Implementation of ORB-SLAM3 using Intel® RealSense D435i RGB-D sensor

Ali Babaei
Robotics and Computer vision Lab
Ryerson University
Toronto, Canada
ali.babaei@ryerson.ca

***Abstract*—This document briefly describes the ORB-SLAM family algorithms, the evolution of them and the main elements or building blocks of the algorithm. Next, the implementation of ORB-SLAM3 on Linux with D435i rgb-d sensor and practical considerations are summarized.**

***Index Terms*—ORB-SLAM, RGB-D, depth, vision, feature, SLAM**

I. Introduction

Simultaneous Localization and Mapping (SLAM) is a crucial process for a robot to explore a previously unknown environment. The tasks of identifying the current position of the robot, generating the coordinates of the robot from inputs of cameras, and creating a global map of the environment are full of the mathematical insights and domain knowledge.

The most significant difference between Visual SLAM and Visual Odometry is that visual SLAM includes a loop closure. The goal of visual SLAM is to build a map and a full trajectory for the camera (or the robot) whereas the goal of visual odometry is to incrementally obtain the next pose for the camera (or the robot). In other words, visual SLAM is targeting on a global consistent trajectory and map whereas visual odometry is targeting on a local consistent trajectory. Before the loop closure, visual SLAM has the same process flow in visual odometry: key-frames collecting, feature extracting and matching, camera pose estimation, and local camera-pose optimization.A visual SLAM system should include the following building blocks: features extraction, feature matching, motion estimation, and bundle adjustment. Visual SLAM related work could be divided into three categories based on their methodologies: filtering methods, feature-based methods, and direct methods.

II. ORB-SLAM

The ORB-SLAM [1]algorithm is categorized into the feature-based method. The name of the algorithm comes from the feature descriptor: ORB (Oriented FAST and Rotated BRIEF) [2]. ORB feature descriptor is evolved from the FAST [3] and BRIEF [4] descriptor. Rublee [2] proposed the ORB descriptor to replace the traditional SIFT (Scale-Invariant Feature Transform) feature detector to improve the performance of a SLAM system. The main advantages of adopting the ORB descriptor include rotation invariance and high noise intolerance. ORB descriptor is faster and more accurate than the original FAST descriptor. When compared to BRIEF, ORB has a lower consumption on computing. The efficiency of ORB feature descriptor enables a better image-matching ability and fast computation. Thus, the benefit of ORB descriptor allows ORB-SLAM to have a good performance.

System Diagram of ORB-SLAM algorithm is shown in Figure 1. Similar to any visual SLAM algorithm, ORB-SLAM consists of three main parts: (1) Local tracking and robot pose estimation, (2) Optimization engine, and (3) Map construction.

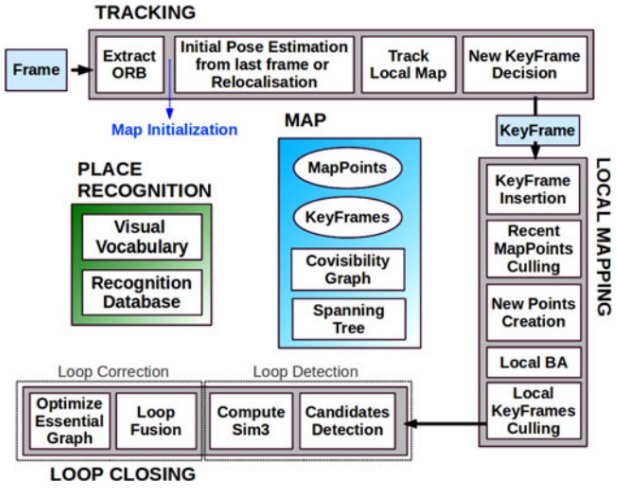

Fig. 1. ORB-SLAM system overview, showing all the steps performed by the tracking, local mapping, and loop closing threads. The main components of the place recognition module and the map are also shown. [1]

A. Initializing the map

To initialize the map starting by computing the relative pose between two scenes, the algorithm computes two geometrical models in parallel, one for a planar scene, a homography and one for non-planar scenes, a fundamental matrix. It then chooses one of both based on a relative score of both. Using the selected model the algorithm estimates multiple motion hypotheses and test if one is significantly better then the others, if so, a full bundle adjustment is done, otherwise the initialization starts over.

B. Tracking

The tracking part localizes the camera and decides when to insert a new key-frame. Features are matched with the previous frame and the pose is optimized using motion-only bundle adjustment. The features extracted are FAST corners. For image resolution up to 752×480 pixels, 1000 corners should suffice, while for higher resolution such as KITTI data-set images, 1241×376 pixels, 2000 corners proofed working well. Multiple scale-levels (factor 1.2) are used and each level is divided into a grid in which 5 corners per cell are attempted to be extracted. These FAST corners are then described using ORB.

The initial pose is estimated using a constant velocity motion model. If the tracking is lost, the place recognition module kicks in and tries to re-localize itself. When there is an estimation of the pose and feature matches, the co-visibility graph of key-frames, that is maintained by the system, is used to get a local visible map. This local map consists of key-frames that share map point with the current frame, the neighbors of these key-frames and a reference key-frame which share the most map points with the current frame. Through re-projection, matches of the local map are searched on the frame and the camera pose is optimized using these matches.

Finally is decided if a new Key-frame needs to be created, new key-frames are inserted very frequently to make tracking more robust. A new key-frame is created when at least 20 frames has passed from the last key-frame, and last global re-localization, the frame tracks at least 50 points of which less then 90% are point from the reference key-frame.

C. Local mapping

First the new key-frame is inserted into the covisibility graph, the spanning tree linking a key-frame to the key-frame with the most points in common, and a 'bag of words' representation of the key-frame (used for data association for triangulating new points) is created.

