RGBiD-SLAM for Accurate Real-time Localisation and 3D Mapping

Daniel Gutierrez-Gomez and Jose J. Guerrero*†

July 24, 2018

Abstract

In this paper we present a complete SLAM system for RGB-D cameras, namely RGB-iD SLAM. The presented approach is a dense direct SLAM method with the main characteristic of working with the depth maps in inverse depth parametrisation for the routines of dense alignment or keyframe fusion. The system consists in 2 CPU threads working in parallel, which share the use of the GPU for dense alignment and keyframe fusion routines. The first thread is a front-end operating at frame rate, which processes every incoming frame from the RGB-D sensor to compute the incremental odometry and integrate it in a keyframe which is changed periodically following a covisibility-based strategy. The second thread is a back-end which receives keyframes from the front-end. This thread is in charge of segmenting the keyframes based on their structure, describing them using Bags of Words, trying to find potential loop closures with previous keyframes, and in such case perform pose-graph optimisation for trajectory correction. In addition, our system allows is able to compute the odometry both with unregistered and registered depth maps, allowing to use customised calibrations of the RGB-D sensor. As a consequence in the paper we also propose a detailed calibration pipeline to compute customised calibrations for particular RGB-D cameras. The experiments with our approach in the TUM RGB-D benchmark datasets show results superior in accuracy to the state-of-the-art in many of the sequences. The code has been made available on-line for research purposes [1].

1 Introduction

In the last years visual SLAM has become one fertile research topic in the fields of computer vision and robotics. Visual SLAM can be addressed either by feature-based or direct feature-less methods. In a previous step feature-based methods extract pixel locations with high gradients and compute the motion by minimisation of the pixelic reprojection error. Feature-less or direct methods skip this step and compute the motion estimate directly from the raw camera input. Regardless which method is used a complete visual SLAM system usually consists of the following modules:

visual SLAM Pipeline:
① tracking camera
② 3D Reconstruction
③ Place Recognition and loop closure.

- Camera tracking
- 3D reconstruction
- Place recognition and loop closure when a zone is revisited

revisited

One of the earliest and most influential works on real time monocular SLAM was presented by Davison [9]. It is a visual feature-based SLAM system, where the camera motion and position of landmarks of the environment are estimated using an Extended Kalman Filter (EKF) from the image displacement of

*D. Gutierrez-Gomez and J. J. Guerrero are with Instituto de Investigacion en Ingenieria de Aragon and Departamento de Informatica e Ingenieria de Sistemas, Universidad de Zaragoza, Maria de Luna 1, 500018, Zaragoza (Spain). e-mail: .

†This work has been supported by projects

[1] https://github.com/dangut/RGBiD-SLAM

the image projections, or features, of such landmarks. Following the success of this work, several contributions introduced new improvements and functionalities like the inverse depth parametrisation of the landmarks [8], a relocalisation module [44], and efficient scheme for data association and outlier rejection [7] or a generalised projection model to support catadioptric sensors [16].

One weakness of this approach is that following a probabilistic filter paradigm, camera tracking and mapping are performed both at frame rate, which limits severally the number of landmarks which could be reconstructed. Klein and Murray [24] addressed this issue with their PTAM (Parallel Tracking and Mapping), where camera tracking and mapping are separated into two different threads. The tracking thread operates at frame rate and is on charge of estimating the camera position given a fixed map; while the mapping thread operates at a lower rate, by building or updating a map by performing bundle adjustment between a set of keyframes selected from the total number of frames. Mur-Artal et al. presented in [28] a visual SLAM approach addressing the weaknesses of PTAM, like handling of landmarks occlusions or mapping scalability, as well as including a novel loop closure and relocalisation method based on Bags of Binary Words using ORB descriptors.

Compared to feature-based methods like the aforementioned direct methods have the advantage of skipping the feature extraction step which performed only an a small fraction of the image pixels and is frequently based on heuristics. However feature-less or direct monocular SLAM methods are more challenging due to the lack of depth measurements from vision sensors and the high computational burden of performing per-pixel operations.

Maybe the earliest and most influencing work in real-time dense monocular method was DTAM (Dense Tracking and Mapping) presented by Newcombe et al. [32]. The authors propose to reconstruct the depth of the scene in keyframes and at the same time estimate the pose of the camera. The problem of estimating a dense depth map from just RGB images is ill-posed and thus authors used an optimisation framework based on variational methods and implemented on GPU. In [13], Engel et al. propose a direct monocular SLAM method which obtains a semi-dense mapping based on variable baseline matching.

Nevertheless, in the last years visual SLAM has been very influenced by two differential events which have rendered direct feature-free approaches more tractable. First the development of a new generation of GPUs and CPUs, which allowed to design algorithms with high parallelisation; and secondly the advent of depth sensors, which are able to provide a dense depth image of the observed environment. The earlier prompted the development of real-time dense visual SLAM methods, while the later enabled SLAM, both feature-based and direct, using RGB and depth channels from a RGB-D camera.

Though RGB-D cameras have the great advantage of providing dense depth information, with respect to RGB cameras they present a more complex calibration. To perform localisation and mapping with RGB, one only needs to calibrate a conventional camera which nowadays is a relatively simple process by using for example Bouguet calibration toolbox [3]. On the other hand, RGB-D sensors require a stereo calibration between the RGB and depth cameras for depth registration. In addition to that, since the depth sensor is really measuring a pixel disparity and then converts it to depth, this relation has to be calibrated also. RGB-D sensors have this functions hard-coded in the hardware, being able depth computation and registration internally. However the use of a fixed factory calibration does not account for imperfections of particular sensors and a customised calibration might be necessary to reach good accuracy in SLAM with RGB-D sensors.

In this paper we present a complete SLAM system for custom calibrated RGB-D sensors. The developed camera tracking module builds upon our dense RGB-D odometry algorithm [17] which presented the novelty of minimising a geometric error parametrised in inverse depth as it fits better the error model of the depth sensor noise. Relying only on the camera tracking for localisation is prone to produce large deformations in the trajectory and the reconstructed map of the environment as the incremental estimate of the odometry drifts. To correct this we incorporate a place recognition and loop closure module which is based in a Bag of Words scheme using ORB de-

scriptors [29]. For the 3D reconstruction of the environment we use a keyframe-based approach where spatially close frames are integrated in one single keyframe to reduce the depth sensor noise. To build the 3D model the point cloud of each new keyframes are concatenated to the map point cloud, avoiding the addition of redundant points. Compared to volumetric approaches, 3D mapping on keyframes is more memory efficient as well as allows to better handle the uncertainty in the depth measures during the frame fusion process. In the experiments we show that our method obtains very accurate results in the trajectory estimation compared against a large selection of the most successful state of the art methods in RGB-D mapping.

Additionally, we present a procedure to completely calibrate RGB-D sensor. The main feature of the proposed method compared to most works in the literature is that in order to calibrate the conversion from the measured depth to the real depth we do not rely in any estimate obtained from the depth camera to generate the ground truth depth. The code has been made available in a public repository for research purposes.

The paper is divided as follows. In Section 2 we make a revision of the existing method on RGB-D SLAM and odometry, as well as RGB-D calibration. Section 3 is dedicated to explain some preliminary concepts of the mathematical representation of rigid body motions. In Section 4, we present the models and the pipeline followed for RGB-D camera calibration. Section 5 presents the different image warping approaches that are used in our SLAM approach. Section 6 explains our algorithm for dense frame alignment required to compute the inter-camera motion and Section 7 describes the complete SLAM system with its different modules. In sections 8 and 9 we present the experiments and the final conclusions.

2 Related Work

Maybe the best example of a feature-based RGB-D SLAM system is the approach proposed by Endres *et al.* [12]. To estimate the camera motion they first compute an initial seed by RANSAC-based 3D alignment of sparse features, followed by a refinement step where they only minimise the point cloud alignment error from the ICP algorithm. The method is improved in [11], including an Environment Measurement Model to prune wrong motion estimates which passed undetected in the RANSAC and ICP steps. The system is able to close loops taking a random sample from a set of keyframes and each frame-to-frame motion calculation between candidates for loop closure is parallelised, thus the computational cost is close to the frame rate of the camera, between 5 and 15 Hz.

KinectFusion by Newcombe *et al.* [31] was a ground-breaking contribution in dense RGB-D SLAM. KinectFusion is composed by two different modules, one for camera tracking and one for dense volumetric mapping. For each new frame first the motion is estimated by frame-to-model ICP alignment of depth maps, *i.e.*, current depth map is aligned with the depth map raycasted from a voxelized 3D model. Then, current depth map is integrated in the 3D model using a truncated signed distance function.

The main constraint of KinectFusion is its limitation to small workspaces, which was nevertheless solved in latter works by using a cyclical buffer to shift the volume as the camera explores the environment [1], [41]. However, artefacts are likely to appear in the reconstructed 3D model when an area is revisited. This is solved in recent works by Whelan *et al.* [42, 43] where a loop closure back-end is introduced. This back-end is able to correct potential artefacts in the 3D volume by enforcing the loop constraints not only on the camera trajectory but also on the dense 3D map using a deformation graph.

Dong *et al.* [10] propose the combination of KinectFusion with a robust odometry algorithm based on keypoint matching and RANSAC alignment between frames. This step provides an initial guess of the sensor pose which is further refined by the ICP step of KinectFusion.

Bylow *et al.* [5] proposed a method which, as KinectFusion, uses only the depth fusion, but rather than raycasting a depth map from the model for posterior ICP- alignment, the camera is tracked by directly minimising the signed distance function be-

tween the current warped depth map and the model surface defined in the voxelised volume. This results in a better accuracy and similar real-time computational performance compared to KinectFusion.

Jaimez and González-Jiménez [21] develop a fast visual odometry method using only depth images. Their method performs frame-to-frame tracking at a camera frame rate of 60Hz and it can be extended to any kind of range sensor.

Following the paradigm of working on 3D models [31], Stuckler and Behnke propose in [36] and [37] converting the RGB and depth images into multi-resolution surfel maps by using a voxel octree representation. Each surfel maintains a shape-texture descriptor, which guide data association between surfels in different maps during camera pose estimation. To alleviate the odometry drift they register the current frame with respect to the latest keyframe. A new keyframe is inserted when camera motion w.r.t. last keyframe is large enough. They also propose a loop closure technique where loop closure candidates are randomly sampled from a probability density function which positively weights the selection of spatially closer keyframes.

Kerl *et al.* [23] propose an RGB-D SLAM system where camera motion is estimated with respect to keyframes, which are switched following an entropy-based criteria. They include of a simple but effective loop closure method based on keyframes spatial proximity to further refine the final odometry estimation. The final 3D is obtained by simply joining all the raw depth maps from the keyframes.

Regarding the 3D reconstruction, though RGB-D sensors already provide a dense depth map of the scene, raw depth maps direct from the sensor show a large noise which grows quadratically with the distance of the observed point. For this reason, in order to obtain an accurate dense 3D reconstruction of the environment one needs to fuse multiple consecutive depth maps. As mentioned earlier in this section approaches like KinectFusion, integrate each of the incoming frames in a 3D volume of voxels.

Contrary to volume-based approaches, Meilland and Comport [27], integrate the depth map of each received frame in a set of close keyframes. To track every new frame they use the integrated keyframes to synthesise a reference keyframe from which the new frame is tracked. Though not reporting the use of any loop closure technique their method shows a compelling accuracy in terms of Absolute Trajectory Error, keeping a low trajectory drift.

When comparing both approaches for 3D mapping, volumetric fusion can be viewed as the integration of every incoming frame within a set of concatenated 2D slices, with each slice corresponding to a quantised depth value. This representation in many depth slices can handle occlusions; however it demands large amounts of memory and also, in the process of quantisation and integration of depth measures in the volume, important information about the depth uncertainty is lost.

In keyframe fusion in turn, integration is performed in a single slice represented by the depth map of the frame at which the keyframe was initialised. The main benefits are two. First, a more lightweight and efficient memory representation and secondly, since integration is performed in the frame of the original image space without discretising the depth values, depth uncertainty can be rigorously handled. The main disadvantage of this representation is that, with a single slice, occlusions between elements in the scene cannot be handled. In our keyframe-based approach, since we generate new keyframes periodically this is not a problem. Also it is worth remarking that a keyframe representation comes on handy when performing applying state of the art appearance-based loop closure strategies in visual SLAM, since it provides a direct mapping of the 3D scene to the RGB image of the keyframe. Furthermore, in our keyframe integration scheme, as in our previous work on direct RGB-D odometry, we take advantage of the usage of inverse depth maps instead of depth maps, which allows us to use a constant uncertainty-based tolerance for the integration of a new inverse depth value.

