SLAM Analysis for Autonomous Robotics Applications

Saardhak Bhrugubanda, Rohan Mukundhan, Arijit Nukala

December 2022

1 Project Overview and Background

Our project was based on the idea of institutional safety, specifically, the development of an autonomous security robot, whose navigation would be based on an algorithm utilizing simultaneous localization and mapping, or SLAM. SLAM is not an algorithm, but a computational challenge, whose object is to construct and frequently update a map of one's surroundings while also defining and tracking one's own location. Solutions to SLAM take many forms, but the central idea is to generate a running model for location tracking and mapping (using Bayes' rule as a framework for updating positions using probabilistic quantities as variables). The basic framework of a model designed to solve this computational challenge consists of both front- and back-end processing, described as follows (with the term vehicle indicating the agent using SLAM) [1]:

1. Front-End (Sensor-Dependent Processing), which depends on motion estimation of both the vehicle and the environment through the use of signals (audio, visual, LiDAR, etc.)
2. Back-End (Sensor-Agnostic or Sensor-Independent Processing) which depends on the generation and frequent optimization of a pose graph

Drawing from this basic framework, many SLAM algorithms exist, varying based on the type of input, method of iteration, and data density (audiovisual vs VSLAM vs LiDAR SLAM, sparse vs semi-dense vs dense data) [2]. Through our combination of a remote-control car and a computer capable of running SLAM, we hoped to implement a Roomba-style autonomous vehicle, on which a camera or sensor could be implemented for security purposes. In order to accomplish this, we chose to research multiple SLAM algorithms to determine the most optimal algorithm to use in our project.

2 Methodology and Procedure

Our methodology for analyzing the different SLAM algorithms began with a detailed literature review of currently-used SLAM algorithms and their applica-

tions, while also finding datasets to use in testing these algorithms. We selected the KITTI database, developed by researchers at Karlsruhe Institute of Technology in conjunction with Toyota. We chose this database as opposed to Waymo or other autonomous driving databases due to the lack of LiDAR reliance displayed by KITTI and its wide applicability and use in SLAM algorithms (according to SLAM literature) [3,4]. We then researched various SLAM algorithms and their applications in autonomous movement and mapping environments, aiming to narrow down to a few algorithms which were easily implementable, practical, varied in mechanisms, and would provide a reliable foundation for expanded application to the theorized security bot. We initially began with SLAMpy and PySLAM, two simple python SLAM algorithms developed more for education than practical use [5,6]. SLAMPy and PySLAM are both monocular (use of a single lens) SLAM algorithms using visual odometry, the usage of cameras to estimate an agent's position and location through iterative image analysis and feature detection. However, these algorithms did not provide realistic processing times for our purposes (processing was slow and discrete, due to Python's library reliance, interpreter use, and lack of reliability at runtime). We chose to switch to a C++ based algorithm due to C++'s reliability and fast processing speeds, lack of reliance on imports, applications of g2o for pose graph optimization, and its reliable, strongly-typed nature. Our final choices for implementable SLAM algorithms were ORB SLAM 2, LSD SLAM, and S-PTAM [7,8,9]. We implemented all algorithms in linux using ROS, ran the algorithms on KITTI datasets, and collected two quantifiable characteristics (absolute trajectory error/ATE and relative position error/RPE) [10]. A description for each SLAM algorithm follows:

2.1 ORB SLAM 2 [11]

ORB SLAM 2 is a successor to the original ORB SLAM algorithm, which incorporates ORB into a SLAM algorithm more efficiently than the initial algorithm. ORB (Oriented FAST and Rotated Brief) itself is a development upon the FAST keypoint detector, a faster and more efficient alternative to SIFT and SURF (also applying BRIEF, a rapid, pixel-to-pixel binary descriptor) [12,13]. ORB SLAM 2 is our algorithm which employs sparse feature tracking.

