RGBDSLAM & RGBDMapping Report

Can it reconstruct 3D object?

Lihang Li

Feb. 2015

RGBDSLAM & RGBDMapping Report

Can it reconstruct 3D object?

1. Papers and Links

(1) RGBDSLAM

Papers:

1) Real-time 3D visual SLAM with a hand-held RGB-D camera-03_engelhard

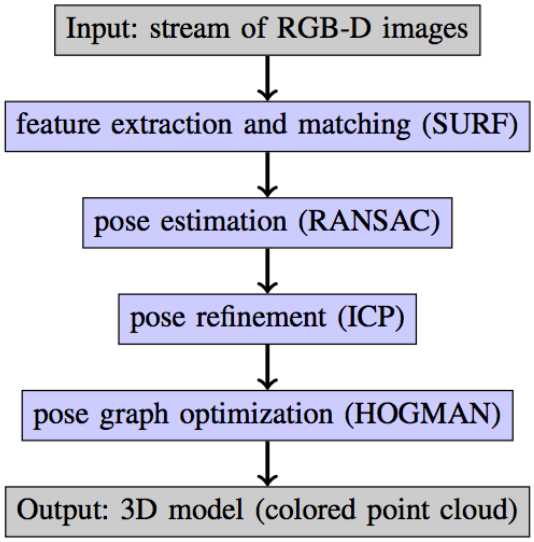

Fig. 1: The four processing steps of our approach. Our approach generates colored 3D environment models from the images acquired with a hand-held Kinect sensor.

2) An evaluation of the RGB-D SLAM system

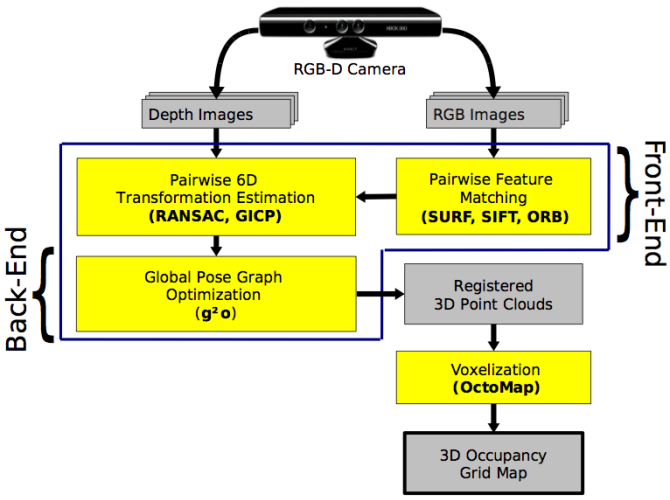

Fig. 2. Schematic overview of our approach. We extract visual features that we associate to 3D points. Subsequently, we mutually register pairs of image frames and build a pose graph, that is optimized using $\mathbf{g^2o}$. Finally, we generate a textured voxel occupancy map using the OctoMapping approach.

3) 3D Mapping with an RGB-D Camera-endres13tro

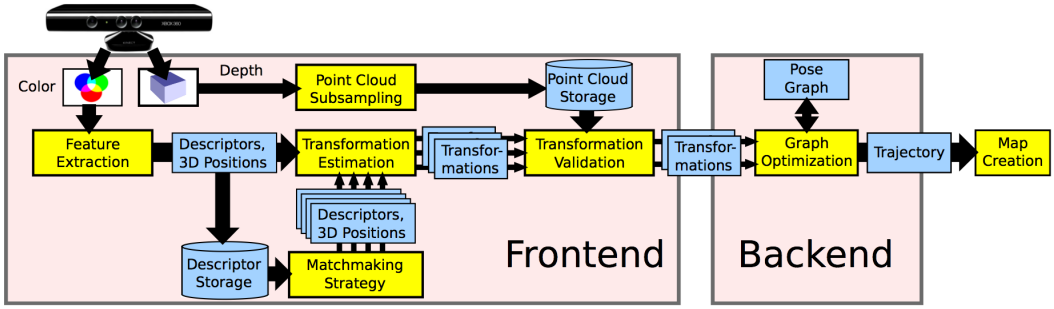

Fig. 3. Schematic overview of our approach. We extract visual features that we associate to 3D points. Subsequently, we mutually register pairs of image frames and build a pose graph, that is optimized using $\mathbf{g^2o}$. Finally, we generate a textured voxel occupancy map using the OctoMapping approach.