Having a custom calibration for one specific camera is key step to achieve a good precision in SLAM. For this reason another objective of this paper is to obtain a RGB-D SLAM system which can use a custom calibration, as well as providing a calibration pipeline to obtain an accurate calibration. Note that the calibration of an RGB-D sensor is more complex than for RGB ones since it requires first the spatial trans-

formation between the RGB and depth cameras, and secondly a function to compute the depth from the raw disparity maps computed by the camera. In the literature, Smisek *et al.* [35] perform a stereo calibration of RGB and depth cameras, using the infra-red channel of the later. For the correction of a spatial depth distortion they use images of a flat surface, and correct the error wrt. the best fitting plane. Herrera *et al.* [18], propose a similar approach, but instead they use the depth image for the stereo calibration and perform the distortion correction in the disparity space rather than depth. Di Cicco *et al.* [6] propose a method for depth distortion correction where some images of a wall are taken. Then a plane is fitted to the pixels in the centre of the images, under the assumption that these pixels suffer less distortion. Finally a pattern of depth multipliers for distortion correction is obtained by regression. Unsupervised approaches for the depth camera intrinsic calibration are proposed by Zhou and Koltun [46] and Teichman *et al.* [39], where the patterns for depth undistortion are computed taking a ground truth trajectory and map built from a SLAM algorithm. In general, the described approaches tend to use the depth images we want to undistort to generate the training examples to compute the correction parameters for those distorted images. In [45], Zeisl and Pollefeys are aware of this issue and propose an auto-calibration method, where a ground truth of camera poses and sparse map points is generated by a Structure from Motion algorithm using solely RGB images. In this work we follow the indication in [45] of generating the ground truth for depth undistortion from RGB data only. However in our case the calibration process is rather supervised by the user. Though it may be a slower and more complex process, we observe that accuracy is increased, since a supervised method allows first to introduce some measure of the scale such as some distance between points, and secondly to control the scene to be only a textured planar surface so that we can generate a dense depth image ground truth from camera poses computed by photogrammetry and the plane constraint.

3 Preliminaries: Rigid Body Transformations

This section introduces some concepts of the theory of rigid body transformations which are widely used in this work. For a more complete explanation we refer the reader to [30]. The motion of a rigid body in the 3D space is described by a rotation and a translation with respect to a reference frame. It is usually denoted by a 4×4 matrix:

$$ {}_W\mathbf{T}^C = \begin{pmatrix} {}_W\mathbf{R}^C & \mathbf{t}_W^C \\ \mathbf{0}^T & 1 \end{pmatrix}, \tag{1} $$

which reads as the translation and rotation of reference frame C, which is attached to the body, wrt. another reference frame W. ${}_W\mathbf{R}^C \in \mathbb{SO}(3)$ and $\mathbf{t}_W^C \in \mathbb{R}^3$ are respectively the rotation matrix and the translation vector. The rigid body transformations belong then to the product space of $\mathbb{R}^3$ and $\mathbb{SO}(3)$ and is denoted as the Special Euclidean Group, $\mathbb{SE}(3) = \mathbb{R}^3 \times \mathbb{SO}(3)$.

The group $\mathbb{SO}(3)$ induces an over-parametrisation of the rotations by 3x3 matrices with only 3 DoF, which is a source of problems in optimisation or handling rotation uncertainty. To alleviate this it is usual to switch between a matrix and an axis-angle representation consisting on a 3D vector, $\boldsymbol{\theta} \in \mathbb{R}^3$. This representation is also denoted as the Lie Algebra $\mathfrak{so}(3)$ of $\mathbb{SO}(3)$. The exponential map $\exp : \mathbb{R}^3 \to \mathbb{SO}(3)$ is used to convert a rotation from axis-angle into a matrix representation:

$$ \exp_{\mathbb{SO}(3)}(\boldsymbol{\theta}) = \exp([\boldsymbol{\theta}]_\times) = \mathbf{I} + \frac{\sin||\boldsymbol{\theta}||}{||\boldsymbol{\theta}||}[\boldsymbol{\theta}]_\times + \frac{1-\cos||\boldsymbol{\theta}||}{||\boldsymbol{\theta}||}[\boldsymbol{\theta}]_\times^2 \tag{2} $$

where $[\cdot]_\times$ denotes the skew symmetric 3x3 matrix built from a 3D vector. A rotation matrix is converted to the axis-angle representation through the logarithmic map $\log : \mathbb{SO}(3) \to \mathbb{R}^3$

$$ \log_{\mathbb{SO}(3)}(\mathbf{R}) = \frac{||\boldsymbol{\theta}||\,(\mathbf{R}-\mathbf{R}^T)}{2\sin||\boldsymbol{\theta}||} \tag{3} $$

$$ ||\boldsymbol{\theta}|| = \arccos\left(\frac{\mathrm{tr}(\mathbf{R})-1}{2}\right) \tag{4} $$

The notion of exponential and logarithmic map can also be defined in the Group $\mathbb{SE}(3)$, where an element

in its Lie algebra $\mathfrak{se}(3)$ is given by an screw motion $\boldsymbol{\xi} = (\boldsymbol{v},\ \boldsymbol{\theta})^T$, with 6 DoF (3 for rotation $\boldsymbol{\theta}$, and 3 for translation $\boldsymbol{v}$) :

$$\exp_{\mathbb{SE}(3)}(\boldsymbol{\xi}) = \exp \begin{pmatrix} [\boldsymbol{\theta}]_\times & \boldsymbol{v} \\ \mathbf{0}^T & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} & \mathbf{Q}(\boldsymbol{\theta})\boldsymbol{v} \\ \mathbf{0}^T & 1 \end{pmatrix} \quad (5)$$

$$\log_{\mathbb{SE}(3)}(\mathbf{T}) = \begin{pmatrix} \mathbf{Q}(\boldsymbol{\theta})^{-1}\mathbf{t} \\ \log(\mathbf{R}) \end{pmatrix} \quad (6)$$

where $\mathbf{Q} = \int_0^1 \exp(\tau\boldsymbol{\theta}))d\tau$ and yielding $\mathbf{t} = \mathbf{Q}(\boldsymbol{\theta})\boldsymbol{v}$. To avoid the rotation having an effect on the translation, in this paper we redefine the exponential and logarithmic maps to act separately in the rotation and translation, that is:

$$\exp_{\mathbb{SE}(3)}(\boldsymbol{\xi}) = \begin{pmatrix} \exp([\boldsymbol{\theta}]_\times) & \boldsymbol{v} \\ \mathbf{0}^T & 1 \end{pmatrix} \quad (7)$$

$$\log_{\mathbb{SE}(3)}(\mathbf{T}) = \begin{pmatrix} \mathbf{t} \\ \log(\mathbf{R}) \end{pmatrix} \quad (8)$$

To simplify the notation, from here in advance exponential and logarithmic functions appearing without the group sub-index will refer to the map of the defined above which matches with the function arguments.

4 Camera model and calibration

A RGB-D sensor is composed by one RGB and one infra-red (IR) camera, as well as a infra-red emitter which projects a dense infra-red 2-dimensional pattern. The projected IR pattern is perceived by the IR camera, and by matching it to a reference pattern obtained at a known depth, a dense disparity map can be computed in a similar way as conventional stereo systems can do sparsely in parts of the image with high intensity gradient. A RGB-D sensor can provide RGB, IR and depth images, where the IR and depth images share the same reference frame and RGB and IR cameras are separated and thus the corresponding images have different coordinate systems. RGB-D cameras include a built-in calibration in their hardware which allows for fast registration of the depth image to the RGB frame. However, this factory calibration does not account for possible imperfections for a particular camera, and thus a custom calibration is necessary to achieve better projection and registration model for a RGB-D sensor.

4.1 RGB and IR intrinsics

Both cameras can be modelled by a typical pinhole camera model with the addition of radial and tangential distortion correction [4]. A scene point $\mathbf{X} = (X,\ Y,\ Z)^T$ is first projected on a normalised plane obtaining the undistorted coordinates $\mathbf{m}_u = (X/Z,\ Y/Z,\ 1)^T$. Then the distortion correction is applied:

$$\mathbf{m}_d = \mathbf{h}(\mathbf{m}_u) = (1 + k_1 r_u^2 + k_2 r_u^4 + k_5 r_u^6)\mathbf{m}_u + \\ + \begin{pmatrix} 2k_3 m_{u,x} m_{u,y} + k_4(r_u^2 + 2m_{u,x}^2) \\ 2k_4 m_{u,x} m_{u,y} + k_3(r_u^2 + 2m_{u,y}^2) \\ -(k_1 r_u^2 + k_2 r_u^4 + k_5 r_u^6) \end{pmatrix} \quad (9)$$

where $r_u = \sqrt{m_{u,x}^2 + m_{u,y}^2}$. The vector $\mathbf{k} = (k_1,\ k_2,\ k_3,\ k_4,\ k_5)^T$ encapsulates the distortion parameters. This step is followed by a linear projection from the normalised plane in the final image plane:

$$\mathbf{p} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \mathbf{m}_d \quad (10)$$

where $\mathbf{K}$ is the pinhole model calibration matrix including the focal length $\mathbf{f} = (f_x,\ f_y)^T$ and the principal point $\mathbf{c} = (c_x,\ c_y,\ 1)^T$. All the previous steps can be encapsulated in a projectivity function $\mathbf{p} = \boldsymbol{\pi}(\mathbf{X})$. Image points can be also lifted to the 3D world with the inverse model:

$$\mathbf{X} = \boldsymbol{\pi}^{-1}(\mathbf{p}) = \mathbf{h}^{-1}(\mathbf{K}^{-1}\mathbf{p}). \quad (11)$$

Note that in this last equation, for $\mathbf{X}$ arbitrarily $Z = 1$ since depth cannot be recovered from one single

projection. It is necessary to know the depth $\mathcal{Z}(\mathbf{p})$ in a pixel to scale $\mathbf{X}$ and obtain a full 3D point.

The depth camera provides a depth image which is obtained from the disparity map using a generic conversion function. Thus, additional calibration parameters are needed to improve the depth map accuracy for a specific camera. We apply the following model to the inverted raw depth map $\mathcal{W}_m = \frac{1}{\mathcal{Z}_m}$ measured by the camera :

$$\mathcal{W}_d(\mathbf{p}) = \beta_1 \mathcal{W}_m(\mathbf{p} - \mathbf{p}_0) + \beta_0 \tag{12}$$

where β_0 and β_1 are the calibration parameters, and $\mathbf{p}_0 = (4,\ 4,\ 0)^T$ is a pixel shift caused by the correlation window used internally by the depth sensor to compute the disparity [25]. The purpose of this function is to correct the the disparity-to-depth conversion computed internally by the sensor using factory constants.

However, the depth sensor introduces also a spatially dependent error. To correct this distortion we apply an undistortion function as follows:

$$\mathcal{W}_u(\mathbf{p}) = \mathcal{D}_1(\mathbf{p})\mathcal{W}_d(\mathbf{p}) + \mathcal{D}_0(\mathbf{p}), \tag{13}$$

where $\mathcal{D}_0(\mathbf{p})$ and $\mathcal{D}_1(\mathbf{p})$ are spatially varying offset and multiplying factor respectively. Each of these terms is calibrated with 8 parameters which are the coefficients of a polynomial in the 2D image coordinates:

$$\begin{aligned}\mathcal{D}_i(\mathbf{p}) = q_{i,0} + q_{i,1}r^2 + q_{i,2}r^4 + q_{i,3}r^6 + q_{i,4}m_x + q_{i,5}m_y + \\ + q_{i,6}m_x m_y + q_{i,7}m_x^2 m_y + q_{i,8}m_x m_y^2,\end{aligned} \tag{14}$$

where $\mathbf{m} = \mathbf{K}^{-1}\mathbf{p}$ and $r = \sqrt{m_x^2 + m_y^2}$.

4.2 RGB and depth cameras extrinsics

The relative pose between the color and the depth camera frames is denoted by the spatial transformation matrix ${}_D\mathbf{T}^C = \{{}_D\mathbf{R}^C, \mathbf{t}_D^C\}$. It is required to register the inverse depth image in the color camera frame.