2.2 LSD-SLAM [14]

Large Scale Direct SLAM is a direct, monocular SLAM algorithm, purpose built to generate large-scale environmental maps. LSD-SLAM utilizes visual odometry to directly optimize image intensities, mitigating the risk of data loss in locations of sparse data availability. Additionally, the algorithm's back-end processing allows for pose-graph optimization (a vital component for SLAM) using g2o and semi-dense construction, allowing for detailed 3D maps without requiring high-end GPUs or prohibitively high processing power [15].

2.3 S-PTAM [16]

Stereo Parallel Tracking and Mapping is a stereo SLAM method which, as its name suggests, exploits the simultaneous need for tracking (location) and mapping (present location and environment) for the project. S-PTAM is a dense feature tracking algorithm, which includes loop detection and correction in the algorithm and stereo constraints to improve mapping, tracking, and computing efficiency.

2.4 Github Link

https://github.com/tikicode/SLAM-Analysis

3 Results and Result Analysis

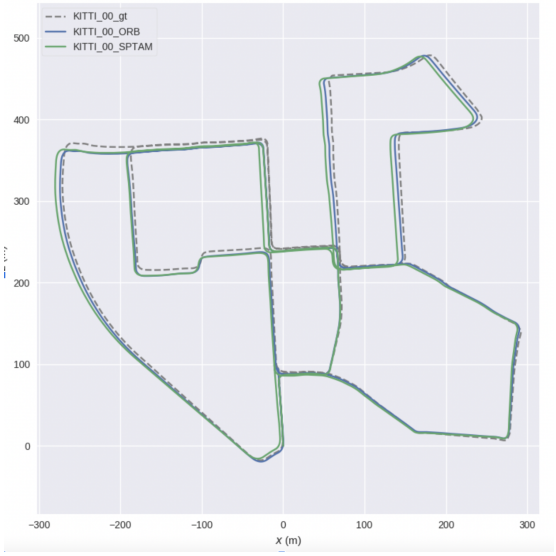

Figure 1: Ground Truth KITTI 00 Data overlaid with ORB-SLAM2 and S-PTAM trajectories

While the trajectories look very similar to the ground truth, it is essential that we quantify the accuracy. In order to evaluate the localization accuracy, we utilized the Absolute Trajectory Error (ATE) metric. ATE is the average of the error between the points of the true trajectory (ground truth) and the estimated trajectory. In order to evaluate the mapping accuracy, we utilized the Relative Pose Error (RPE) metric. RPE is the average of the error between the frame estimation of the SLAM algorithm and the ground truth frame. With both these metrics we believe we can assess accuracy and compare the SLAM algorithms against one another and with ground truth [17]. As can be seen

in the tables below, ORB-SLAM2 has significantly less error than S-PTAM compared to the ground truth trajectory for both the ATE and RPE metrics. This is a surprising result for us as we expected S-PTAM to be more accurate. S-PTAM, as aforementioned, utilizes a stereo camera and is built to extract 3D information from using its two cameras. Additionally, S-PTAM is categorized as a Dense-feature SLAM algorithm and is therefore able to extract more features per unit area from a frame. Surprisingly, ORB-SLAM2 was more accurate than S-PTAM regardless of the stereo camera and the dense-feature categorization. We suspect that ORB-SLAM2's ability to increase positioning accuracy via loop-closure is responsible for its results [18]. Additionally, ORB-SLAM2 can recompute its pose when motion between frames is too high. The KITTI dataset is captured at a low frame rate. Consequently, during turns, motion between frames is abnormally high. From the ATE over time graph, we see that towards the middle, when the KITTI agent performs a turn, S-PTAM's ATE estimate experiences increased error whereas ORB-SLAM2 is able to handle the turn.

