Investigating the real-time performance by computing Utilization using ROS

Om Tiwari
Department of Informatik
RPTU Kaiserslautern-Landau
Kaiserslautern
tiwari@rptu.de

Amjad Haider
Grad. School Commercial Vehicle Tech.
RPTU Kaiserslautern-Landau
Kaiserslautern
ahaider@rptu.de

Vidit Goyal
Department of Informatik
RPTU Kaiserslautern-Landau
Kaiserslautern
goyal@rptu.de

August 26, 2023

Abstract

Recent advancements in autonomous vehicles such as delivery robots, maritime and commercial vehicles have resulted in renewed interest in the field of autonomous driving. One of the important aspects involves quantizing how well the technologies involved in autonomous driving execute and research in this field has been of vital importance. Simultaneous Localization and Mapping (SLAM) is an approach that enables machines to create maps and understand their location in real-time. Visual SLAM represents a distinct variant within the realm of SLAM systems, utilizing 3D vision to execute tasks of localization and mapping even in situations where both the sensor's location and the surrounding environment remain unfamiliar The goal of this paper is to document the execution of OV^{2}Slam Algorithm with a subset of KITTI data and also benchmark the results with boxplots and calculations involved. We also have a look at the utilization time of a chunk of KITTI data in OV^{2}Slam.

1 Introduction

Robotics, augmented reality, and autonomous vehicles have been at the forefront of technological evolution, significantly influenced by the capabilities of computational frameworks such as the Robot Operating System (ROS). Visual SLAM (Simultaneous Localization and Mapping) technology stands as a pivotal innovation in this domain, enabling camera-equipped devices to adeptly navigate unknown terrains while concurrently constructing maps. A notable contribution to this field is OV^{2}SLAM, an online VSLAM algorithm which stands out for its real-time capability, accuracy, and robustness. With its unique four-thread architecture comprising the Front End, Mapping, State Optimization, and Loop Closer, OV^{2}SLAM optimizes efficiency through multithreading, presenting state-of-the-art accuracy and real-time performance when juxtaposed against its counterparts.

In light of the above, this paper embarks on an in-depth exploration of real-time performance by computing utilization using ROS in tandem with OV^{2}SLAM. By incorporating the KITTI dataset, a benchmark in the autonomous driving domain, our methodology discerns the utilization of various processes in relation to the frame rate. This intersection of theoretical paradigms and practical applications aspires to provide developers and researchers with holistic insights. The rest of this paper first discusses related work in Section 2 which builds an understanding for evaluation metrics, and then describes our implementation in Section 4. Section 5 describes how we evaluated our system and presents the results. Section 6 presents our conclusions and describes future work.

2 Related Work

Several papers in the domain of robotic autonomy have proposed innovative methodologies to address challenges related to energy management and SLAM performance.

Towards A Multi-Mission QoS and Energy Manager for Autonomous Mobile Robots [1] pointed out several areas that, while notable, still had room for improvement when juxtaposed against our methods. One key methodology, energy-aware path and motion planning, proposed in previous works like [2] and [3], focuses on path optimization to conserve energy. However, they tend to overlook the constraints posed by the robot's limited computing resources. This oversight implies that despite following an energy-efficient trajectory, the robot could still confront computational challenges due to resource limitations. Furthermore, in [4], the emphasis on battery management compromises between the peril of battery exhaustion and the achievement of mission objectives. While the redirection of robots to docking stations for recharging is factored in, this method primarily pivots around energy management without adequately addressing limited computational resources, crucial for ensuring a robot's autonomy.

In another significant study, *Towards a Framework for SLAM Performance Investigation on Mobile Robots* [5], a method to assess two SLAM algorithms, Google Cartographer and Hector SLAM, was proposed. Utilizing ROS as middleware and Nvidia Jetson TX2 as the computational platform, they evaluated the algorithms based on metrics like accuracy, processing time, and hardware resource consumption (CPU and RAM usage). This analysis proved instrumental in our research. Their findings indicated Hector SLAM's superior accuracy compared to Cartographer, albeit at the cost of Cartographer's exceptional time performance, which demanded a higher CPU and RAM allocation.

3 Datasets used

The KITTI dataset [6], a cornerstone in computer vision research related to autonomous vehicles, encompasses a vast array of data. However, given the challenges posed by limited computational resources and storage capacities often encountered in academic settings, it's sometimes necessary to focus on smaller portions of this expansive dataset. In our study, we particularly concentrated on Dataset 0096. This subset retains a comprehensive collection of the raw images typically found in KITTI, including data from both left and right camera perspectives. While Dataset 0096 is more compact, it offers a representative sample of the larger dataset, ensuring that our analyses and findings maintain validity and relevance. The ability to work with such a subset empowers researchers to continue pushing the boundaries of knowledge without being hamstrung by technological constraints.