4.3 Calibration pipeline

The calibration of the RGB-D camera is performed in two steps. The first step consists in a conventional calibration of the IR and RGB cameras intrinsics and stereo transformation using the Bouguet calibration Toolbox [3]. With this procedure we obtain the RGB and depth intrinsics $\mathbf{f}_{\{C,D\}}$, $\mathbf{c}_{\{C,D\}}$, $\mathbf{k}_{\{C,D\}}$ and the spatial transformation ${}_D\mathbf{T}^C$ between the RGB and depth camera frames.

The second step in the calibration is the computation of the depth intrinsics for the correction of the raw inverse depth maps. For this step, we need M synchronised depth and RGB shots taken at different distances from a wall filling the whole image with a checker-board pattern and some posters attached to it. The real inverse depth map is computed by photogrammetry from the RGB images. After corner extraction, image matching and applying bundle adjustment techniques we obtain the pose ${}_W\mathbf{T}_i^C$ of each of the M RGB frames and a 3D sparse point cloud. Since the dimensions of the checker-board pattern are known, the estimated variables are in the true scale. The plane $\mathbf{\Pi} = (\mathbf{n}_W, d)$ which best fits the point cloud is also computed.

With the poses ${}_W\mathbf{T}_i^C$, and the stereo camera intrinsics from the previous step all the pixels in the inverse depth image can be lifted and represented in the global reference frame:

$$\begin{aligned}\mathbf{X}_W^i(\mathbf{p}) &= {}_W\mathbf{R}_i^C\, {}_C\mathbf{R}^D \frac{1}{\mathcal{W}(\mathbf{p})}\boldsymbol{\pi}_D^{-1}(\mathbf{p}) + {}_W\mathbf{R}_i^C\mathbf{t}_C^D + \mathbf{t}_{W,i}^C \\ &= {}_W\mathbf{R}_i^D \frac{1}{\mathcal{W}(\mathbf{p})}\boldsymbol{\pi}_D^{-1}(\mathbf{p}) + \mathbf{t}_{W,i}^D,\end{aligned} \tag{15}$$

And by applying the restriction of all the image points being contained in a plane $\mathbf{n}_W^T\mathbf{X}_W^i(\mathbf{p}) + d = 0$, we can compute a ground truth for the inverse depth images:

$$\mathcal{W}_{GT,i}(\mathbf{p}) = -\frac{\mathbf{n}_W^T\, {}_W\mathbf{R}_i^D\boldsymbol{\pi}_D^{-1}(\mathbf{p})}{\mathbf{n}_W^T\mathbf{t}_{W,i}^D + d} \tag{16}$$

Then, the intrinsic depth correction parameters are computed in two steps. In the first step we compute only the parameters of the linear relation:

wrapping function

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{M} \sum_{\substack{\mathbf{p}\in\Omega \\ \mathbf{p}-\mathbf{p}_0\in\Omega}} ||\mathcal{W}_{GT,i}(\mathbf{p}) - (\beta_1 \mathcal{W}_{m,i}(\mathbf{p}-\mathbf{p}_0)+\beta_0)||^2 \tag{17}$$

In the second step after applying (12) we compute the coefficients of the distortion polynomials:

$$\underset{\boldsymbol{q}_0,\boldsymbol{q}_1}{\operatorname{argmin}} \sum_{i=1}^{M} \sum_{\mathbf{p}\in\Omega} ||\mathcal{W}_{GT,i}(\mathbf{p}) - (\mathcal{D}_1(\mathbf{p})\mathcal{W}_{d,i}(\mathbf{p})+\mathcal{D}_0(\mathbf{p}))||^2 \tag{18}$$

The reason of optimising in two separate steps is obtaining a calibration where we can choose whether use or not use the spatial distortion correction.

5 Warping functions

A key element in direct methods for visual slam is the image warping functions. These are need for distortion correction or to obtain representations of the scene in some image as viewed from another different pose. Depending on what we need to achieve we use different warping approaches.

5.1 Inverse warping

Given a source image $\mathcal{I}_A$ and a destination image $\mathcal{I}_B$, inverse warping consists into mapping every pixel $\mathbf{p} \in \Omega$ in the destination image to a position in the source image, and then obtain the corresponding value by bilinear interpolation at the source:

$$\mathcal{I}_B(\mathbf{p}) = \mathcal{I}_A(\mathbf{f}_w(\mathbf{p})) \tag{19}$$

where the $\mathbf{f}_w$ is the warping function. We use inverse warping for undistortion of intensity and inverse depth maps, where $\mathbf{f}_w(\mathbf{p}) = \mathbf{K}\mathbf{h}(\mathbf{K}^{-1}\mathbf{p})$.

5.2 Forward registration

Forward registration is required when we need to represent an inverse depth map in another reference frame, like for example the one of the RGB camera. Generically, given the spatial transformation ${}_A\mathbf{T}^B$ and assuming that images have been undistorted, a pixel $\mathbf{p}$ in a depth map $\mathcal{W}_A$ at source frame A is mapped to a pixel $\mathbf{p}_B$ in a destination frame B through the following equation:

$$\frac{1}{w_B}\mathbf{p}_B = {}_B\tilde{\mathbf{R}}^A \left(\frac{1}{\mathcal{W}_A(\mathbf{p})}\mathbf{p} - \tilde{\mathbf{t}}_A^B \right), \tag{20}$$

where ${}_B\tilde{\mathbf{R}}^A = \mathbf{K}_B\, {}_B\mathbf{R}^A \mathbf{K}_A^{-1}$ and $\tilde{\mathbf{t}}_A^B = \mathbf{K}_A \mathbf{t}_A^B$.

A naive approach would be to compute $\mathbf{p}_B$, round to the nearest pixel location and then update the inverse depth value at this location. Unfortunately this approach is liable to produce holes in the new registered depth map. Instead we use a solution based on the relief texture mapping algorithm [33], performing the registration in two steps.

In the first step the depth image is registered to an intermediate frame $B_\mathbf{t}$ applying only the shift due to translation:

$$\frac{1}{w_{B_\mathbf{t}}}\mathbf{p}_{B_\mathbf{t}} = \frac{1}{\mathcal{W}_A(\mathbf{p})}\mathbf{p} - \tilde{\mathbf{t}}_A^B. \tag{21}$$

From this equation we obtain:

$$w_{B_\mathbf{t}} = \frac{\mathcal{W}_A(\mathbf{p})}{1 - \mathcal{W}_A(\mathbf{p})\mathbf{e}_\mathbf{z}^T \tilde{\mathbf{t}}_A^B} \tag{22}$$

$$\mathbf{p}_{B_\mathbf{t}} = \frac{\mathbf{p} - \mathcal{W}_A(\mathbf{p})\tilde{\mathbf{t}}_A^B}{1 - \mathcal{W}_A(\mathbf{p})\mathbf{e}_\mathbf{z}^T \tilde{\mathbf{t}}_A^B} \tag{23}$$

to prevent the arising of holes we map the whole pixel surface to the frame $B_\mathbf{t}$. Adding $\boldsymbol{\delta}\mathbf{p} = (\delta p_x,\ \delta p_y,\ 0)$ to $\mathbf{p}$ in (21) and assuming that $\mathcal{W}_A(\mathbf{p}) = \mathcal{W}_A(\mathbf{p}+\boldsymbol{\delta}\mathbf{p})$ $\forall\, ||\boldsymbol{\delta}\mathbf{p}||_\infty \leq 0.5$, we get:

$$\boldsymbol{\delta}\mathbf{p}_{B_\mathbf{t}} = \frac{w_{B_\mathbf{t}}}{\mathcal{W}_A(\mathbf{p})}\boldsymbol{\delta}\mathbf{p}. \tag{24}$$

Taking the four pixel corners at frame B at $\delta p_x \pm 0.5$ and $\delta p_y \pm 0.5$ and rounding the corresponding $\mathbf{p}_{B_\mathbf{t}} + \boldsymbol{\delta}\mathbf{p}_{B_\mathbf{t}}$ to the nearest integer, we obtain a square containing all the pixels where $w_{B_\mathbf{t}}$ should be mapped at frame $B_\mathbf{t}$. For every pixel $\mathbf{p}_s$ in this square $\mathcal{W}_{B_\mathbf{t}}(\mathbf{p}_s)$ is updated if $w_{B_\mathbf{t}}$ is the greatest or the first inverse depth value computed in that pixel.

In the second step, the remaining rotation is applied to the warped frame. Noting that the ${}_B\tilde{\mathbf{R}}^A$

CNNSVO - RGBiD. SLAM

represents an homography, and thus a linear mapping, inverse warping can be applied taking pixels in the destination frame B:

$$\frac{1}{\mathcal{W}_B(\mathbf{p})}\mathbf{p} = {}_B\tilde{\mathbf{R}}^A \frac{1}{\mathcal{W}_{B_\mathbf{t}}(\mathbf{p}_{B_\mathbf{t}})}\mathbf{p}_{B_\mathbf{t}} \tag{25}$$

From this equation $\mathbf{p}_{B_\mathbf{t}} = \frac{{}_A\tilde{\mathbf{R}}^B\mathbf{p}}{\mathbf{e}_\mathbf{z}^T\,{}_A\tilde{\mathbf{R}}^B\mathbf{p}}$, with ${}_A\tilde{\mathbf{R}}^B = \left({}_B\tilde{\mathbf{R}}^A\right)^{-1}$. Then $\mathcal{W}_{B_\mathbf{t}}(\mathbf{p}_{\mathbf{B_t}})$ is obtained by nearest neighbour interpolation and finally the inverse depth value $\mathcal{W}_B(\mathbf{p}) = \mathcal{W}_{B_\mathbf{t}}(\mathbf{p}_{\mathbf{B_t}})\mathbf{e}_\mathbf{z}^T\,{}_A\tilde{\mathbf{R}}^B\mathbf{p}$ in the registered image.

5.3 Inverse geometric warping

If inverse depth and intensity maps are available at both frames A and B, inverse warping can be performed by using (20) as the warping function. Note that since we are performing inverse warping, the roles of the frames A and B in this equation is interchanged (A is the destination, and B is the source). This is computationally faster than forward warping and is useful, for example, to fuse inverse depth maps taken at different poses. The drawback is that occlusions cannot be handled and thus the motion between frames must be small. Given that the pixel position $\mathbf{p}_B$ computed by (20), the images at frame $B*$ warped towards frame A are computed by the following relations:

$$\mathcal{I}_{B*}(\mathbf{p}) = \mathcal{I}_B(\mathbf{p}_B) \tag{26}$$

$$\mathcal{W}_{B*}(\mathbf{p}) = \frac{\mathbf{e}_\mathbf{z}^T\,{}_B\tilde{\mathbf{R}}^A\mathbf{p}}{1-\mathcal{W}_B(\mathbf{p}_B)\,\mathbf{e}_\mathbf{z}^T\tilde{\mathbf{t}}_B^A}\mathcal{W}_B(\mathbf{p}_B) \tag{27}$$

5.4 GPU implementation

All the warping functions have been implemented for CUDA capable NVIDIA GPUs. In the inverse warping functions, the mapping of source images to the GPU texture memory allows for efficient interpolation via texture fetching. In the case of the translation-only warping step in the forward registration procedure, a pixel in the destination images might be accessed at the same time by two or more GPU threads for a writing operation. In order to avoid race conditions and serialise the access for writing we use atomic operations.

6 Dense motion estimation

Dense frame alignment consists into estimating the camera motion ${}_A\hat{\mathbf{T}}^B = ({}_A\hat{\mathbf{R}}^B, \hat{\mathbf{t}}_A^B)$ between two frames A and B by pixel-wise minimisation of the photometric and inverse depth error between both frames.

