Using Dense 3D Reconstruction for Visual Odometry Based on Structure from Motion Techniques

Marcelo de Mattos Nascimento,
Manuel Eduardo Loaiza Fernandez(✉), and Alberto Barbosa Raposo

Department of Informatics, Pontifical Catholic University of Rio de Janeiro, PUC-Rio, Rio de Janeiro, RJ, Brazil
mzumbin@gmail.com,
{manuel,abraposo}@tecgraf.puc-rio.br

Abstract. Aim of intense research in the field computational vision, dense 3D reconstruction achieves an important landmark with first methods running in real time with millimetric precision, using RGBD cameras and GPUs. However, these methods are not suitable for low computational resources. The goal of this work is to show a method of visual odometry using regular cameras, without using a GPU. The proposed method is based on techniques of sparse Structure from Motion (SFM), using data provided by dense 3D reconstruction. Visual odometry is the process of estimating the position and orientation of an agent (a robot, for instance), based on images. This paper compares the proposed method with the odometry calculated by Kinect Fusion. Odometry provided by this work can be used to model a camera position and orientation from dense 3D reconstruction. AQ1

1 Introduction

Augmented reality applications using cameras are increasingly present in live broadcasts, advertisements and games. The critical requirement in the above cases is to estimate the camera position and orientation. This process of estimating the orientation and position with a camera is known as visual odometry. The first work on this subject, developed by Moravec [1], dates from the early 80's and was designed to control a robotic probe. The visual odometry is currently a subset of a more general problem known as structure from motion (SFM) [2]. It focuses on calculating the camera position and direction sequentially as it processes new frames. When it generates a map of the environment, we have methods called simultaneous localization and mapping (SLAM), as Davison et al. [3], Klein and Murray [4], and Grisetti et al. [5] has presented. Unlike the visual SLAM methods that seek real-time, the methods of dense reconstruction try to recover the geometry of a scene, also based on visual odometry. An example of a dense reconstruction system is Kinect Fusion [6], which uses a dense global model and an RGBD camera, achieving great precision in the reconstruction. The applications of such methods are diverse.

DOI: 10.1007/978-3-319-50832-0_47

In this work, we implemented a method for calculating the visual odometry with a pipeline based on sparse features with local adjustment and using information from dense reconstruction, in this case, the 3D position of reconstruction points with 2D information of features. To assess the accuracy and performance of the system proposed here, the results are compared with those obtained with Kinect Fusion. We have put together data from a dense 3D reconstruction previously made by Kinect Fusion, with position calculated by a classic SFM pipeline with sparse features, without the use of a GPU or an RGBD camera. Thus, information such as dense geometry of a scene is available and the odometry may be calculated in devices with less computational power.

The text is organized as follows. Section 2 provides a summary of the work related to dense 3D reconstruction and not dense visual odometry. The following section presents the proposed method. Section 4 describes the results of the proposed model with Kinect Fusion in two scenes. Finally, the last section presents conclusions.

2 Related Work

Visual odometry is a widely studied topic [7]; it has several techniques that leverage the use of common RGB cameras, as well as new RGBD cameras with depth information. In this paper, we focus on real-time techniques using a single RGB calibrated camera and an RGBD camera. The RGBD camera is used in a first step to make the dense reconstruction of a predetermined environment, and in a second step, the visual odometry is made with an RGB camera, allowing this device to know where it is within the previously mapped environment. Visual Odometry techniques can be classified into two types, those based on sparse environment information and those that use all environmental information by creating a dense representation of the same.

The visual odometry techniques using sparse points were successfully implemented to support augmented reality and robot control applications. The typical pipeline of these methods, as described by Nister et al. [8], is to first extract points of interest with the use of detectors like Harris [9] and FAST [10] and then establish the correspondence between the points of interest calculated in the current frame and the equivalent points in the previous one. The correspondence is made by comparing an area around the points, using some metrics based on descriptors such as SURF [11] and FREAK [12]. Finally, the rigid body transformation between consecutive frames is estimated with 2D-2D methods decomposing the essential matrix, or 2D-3D methods using triangulation [13] between pairs of previous 3D reconstructed points. The stage of points of interest correspondence is not free of errors and the amount of points is important for filtering outliers using methods like RANSAC [14].