4) Robust Odometry Estimation for RGB-D Cameras-kerl13icra

***Abstract*— The goal of our work is to provide a fast and accurate method to estimate the camera motion from RGB-D images. Our approach registers two consecutive RGB-D frames directly upon each other by minimizing the photometric error. We estimate the camera motion using non-linear minimization in combination with a coarse-to-fine scheme. To allow for noise and outliers in the image data, we propose to use a robust error function that reduces the influence of large residuals. Furthermore, our formulation allows for the inclusion of a motion model which can be based on prior knowledge, temporal filtering, or additional sensors like an IMU. Our method is attractive for robots with limited computational resources as it runs in real-time on a single CPU core and has a small, constant memory footprint. In an extensive set of experiments carried out both on a benchmark dataset and synthetic data, we demonstrate that our approach is more accurate and robust than previous methods. We provide our software under an open source license.**

The main contributions of this paper are:

- a probabilistic formulation for direct motion estimation based on RGB-D data,
- a robust sensor model derived from real world data,
- the integration of a temporal prior,
- an open-souce implementation that runs in real-time (30 Hz) on a single CPU core,

5) Dense Visual SLAM for RGB-D Cameras-kerl13iros

In this paper, we extend our dense visual odometry [7] by several key components that significantly reduce the drift and pave the way for a globally optimal map. Figure 1 shows an example of an optimized trajectory and the resulting consistent point cloud model. The implementation of our approach is available at:

`vision.in.tum.de/data/software/dvo`

The main contributions of this paper are:

- a fast frame-to-frame registration method that optimizes both intensity and depth errors,
- an entropy-based method to select keyframes, which significantly decreases the drift,
- a method to validate loop closures based on the same entropy metric, and
- the integration of all of the above techniques into a general graph SLAM solver that further reduces drift.

Links:
http://wiki.ros.org/rgbdslam
http://wiki.ros.org/rgbdslam_electric
http://felixendres.github.io/rgbdslam_v2/
http://www2.informatik.uni-freiburg.de/~endres/

(2) RGBDMapping

Papers:

1) RGB-D Mapping_ Using Depth Cameras for Dense 3D Modeling of Indoor Environments-3d-mapping-iser-10-final

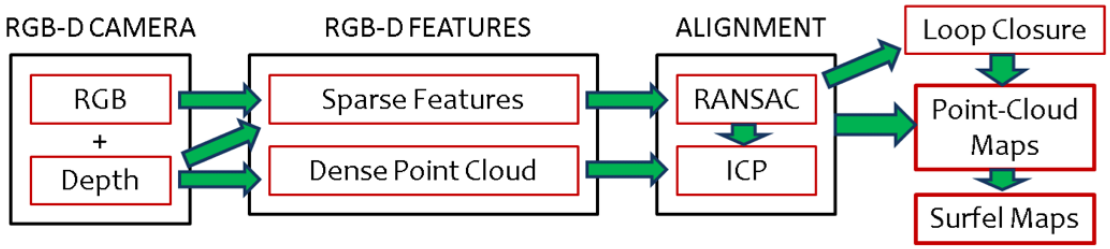

Fig. 2 Overview of RGB-D Mapping. The algorithm uses both sparse visual features and dense point clouds for frame-to-frame alignment and loop closure detection. The surfel representation is updated incrementally.

RGBD-ICP $(\mathbf{P}_s, \mathbf{P}_t)$**:**

1: $F = Extract\_RGB\_point\_features(\mathbf{P}_s)$
2: $F_{\text{target}} = Extract\_RGB\_point\_features(\mathbf{P}_t)$
3: $(\mathbf{t}^*, A_f) = Perform\_RANSAC\_Alignment(F, F_{\text{target}})$
4: ***repeat***
5: $A_d = Compute\_Closest\_Points(\mathbf{t}^*, \mathbf{P}_s, \mathbf{P}_t)$
6: $\mathbf{t}^* = \text{argmin}_{\mathbf{t}}\, \alpha \left(\frac{1}{|A_f|} \sum_{i \in A_f} w_i \left| \mathbf{t}(f_s^i) - f_t^i \right|^2 \right) + (1-\alpha) \left(\frac{1}{|A_d|} \sum_{j \in A_d} w_j \left| (\mathbf{t}(p_s^j) - p_t^j) \cdot n_t^j \right|^2 \right)$
7: ***until*** $(Error\_Change(\mathbf{t}^*) \leq \theta)$ *or (maxIter reached)*
8: ***return*** $\mathbf{t}^*$

Table 1 RGBD-ICP algorithm for matching two RGB-D frames.