6.1 Photometric and geometric constraints

Let us denote two camera frames as A and B, at instants t and $t+\Delta t$ respectively. Given the intensity images $\mathcal{I}_A$ and $\mathcal{I}_B$, and inverse depth maps $\mathcal{W}_A$ and $\mathcal{W}_B$ defined over the image domain $\Omega \subset \mathbb{P}^2$, for an image point $\mathbf{p} = (u\ v\ 1)^T \in \Omega$ in frame A, the following constraints hold:

$$\mathcal{I}_B(\mathbf{p}+\mathbf{\Delta p}) = \mathcal{I}_A(\mathbf{p}) \tag{28}$$

$$\mathcal{W}_B(\mathbf{p}+\mathbf{\Delta p}) = \frac{\mathcal{W}_A(\mathbf{p})}{1+\mathcal{W}_A(\mathbf{p})\mathbf{e}_\mathbf{z}^T\mathbf{\Delta X_p}}, \tag{29}$$

where $\mathbf{\Delta X_p}$ and $\mathbf{\Delta p}$ are respectively the scene and the optical flow of one point from frame A to B, and $\mathbf{e}_\mathbf{z}^T = (0\ \ 0\ \ 1)$. The constraint in intensity assumes constant illumination of one scene point. The second constraint models the change of one point's inverse depth due to the relative motion along the optical axis of the camera.

It can be verified that the scene and the optical flow satisfy the following equation:

$$\mathcal{W}_A(\mathbf{p})\left(\mathbf{K}-\mathbf{p}\mathbf{e}_\mathbf{z}^T\right)\mathbf{\Delta X_p} = (1+\mathcal{W}_A(\mathbf{p})\mathbf{e}_\mathbf{z}^T\mathbf{\Delta X_p})\mathbf{\Delta p} \tag{30}$$

With this relation and computing the first order Taylor expansion with respect to the scene flow we get the linearised constraints:

$$\mathcal{W}_A(\mathbf{p})\nabla\mathcal{I}_A(\mathbf{p})\left(\mathbf{K}-\mathbf{p}\mathbf{e}_\mathbf{z}^T\right)\boldsymbol{\Delta}\mathbf{X}_\mathbf{p}+\mathcal{I}_B(\mathbf{p})-\mathcal{I}_A(\mathbf{p})=0 \tag{31}$$

$$\mathcal{W}_A(\mathbf{p})\left(\nabla\mathcal{W}_A(\mathbf{p})\left(\mathbf{K}-\mathbf{p}\mathbf{e}_\mathbf{z}^T\right)+\mathcal{W}_B(\mathbf{p})\mathbf{e}_\mathbf{z}^T\right)\boldsymbol{\Delta}\mathbf{X}_\mathbf{p}+ +\mathcal{W}_B(\mathbf{p})-\mathcal{W}_A(\mathbf{p})=0. \tag{32}$$

Since we are estimating the motion between frames we are working under the assumption of a rigid scene. Assuming a small motion described by the rotation and translation pair $({}_A\mathbf{R}^B, \mathbf{t}_B^A) \in \mathbb{SE}(3)$ we have:

$$\begin{aligned}\boldsymbol{\Delta}\mathbf{X}_\mathbf{p} &= \left({}_B\mathbf{R}^A-\mathbf{I}\right)\mathbf{X}_A+\mathbf{t}_B^A\\ &= [\boldsymbol{\theta}_B^A]_\times\mathbf{X}_A+\mathbf{t}_B^A+\mathcal{O}\left(\left\|[\boldsymbol{\theta}_B^A]_\times^2\mathbf{X}_A\right\|\right)\\ &\approx \mathbf{t}_B^A-[\mathbf{X}_A]_\times\boldsymbol{\theta}_B^A=\mathbf{M}(\mathbf{p})\boldsymbol{\xi}_B^A.\end{aligned} \tag{33}$$

where $\mathbf{X}_A=\mathbf{K}^{-1}\mathbf{p}\frac{1}{\mathcal{W}_A(\mathbf{p})}$ and $\boldsymbol{\xi}_B^A=(\mathbf{t}_B^A,\ \boldsymbol{\theta}_B^A)^T$. $[\cdot]_\times$ denotes the antisymmetric matrix from a vector and $\boldsymbol{\theta}_B^A$ is the axis-angle representation of ${}_B\mathbf{R}^A$.

6.2 Iterative optimisation

The inter-frame motion is computed iteratively in a coarse-to-fine manner using image pyramids, performing a given number of iterations at each pyramid level. At the start and every time we step down in the image pyramid, we downsample $\{\mathcal{I}_A,\ \mathcal{W}_A\}$ and compute the gradients $\{\nabla\mathcal{I}_A,\ \nabla\mathcal{W}_A\}$ at current pyramid level. The initial estimate of the rigid motion is set to the identity, ${}_A\hat{\mathbf{T}}_{(0)}^B=\mathbf{I}$, unless an initial guess is provided.

At each iteration γ, first intensity and depth maps in frame B, $\{\mathcal{I}_B,\ \mathcal{W}_B\}$, are warped towards frame A taking the up-to-date motion estimate ${}_A\hat{\mathbf{T}}_{(\gamma)}^B$. By using the reverse warping approach described in Section 5.3 we obtain the warped images $\{\mathcal{I}_B^{(\gamma)},\ \mathcal{W}_B^{(\gamma)}\}$. These images are downsampled to the pyramid level at which current iteration is taking place. The formulae for the residuals at current iteration are given by:

$$\begin{aligned}r_\mathcal{I}(\mathbf{p},\boldsymbol{\xi}) &= \mathcal{W}_A(\mathbf{p})\nabla\mathcal{I}_A(\mathbf{p})(\mathbf{K}-\mathbf{p}\mathbf{e}_\mathbf{z}^T)\mathbf{M}(\mathbf{p})\boldsymbol{\xi}+\\ &\quad+\mathcal{I}_B^{(\gamma)}(\mathbf{p})-\mathcal{I}_A(\mathbf{p})\end{aligned} \tag{34}$$

$$\begin{aligned}r_\mathcal{W}(\mathbf{p},\boldsymbol{\xi}) &= \mathcal{W}_A(\mathbf{p})\left(\nabla\mathcal{W}_A(\mathbf{p})(\mathbf{K}-\mathbf{p}\mathbf{e}_\mathbf{z}^T)+\mathcal{W}_B^{(\gamma)}(\mathbf{p})\mathbf{e}_\mathbf{z}^T\right)\mathbf{M}(\mathbf{p})\boldsymbol{\xi}+\\ &\quad+\mathcal{W}_B^{(\gamma)}(\mathbf{p})-\mathcal{W}_A(\mathbf{p}),\end{aligned} \tag{35}$$

The next step is the computation of the update of the motion estimate between frames. This is achieved by optimisation with Iteratively Reweighted Least Squares algorithm (IRLS) [19], which results in a linear least-squares problem to be solved at each iteration:

$$\begin{aligned}\hat{\boldsymbol{\xi}}_{B^{(\gamma)}}^A = \underset{\boldsymbol{\xi}}{\operatorname{argmin}}\sum_{\mathbf{p}\in\Omega}&\omega\left(\frac{r_\mathcal{I}(\mathbf{p},\mathbf{0})-\mu_\mathcal{I}}{\sigma_\mathcal{I}}\right)\frac{r_\mathcal{I}^2(\mathbf{p},\boldsymbol{\xi})}{\sigma_\mathcal{I}^2}\\ &+\lambda_\mathbf{n}(\mathbf{p})\omega\left(\frac{r_\mathcal{W}(\mathbf{p},\mathbf{0})-\mu_\mathcal{W}}{\sigma_\mathcal{W}}\right)\frac{r_\mathcal{W}^2(\mathbf{p},\boldsymbol{\xi})}{\sigma_\mathcal{W}^2},\end{aligned} \tag{36}$$

The weighting function $\omega(x)$ depends on the assumed probability distribution of the residuals. $\mu_{\{\mathcal{I},\mathcal{W}\}}$ and $\sigma_{\{\mathcal{I},\mathcal{W}\}}$ are the location and scaling parameters which capture the bias and the uncertainty in intensity and inverse depth residuals, and allow for normalisation of residuals in different magnitudes. The details on the computation of the weighting function and the location and scaling parameters are presented in the next subsection 6.3. To account for noisier depth measurements from surfaces more oblique to the camera, $\lambda_\mathbf{n}(\mathbf{p})=\mathbf{n}^T\frac{\mathbf{p}}{||\mathbf{p}||}$ is used to down-weight the inverse depth residuals, where the normal $\mathbf{n}^T$ is computed as:

$$\mathbf{n}^T=\alpha\left(\frac{1}{\mathcal{W}_A(\mathbf{p})}\nabla\mathcal{W}_A(\mathbf{p})\left(\mathbf{K}-\mathbf{p}\mathbf{e}_\mathbf{z}^T\right)+\mathbf{e}_\mathbf{z}^T\right), \tag{37}$$

where α is the normalising constant.

The minimisation of (36) leads us to solve the following linear system:

$$\mathbf{H}^{(\gamma)}\hat{\boldsymbol{\xi}}_{B^{(\gamma)}}^A=\mathbf{b}^{(\gamma)}, \tag{38}$$

and after computing $\boldsymbol{\xi}_{B^{(\gamma)}}^A$ we update the inter-frame motion estimate for the next iteration:

$$ {}_A\hat{\mathbf{T}}^B_{(\gamma+1)} = \begin{pmatrix} \exp([\hat{\boldsymbol{\theta}}^A_{B(\gamma)}]_\times) & \mathbf{t}^A_{B(\gamma)} \\ 0 & 1 \end{pmatrix}^{-1} {}_A\hat{\mathbf{T}}^B_{(\gamma)}. \quad (39) $$

The covariance of the motion estimate is computed as the inverse of the Hessian. The Hessian depends on the intensity and depth maps gradients computed in the frame A. Due to sensor noise it is possible that some pixels provide misleading information making the Hessian larger, which would yield a wrongly optimistic covariance matrix. In order to prevent this, once we have computed the final iteration for the motion estimate, we estimate the Hessian by performing an extra iteration where the motion estimate is not updated and where $\{\mathcal{I}_A,\ \mathcal{W}_A\}$ are applied a bilateral filter prior to the computation of their gradients:

$$ \boldsymbol{\Sigma}^{A\to B}_A = \boldsymbol{\Sigma}^{B\to A}_A = \mathbf{H}^{-1}_{filt} \quad (40) $$

6.3 Error distribution and robust optimisation

The underlying assumption behind the IRLS method is that the residuals we want to minimise are modelled by a probability density function (pdf). This function is related to the weights $\omega(x)$ we apply in the cost function. The purpose of IRLS is that, by selecting a pdf with heavier tails than the typically assumed Gaussian distribution, gross errors can be accommodated and and thus we gain robustness to outliers. In this work we use the student t-distribution whose pdf is given by:

$$ f_{St}(r_i \mid \mu,\ \sigma,\ \nu) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)\sigma}\left(1+\frac{x_i^2}{\nu}\right)^{-\frac{\nu+1}{2}} \quad (41) $$

with $x_i = \frac{r_i-\mu}{\sigma}$, and where μ, σ are the location, and the scale of the pdf, while ν is a characteristic parameter of the Student's t-distribution and denotes the degrees of freedom. In robust estimation the parameter ν is associated to the achieved robustness. Taking the logarithm of (41) and eliminating constants we obtain the log-likelihood function:

$$ \begin{aligned} \log\mathcal{L}(\mu,\ \sigma,\ \nu \mid r_i) = {} & \ln\left(\Gamma\left(\tfrac{\nu+1}{2}\right)\right) - \ln\left(\Gamma\left(\tfrac{\nu}{2}\right)\right) \\ & - \ln(\sigma) + \tfrac{\nu}{2}\ln(\nu) - \underbrace{\tfrac{\nu+1}{2}\ln(\nu+x_i^2)}_{\rho(x_i,\ \nu)} \end{aligned} \quad (42) $$

The location and scaling parameters of the residuals are obtained by maximisation of their joint log-likelihood function. That is:

$$ \begin{aligned} [\mu,\ \sigma] &= \underset{\mu,\sigma}{\operatorname{argmax}} \sum_i \log\mathcal{L}(\mu,\ \sigma,\ \nu \mid r_i) \\ &= \underset{\mu,\sigma}{\operatorname{argmin}} \sum_i \ln(\sigma) + \rho(x_i,\nu) \end{aligned} \quad (43) $$

Taking the derivatives with respect to μ and σ we obtain respectively the following equations:

$$ \sum_i \omega(x_i,\ \nu)(r_i-\mu) = 0 \quad (44) $$

$$ \sum_i \left(1 - \omega(x_i,\ \nu)\frac{(r_i-\mu)^2}{\sigma^2}\right) = 0 \quad (45) $$

where $\omega(x,\ \nu) = \frac{1}{x}\frac{\partial\rho(x,\ \nu)}{\partial x}$. This system of equations can be solved iteratively using the previous estimates of μ and σ to compute $\omega(x_i,\ \nu)$ before each iteration.