The techniques of dense visual odometry, unlike the sparse ones, use information of the whole picture. There are two main approaches: those based on photometric error and those based on iterative closest point (ICP) [15]. A work that shows the first approach based on photogrammetric error is the work described as DTAM [16], where the authors were able to extract the depth of a picture through the analysis of a set of images filling a distance *d*, building a discrete cost volume to assign a depth value to a certain pixel. In the line of ICP, the methods seek to align, with a rigid body transformation, a set of 3D points to another set of 3D points, minimizing the distances between corresponding points. There are several variations of ICP. A widely used is the plane to point [17], where their normals are also used for calculating the alignment. When depth information is available in a depth map from a projection as in an RGBD camera, the technique known as projective date association [18] can be used with better performance to search nearest neighbors, commonly used in ICP methods for the calculation of the corresponding points. The technique that better uses this approach is the Kinect Fusion [6], which is the basis for our proposal in the first stage.

Our proposal is to create a hybrid method that in a first step uses dense odometry data to create a 3D map of an environment that in a second step can be used by an ordinary RGB camera to perform a sparse odometry without the high computational costs derived from the image processing required by these techniques.

3 The Proposed Method

The method proposed in this work is divided into two blocks:

- The first block carries out the construction of an offline time 3D map using the dense reconstruction offered by Kinect.
- The second one uses the extracted data from the 3D map to perform a visual odometry in real time. For this, it uses keypoints and descriptors defined from the data captured in the first block.

The procedures performed in each block are described below.

3.1 Offline Map Generation

Kinect has two cameras, an infrared used to generate the depth map and an RGB camera. As the cameras are tightly coupled and calibrated, there is a transformation that takes a pose from the RGB camera to the IR camera. With this information, Kinect Fusion performs the dense 3D reconstruction of the environment by following these steps:

- Obtain the depth map, smooth the map maintaining edges, and calculate normal for each point.
- Estimate the pose of the new depth map.
- Update the global volume with the new depth map.
- Generate a depth map through raycast using a prediction for next pose and use this map to calculate the next pose.

Since the Kinect Fusion works at 30 FPS, the same capture rate of the cameras, then it is possible to calculate a 3D pose for each captured frame. With this information and the RGB stream (Fig. 1) a keyframes map is constructed, being composed of a set of keypoints, a descriptor, and a 3D pose for each certain interval of the 3D map.

This dataset will be used to calculate the visual odometry in the next step. This process starts reading the stream of IR and RGB cameras sequentially (Algorithm 1, lines 2–3). After running the fusion pipeline (line 4), it is checked whether the tracking is valid (convergence given by Fusion Kinect API). Moreover, if the time interval between capturing the two images is below a threshold and there was a movement to a distance greater than 10 cm or an angle larger than 10° between the last image stored in the buffer and the current one. After this another raycast of the volume using the RGB camera pose and its intrinsic parameters is executed, and with this data an image where the pixels have the 3D coordinates of the world is generated (line 7). RGB and 3D images are stored in a buffer. After a certain number of frames the process is finished and the buffer is written to disk. Thereafter, for each 3D and RGB images set their respective pairs of SURF keypoints and FREAK descriptors are generated.

Algorithm 1: Create the Keyframe map

```
1: loop
2:     $Im_{ir} \leftarrow$ readImageIR
3:     $Im_{RGB} \leftarrow$ readImageRGB
4:     fusion($Im_{ir}$)                              execute fusion pipeline
5:     $T^{ir}_{w} \leftarrow$ fusion.readPose($Im_{ir}$)
6: If changeAngle($T_{ir}$) && timeStampOK && fusion.trackOK then
7:     $Im_{3d} \leftarrow$ fusion.raycast($T^{rgb}_{ir}$ $T^{ir}_{w}$, $K_{rgb}$)   new raycast
8:     buffer $\leftarrow$ ($Im_{rgb}$, $Im_{3d}$,, $T^{rgb}_{ir}$ $T^{ir}_{w}$)       save 3D image, 2D image and pose
9: end if
10: end loop
11: for all($Im_{rgb}$, $Im_{3d}$,,Pose) belonging to buffer do
12:     keypoints $\leftarrow$ surf($Im_{rgb}$)
13:     descriptors $\leftarrow$ freak(keypoints, $Im_{rgb}$)
14:     map $\leftarrow$ (pose, keypoints, descriptors, $Im_{3d}$)     add tuple to map
15: end for
```

The SURF keypoints and FREAK descriptors, lines 12 and 13, are calculated using the OpenCV library [19]. The calculation of the non max suppression is done by comparing every neighbor, pixel by pixel in a simple way, although there are algorithms with higher performance. This allows the implementation to be parallelizable. The parameters used in the calculation of keypoints are: value above 300 for determining the Hessian, 4/8 and two scales per octave. The implementation of FREAK descriptor uses the size of keypoints (in the case of SURF keypoint, the kernel size used in the range where the keypoint was found), in order to adjust the position and area of the sample. The scale in the area of the sample allows the FREAK descriptor to be resistant to scale operations.