ATE Results	ORB-SLAM2	S-PTAM
Max	3.587949	7.768977
Mean	1.156997	3.490977
Median	1.065625	3.642585
Min	0.069313	0.694787
RMSE	1.30345	3.738488
SSE	7715.07344	63466.34122
STD	0.600282	1.337675

(a) ATE Results

RPE Results	ORB-SLAM2	S-PTAM
Max	0.302712	1.136074
Mean	0.019301	0.02340
Median	0.014709	0.01916
Min	0.000312	0.00097
RMSE	0.02812	0.034919
SSE	3.59003	5.535905
STD	0.02045	0.025913

(b) RPE Results

Table 1: ATE (left) and RPE (right) results for ORB-SLAM2 and S-PTAM on KITTI 00 Data

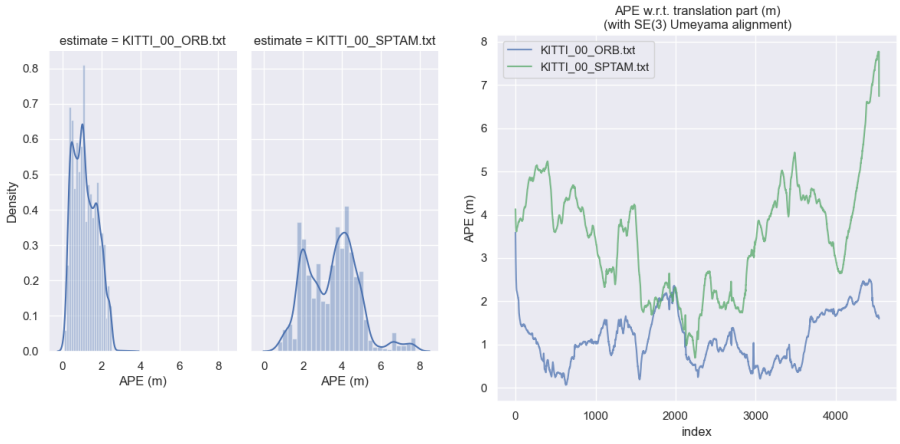

Figure 2: ATE vs Density (left) and ATE over time (right) for ORB-SLAM2 and S-PTAM

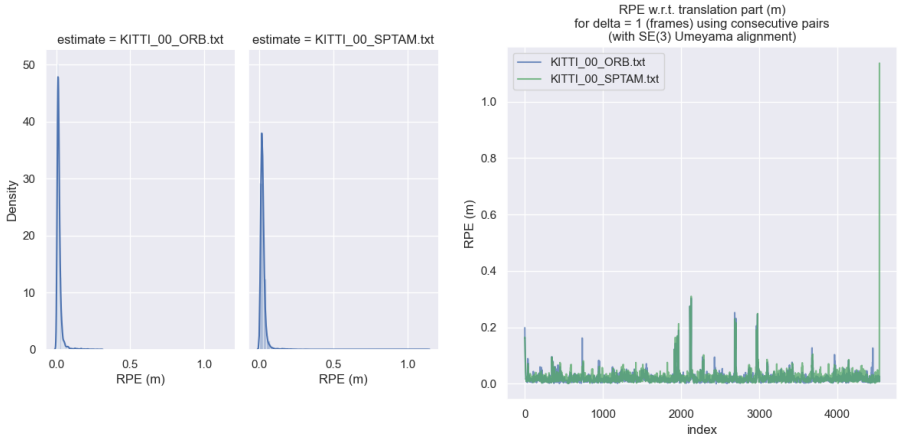

Figure 3: RPE vs Density (left) and RPE over time (right) for ORB-SLAM2 and S-PTAM

4 Discussion

Our initial project goals were to find and implement the optimal SLAM algorithm with any necessary modifications made based on our intended purpose and, as an extension on this, apply this SLAM to an autonomous vehicle with a camera mounted to act as a surveillance vehicle. Through our work on this project, we achieved our primary goal of researching SLAM and finding the

optimal SLAM algorithm out of many tested. Our results indicated that ORB-SLAM2 was the best SLAM algorithm, based on our implementation and testing of the algorithms on the KITTI datasets. We were unable to achieve the extended implementation of running these algorithms on an autonomous vehicle due to the cost of this implementation being far too high (buying an RC vehicle and a computer capable of running a full SLAM algorithm proved far too expensive for us) .