Note that up to this point we have ignored the estimation of the parameter ν. Typically this parameter is set to $\nu = 5$ which provides enough robustness and also behaves well in case the residuals have a Gaussian distribution. However, ν can be estimated as well, which allows to adapt the required robustness to the distribution of current residuals. By maximising (42) with respect to ν [26] we obtain the following non-linear equation:

$$ \begin{aligned} \sum_i \Bigg(-\phi\left(\frac{\nu}{2}\right) + \ln\left(\frac{\nu}{2}\right) + \phi\left(\frac{\nu+1}{2}\right) - \ln\left(\frac{\nu+1}{2}\right) \\ + 1 + \ln(\omega(x_i,\nu)) - \omega(x_i,\nu)\Bigg) = 0, \end{aligned} \quad (46) $$

where $\phi(y)$ denotes the digamma function. Note that ideally robustness parameters should be estimated at

the same time as location and scaling. However, unlike location and scaling equations, fixing the weights in (46) with the previous estimate does not produce a linear system, requiring from some sub-iterations to compute ν within current iteration. Moreover fixing ν in the weights to its previous value produces a slow convergence.

However letting ν to be an unknown in the weights implies to perform a sub-loop of GPU reduction operations within the loop in which the distribution parameters are estimated. Therefore, to keep computational cost low we estimate first the final location and scaling values for intensity and depth residuals fixing $\nu = 5$, and then we solve (46) with the computed values using the bisection method assuming $\nu \in [2, 10]$ with a lower ν implying more robustness to outliers. This is performed both for the intensity and inverse depth residuals yielding $\nu_{\mathcal{I}}$ and $\nu_{\mathcal{W}}$,

In dense RGB-D tracking the main source of outliers are the high inverse depth residuals caused by occlusions which are due to dynamic objects or areas which simply become revealed or hidden by the camera motion, *i.e.*, outliers are caused by large inverse depth residuals rather than intensity ones. For this reason and because too much robustness can lead to a bad or slow convergence to the solution, we select the highest of the computed $\nu_{\mathcal{W}}$ and $\nu_{\mathcal{I}}$ for the intensity residuals.

$$\nu_{\mathcal{I}} = \max(\nu_{\mathcal{I}}, \nu_{\mathcal{W}}). \tag{47}$$

A naive computation of the location, scaling and robustness parameters would use all the residuals. Given that every residual in intensity and inverse depth corresponds to a single pixel, the number of samples would be extremely large, more than 300000 samples at the highest resolution level, which would involve a high computational cost. To reduce this burden we compute the scaling parameters in a sample with a maximum size of $N = 19200$ points obtained by systematic selection. This sample size guarantees first an integer stride for pixel sampling at all the considered resolutions; and secondly, after leading an statistical analysis, it also guarantees a relative precision of 0.03 with a confidence level of 99.7%.

6.4 Dense frame covisibility ratio

Computing the covisibility score between two given frames is a key element to prune redundant frames, selection of reference frames or keyframes, or computing visibility graphs. In this work, taking advantage of the availability of dense depth maps, we use a covisibility computed from all the pixels in the image. Given two frames A and B and the camera motion estimate between them $({}_B\mathbf{R}^A, \mathbf{t}_B^A) \in \mathbb{SE}(3)$, we transfer pixels from A to B using (20) and compute the following pixel subsets:

$$S_{\bar{H}} = \{\mathbf{p} \in \Omega \mid \mathcal{W}_A(\mathbf{p}) \neq \emptyset\}, \tag{48}$$

$$S_V = S_{\bar{H}} \cap \{\mathbf{p} \in \Omega \mid (\mathbf{p}_B \in \Omega) \wedge (|\mathcal{W}_B(\mathbf{p}_B) - w_B| < 3\sigma_{\mathcal{W}})\}. \tag{49}$$

$S_{\bar{H}}$ is the subset of pixels in frame A with valid inverse depth values (no-hole pixels) and S_V the set of visible pixels when transferred to frame B. A pixel is tagged visible if it is within the image bounds, and it is not occluded in frame B. A pixel is considered occluded if its mapped inverse depth value is above an uncertainty allowed by the sensor noise model. Then the covisibility ratio is computed:

$$vr_{AB} = \frac{|S_V|}{|S_{\bar{H}}|} \tag{50}$$

This procedure is repeated switching the role of A and B, and then we select the minimum ratio as the final covisibility ratio.

7 RGB-ID SLAM system

Our RGB-ID SLAM system is implemented in two CPU threads running concurrently. One thread executes a front-end for every incoming frame from the RGB-D sensor, performing camera tracking and fusion of inverse depth measurements in a single keyframe. For both of these tasks the CPU thread calls functions which execute in GPU where pixel-wise operations can be easily parallelised. The second thread executes a back-end in charge of managing the keyframes passed by the front-end through a buffer.

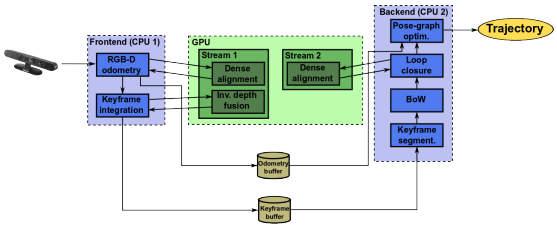

Figure 1: Scheme of our complete RGB-ID SLAM system

First it performs a segmentation of the keyframe in juts using the map of normals. Secondly for every keyframe it obtains different BoW histograms and stores the keyframe in a database. Finally it attempts to close loops comparing the last processed keyframe against the keyframes in the database. All the tasks of the back-end thread are performed in CPU except for a dense alignment in GPU between keyframes when a successful loop is detected. In order to allow for simultaneous access to GPU by both threads without blocking, each thread calls its GPU functions on its own CUDA stream which is executed asynchronously.

7.1 Camera tracking

For every upcoming frame $\{\mathcal{I}_k,\ \mathcal{W}_k\}$ we have to compute the rigid body transformation ${}_{rf}\hat{\mathbf{T}}^k$ which best aligns it with a given reference frame $\{\mathcal{I}_{rf},\ \mathcal{W}_{rf}\}$. This motion is computed by the dense RGB-D alignment method described in Section 6. We provide an initial guess of the motion estimate using a constant velocity model, *i.e.*, ${}_{rf}\hat{\mathbf{T}}^k_{(0)} = {}_{rf}\hat{\mathbf{T}}^{k-1}\,{}_{k-2}\hat{\mathbf{T}}^{k-1}$.

Once we compute the motion estimate between the reference and current frame ${}_{rf}\hat{\mathbf{T}}^k$ and its covariance $\mathbf{\Sigma}^{rf\to k}_{rf}$, we compute the sequential odometry constraint as it will be required in the latter pose-graph optimisation step.

$$ {}_{k-1}\mathbf{T}^k = \left({}_{rf}\mathbf{T}^{k-1}\right)^{-1}\,{}_{rf}\mathbf{T}^k \tag{51} $$

$$ \mathbf{\Sigma}^{k-1\to k}_{k-1} = \frac{\partial\,{}_{k-1}\mathbf{T}^k}{\partial\,{}_{rf}\mathbf{T}^{k-1}}\mathbf{\Sigma}^{rf\to k-1}_{rf}\left(\frac{\partial\,{}_{k-1}\mathbf{T}^k}{\partial\,{}_{rf}\mathbf{T}^{k-1}}\right)^T + $$
$$ + \frac{\partial\,{}_{k-1}\mathbf{T}^k}{\partial\,{}_{rf}\mathbf{T}^{k}}\mathbf{\Sigma}^{rf\to k}_{rf}\left(\frac{\partial\,{}_{k-1}\mathbf{T}^k}{\partial\,{}_{rf}\mathbf{T}^{k}}\right)^T \tag{52} $$

where the computation of the Jacobians of the transformations is detailed in the Appendix A.

7.2 Keyframe fusion

After the visual odometry, the inverse depth map of current frame is fused in the last selected keyframe. The criteria for keyframe selection is the same covisibility criteria as used for the reference frames taken for odometry. We must note that generally keyframes do not have to coincide with the odometry reference frames, since we allow for using a different covisibility threshold. Normally it will be set to a lower value for keyframe switching, *i.e.*, we will initialise new keyframes at a lower rate than reference frames, using a visibility threshold of 0.7. Once a keyframe is initialised, the inverse depth maps of the incoming frames are fused with the keyframe's. We not only fuse the incoming frames but also the frames recorded prior to the keyframe selection in order to integrate inverse depth measurements in parts of the image no longer observed after keyframe initialisation. To do so, frames are stored temporally in a buffer and erased once integrated in the later keyframe. Every time we integrate an incoming frame, we integrate also the frame closest in time of the ones remaining in the buffer.

Fusion in keyframes is performed in GPU using the reverse warping approach detailed in section 5.3, where the frame to be integrated is warped towards the keyframe. Note that since we are using a reverse warping scheme parts of the scene with missing depth measurements in the frame from which the keyframe is initialised cannot be reconstructed from subsequent frames. We find this a minor disadvantage given first, that reverse warping in CUDA is computationally cheap and easy to implement and secondly, that the

generation of many keyframes generally guarantees that zones missing in one keyframe can be observed in another one.

Once the frame has been warped we have to integrate every new inverse depth value for every pixel in the keyframe. To do this we first verify that the pixels on both the keyframe and the warped frame correspond to the same scene point by performing the same covisibility check as in Section 6.4. If a pixel passes this test we update its inverse depth value $\mathcal{W}_{kf}(\mathbf{p})$ as well as the corresponding weight $\mathcal{C}_{kf}(\mathbf{p})$ in the keyframe as follows:

$$\mathcal{W}_{kf}(\mathbf{p}) \leftarrow \frac{\mathcal{W}_{kf}(\mathbf{p})\mathcal{C}_{kf}(\mathbf{p}) + \mathcal{C}_k(\mathbf{p}_w)\mathcal{W}_k(\mathbf{p}_w)}{\mathcal{C}_{kf}(\mathbf{p}) + \mathcal{C}_k(\mathbf{p}_w)} \tag{53}$$

$$\mathcal{C}_{kf}(\mathbf{p}) \leftarrow \mathcal{C}_{kf}(\mathbf{p}) + \mathcal{C}_k(\mathbf{p}_w) \tag{54}$$

The weight $\mathcal{C}_k(\mathbf{p}_w) = \left(\frac{(1-\mathcal{W}_B(\mathbf{p}_w)\mathbf{e}_\mathbf{z}^T\tilde{\mathbf{t}}_B^A)^2}{\mathbf{e}_\mathbf{z}^T{}_B\tilde{\mathbf{R}}^A\mathbf{p}}\right)^2$ stems from the propagation of the inverse depth variance through (20). When a new keyframe is initialised the values of the weight map $\mathcal{C}_{kf}$ is reset to 1 and the last keyframe is stored in a buffer, awaiting to be processed by the back-end in the second CPU thread.

7.3 Keypoint extraction and BoW histograms

Loop detection primarily relies on the DBoW2 place recognition system based in Bags of Binary Words proposed by Gálvez-López and Tardós [15]. Mur-Artal et al. [29] used this approach with ORB descriptors obtaining a quite reliable and fast loop closing system. As the authors suggest, we compute a pyramid of 8 levels and a relative scale of 1.2 between levels from the intensity image, and then we extract FAST keypoints at each pyramid level. In order to distribute the keypoints uniformly over the image we divide the image in cells and set the firing threshold for the FAST extractor adaptively to extract the desired number of points at each cell. Once the keypoints are extracted, they are lifted to 3D points using the inverse depth map of the keyframe and ORB descriptors are computed in the intensity image. For the classification of the ORB descriptors in a BoW we use the same dictionary as in [29]. For each keyframe a BoW histogram is computed with the frequency of each word in the given image. Finally the processed keyframe is stored in a keyframe database from which keyframes are queried in the loop closure process.

7.4 Superjuts and entropy of normals

For every keyframe which is passed to the back-end we generate a map of the degree of disarrangement of the elements in the scene,based on the entropy of the normals. For a given keyframe, we generate a normal map from its inverse depth map. Then we perform an object-like segmentation based on the algorithm for RGB images developed by Felzenszwalb and Huttenlocher [14], and which is similar to the methods presented in [40] and [22]. The main difference is that we perform the segmentation directly on the keyframe pixels rather than on a lifted point cloud.