In our research, to thoroughly analyze the values of all frames within the Dataset 0096 of the KITTI data, we processed the dataset through OV^{2}Slam. This allowed us to extract precise measurements for each consecutive frame. Based on the extracted data, we then generated a boxplot to visually represent the distribution and variance of values across frames. Additionally, we computed descriptive statistics, identifying the minimum, maximum, and average values. This rigorous approach ensured a comprehensive understanding of the dataset's characteristics.

4 Implementation

Why did we choose ROS [7]?

The Robot Operating System (ROS) [7] has been a transformative middleware in the domain of robotics and autonomous systems. When considering its implementation, several reasons underscore why ROS is particularly suitable:

- Modularity: ROS's node-based architecture simplifies complex tasks, enabling efficient development and easier debugging.
- Language Independence: With support for multiple programming languages like Python and C++, developers can pick the best fit for their task.
- Open-Source Nature: Being open-source, ROS boasts a vast community that contributes to its continuous enhancement, diverse tools, and extensive documentation.
- Middleware Abstraction: ROS abstracts hardware details, allowing developers to focus on software without getting entangled with specific hardware intricacies.
- Interprocess Communication: Robust communication between nodes is facilitated by ROS's in-built system for passing messages and data seamlessly.

Installation of OV^{2}Slam

Introduction:
OV²SLAM is a real-time Visual SLAM algorithm suitable for both Stereo and Monocular cameras. It offers a comprehensive SLAM pipeline with features like Tracking, Mapping, Bundle Adjustment, and Loop Closure. The algorithm is designed with a multi-threaded architecture to ensure real-time performance.

Our installed dependencies:

1. **Operating System & Tools:** We tested OV^{2}Slam on Ubuntu 20.04.6, OpenCV 3, though not extensively tested.
2. **Programming Language:** The library utilizes C++11 features, so it requires a compiler supporting C++11 or higher.
3. **Dependencies:**
   - **Eigen3:** Used throughout OV²SLAM. Compatibility with version 3.3.0 or above is essential.
   - **OpenCV:** Developed with OpenCV 3. For BRIEF descriptor functionality, the installation of `opencv_contrib` is necessary. OpenCV 4 might also be compatible but hasn't been fully tested.
   - **iBoW-LCD:** An online Bag of Words method. A modified version is included in the Thirdparty folder.
   - **Sophus:** Used for SE(3) and SO(3) elements representation. A copy is provided in the Thirdparty folder for convenience.
   - **Ceres Solver:** Handles optimization tasks like PnP, Bundle Adjustment, and PoseGraph Optimization. A version is included in the Thirdparty directory.
   - **OpenGV (Optional):** Useful for Multi-View Geometry operations. If not installed, alternatives using OpenCV functions are available, though performance might vary.

Installation Process:

1. **ROS Workspace:** ROS-Noetic was chosen as the ROS distribution, a workspace was created with required folders.
2. **Repository Cloning:** Clone the OV²SLAM repository into your catkin workspace.

3. **Third-party Libraries:** A script is provided to facilitate the compilation of third-party libraries. This script builds obindex2, ibow-lcd, sophus, and ceres. If OpenGV is desired, it should be installed separately.

4. **Building OV²SLAM:** Once the third-party libraries are set up, OV²SLAM can be built using the catkin build tool.

- We have a chunk of data from the KITTI data that we have successfully managed to run in OV^{2}Slam, generated an ROS graph that visualizes the topics that are being subscribed to. Figure 1 shows a ROS graph generated by our method:

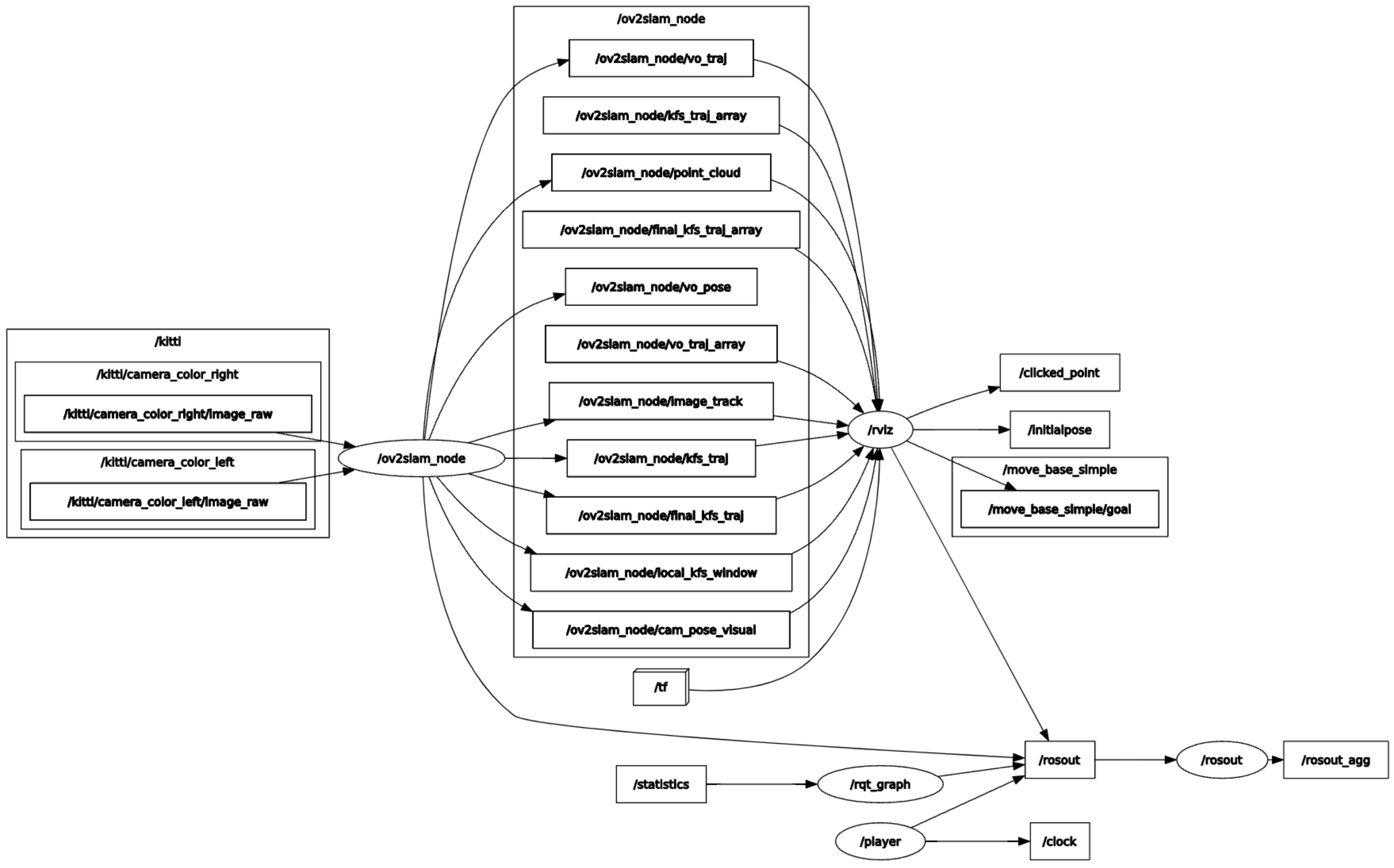
/ov2slam_node
/ov2slam_node/vo_traj
/ov2slam_node/kfs_traj_array
/ov2slam_node/point_cloud
/ov2slam_node/final_kfs_traj_array
/ov2slam_node/vo_pose
/ov2slam_node/vo_traj_array
/ov2slam_node/image_track
/ov2slam_node/kfs_traj
/ov2slam_node/final_kfs_traj
/ov2slam_node/local_kfs_window
/ov2slam_node/cam_pose_visual
/kitti
/kitti/camera_color_right
/kitti/camera_color_right/image_raw
/kitti/camera_color_left
/kitti/camera_color_left/image_raw
/ov2slam_node
/rviz
/clicked_point
/initialpose
/move_base_simple
/move_base_simple/goal
/tf
/rosout
/rosout
/rosout_agg
/statistics
/rqt_graph
/player
/clock

Figure 1: ROS graph Representation

- We managed to get a log files that was generated during the execution of the 96 record of KITTI data through the aid of launch file . The log file consists of various values for each frame from the data. From the log file, we computed the time taken for each process from each successive frames.

5 Evaluation

Obtaining the maximum, minimum and the average values from the log files

This script we made processes a specified log file to extract numerical timing details associated with different processes. It identifies both raw start times for processes and a summary of their time statistics, computing the time differences between consecutive start times. The script then calculates the maximum, minimum, and average differences for each process. Also, it extracts a summarized time statistic from the log's end section, which it then uses to compute and display the utilization ratios by comparing with the initially computed statistics. The table below shows the values respectively:

Table 1: Utilization Statistics for Various Processes. The table presents the utilization statistics for different processes, showcasing their average, minimum, and maximum values. It provides insights into the efficiency and performance of each process, enabling us to identify potential bottlenecks or areas of optimization.

