Towards Understanding a Basic Stereo VO SLAM Framework by Building It from Scratch

Shubhra Aich[1] and Sai Yarlagadda[2] and Akshay Badagabettu[3]

I. Introduction

In this project, instead of playing with any particular SLAM implementation or trying out a new idea, we have decided to understand the engineering implementation of a basic SLAM framework by developing it in C++ from scratch. This is because from the initial exploratory search, we perceive that none of the SLAM implementations are essentially the same which oftentimes makes it difficult to choose just one implementation over others and build on top of that. Therefore, we have decided to build a simple one from scratch in a low-level language (C++) to get a complete understanding of the subtleties associated with building a basic pipeline. We believe this will provide us a better foundational knowledge to pursue cutting edge research on SLAM in the future.

Technically, we plan to implement a simple stereo VO SLAM on the KITTI dataset. The reason behind choosing stereo is twofold. First is the strong dependency on the stereo camera information for the projects with which the authors are affiliated. Second is the simplicity of the stereo vision in terms of implementation and initialization.

Following the classic SLAM framework, we will have two main modules in our framework – frontend and backend. A high-level overview of the features that we plan to include in these modules is briefed below:

- **Frontend:** We assume a calibrated stereo pair (or frame) as input to the frontend. We will explore classical feature extraction algorithms, such as ORB [1], SIFT [2], etc., to extract and associate or match the keypoint information across frames in the sequence. Although deep learning based stereo depth estimation now surpasses the classical approaches in most cases, we will stick to the classical approaches here. However, we will create a modular pipeline to make it easier to replace the conventional approaches with deep learning based ones in the near future for our own research. We will also track the features with basic optical flow computation, and update the complete map points based on the optical flow results. Finally, the output of the frontend will be fed to the backend for slower optimization.
- **Backend:** The landmark points from the frontend will further be optimized into the backend for better results. We will focus on updating or controlling the list of landmarks and map points so that the computational complexity of the backend does not grow over time.

[1]Shubhra Aich is with AirLab at CMU RI saich@andrew.cmu.edu

[2]Sai Yarlagadda is pursuing Master's of AI Engineering at CMU saisravy@andrew.cmu.edu

[3]Akshay Badagabettu is pursuing Master's of AI Engineering at CMU abadagab@andrew.cmu.edu

*This work was completed using the computing resources provided by the AirLab https://theairlab.org

II. Experiments

A. Datasets

Evaluation will be performed on the KITTI odometry [3] dataset. We will mainly use qualitative visualization for the evaluation. Moreover, we may include quantitative measures following the KITTI benchmark in the final submission.

B. Implementation Details

We will develop our framework based on the open-source libraries like OpenCV [4], PCL [5], g2o [6], ceres [7]. Further implementation details will be provided in the final report.

References

[1] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *ICCV*, 2011.

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *IJCV*, 2004.

[3] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *CVPR*, 2012.

[4] "OpenCV - Open Computer Vision Library," https://docs.opencv.org/4.x.

[5] "PCL - Point Cloud Library," https://pointclouds.org/.

[6] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *ICRA*, 2011.

[7] S. Agarwal, K. Mierle, and T. C. S. Team, "Ceres Solver," 2023.