3.2 Odometry in Real Time

The visual odometry provides a pose g(*R*, *T*) using the stream of an RGB camera and keyframes map. The pose is calculated according to the following steps:

- Read the RGB image stream (algorithm 2, line 2)
- Calculate the SURF keypoints and FREAK descriptors; search similar descriptors of the current image in keyframes and create the set of 2D and 3D points (lines 3–5)

- Calculate the pose minimizing the reprojection error of 3D points in the current image (line 6)
- Find nearest keyframe following a metric between angle and distance of the current pose with the calculated pose (line 7).

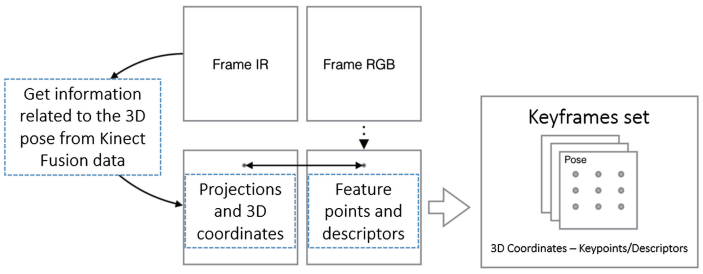

Frame IR
Frame RGB
Get information related to the 3D pose from Kinect Fusion data
Projections and 3D coordinates
Feature points and descriptors
Keyframes set
Pose
3D Coordinates – Keypoints/Descriptors

Fig. 1. Flow process for the construction of offline map.

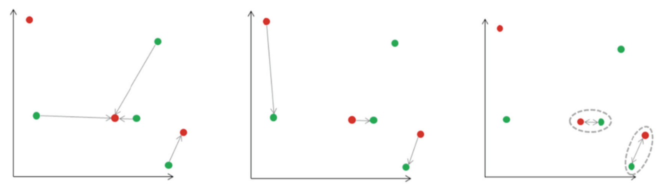
3.3 Matching

This step makes the association between the descriptors of the current frame and keyframe descriptors. Initially the SURF keypoints are calculated (line 3 Algorithm 2) and after the FREAK descriptors are calculated, line 4.

```
Algorithm 2: Odometry
1: loop
2:     Im_RGB ← readImageRGB
3:     keypts ← surf(Im_rgb)
4:     descriptors ← freak(keypts, Im_rgb)
5:     (2D keypts, 2D current, 3D) ← matching(k.keypts, k.descriptors, keypts, descriptors)
6:     pose ← calcPose(2D keypts, 2D current, 3D, k.pose, poseCurrent)
7:     k ← searchNext(map, pose)
8: end loop
```

This is followed by a search of the next neighbor (NN) using as the metric the Hamming distance, which is given by SUM (XOR(s1, s2)), being s1 and s2 two binary strings, the xor or exclusive and the SUM function has the numbers 1 in the resulting string. The search is done by brute force comparing each descriptor set of the current frame with the set of descriptors of keyframes by selecting the descriptor with less distance for each pair. We used OpenCV BFMatcher class that has optimizations.

After the search for the closest neighbor, a filter is created to select the strongest correspondences, making sure that the shortest distance to the "**i**" assembly descriptor A is the "**j**" descriptor set B, and if this relationship is valid in the opposite direction too (Fig. 2). This calculation is done crossing the tuples calculated in search nn(A,B), for each tuple ($\mathbf{a_i,b_j,d}$) update the tuple ($\mathbf{b_j,k,d_{min}}$) and ($\mathbf{b_j;j;d}$) if $d < d_{min}$.



(a) nn(green, red) (b) nn(red, green) (c) cross

Fig. 2. Filtering procedure of the strongest correspondences. (Color figure online)

The resulting tuple contains the filtered pairs. Figure 2 illustrates the process with Euclidean distance, but the same is valid for the Hamming distance that satisfies the axioms of distance, especially in the symmetry case $d(a, b) = d(b, a)$.