A graph $G = (V, E)$ is constructed, where each vertex in V represents a 3D point $\mathbf{X}_i$ lifted from a pixel location and the edges E represent the neighbouring relations between points. Edges are inserted between points with adjacent pixel locations on the image. Having the normals at every point, a weight w_{ij} for the edge joining vertices i and j is computed:

$$w_{ij} = \begin{cases} (1 - \mathbf{n}_i^T\mathbf{n}_j)^2 & \text{if } \mathbf{n}_j^T(\mathbf{X}_j - \mathbf{X}_i) > 0 \\ 1 - \mathbf{n}_i^T\mathbf{n}_j & \text{if } \mathbf{n}_j^T(\mathbf{X}_j - \mathbf{X}_i) \leq 0 \end{cases} \tag{55}$$

where the squared weight is applied to convex edges, reflecting the fact that convex regions usually contain points belonging to the same object and concave regions are likely to arise in frontiers between objects. After computing the weights, the segmentation algorithm is run and essentially groups points sharing edges with low weights in the same superjut. A parameter k must be tuned such that the higher k, the larger segments will be obtained. We set this value always to 0.6.

After the segmentation we compute a histogram of normals $F(\mathbf{n})$. Each of the M bins in the histogram corresponds to a Voronoi cell associated to

React.

one point out of M points uniformly distributed over the sphere. Since there is no analytical solution to the problem of evenly distributing M points on the sphere for any M, we use a simple approximate solution based on the golden section spiral [2, 34], which results in a discretisation with bins covering areas of similar size. Finally given the histogram of normals of a jut, we can compute its entropy:

$$ent(F) = \sum_{i=1}^{M} F(\mathbf{n}_i) \log(F(\mathbf{n}_i)). \tag{56}$$

7.5 Loop closure

After extracting keypoints and computing the BoW histogram the keyframe database is searched for potential keyframe candidates for loop closures. Given two keyframes, DBoW2 returns a similarity score based on the distance between their BoW histograms, which we normalise with the score between the currently querying keyframe and the previous keyframe. If this score is above a threshold for any of the histogram comparisons a potential loop candidate is stored. In order to ensure that potential loop closures are obtained between temporally distant keyframes we establish a minimum keyframe separation.

For every possible keyframe candidate we match its 3D points with the 3D points of current keyframe using their ORB descriptors. Then we perform a geometric validation of the detected loop by computing a rigid motion estimate applying the method described in [20] for alignment of 3D point clouds in a 3-point RANSAC scheme. If more than 10 points agree with the computed motion between keyframes, and the convex hull spanned by these points takes up more than 5% of the image, the loop is tagged as correct. In order to compute the loop constraint, containing both a motion estimate and its covariance, we align both keyframes using the approach described Sec. 6 where we pass the RANSAC rigid motion estimate as initial guess.

7.5.1 Pose-graph optimisation

When a new loop is detected it is added to a cluster containing loop constraints which are temporally close, allowing a maximum separation of 10 keyframes between any two of the keyframes in the cluster. Pose-graph optimisation is applied when one of the clusters is no longer accepting new loop constraints, updating the graph with the new constraints in the cluster, as well as the poses and odometry constraints computed by the front-end, since the last graph update. Then, given a graph with a set of odometry constraints, $\mathcal{S} = \{(1,2), ..., (N-1, N)\}$, and a set of loop constraints, $\mathcal{R} = \{(i_1, j_1), ..., (i_L, j_L)\}$, we want to find the trajectory $\mathbf{x} = \left\{ {}_W\mathbf{T}^1,\ {}_W\mathbf{T}^2,\ ...,\ {}_W\mathbf{T}^N \right\}$ which minimises the following cost function:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{(i,j)\in\mathcal{S}} \mathbf{e}_{ij}^T(\mathbf{x}) \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}(\mathbf{x}) + \sum_{(i,j)\in\mathcal{R}} \mathbf{e}_{ij}^T(\mathbf{x}) \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}(\mathbf{x}), \tag{57}$$

where $\boldsymbol{\Omega_{ij}}$ is the information matrix of the constraint and the error term $\mathbf{e}_{ij}(\mathbf{x})$ represents the residual between a constraint ${}_i\hat{\mathbf{T}}^j$ and the relative motion between the trajectory poses ${}_W\mathbf{T}^i$ and ${}_W\mathbf{T}^j$ it is connecting, expressed with a minimal parametrisation:

$$\mathbf{e}_{ij}(\mathbf{x}) = \log_{\mathbb{SE}(3)} \left({}_i\hat{\mathbf{T}}^j ({}_W\mathbf{T}^j)^T {}_W\mathbf{T}^i \right) \tag{58}$$

From this error term, the Jacobians and information matrix required for each constraints can be computed as detailed in Appendix B.

To speed-up the process we perform the optimisation in a multi-layered scheme. In the first layer, odometry constraints between consecutive frames, are substituted by constraints between consecutive keyframes, which are also computed by the tracking front-end, and thus we only optimise the poses of the keyframes. In the second level the keyframe poses are fixed, and the poses of the rest of the frames are optimised by enforcing the odometry constraints.

8 Experiments

We first compare our method quantitatively with the state-of-the-art in terms of trajectory estimation. Then we perform a qualitative evaluation of the 3D reconstructions both in the Freiburg and our

own RGB-D sequences, and finally we measure the computational performance, both of the front-end, in charge of camera tracking, and the back-end, performing keyframe processing and loop closing.

8.1 Trajectory Estimation

We first compare our method to state-of-the-art visual odometry and SLAM approaches with RGB-D systems. For the comparison we use the TUM benchmarking dataset [38]. The evaluation has been carried out taking the error metric of the Absolute Trajectory Error (ATE) in meters (see Table 1). However it is worth noticing that there exists a large improvement in the $fr2/desk$ dataset. Apparently the reason is a correction we applied to the depth maps of the Freiburg 2 sequences by a constant scaling factor. This correction was in principle reported as done by the authors of the dataset. However it seems from the observation made by Mur-Artal et al. in [29] of a scale bias in the RGB-D SLAM system of Endres et al. [11], that it has not been applied in the Freiburg 2 datasets available for download.

The table shows that our method outperforms most of the odometry and SLAM methods in the literature and it is close to the ElasticFusion of Whelan et al. [43], which shows the best performance in the state-of-the-art in the datasets $fr3/office$ and $fr3/nst$. Among the methods performing only odometry, ElasticFusion in tracking-only mode outperforms them and even some of the methods performing loop closure. However in other datasets our method shows overall the best performance both in tracking-only mode and with loop closures. It is worth noticing that challenging datasets like $fr1/desk2$ and $fr1/room$, showing a fast camera motion, our method provides a great precision.

8.2 3D reconstruction

In this section we perform a qualitative evaluation of the 3D models reconstructed of our SLAM system. To build the 3D model of the scene we concatenate the point clouds of different keyframes in real-time. To avoid unnecessary redundant points we only concatenate partial point clouds which correspond to novel parts of the scene between consecutive keyframes. Redundant points due too loop closure are eliminated at the end of the execution of our method by performing a voxel grid filtering with a voxel size of 1 cm. In Fig. 2 we show the 3D models of the evaluated TUM datasets, showing a great detail of the reconstructions.

8.3 Computational performance

The computational performance of our system has been evaluated in a laptop PC with an Intel Core i7-4710HQ CPU at 2.50GHz, 16GB of RAM and a nVidia GeForce GTX 850M GPU with 4GB of memory. As shown in Table 2, the execution time of the front-end varies depending on whether the system operates in tracking only-mode, *i.e.*, with the back-end disabled or with its complete functionalities. This is due to the division of the GPU resources upon successful loop closure between the visual odometry and integration modules from the front-end, and the keyframe alignment module between loop keyframes from the back-end.

In Fig. 3 we show graphically the computational cost of the back-end pipeline, which has been dissected in the different processes involved. As we can observe there is a nearly constant cost for keyframe segmentation and descriptor extraction, while the cost for loop closure detection and pose-graph optimisation shows a great variability. The reason is that most of the cost of loop closure detection is governed by the step of geometric verification and loop constraint computation, which is run only on a reduced list of loop candidates delivered by the efficient appearance-based BoW algorithm for loop detection. Red dashed line represents the time it takes the front-end to deliver a new keyframe to the back-end, assuming that it is able to process al the frames in the sequence at the frame rate of 30 Hz. We can observe that in datasets where the camera is moved slowly in a loop around the same scene, the back-end processing time is widely below the average keyframe rate. However in datasets like $fr1/desk2$ or $fr1/room$, where the camera moves quickly and/or the area being mapped changes a lot, the back-end

Table 1: Absolute trajectory error (RMSE, median and max) in meters of our method with and without loop closure and comparison with state-of-the-art approaches. For a given error measure in a dataset, we show in bold the best approach in the state of the art and ours if it is the absolute best.

Approach	fr1/desk			fr1/desk2			fr1/room			fr2/desk			fr3/office			fr3/nst		
	RMSE	median	max	RMSE	median	max	RMSE	median	max	RMSE	median	max	RMSE	median	max	RMSE	median	max
Ours (w/o loop closure)	0.034	0.030	0.087	0.054	0.037	0.137	0.087	0.072	0.203	0.037	0.026	0.084	0.057	0.042	0.118	0.041	0.030	0.106
Ours (w/ loop closure)∗	**0.018**	**0.014**	**0.052**	0.036	**0.029**	**0.104**	**0.040**	**0.035**	**0.140**	0.017	**0.015**	**0.036**	0.025	**0.024**	**0.046**	**0.015**	0.012	0.051
[17]	0.033	0.026	0.086	0.066	0.044	0.180	0.097	0.086	0.195	0.075	0.077	f0.104	0.082	0.036	0.143	-	-	-
Unified VP [27]	-	**0.018**	**0.066**	-	-	-	-	0.144	0.339	-	0.093	0.116	-	-	-	-	-	-
ICP+RGB-D [41]	-	0.069	0.234	-	-	-	-	0.158	0.421	-	0.119	0.362	-	-	-	-	-	-
6D RGB-D odometry [10]	-	-	-	-	-	-	0.095	**0.067**	0.254	0.197	0.174	0.416	-	-	-	-	-	-
SDF tracking [5]	0.035	-	-	0.062	-	-	0.078	-	-	-	-	-	0.040	-	-	-	-	-
DIFODO [21]	0.047	-	-	0.094	-	-	0.109	-	-	0.342	-	-	-	-	-	-	-	-
SLAC [46]	0.026	-	-	**0.035**	-	-	0.059	-	-	-	-	-	0.022	-	-	-	-	-
ElasticFusion (w/o loop closure) [43]	0.022	-	-	-	-	-	-	-	-	-	-	-	0.025	-	-	0.027	-	-
RGB-D SLAM [11]∗	0.023	-	-	0.043	-	-	0.084	-	-	0.057	-	-	0.032	-	-	0.017	-	-
MRSMap [36]∗	0.043	-	-	0.049	-	-	0.069	-	-	0.052	-	-	0.042	-	-	2.018	-	-
DVO-SLAM [23]∗	0.021	-	-	0.046	-	-	**0.053**	-	-	**0.017**	-	-	0.035	-	-	0.018	-	-
Kintinious [42]∗	0.037	0.031	0.078	0.071	-	-	0.075	0.068	**0.231**	0.034	**0.028**	**0.079**	0.030	-	-	0.031	-	-
ElasticFusion (w/ loop closure) [43] ∗	**0.020**	-	-	0.048	-	-	0.068	-	-	0.071	-	-	**0.017**	-	-	**0.016**	-	-

∗ with loop closure and pose-graph optimisation (deformation graph for ElasticFusion)

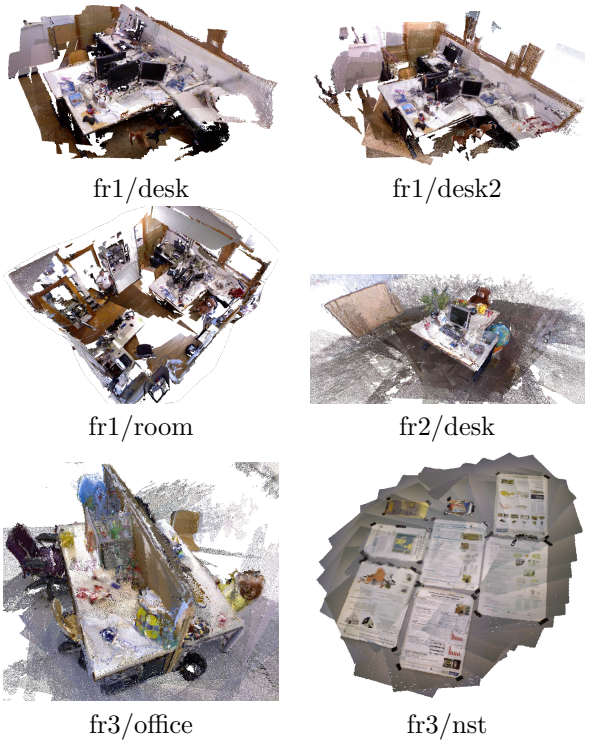

Figure 2: Final 3D models from the TUM datasets on which our method has been evaluated. Despite the discontinuities in color, we can appreciate a good level of detail in the final reconstruction.

struggles to operate at keyframe rate. These observations are embodied numerically in Table 3.