2) RGB-D mapping_ Using Kinect-style depth cameras for dense 3D modeling of indoor environments-henry-ijrr12-rgbd-mapping

This paper is an extension of our previous work presented at the International Symposium on Experimental Robotics (Henry et al. 2010). The major additions are:

- we utilize re-projection error for frame-to-frame alignment RANSAC and demonstrate that it performs better than the Euclidean-space RANSAC used in Henry et al. (2010);
- we use FAST features and Calonder descriptors instead of scale-invariant feature transform (SIFT);
- we improve the efficiency of loop closure detection through place recognition;
- we improve global optimization by using *sparse bundle adjustment* (SBA) and compare it with TORO;
- we show how to incorporate ICP constraints into SBA, a crucial component in featureless areas.

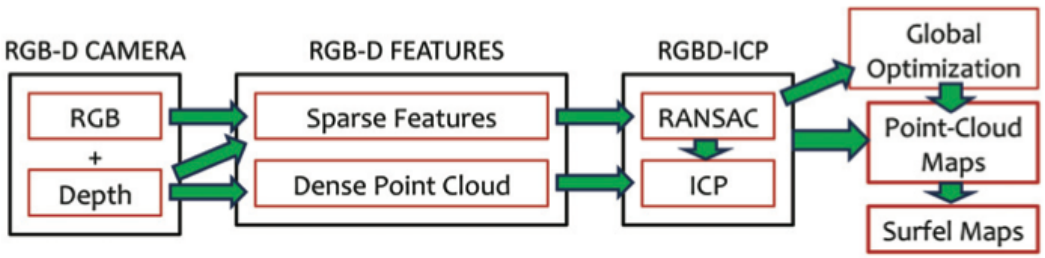

Fig. 2. Overview of RGB-D Mapping. The algorithm uses both sparse visual features and dense point clouds for frame-to-frame alignment and loop-closure detection, which run in parallel threads.

Algorithm 1: RGB-D ICP algorithm for matching two RGB-D frames.

input : source RGB-D frame P_s, target RGB-D frame P_t, previous transformation $\mathbf{T}_\mathrm{p}$
output: Optimized relative transformation $\mathbf{T}^*$

$F_s \leftarrow$ `Extract_RGB_Point_Features` (P_s);
$F_t \leftarrow$ `Extract_RGB_Point_Features` (P_t);
$(\mathbf{T}^*, A_f) \leftarrow$ `Perform_RANSAC_Alignment` (F_s, F_t);
if $|A_f| < \gamma$ **then**
    $\mathbf{T}^* = \mathbf{T}_\mathrm{p}$;
    $A_f = \emptyset$;
end
repeat
    $A_d \leftarrow$ `Compute_Closest_Points` $(\mathbf{T}^*, P_s, P_t)$;
    $\mathbf{T}^* \leftarrow$ `Optimize_Alignment` $(\mathbf{T}^*, A_f, A_d)$;
until *(*`Change` $(\mathbf{T}^*) \leq \theta$*)* **or** *(*Iterations > MaxIterations*)* ;
return $\mathbf{T}^*$;

Algorithm 2: Two-Stage RGB-D ICP algorithm for more quickly matching two RGB-D frames.

input : source RGB-D frame P_s and target RGB-D frame P_t
output: optimized relative transformation $\mathbf{T}^*$

$F_s \leftarrow$ `Extract_RGB_Point_Features` (P_s);
$F_t \leftarrow$ `Extract_RGB_Point_Features` (P_t);
$(\mathbf{T}^*, A_f) \leftarrow$ `Perform_RANSAC_Alignment` (F_s, F_t);
if $|A_f| < \gamma$ **then**
    $\mathbf{T}^* = \mathbf{T}_\mathrm{p}$;
    $A_f = \emptyset$;
end
if $|A_f| \geq \phi$ **then**
    return $\mathbf{T}^*$;
else
    repeat
        $A_d \leftarrow$ `Compute_Closest_Points` $(\mathbf{T}^*, P_s, P_t)$;
        $\mathbf{T}^* \leftarrow$ `Optimize_Alignment` $(\mathbf{T}^*, A_f, A_d)$;
    until *(*`Change` $(\mathbf{T}^*) \leq \theta$*)* **or** *(*Iterations > MaxIterations*)* ;
    return $\mathbf{T}^*$;
end

Links:
http://www.cs.washington.edu/robotics/projects/rgbd-3d-mapping/
https://www.cs.washington.edu/node/3544/

2. Methods

1) RGBDSLAM

The system is reported in their papers to be divided into Frontend and Backend. Visual odometry is done in frontend utilising sparse-feature extraction(SIFT, SURF, ORB, etc) and thus the pose of each frame is obtained. By composing a pose graph, there is a graph optimiser running in the backend to deal with the drift due to frame-to-frame registration. Once final trajectory is obtained, 3D point cloud is generated. After that, the authors use Octomap(Voxelization) to represent the map.