3.4 Pose Calculation

The pose calculation is done by minimizing the reprojection error of 3D points found in matching and projected in the current image. The residual is given by ri = π(Xi) – xi. Where π is the projection matrix, ***Xi*** is a 3D point coordinate and ***xi*** is a pixel coordinate found in the current frame.

$$\frac{\partial\pi\left(e^{\widehat{\xi}}\oplus G\oplus X\right)}{\partial\xi} = \left.\frac{\partial\pi(X')}{\partial X'}\right|_{X'=G\oplus X=g} \left.\frac{\partial e^{\widehat{\xi}}\oplus G\oplus X}{\partial\xi}\right|_{\xi=0}$$

This is a problem of nonlinear least squares containing outliers. The projection can be parameterized using the tangent space of the rigid body transformation, and assuming a small transformation $\xi \approx 0$ in relation to a previous pose G(*R*,*T*). This assumption allows to calculate the Jacobian of projection π in a simple way with the formula, as described in [20]. The simplification $\xi = 0$ is not always the case of the current problem. The use of a robust descriptor allows the matching function with considerable difference between poses and parameterization, which must take into account this characteristic. Thus, the projection π is parameterized with the equation:

$$K\left(e^{\widehat{\omega}}X + T\right) = X_\pi \; x = X_\pi^1 / X_\pi^3 \; y = X_\pi^2 / X_\pi^3$$

with the variable ***x*** = (*w1*,*w2*,*w3*,*t1*,*t2*,*t3*). This problem is solved with the aid of Ceres Solver library [21], using with automatic derivatives calculations [22] and the Levenberg- Marquardt method (LM).

3.5 Select the Keyframe

The keyframe that will be used for matching is the one with the minimum value of the formula: $(2-\cos(\mathbf{a}))^2\boldsymbol{d}$, where $\boldsymbol{d}$ is the distance between the origin of the current pose **P** and keyframe “$\boldsymbol{k_i}$”, and “$\boldsymbol{a}$” is the angle between the **Z** components (Fig. 3).

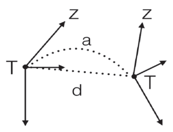


Fig. 3. Angle and distance between two poses.

The poses P and Pk (pose in the kth keyframe) transforms the world coordinates to the camera and can be seen as matrices that change the orthonormal basis with change of origin. The Z component points to the projection center and T is the distance to the origin of the world coordinates. The cosine of the angle and the distance **d** are given by:

$$\cos(a) = \frac{Z_k \,.\, Z_p}{||Z_k||\;||Z_p||} \quad d = ||T_k - T_p||$$

4 Results

In this section, we present a comparison of the proposed method with the localization calculated by the Kinect Fusion algorithm and analyze the performance achieved. To validate our method, a test composed of two parts was designed: first, mounting and generating a 3D map of the environment, and capturing and saving some keyframes to this environment based on the data generated by the Kinect Fusion algorithm. A second part does the analysis and comparison between the odometry calculated with Kinect Fusion and with the proposed method based on the approximations and distances to saved keyframes.

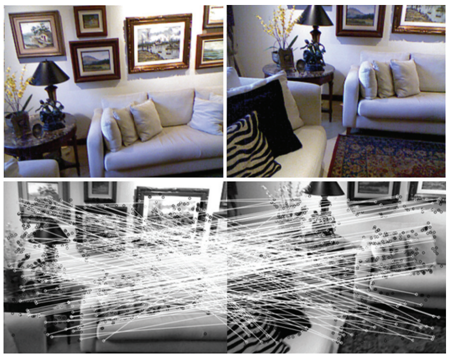


Fig. 4. Outliers present in matching.

The RGB camera of the Kinect device was used in our method to calculate its position and navigate in the environment using the keyframe data, the descriptors and the poses mapped to the first part of the proposed method. 3D maps were made using the Kinect Fusion algorithm with a resolution of 128 voxels per meter and $512 \times 384 \times 512$ voxels, resulting in a volume of $4 \times 3 \times 4$ m and resolution by voxel $8 \times 10 \times 8$ mm.

The capture of the keyframes used in the mapping environment module was done at every movement with a separated distance of 15 cm between frames or 10° angle, both measures were extracted from the data of Kinect Fusion odometry. The calculation of the intrinsic parameters of the RGB and IR cameras, and the relative pose between them cameras was made with MRPT [23]. Two cases with two different scenarios were chosen for our tests, where Kinect RGB camera is used to calculate the visual odometry with our method. The camera works with the resolution of 640×480 pixels and a frame rate of 20 FPS. Figure 4 shows views of the two test scenarios.

