measured by the RMSE.

The effectiveness of our proposal can also be judged from qualitative results attained on sequences captured by an Asus Xtion Pro Live camera moved freely in the workspace. Purposely, we show qualitative results addressing three diverse scenarios: object scanning (Fig. 1), complete reconstruction of a room (Fig. 1 and Fig. 5) and robot navigation within an apartment (Fig. 6). Finally, we provide additional qualitative results in the supplementary material. In particular, a video shows detailed navigation within the 3D reconstruction depicted in the right image of Fig. 1 and around the scanned object in the central image of Fig. 1. Moreover, we include a sequence that shows real-time 3D reconstruction of a Lab environment and demonstrates the ability of SlamDunk to recover from tracking failures due to the camera undergoing motion while the lens is temporarily occluded (as discussed in Sec. 4.2).

6 Final Remarks

We have presented SlamDunk, a novel real-time algorithm which allows for solving effectively the Visual SLAM problem with a few CPU-threads, small memory requirements and no hardware acceleration. Thanks to its peculiar traits, the approach de-

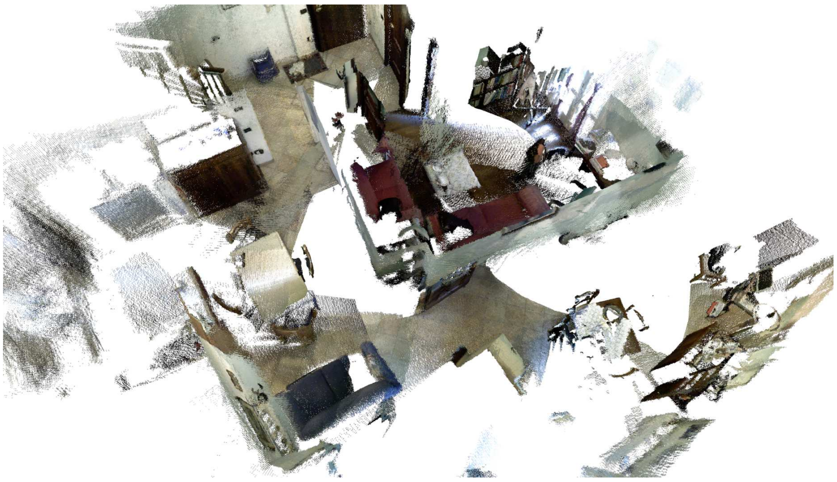

Fig. 6. A simulated robot navigation through an apartment.

scribed in this paper may represent a relevant step towards rendering real-time dense SLAM amenable to low-cost, low-power platforms, so to foster further research and pave the way for a host of new scenarios. In particular, the realm of mobile applications may soon become addressable, should the foreseeable advent of RGB-D sensing on off-the-shelf tablets and smartphones turn into reality [1]. Hence, we are currently working to create an implementation of SlamDunk suited to Android tablet devices.

[1] For example, the *Structure* sensor (http://structure.io/) as well as the Project Tango tablet by Google are scheduled for availability in 2014