New map points are created by triangulating ORB from connected key-frames in the covisibility graph. The unmatched ORB in a key-frame are compared with other unmatched ORB in other key-frames. The match must fulfill the Epipolar constraint to be valid. To be a match, the ORB pairs are triangulated and checked if in both frames they have a positive depth, and the parallax, re-projection error and scale consistency is checked. Then the match is projected to other connected key-frames to check if it is also in these.

The new map points first need to go through a test to increase the likelihood of these map points being valid. They need to be found in more than 25% of the frames in which it is predicted to be visible and it must be observed by at least three key-frames. Then through local bundle adjustment, the current key-frame, all key-frames connected to it through the covisibility graph and all the map points seen by these key-frames are optimized using the key-frames that do see the map points but are not connected to the current key-frame.

Finally key-frames that are abundant are discarded to remain a certain simplicity. Key-frames from which more than 90% of the map points can be seen by three other key-frames in the same scale-level are discarded.

D. Loop Closing

To detect possible loops, the algorithm checks the bag of words vectors of the current key-frame and its neighbors in the covisibitlity graph. The minimum similarity of these bag of words vectors is taken as a benchmark and from all the key-frames with a bag of words similarity to the current key frame that is greater that this benchmark, all the key-frames that are already connected to the current key-frame are removed. If three loop candidates that are consistent are detected consecutively, this loop is regarded as a serious candidate.

For these loops, the similarity transformation is calculated (7DOF, 3 translations, 3 rotations, and 1 scale). RANSAC is then performed to find the outliers and these are then optimized after which more correspondences are searched and then again an optimization is preformed. If the similarity is supported by having enough inliers, the loop is accepted.

The current key-frame pose in then adjusted and this is propagated to its neighbors and the corresponding map-points are fused. Finally a pose graph optimization is preformed over the essential graph to take out the loop closure created errors along the graph. This also corrects for scale drift.

III. ORB-SLAM2

ORB-SLAM developed based on using just a monocular camera [1], which is the cheapest and smallest sensor setup. However as depth is not observable from just one camera, the scale of the map and estimated trajectory is unknown [5]. In addition the system bootstrapping require multi-view or filtering techniques to produce an initial map as it cannot be triangulated from the very first frame. Last but not least, monocular SLAM suffers from scale drift and may fail if performing pure rotations in exploration [5]. By using a stereo or an RGB-D camera all these issues are solved and allows for a more reliable Visual SLAM solutions.

ORB-SLAM2 uses depth information to synthesize a stereo coordinate for extracted features on the image. This way the system is agnostic of the input being stereo or RGB-D. Differently to all other methods the ORB-SLAM2 back-end is based on bundle adjustment and builds a globally consistent sparse reconstruction. Therefore the proposed method is lightweight and works with standard CPUs. The goal of ORB-SLAM is long-term and globally consistent localization instead of building the most detailed dense reconstruction. However from the highly accurate key-frame poses one could fuse depth maps and get accurate reconstruction on-the-fly in a local area or post-process the depth maps from all key frames after a full BA and get an accurate 3D model of the whole scene.

IV. Implementation

The goal of this assignment is to implement ORB-SLAM2 with Intel RealSense depth camera. We used D435i model, as illustrated in Figure 2.

To implement the ORB-SLAM2 algorithm there are a handful of prerequisites and dependencies that should be taken care of. These include C++11 compiler, Python with NumPy module, Pangolin for visualization and interface, OpenCV 3.2, Eigen 3.1.0 or newer, the DBoW2 library to perform place recognition and g2o library to perform non-linear optimizations. Having ROS is optional, but the examples provided are based on process input of a monocular, monocular-inertial, stereo, stereo-inertial or RGB-D camera using ROS. The authors provided that all their implementation was in ROS melodic under Ubuntu 18.04.

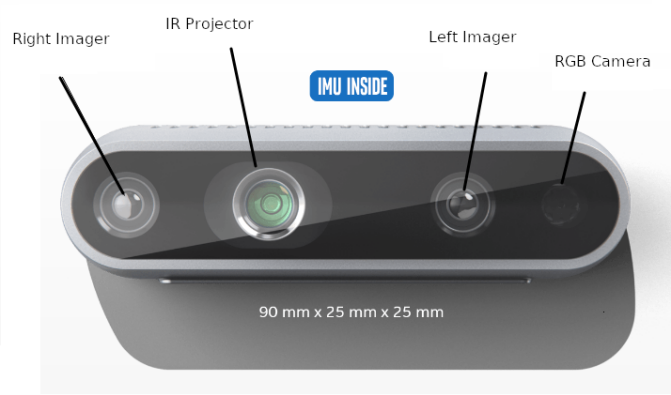

Fig. 2. Intel RealSense D435i RGB-D sensor

Building ORB-SLAM2 looks pretty straightforward. However, one may face numerous errors which should be dealt with carefully. [6] and [7] are some useful resources to look in case of any issues.

As it was discussed before, the ORB-SLAM goal is long-term and globally consistent localization instead of building the most detailed dense reconstruction. SO there is no dense point cloud as output. A video of running ORB-SALM3 with depth camera in my office is available at *https://youtu.be/xNlgA1YarDQ*.

References

[1] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[2] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[3] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*. Springer, 2006, pp. 430–443.

[4] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*. Springer, 2010, pp. 778–792.

[5] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE transactions on robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[6] Authors. Orb-slam2 configuration process. [Online]. Available: https://www.programmersought.com/article/4527375693/

[7] M. Yip. Orb-slam3 configuration process under ubuntu 20.04. [Online]. Available: https://githubmemory.com/repo/Mauhing/ORB$_SLAM3$/