During the matching phase and the creation of the map, the SURF detector found a high amount of keypoints, with an average of 652 per frame. In Fig. 4 there is a matching display showing on the right, an image of one frame and on the left, a picture in the keyframe saved with which the odometry will be calculated. The white dots are the keypoints/descriptors and the lines are the matches found between them. In Fig. 4, in the region in the table in the upper left corner we can see the outliers. Although there are several outliers, we use the RANSAC algorithm to reduce them in the correspondence process. The distance and the orientation obtained for the calculated pose in relation to the value obtained by Fusion remained smaller than 4 cm and less than 2° (Fig. 5, up) throughout the experiment in the case of the scene 1.

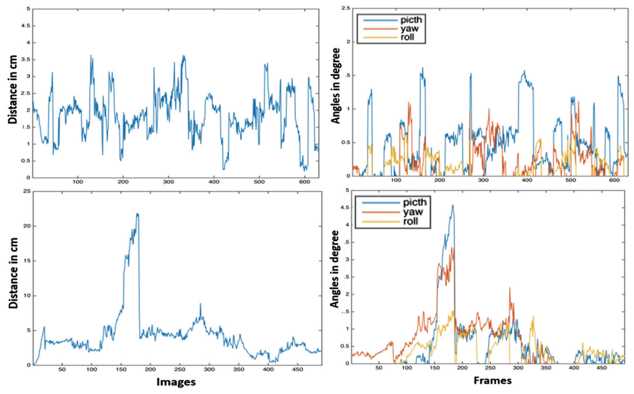


Fig. 5. Position and orientation errors for scene 1 (Up) and scene 2 (bottom).

In Scene 1, the maximum value for the distance between the pose calculated by the Fusion and the proposed method was 3.75 cm with minimum of 0.14 cm. The average value was 1.83 cm and standard deviation of 0.67 cm.

LM	3,94s	18,6%
SURF	11,9s	56%
FREAK	1,03s	4,6%
NN	3,97s	19%
Total	21,2s	
FPS*	29,8	

LM	4,12s	17,8%
SURF	11,6s	50%
FREAK	1,18s	5%
NN	5,9s	25%
Total	23,2s	
FPS*	21,2	

Fig. 6. Execution times of visual odometry: scene 1 (left), scene 2 (right).

Figure 6, shows the execution times for each components of our method. In the case of the experiment on the scene 2, we had a noticeably poorer quality compared to the results of scene 1. The maximum value for the distance in relation to the Fusion was 22.6 cm with minimum of 0.18 cm. The average value was 4.63 and standard deviation of 3.80 cm (Fig. 5, bottom). Figure 6 shows the time values that each subprocess of odometry calculations, as it is showed the bottleneck of the process is the step of finding the characteristic points with SURF detection algorithms. Figure 7 shows the graphic results of odometry process using the RGB camera. The white dots represent keypoints/map descriptors and the blue line is the trajectory of the camera calculated by Kinect Fusion algorithm.

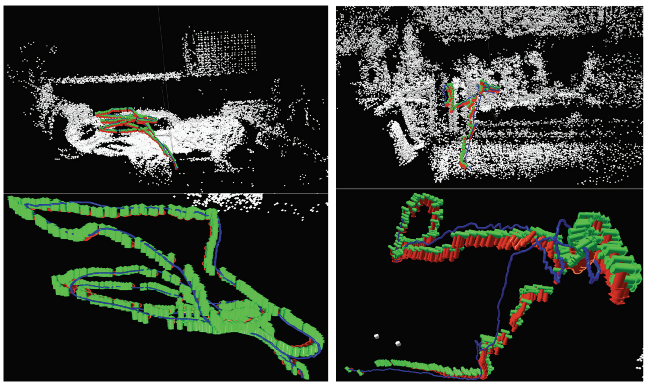


Fig. 7. Visual odometry results for scene 1(left) and scene 2 (right). (Color figure online)

The green arrows (Z axis) and red (Y-axis) represents the pose and the position calculated by the visual odometry based on the keyframes saved as a part of our proposed method.