Besides the whole framework, it is worth noting other strategies like use the Enviroment Measurement Model(EMM) to assess the quality of visual odometry edges and define keyframes to do efficient loop closure detection.

2) RGBDMapping

This might be the first system reported for dense 3D modelling of indoor environments. Also the framework is typical that, after frame-to-frame registration, a pose graph is constructed. Then loop closure is detected using keyframes and the graph is optimised using TORO. The core algorithm is RGBD-ICP which is described in the 2010 conference paper, then variants are

developed like the Two-Stage RGBD-ICP in the 2012 journal paper. Also SBA is discussed and compared with TORO. The authors utilise surfers to represent the 3D map.

3. Source code

The RGBDSLAM released source code version 1 and version 2 both based on ROS. Version 1 is run on Fuerte while version 2 is powered by Hydro. However, the approaches didn't differ much.

RGBDMapping from Washington University did not release the source code, however the method is clearly described within their papers.

4. Compared with RTABMAP

After reading all these papers and done some experiments with the available open-source systems, we can have some comparisons:
(1) Main Method

As can be seen, all the three systems follow the standard approach:
frame-to-frame registration —> loop closure detection —> pose graph optimization —> point cloud post-processing —> map representation.

However, we find that the main difference may lie in the loop closure detection. RGBDSLAM and RGBDMapping prefer to define keyframes while RTABMAP explicitly employs Bag of Words. Also the memory management is superior in RTABMAP as it implements a good strategy which consists of STM(Short Term Memory), WM(Working Memory) and LTM(Long Term Memory). When trying scanning with RGBDSLAM, the system crashes due to 'bad_alloc' exception and examined with 'top', we find the footprint is rather big. When the memory usage reaches about 30% of the system memory, the system crashes.

(2) Timing

All these systems are strictly near real-time. about 1-3 Hz.

For visual odometry, RANSAC is effective and efficient in mose cases than ICP. Feature detection and extraction costs about 200ms and RANSAC 100ms, however ICP may cost up to 500ms.

Loop closure detection is curtail to all systems, and experiments tell us that RTABMAP is doing great in both timing and effectiveness.

Pose graph optimization is dependent on the number of nodes and edges, pruning techniques might be necessary here.

Detailed timing will be look closely when diving into the source code of RTABMAP.

(3) Adapbility

Based on the discussions above, we decide to follow RTABMAP to begin our 3D object modelling project for such reasons:

1) Good experimental results
2) Good timing performance
3) Good source code, purely written in C++ using Qt, PCL, OpenCV, etc

For RGBDSLAM, we find it hard to save 3D point cloud as '.ply' files and also the footprint is too large. Also the reconstruction results are not encouraging though. For RGBDMapping, the results are at the level with RGBDSLAM, but the mot hod sounds more reasonable. Without source code of RGBDMapping, no evaluation can be performed. However the RGBD-ICP is worth trying.

5. Things learned

(1) People are doing lots of work with RGBD cameras, mainly for visual SLAM, indoor environments 3D modelling and also large scale outdoor SLAM
(2) For 3D object modelling, we find 4 possibilities, which are RGBDSLAM, RGBDMapping, KinectFusion and RTABMAP
(3) According to experimental results, we find RTABMAP a good framework to follow and superior in both timing and point cloud visualisation than RGBDSLAM

(4) Typical systems consist of mainly three components: frame-frame registration, loop closure detection and pose graph optimisation
(5) KinectFusion(and also similar systems) is not suited here as it heavily depends on the GPU computing capabilities
(6) Besides to assessing the system capability from the point cloud subjectly, we need quantitative methods to evaluate the algorithms, the trajectory, pose graph, point cloud, etc

6. Technology roadmap

Follow RTABMAP and fuse the methods and strategies used in RGBDSLAM and RGBDMapping, etc.