8.4 Sensor Calibration and Depth Correction

In this section we evaluate the calibration model and the procedure proposed in Section 4 to compute a custom calibration for a specific sensor. The calibration is done in two steps. In the first step we take pairs of RGB and IR images compute the RGB and IR intrinsics as well as the rigid transformation between the RGB and depth reference frames, using the Bouguet Toolbox [3]. In the second step we calibrate the disparity-to-depth conversion of the depth sensor. To do this, we take 18 shots of synchronised RGB and depth images with the camera pointing a textured wall. From the RGB images we have generated a ground truth, by manually selecting and matching some dozens of points across images, and then performing a Bundle Adjustment with the Photomodeler software to obtain the camera poses as well as a 3D point cloud from which the best fitting plane is computed. Then, using (16) with the calibration parameters computed in the first step, we generate a dense inverse depth map which is used as ground truth in the optimisation function used for calibration of the disparity-to-depth conversion. To test the accuracy of our method we have selected the odd images for calibration and the even ones for validation. The comparison of our calibration, with and without

Table 2: Computational cost of the front-end pipeline in milliseconds

	fr1/desk	fr1/desk2	fr1/room	fr2/desk	fr3/office	fr3/nst
w/o loop closure	39	39	42	38	38	36
w/ loop closure	46	55	76	49	53	41

Table 3: Mean computational cost of the processes involved in the back-end pipeline in milliseconds. To achieve online performance the mean total cost should be lower than the mean keyframe rate.

	fr1/desk	fr1/desk2	fr1/room	fr2/desk	fr3/office	fr3/nst
Segmentation	147	156	152	151	174	203
ORB desc. + BoW hist.	18	19	18	17	18	19
Loop detection	54	61	52	46	33	36
Pose-graph optim.	1	1	1	1	1	1
Total keyframe proc.	243	260	246	237	250	286
Mean keyframe rate	320	222	246	1017	773	1735

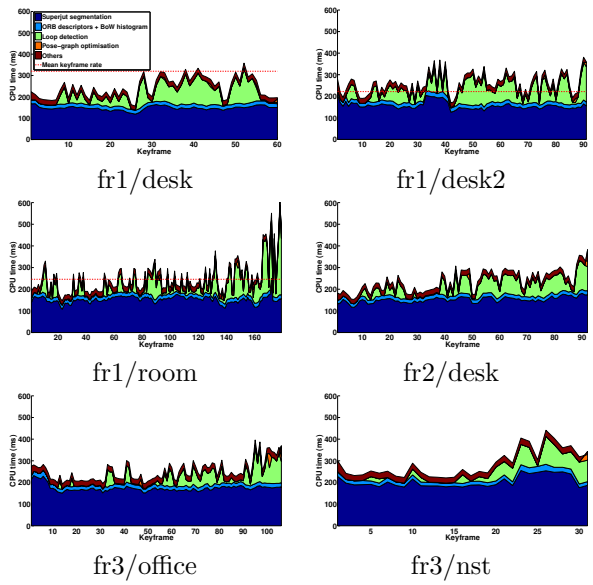

Figure 3: Computational cost of the keyframe processing in the back-end divided by processes (area graph), and average acquisition time of a keyframe (dashed red). In the datasets where the average acquisition time is not shown, it means that it is out of the plot bounds (above 600 ms). This usually occurs in datasets when the camera executes a loop around some scene, which normally involves that keyframes are switched with low frequency.

depth spatial distortion correction, and the factory calibration is shown in Fig. 4. We can observe in the figure, that there exist a bias in the raw depth map which is completely corrected by calibrating only the linear model. However, after this adjustment it still remains a zero-mean error with a spatial pattern, which is corrected by the calibrated model of the depth spatial distortion. Note that with a custom calibration the depth error is reduced in both the calibration and the validation images, which indicates that the disparity-to-depth conversion is well modelled by our calibration model.

To evaluate the performance of our RGB-iD SLAM system with a custom calibration we have run our method on one sequence taken with synchronised RGB and depth streams but with unregistered depth maps. The obtained reconstruction is shown in Fig. 5

9 Conclusions

In this paper we have presented a complete RGB-ID SLAM system which combines an accurate dense and direct RGB-D odometry system and state-of-the-art loop closing procedure to achieve a great accuracy in the estimation of the trajectory. The algorithm also performs the integration of the tracked frames into keyframes with smooth inverse depth maps. The fu-

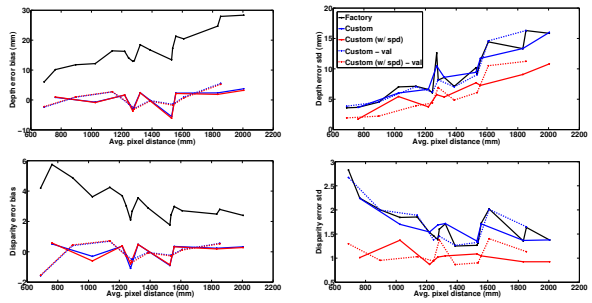

Figure 4: Bias and standard deviation of the calibration error, varying with the average pixel distance of each image for different calibration methods. Error is shown in depth units (top) as well as in disparity units (bottom).

Figure 5: 3D reconstruction obtained by our RGB-iD SLAM system in an office environment using a custom calibration for the RGB-D camera, and with unregistered depth images.

sion of depth measurements in keyframes also allows to achieve a greater accuracy in the computation of relative camera transformation at loop closure. The concatenation of the novel parts between consecutive keyframes allow to obtain precise 3D models of the environment. In the comparison with some of the best state of-the-art methods for RGB-D mapping our method shows to have results with greater or similar accuracy in most of the tested datasets. Our method is able to work in real time with a mid-range laptop GPU. Our system is also to work with custom calibrated RGB-D sensors without need of hardware registering the depth image to the RGB frame. With this in mind we have also proposed a pipeline for accurate calibration of RGB-D cameras. The code for the RGBiD-SLAM system has been made available publicly [2].

A Covariance propagation in composition of spatial transformation

Let us define the following composition operation of spatial transformations ${}_A\mathbf{T}^B = \left({}_W\mathbf{T}^A\right)^{-1} {}_W\mathbf{T}^B$. Let us suppose that ${}_W\mathbf{T}^A$) and ${}_W\mathbf{T}^B$ have covariances $\mathbf{\Sigma}_W^{WA}$ and $\mathbf{\Sigma}_W^{WB}$ respectively expressed in the reference W, and we want to compute the covariance $\mathbf{\Sigma}_A^{AB}$. To compute the Jacobians we must apply differential perturbations in the side in which the covariance is referenced. In this case all of them are left perturbations. Taking separately the translation and rotation we have:

$$\mathbf{t}_A^B + \boldsymbol{\delta}\mathbf{t}_A^{AB} = \left(\exp\left(\boldsymbol{\delta\theta}_W^{WA}\right) {}_W\mathbf{R}^A\right)^T \left(\mathbf{t}_W^B + \boldsymbol{\delta}\mathbf{t}_W^{WB} - \mathbf{t}_W^A - \boldsymbol{\delta}\mathbf{t}_W^{WA}\right) \tag{59}$$

$$\exp\left(\boldsymbol{\delta\theta}_A^{AB}\right) {}_A\mathbf{R}^B = \left(\exp\left(\boldsymbol{\delta\theta}_W^{WA}\right) {}_W\mathbf{R}^A\right)^T \exp\left(\boldsymbol{\delta\theta}_W^{WB}\right) {}_W\mathbf{R}^B \tag{60}$$

Applying that $\exp(\boldsymbol{\delta\theta}) \approx \mathbf{I} + [\boldsymbol{\delta\theta}]_\times$ and $[\mathbf{a}]_\times\mathbf{b} = -[\mathbf{b}]_\times\mathbf{a}$ in (59); and $\mathbf{R}\exp(\boldsymbol{\theta})\mathbf{R}^T = \exp(\mathbf{R}\boldsymbol{\theta})$ and $\log(\exp(\boldsymbol{\delta\theta}_1)\exp(\boldsymbol{\delta\theta}_2)) \approx \boldsymbol{\delta\theta}_1 + \boldsymbol{\delta\theta}_2$ in (60) we obtain the following expression for the covariance propagation:

[2] https://github.com/dangut/RGBiD-SLAM

$$\boldsymbol{\Sigma}_A^{AB} = \frac{\boldsymbol{\partial}_A\mathbf{T}^B}{\boldsymbol{\partial}_W\mathbf{T}^A}\boldsymbol{\Sigma}_W^{WA}\left(\frac{\boldsymbol{\partial}_A\mathbf{T}^B}{\boldsymbol{\partial}_W\mathbf{T}^A}\right)^T + \frac{\boldsymbol{\partial}_A\mathbf{T}^B}{\boldsymbol{\partial}_W\mathbf{T}^B}\boldsymbol{\Sigma}_W^{WB}\left(\frac{\boldsymbol{\partial}_A\mathbf{T}^B}{\boldsymbol{\partial}_W\mathbf{T}^B}\right)^T \tag{61}$$

with

$$\frac{\boldsymbol{\partial}_A\mathbf{T}^B}{\boldsymbol{\partial}_W\mathbf{T}^A} = \frac{\boldsymbol{\partial}(\boldsymbol{\delta}\mathbf{t}_A^{AB},\ \boldsymbol{\delta}\boldsymbol{\theta}_A^{AB})}{\boldsymbol{\partial}(\boldsymbol{\delta}\mathbf{t}_W^{WA},\ \boldsymbol{\delta}\boldsymbol{\theta}_W^{WA})} = -\begin{pmatrix} {}_A\mathbf{R}^W & {}_A\mathbf{R}^W[\mathbf{t}_W^A - \mathbf{t}_W^B]_\times \\ \mathbf{0} & {}_A\mathbf{R}^W \end{pmatrix} \tag{62}$$

$$\frac{\boldsymbol{\partial}_A\mathbf{T}^B}{\boldsymbol{\partial}_W\mathbf{T}^B} = \frac{\boldsymbol{\partial}(\boldsymbol{\delta}\mathbf{t}_A^{AB},\ \boldsymbol{\delta}\boldsymbol{\theta}_A^{AB})}{\boldsymbol{\partial}(\boldsymbol{\delta}\mathbf{t}_W^{WB},\ \boldsymbol{\delta}\boldsymbol{\theta}_W^{WB})} = \begin{pmatrix} {}_A\mathbf{R}^W & \mathbf{0} \\ \mathbf{0} & {}_A\mathbf{R}^W \end{pmatrix} \tag{63}$$