5 Conclusion and Future Work

This paper proposed a method to use data from a 3D dense reconstruction made by Kinect Fusion algorithm, joined to a pipeline that calculates the visual odometry using sparse features. In the experiments, the proposed method calculates in real time the pose of an external camera, based on information of pre-processed 3D mapping information and 2D features captured and saved for some keyframes in the tracking environment. In the worst case, our method gets a distance error of 23 cm in relation to Kinect Fusion odometry calculation.

The limitations of the 3D mapping using the Kinect device restricted our tests for the proposed method to internal areas with controlled lighting and not allowed the camera to be moved with great speed, the maximum angle recorded between pose and keyframe was 20°, and the capture frame rate of Kinect device was 30 FPS.

One practical application for the presented method would be to use the visual odometry calculation by the external cameras to give support to augmented reality applications. For example, an initial user might be generating keyframes with the 3D environment-mapping module, and other users in the same environment can simultaneously use the odometry information to view or interact with virtual objects placed in the scene, without the need to have an RGBD camera. In this case, the user can use their standard RGB cameras found in mobile devices such as smartphones or tablets.

References

1. Moravec, H.: Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, Carnegie Mellon University, CMU-RI-TR-80-03 Document (1980)
2. Ma, Y., Soatto, S., Kosecka, J., Sastry, S.: An Invitation to 3-D Vision: From Images to Geometric Models, vol. 26. Springer Science & Business Media, New York (2001)
3. Davison, A., Reid, I., Molton, N., Stasse, O.: Monoslam: real-time single camera slam. IEEE Trans. Pattern Anal. Mach. Intell. **29**(6), 1052–1067 (2007)
4. Klein, G., Murray, D.: Parallel tracking and mapping for small ar workspaces. In: 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR, pp. 225–234 (2007)
5. Grisetti, G., Kummerle, R., Stachniss, C., Burgard, W.: A tutorial on graph-based slam. Intell. Transp. Syst. Mag. IEEE **2**(4), 31–43 (2010)
6. Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., Fitzgibbon, A.: Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, UIST, pp. 559–568, NY, USA (2011)
7. Scaramuzza, D., Fraundorfer, F.: Visual odometry [tutorial]. Robot. Autom. Mag. IEEE **18** (4), 80–92 (2011)
8. Nister D., Naroditsky O., Bergen, J.: Visual odometry. In: Proceedings of IEEE Computer Society Conference Computer Vision and Pattern Recognition, CVPR, vol. 1, pp. I–652 (2004)
9. Harris, C., Pike, J.: 3d positional integration from image sequences. Image Vis. Comput. **6** (2), 87–90 (1988)
10. Rosten, E., Drummond, T.: Machine learning for high-speed corner detection. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3951, pp. 430–443. Springer, Heidelberg (2006). doi:10.1007/11744023_34
11. Bay, H., Tuytelaars, T., Gool, L.: SURF: speeded up robust features. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3951, pp. 404–417. Springer, Heidelberg (2006). doi:10.1007/11744023_32

12. Alahi, A., Ortiz, R., Vandergheynst, P.: Freak: fast retina keypoint. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 510–517 (2012)
13. Hartley, R., Sturm, P.: Triangulation. Comput. Vis. Image Underst. **68**(2), 146–157 (1997)
14. Fischler, M., Bolles, R.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**(6), 381–395 (1981)
15. Besl, P., McKay, N.A.: Method for registration of 3-d shapes. IEEE Trans. Pattern Anal. Mach. Intell. **14**(2), 239–256 (1992)
16. Newcombe, R., Lovegrove, S., Dtam, D.A.: Dense tracking and mapping in real-time. In: IEEE International Conference on ICCV, pp. 2320–2327 (2011)
17. Chen, Y., Medioni, G.: Object modeling by registration of multiple range images. In: IEEE International Conference on Robotics and Automation, pp. 2724–2729 (1991)
18. Rusinkiewicz, S., Levoy, M.: Efficient variants of the ICP algorithm. In: Proceedings of Third International Conference on 3-D Digital Imaging and Modeling, pp. 145–152 (2001)
19. Bradski, G.: The OpenCV library. Dr. Dobb's J. Softw. Tools (2000)
20. Blanco, J.L.: A tutorial on SE(3) transformation parameterizations and on-manifold optimization (2014)
21. Agarwal, S., Mierle, K., et al.: Ceres solver (2016). http://ceres-solver.org
22. Wikipedia: Automatic differentiation (2016) http://en.wikipedia.org/wiki/Automatic_differentiation
23. Blanco, J.L.: Mobile Robot Programming Toolkit (MRPT) (2016). http://www.mrpt.org