Multiple CV methods were used, both our project approach and foundation as well as those which were included in our solutions and tools. A list of the methods used as well as a brief elaboration on each follows: Feature Detection-versions of SIFT, SURF, FAST, BRIEF, an ORB were all used in all SLAM algorithms to detect salient features between maps and update self-determined position and location Stereo Vision-Monocular vision used, specifically in ORB-SLAM 2 and LSD-SLAM Photogrammetry-Utilized in visual odometry, specifically in LSD-SLAM

Throughout the course of our project, we discovered key limitations of our algorithms which ranged in their effects on algorithm applicability and efficiency. ORB-SLAM2's major shortcoming was its potential inability to map based on insufficient information (sparse algorithm) or due to large camera variations and rotations. This could be addressed in the future by implementing an adaptive algorithm to threshold image entropy and mitigate the risk of the algorithm responding negatively to camera rotation variation. LSD-SLAM relies on a high capture rate for its processing ability, rendering it unable to process the KITTI data to the extent and accuracy it could process general data. In response to this, we suggest implementing a system or model for capture rate conversion which allows LSD-SLAM to normalize images to its "sweet spot" capture rate, allowing it to process more images. S-PTAM SLAM suffered from localization problems due to abrupt motion changes. Including an IMU for calibration in this particular workflow would allow for compensation for this. In the future, implementation of SLAM with more metrics (dedicated RMSE, processing time/power required, external device requirements (GPU, calibration devices, etc) would allow for easier quantification of algorithm success. Even potential implementation of a "SLAM Success Score" which weighted all metrics based on importance and returned a value on a normalized scale of efficiency would improve analytics. Better visualization through ROS allowing for overlay and mapping would also greatly improve ease of observation and speed for different SLAM algorithms. Finally, our usage of algorithms was restrained by the hardware we had access to (our best computer had 8GB RAM and an Apple M2 processor). Usage of computers with more computing power dedicated to the desired processes would allow for testing of denser data and feature detection, implementation of ML/DL/general CNNs and RNNs, and usage of dense point clouds.

5 Final Questions

5.1 The two or three things you learned during this project.

We learned more about back-end SLAM processing, specifically the importance of the pose graph and pose graph creation, updating, and optimization in mapping a 3D environment rapidly and accurately. The nature of our project also required us to familiarize ourselves with Ubuntu, Linux, and ROS.

5.2 Advice you would give to next year's CV students (and instructors).

Get started on HW and the CV final project early, you'll never know what will go wrong!

6 Sources

1. https://www.researchgate.net/publication/304163768_Simultaneous_Localization_And_Mapping_Present_Future_and_the_Robust-Perception_Age
2. https://www.mathworks.com/discovery/slam.html
3. https://www.cvlibs.net/datasets/kitti/
4. https://waymo.com/open/about/
5. https://github.com/luigifreda/pyslam
6. https://github.com/GiordanoLaminetti/SlamPy
7. https://arxiv.org/abs/1610.06475
8. https://jakobengel.github.io/pdf/engel14eccv.pdf
9. https://webdiis.unizar.es/ jcivera/papers/pire_etal_ras17.pdf
10. https://github.com/MichaelGrupp/evo
11. https://github.com/raulmur/ORB_SLAM2
12. https://ieeexplore.ieee.org/document/6126544
13. https://medium.com/data-breach/introduction-to-brief-binary-robust-independent-elementary-features-436f4a31a0e6
14. https://github.com/tum-vision/lsd_slam
15. https://ieeexplore.ieee.org/document/7139836
16. https://github.com/lrse/sptam

17. https://arxiv.org/pdf/1910.04755.pdf

18. https://www.robots.ox.ac.uk/ mobile/Papers/NewmanHoICRA05.pdf