Process Name	Average	Max	Min
0.Full-Front_End	0.095060	0.230849	0.020553
0.Keyframe-Processing_Mapper	0.033791	0.166336	0.004097
0.LC_ProcessKF	0.079148	0.740404	0.003095
1.BA_localBA	0.122130	0.976185	0.005719
1.BA_map-filtering	0.016737	0.090100	0.000942
1.FE_Track-Mono	0.051019	0.115907	0.020506
1.FE_createKeyframe	0.044586	0.182030	0.009212
1.KF_MatchingToLocalMap	0.007968	0.045931	0.000156
1.KF_TriangulateStereo	0.000281	0.001912	0.000026
1.KF_TriangulateTemporal	0.000207	0.002195	0.000008
1.KF_stereoMatching	0.017413	0.111471	0.001980
1.KF_updateFrameCovisilbity	0.003795	0.022414	0.000065
2.BA_L2-Refinement	0.040357	0.271293	0.001773
2.BA_Optimize	0.059630	0.608397	0.002693
2.BA_SetupPb	0.016571	0.072633	0.000859
2.BA_Update	0.002625	0.034062	0.000114
2.FE_CF_extractKeypoints	0.043987	0.180343	0.009077
2.FE_CF_prepareFrame	0.000298	0.014873	0.000017
2.FE_TM_EpipolarFiltering	0.024502	0.053645	0.011439
2.FE_TM_KLT-Tracking	0.013027	0.066186	0.006514
2.FE_TM_checkNewKfReq	0.000461	0.001304	0.000237
2.FE_TM_computePose	0.009597	0.054400	0.004221
2.FE_TM_preprocessImage	0.006274	0.032390	0.003928