B Pose-graph Jacobians and Information matrix

Frequently in optimisation problems, the measurement contribution to the error term is expressed through an addition or subtraction, and thus the information matrix of the constraint is trivially the inverse of the measurement's covariance. However in a pose-graph problem the error is expressed as:

$$\mathbf{e}_{ij} = \log_{\mathbb{SE}(3)}\left({}_i\hat{\mathbf{T}}^j({}_W\mathbf{T}^j)^{-1}{}_W\mathbf{T}^i\right) \tag{64}$$

and thus we need to compute the derivative of $\mathbf{e}_{ij}$ wrt. ${}_i\hat{\mathbf{T}}^j$. Assuming that the covariance of ${}_i\hat{\mathbf{T}}^j$, $\boldsymbol{\Sigma}_i^{ij}$, is computed on the left side, we will compute the derivative applying a perturbation at the left of ${}_i\hat{\mathbf{T}}^j$:

$$\mathbf{e}_{ij} + \boldsymbol{\delta}\mathbf{e}_{ij} = \log\left(\exp(\boldsymbol{\delta}\mathbf{t}_i^{ij},\ \boldsymbol{\delta}\boldsymbol{\theta}_i^{ij})\mathbf{E}_{ij}\right) \tag{65}$$

$$\approx \begin{pmatrix} \boldsymbol{\Delta}\mathbf{t}_{ij} - [\boldsymbol{\Delta}\mathbf{t}_{ij}]_\times\boldsymbol{\delta}\boldsymbol{\theta}_i^{ij} + \boldsymbol{\delta}\mathbf{t}_i^{ij} \\ \log\left(\exp(\boldsymbol{\delta}\boldsymbol{\theta}_i^{ij})\boldsymbol{\Delta}\mathbf{R}_{ij}\right) \end{pmatrix} \tag{66}$$

with

$$\mathbf{E}_{ij} = {}_i\hat{\mathbf{T}}^j({}_W\mathbf{T}^j)^{-1}{}_W\mathbf{T}^i = \begin{pmatrix} \boldsymbol{\Delta}\mathbf{R}_{ij} & \boldsymbol{\Delta}\mathbf{t}_{ij} \\ \mathbf{0}^T & 1 \end{pmatrix} \tag{67}$$

The derivative of the error wrt. the relative pose measurement remains:

$$\frac{\partial\mathbf{e}_{ij}}{\partial_i\hat{\mathbf{T}}^j} = \frac{\partial\mathbf{e}_{ij}}{\boldsymbol{\partial}(\boldsymbol{\delta}\mathbf{t}_i^{ij},\ \boldsymbol{\delta}\boldsymbol{\theta}_i^{ij})} = \begin{pmatrix} \mathbf{I} & -[\boldsymbol{\Delta}\mathbf{t}_{ij}]_\times \\ \mathbf{0} & \boldsymbol{\Delta}\mathbf{Q}_{ij}^{-1} \end{pmatrix} \tag{68}$$

where $\boldsymbol{\Delta}\mathbf{Q}_{ij} = \int_0^1 \exp(\tau\log(\boldsymbol{\Delta}\mathbf{R}_{ij}))d\tau$. And hence we obtain a expression for information update:

$$\boldsymbol{\Omega}_{ij} = \left(\frac{\partial\mathbf{e}_{ij}}{\partial_i\hat{\mathbf{T}}^j}\boldsymbol{\Sigma}_i^{ij}\left(\frac{\partial\mathbf{e}_{ij}}{\partial_i\hat{\mathbf{T}}^j}\right)^T\right)^{-1} \tag{69}$$

To obtain the Jacobians wrt. to the state of the optimised nodes we proceed similarly. The incremental updates are to be applied in the form of transformation matrices by post-multiplication. Thus in this case the perturbations to compute the Jacobians are applied on the right side of the absolute camera poses in (64). Thus we obtain:

$$\frac{\partial\mathbf{e}_{ij}}{\partial_W\hat{\mathbf{T}}^i} = \frac{\partial\mathbf{e}_{ij}}{\boldsymbol{\partial}(\boldsymbol{\delta}\mathbf{t}_i^{Wi},\ \boldsymbol{\delta}\boldsymbol{\theta}_i^{Wi})} = \begin{pmatrix} \boldsymbol{\Delta}\mathbf{R}_{ij} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Delta}\mathbf{Q}_{ij}^{-1}\boldsymbol{\Delta}\mathbf{R}_{ij} \end{pmatrix} \tag{70}$$

$$\frac{\partial\mathbf{e}_{ij}}{\partial_W\hat{\mathbf{T}}^j} = \frac{\partial\mathbf{e}_{ij}}{\boldsymbol{\partial}(\boldsymbol{\delta}\mathbf{t}_j^{Wj},\ \boldsymbol{\delta}\boldsymbol{\theta}_j^{Wj})} = \begin{pmatrix} -{}_i\hat{\mathbf{R}}^j & [\hat{\mathbf{t}}_{ij}]_\times{}_i\hat{\mathbf{R}}^j \\ \mathbf{0} & -\boldsymbol{\Delta}\mathbf{Q}_{ij}^{-1}{}_i\hat{\mathbf{R}}^j \end{pmatrix} \tag{71}$$

References

[1] Egor Bondarev, Francisco Heredia, Rafael Favier, Lingni Ma, and Peter H. N. de With. On photo-realistic 3d reconstruction of large-scale and arbitrary-shaped environments. In *IEEE Consumer Communications and Networking Conf. (CCNC)*, pages 621–624, 2013.

[2] Patrick Boucher. Points on a sphere. http://www.softimageblog.com/archives/115, 2006.

[3] J. Y. Bouguet. Camera calibration toolbox for Matlab, 2008.

[4] D.C. Brown. Decentering distortion of lenses. *Photometric Engineering*, 32:444–462, 1966.

[5] Erik Bylow, Jrgen Sturm, Christian Kerl, Fredrik Kahl, and Daniel Cremers. Real-time camera tracking and 3d reconstruction using signed distance functions. In *Robotics: Science and Systems (RSS)*, 2013.

[6] Maurilio Di Cicco, Luca Iocchi, and Giorgio Grisetti. Non-parametric calibration for depth sensors. *Robotics and Autonomous Systems*, 74, Part B:309 – 317, 2015.

[7] J. Civera, O. G. Grasa, A. J. Davison, and J. M. M. Montiel. 1-Point RANSAC for EKF Filtering: application to real-time structure from motion and visual odometry. *J. of Field Robotics*, 27(5):609–631, 2010.

[8] Javier Civera, Andrew J. Davison, and J. M. M. Montiel. Inverse depth parametrization for monocular slam. *IEEE Trans. on Robotics*, 24(5):932–945, 2008.

[9] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Int. Conf. on Computer Vision (ICCV)*, volume 2, pages 1403–1410, 2003.

[10] Haiwei Dong, Nadia Figueroa, and Abdulmotaleb El Saddik. Towards consistent reconstructions of indoor spaces based on 6d rgb-d odometry and kinectfusion. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1796–1803, 2014.

[11] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3D mapping with an RGB-D camera. *IEEE Trans. on Robotics (T-RO)*, 30(1):177–187, 2014.

[12] Felix Endres, Juergen Hess, Nikolas Engelhard, Juergen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the RGB-D SLAM system. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2012.

[13] Jakob Engel, Jurgen Sturm, and Daniel Cremers. Semi-dense visual odometry for a monocular camera. In *IEEE Int. Conf. on Computer Vision (ICCV)*, 2013.

[14] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *Int. Journal of Computer Vision (IJCV)*, 59(2):167–181, 2004.

[15] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Trans. on Robotics (T-RO)*, 28(5):1188–1197, 2012.

[16] Daniel Gutiérrez, Alejandro Rituerto, J. M. M. Montiel, and J. J. Guerrero. Adapting a real-time monocular visual slam from conventional to omnidirectional cameras. In *11th OMNIVIS, held with International Conference on Computer Vision (ICCV)*, 2011.

[17] Daniel Gutiérrez-Gómez, Walterio Mayol-Cuevas, and J. J. Guerrero. Dense rgb-d visual odometry using inverse depth. *Robotics and Autonomous Systems (RAS)*, 75(Part B):571 – 583, 2016. Special Section on 3D Perception with PCL.

[18] Daniel Herrera C, Juho Kannala, and Janne Heikkila. Joint depth and color camera calibration with distortion correction. *IEEE Trans. on Pattern Analysis and Machine Intelligence (T-PAMI)*, 34(10):2058–2064, 2012.

[19] P. W. Holland and R. E. Welsch. Robust Regression Using Iteratively Reweighted Least-Squares. *Communications in Statistics: Theory and Methods*, A6:813–827, 1977.

[20] Berthold K. P. Horn, H.M. Hilden, and Shariar Negahdaripour. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society America*, 5(7):1127–1135, 1988.

[21] Mariano Jaimez and Javier González-Jiménez. Fast visual odometry for 3-d range sensors. *IEEE Transactions on Robotics*, 31(4):809–822, 2015.

[22] Andrej Karpathy, Stephen Miller, and Li Fei-Fei. Object discovery in 3d scenes via shape analysis. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2088–2095, 2013.

[23] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Dense visual slam for rgb-d cameras. In

IEEE/RSJ Conf. on Intelligent Robots and Systems (IROS), pages 2100–2106, 2013.

[24] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *IEEE and ACM Int. Symp. on Mixed and Augmented Reality (ISMAR)*, 2007.

[25] Kurt Konolige and Patrick Mihelich. Technical description of kinect calibration. `http://wiki.ros.org/kinect_calibration/technical`, 2010, [Online; accessed 25-September-2014].

[26] Chuanhai Liu and Donald B. Rubin. ML estimation of the t distribution using EM and its extensions, ECM and ECME. *Statistica Sinica*, 5:19–39, 1995.

[27] M. Meilland and A.I. Comport. On unifying key-frame and voxel-based dense visual SLAM at large scales. In *IEEE/RSJ Int. Conf.on Intelligent Robots and Systems (IROS)*, 2013.

[28] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Trans. on Robotics (T-RO)*, 31(5):1147–1163, 2015.

[29] Raul Mur-Artal and Juan D. Tardós. Fast relocalisation and loop closing in keyframe-based slam. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2014.

[30] Richard M. Murray, S. Shankar Sastry, and Li Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., 1994.

[31] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Int. Symp. on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011.

[32] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. In *Int. Conf. on Computer Vision (ICCV)*, pages 2320–2327, 2011.

[33] Manuel M. Oliveira, Gary Bishop, and David McAllister. Relief texture mapping. In *Proc. of Annual Conf. on Computer graphics and interactive techniques (SIGGRAPH)*, pages 359–368, 2000.

[34] Edward B Saff and A BJ Kuijlaars. Distributing many points on a sphere. *The mathematical intelligencer*, 19(1):5–11, 1997.

[35] Jan Smisek, Michal Jancosek, and Tomás Pajdla. 3d with kinect. In *IEEE Int. Conf. on Computer Vision Workshops, ICCV-W*, pages 1154–1160, 2011.

[36] J Stuckler and Sven Behnke. Integrating depth and color cues for dense multi-resolution scene mapping using rgb-d cameras. In *IEEE Conf. on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 162–167, 2012.

[37] Jörg Stückler and Sven Behnke. Multi-resolution surfel maps for efficient dense 3d modeling and tracking. *Journal of Visual Communication and Image Representation*, 25(1):137–147, 2014.

[38] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IEEE/RSJ Int. Conf. on Intelligent Robot Systems (IROS)*, 2012.

[39] Alex Teichman, Stephen Miller, and Sebastian Thrun. Unsupervised intrinsic calibration of depth sensors via slam. In *Robotics: Science and Systems (RSS)*, 2013.

[40] Rudolph Triebel, Jiwon Shin, and Roland Siegwart. Segmentation and unsupervised part-based discovery of repetitive objects. In *Robotics: Science and Systems (RSS)*, 2010.

[41] Thomas Whelan, Hordur Johannsson, Michael Kaess, John J Leonard, and John McDonald. Robust real-time visual odometry for dense rgb-d mapping. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2013.

[42] Thomas Whelan, Michael Kaess, Ross Finman, Maurice Fallon, Hordur Johannsson, John

Leonard, and John McDonald. Real-time large scale dense rgb-d slam with volumetric fusion. *The Int. Journal of Robotics Research (IJRR)*, 34(4-5):598–626, 2014.

[43] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. Elasticfusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research (IJRR)*, 35(14):1697–1716, 2016.

[44] B. Williams, G. Klein, and I. Reid. Real-time SLAM relocalisation. In *Int. Conf. on Computer Vision (ICCV)*, 2007.

[45] Bernhard Zeisl and Marc Pollefeys. Structure-based auto-calibration of RGB-D sensors. In *IEEE Int. Conf. on Robotics and Automation, ICRA*, pages 5076–5083, 2016.

[46] Qian-Yi Zhou and Vladlen Koltun. Simultaneous localization and calibration: Self-calibration of consumer depth cameras. In *IEEE Conf. on Computer Vision and Pattern Recognition, CVPR*, pages 454–460, 2014.