References

1. Arun, K.S., Huang, T.S., Blostein, S.D.: Least-squares fitting of two 3-d point sets. Pattern Analysis and Machine Intelligence (PAMI), IEEE Trans. on 9(5), 698–700 (sept 1987)
2. Bay, H., Ess, A., Tuytelaars, T., Gool, L.V.: Speeded-up robust features (SURF). Computer Vision and Image Understanding 110(3), 346 – 359 (September, 10 2008)
3. Borrmann, D., Elseberg, J., Lingemann, K., NÃijchter, A., Hertzberg, J.: Globally consistent 3d mapping with scan matching. Journal of Robotics and Autonomous Systems 56, 130–142 (Feb 2008)
4. Bradski, G.: Dr. Dobb's Journal of Software Tools
5. Bylow, E., Sturm, J., Kerl, C., Kahl, F., Cremers, D.: Real-time camera tracking and 3d reconstruction using signed distance functions. In: Robotics: Science and Systems (RSS). Berlin, Germany (2013)
6. Davison, A., Reid, I.D., Molton, N.D., Stasse, O.: MonoSLAM: Real-time single camera SLAM. Pattern Analysis and Machine Intelligence (PAMI), IEEE Trans. on 29(6), 1052–1067 (Jun 2007)
7. Dryanovski, I., Valenti, R., Xiao, J.: Fast visual odometry and mapping from rgb-d data. In: Robotics and Automation (ICRA), IEEE Int'l Conf. on. pp. 2305–2310 (May 2013)
8. Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D., Burgard, W.: An evaluation of the RGB-D SLAM system. In: Robotics and Automation (ICRA), IEEE Int'l Conf. on. St. Paul, MA, USA (May 2012)
9. Fioraio, N., Konolige, K.: Realtime visual and point cloud SLAM. In: RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras. Los Angeles (CA), USA (2011)
10. Henry, P., Krainin, M., Herbst, E., Ren, X., Fox, D.: RGB-D mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. The International Journal of Robotics Research 31(5), 647–663 (Feb 2012)
11. Hornung, A., Wurm, K.M., Bennewitz, M., Stachniss, C., Burgard, W.: OctoMap: An efficient probabilistic 3D mapping framework based on octrees. Autonomous Robots (2013), http://octomap.github.com, software available at http://octomap.github.com
12. Kerl, C., Sturm, J., Cremers, D.: Dense visual SLAM for RGB-D cameras. In: Intelligent Robots and Systems, IEEE/RSJ Int'l Conf. on (2013)
13. Klein, G., Murray, D.: Parallel tracking and mapping for small ar workspaces. In: Mixed and Augmented Reality (ISMAR), IEEE and ACM Int'l Symp on. pp. 225 –234 (nov 2007)
14. Konolige, K., Agrawal, M.: FrameSLAM: From bundle adjustment to real-time visual mapping. IEEE Transactions on Robotics 24(5), 1066–1077 (Oct 2008)
15. Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., Burgard, W.: g2o: A general framework for graph optimization. In: International Conference on Robotics and Automation (ICRA). Shanghai, China (may 2011)
16. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. IJCV 60(2), 91–119 (January, 5 2004)
17. Montemerlo, M., Thrun, S.: Large-scale robotic 3-d mapping of urban structures. In: International Symposium on Experimental Robotics (ISER). Singapore (2004)
18. Newcombe, R., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A., Kohli, P., Shotton, J., Hodges, S., Fitzgibbon, A.: KinectFusion: Real-time dense surface mapping and tracking. In: 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR). pp. 127–136. Washington, DC, USA (2011)
19. Newcombe, R., Lovegrove, S., Davison, A.: DTAM: Dense tracking and mapping in real-time. In: International Conference on Computer Vision (ICCV). pp. 2320–2327 (Nov 2011)
20. Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: Orb: An efficient alternative to sift or surf. In: Computer Vision (ICCV), IEEE Int'l Conf. on. pp. 2564–2571 (Nov 2011)

21. Scherer, S., Zell, A.: Efficient onbard rgbd-slam for autonomous mavs. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ Int'l Conf. on. pp. 1062–1068 (Nov 2013)
22. Sibley, G., Mei, C., Reid, I., Newman, P.: Adaptive relative bundle adjustment. In: Robotics: Science and Systems (RSS). Seattle, USA (2009)
23. Steinbruecker, F., Kerl, C., Sturm, J., Cremers, D.: Large-scale multi-resolution surface reconstruction from RGB-D sequences. In: International Conference on Computer Vision. Sydney, Australia (2013)
24. Strasdat, H., Davison, A.J., Montiel, J., Konolige, K.: Double window optimisation for constant time visual SLAM. In: International Conference on Computer Vision (ICCV). pp. 2352–2359. Los Alamitos, CA, USA (2011)
25. Sturm, J., Engelhard, N., Endres, F., Burgard, W., Cremers, D.: A benchmark for the evaluation of RGB-D SLAM systems. In: Proc. of the International Conference on Intelligent Robot Systems (IROS) (Oct 2012)
26. Whelan, T., Mcdonald, J., Kaess, M., Fallon, M., Johannsson, H., Leonard, J.: Kintinuous: Spatially extended KinectFusion. In: RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras. Sydney, Australia (2012)
27. Wu, C.: SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). http://cs.unc.edu/ ccwu/siftgpu (2007)