Visualizing the values

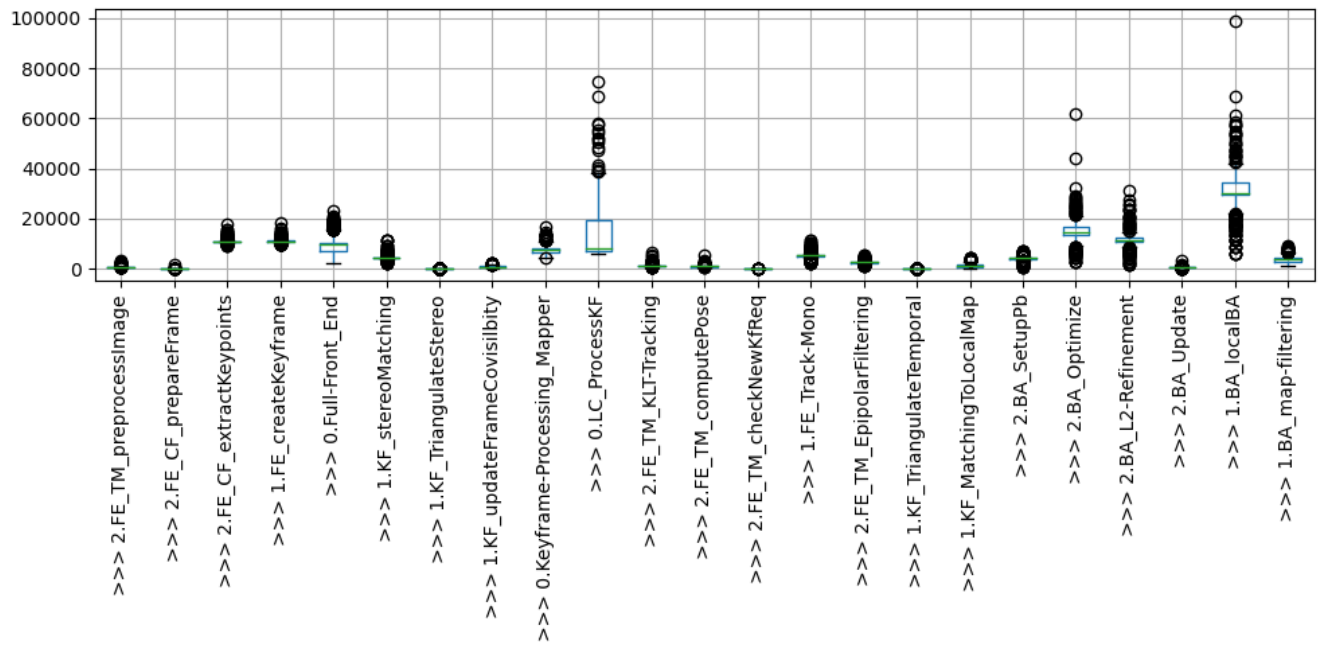

Figure 2: Visualization of values using boxplot

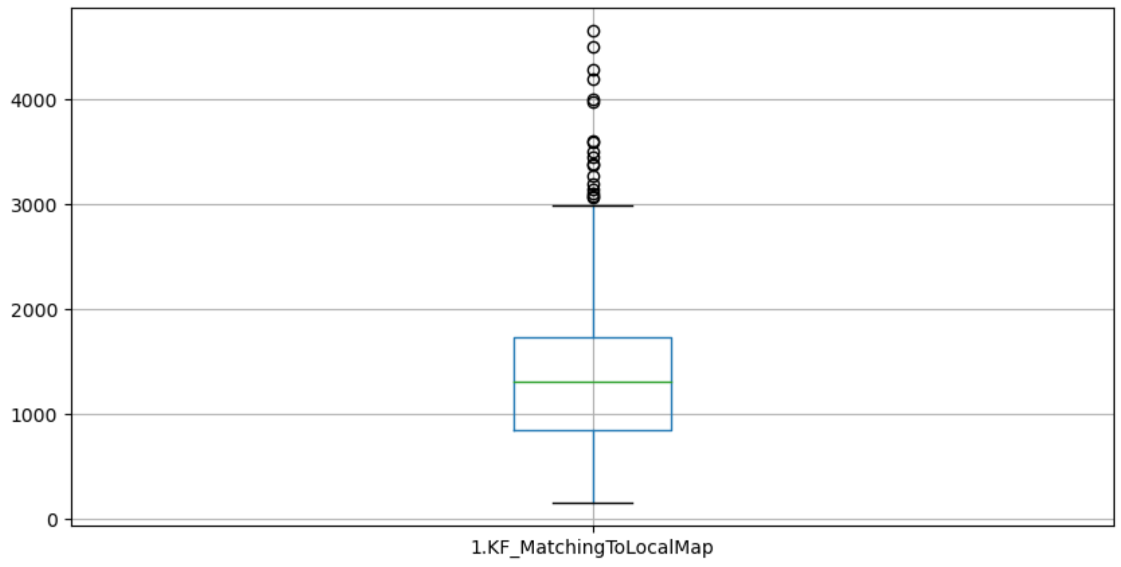

Figure 3: Visualization single parameter using boxplot

From the above values, it is evident that thread 0 Full-Front_End has an average utilization of 0.095. Furthermore, thread 2 is burdened with BA_localBA process with an average utilization of 0.122.

6 Results

The paper documents an insightful study on the OV^{2}Slam, a Visual SLAM algorithm that is suitable for real-time capability in terms of accuracy and robustness. This is evident from the analysis of the utilization calculated for the four threads namely the visual front-end, mapping, state optimization and loop closing threads. Furthermore, OV^{2}Slam was not easily impacted by the runtime requirements of the algorithm. Its adaptability to dynamic environments for task execution with various threads. As work on VSLAM algorithms continue to progress,OV^{2}Slam can be utilized for its capabilities in autonomous navigation .This study has analysed the resource competition between different threads with respect to its behaviour and will aid in further studies for resource allocation.

References

[1] D.-K. Ho, K. B. Chehida, B. Miramond, and M. Auguin, "Towards a multi-mission qos and energy manager for autonomous mobile robots," in *2018 Second IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2018, pp. 270–273.

[2] M. Rappaport, "Energy-aware mobile robot exploration with adaptive decision thresholds," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*. VDE, 2016, pp. 1–8.

[3] B. Zhang, L. Tang, J. DeCastro, M. Roemer, and K. Goebel, "Autonomous vehicle battery state-of-charge prognostics enhanced mission planning," *International Journal of Prognostics and Health Management*, vol. 5, no. 2, 2014.

[4] V. Berenz and K. Suzuki, "Risk and gain battery management for self-docking mobile robots," in *2011 IEEE International Conference on Robotics and Biomimetics*. IEEE, 2011, pp. 1766–1771.

[5] D.-T. Ngo and H.-A. Pham, "Towards a framework for slam performance investigation on mobile robots," in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, 2020, pp. 110–115.

[6] Y. Liao, J. Xie, and A. Geiger, "Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 3, pp. 3292–3310, 2022.

[7] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.