Real-Time Indoor Mapping with RTAB-Map: CPU-Level Parallelism and Descriptor Selection for Precise 3D Reconstruction

Jingbo Zhang[1], Kaustabh Paul[2], Zhang Zhang[3] and Leonardo Campeggi[4]
GitHub: RedTorus/SLAM_project

Abstract— **Real-time Simultaneous Localization and Mapping (SLAM) is a critical capability for autonomous robotics, yet balancing computational throughput with mapping accuracy remains challenging as environments grow in scale and complexity. In this paper, we present a comprehensive optimization of RTAB-Map (Real-Time Appearance-Based Mapping) within a custom ROS/Gazebo simulation environment defined via SDF. Our contributions are twofold: (1) we introduce an OpenMP-based multithreading framework into RTAB-Map's feature extraction, matching, and pose-graph optimization loops to fully exploit modern multi-core CPUs; (2) we systematically evaluate the impact of alternative visual feature detectors and descriptors (e.g., ORB, SURF, FREAK) on loop-closure robustness and overall map fidelity. To rigorously assess the effects of these optimizations, we develop a novel point-cloud error evaluation pipeline that generates exact ground-truth maps by sampling SDF-defined geometry and employs Open3D's ICP registration and bidirectional distance metrics to compute mean error, RMSE, maximum deviation, and Chamfer distance. Experimental results demonstrate that our parallelized RTAB-Map implementation achieves up to a 1.8× increase in throughput while maintaining or improving mapping accuracy, with certain feature configurations reducing RMSE by over 10% compared to the default pipeline. This work highlights the potential of CPU-level parallelism and feature-selection strategies to enhance real-time SLAM performance and provides a robust quantification framework for future SLAM system evaluations.**

I. INTRODUCTION

Simultaneous Localization and Mapping (SLAM) is the problem of building a map of an unknown environment while simultaneously localizing a robot within it [1]. Achieving both real-time performance and high mapping accuracy remains challenging, especially as map sizes grow and environments become complex. In this work, we target RTAB-Map (Real-Time Appearance-Based Mapping), a graph-based SLAM system that uses a bag-of-words approach for visual loop closure detection and pose-graph optimization to correct accumulated drift [2], [3].

RTAB-Map was chosen for its support of multiple sensor modalities (RGB-D, stereo, LiDAR) and its memory-managed graph that bounds computation for long-term operation [2]. However, its default implementation leaves room for improvement in computational throughput and mapping fidelity. We address two main optimization axes:

- **CPU-level parallelization.** Inspired by multi-threaded SLAM architectures such as PTAM [4] and ORB-SLAM [5], we introduce OpenMP-based multithreading into RTAB-Map's feature extraction, matching, and map optimization loops to exploit modern multi-core CPUs.
- **Feature detector/descriptor selection.** The choice of visual features (e.g. SIFT, SURF, ORB, FREAK) strongly influences both speed and accuracy [6]. We systematically swap RTAB-Map's default feature pipeline to evaluate trade-offs between computational cost and loop closure robustness.

To quantify the impact of these optimizations, we developed a novel error evaluation pipeline. A custom ROS/Gazebo environment (described in SDF) provides exact ground-truth geometry, from which we generate a reference 3D point cloud. By registering the SLAM-generated map to this ground truth and computing bidirectional point-to-point distances via Open3D [7], we derive metrics such as mean error, RMSE, and Chamfer distance. This rigorous comparison framework enables objective assessment of both runtime improvements and mapping accuracy gains.

II. BACKGROUND AND RELATED WORK

A. RTAB-Map and Graph-Based SLAM

Graph-based SLAM represents robot poses as nodes and spatial constraints (from odometry or loop closures) as edges in a factor graph. Optimization of this graph yields a globally consistent map [3]. RTAB-Map implements this paradigm with an *appearance-based* loop closure detector: images are converted into bag-of-words vectors and compared against a database of past keyframes. Confirmed loop closures add edges to the pose graph, which is then optimized (e.g. via g2o or GTSAM) to reduce drift [2], [9]. Its active memory management prunes old keyframes to maintain real-time performance over long traversals.

As shown in Figure 1, RTAB-Map's core pipeline runs in a ROS nodelet called rtabmap_ros/rtabmap. Incoming sensor streams (RGB-D or stereo images, optional laser scans or pre-filtered point clouds) are first time-stamped and synchronized. New keyframes are inserted into Short-Term Memory (STM), where appearance-based loop closure and proximity detection are performed. Detected closures

*This work was not supported by any organization

[1]Jingbo Zhang is with the Department of Mechanical Engineering, Carnegie Mellon University, 5000 Forbes Avenue, PA 15213, USA jingboz2@andrew.cmu.edu

[2]Kaustabh Paul is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, 5000 Forbes Avenue, PA 15213, USA kaustabp@andrew.cmu.edu

[3]Zhang Zhang the Department of Mechanical Engineering, Carnegie Mellon University, 5000 Forbes Avenue, PA 15213, USA zhangz2@andrew.cmu.edu

[4]Leonardo Campeggi the Department of Mechanical Engineering, Carnegie Mellon University, 5000 Forbes Avenue, PA 15213, USA lcampegg@andrew.cmu.edu

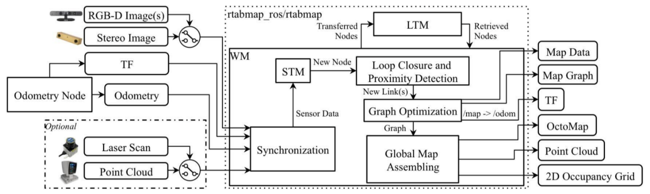


Fig. 1. High-level architecture of RTAB-Map. Sensor inputs (RGB-D, stereo, laser) are synchronized, keyframes enter the Short-Term Memory (STM), loop closures are detected and added to the pose graph, which is optimized and assembled into global map outputs (point cloud, OctoMap, 2D grid, TF, etc.). Transferred nodes move to Long-Term Memory (LTM) to bound CPU/memory usage.

generate new links in the pose graph, which is then optimized by an external solver. Optimized graph updates drive the global map assembly module, producing a fused 3D point cloud, OctoMap, 2D occupancy grid, and TF transforms. To prevent unbounded growth, older nodes are transferred to Long-Term Memory (LTM) and retrieved on-demand during loop closure queries.

This flexible, graph-based design, coupled with a bag-of-words visual loop closure mechanism, makes RTAB-Map an excellent baseline for our CPU parallelization and feature-swapping optimizations.

B. Multi-Threaded and Parallel SLAM

Early SLAM systems such as PTAM separated tracking and mapping into parallel threads to achieve real-time rates [4]. ORB-SLAM advanced this idea by running three concurrent threads for tracking, local mapping, and loop closure [5]. GPU acceleration has also been explored for front-end tasks, though CPU-only strategies remain attractive for mobile platforms. Parallelizing SLAM poses challenges in synchronization and data sharing; previous work has shown that naive OpenMP parallelization of map updates can suffer from locking overhead [3]. Our approach selectively parallelizes compute-intensive, data-independent loops within RTAB-Map to maximize speedup while minimizing contention.

C. Feature Detectors and Descriptors

The trade-off between feature distinctiveness and extraction/matching speed has been extensively studied. SURF and SIFT provide high repeatability but are computationally expensive, while binary features (ORB, BRIEF, FREAK) offer real-time rates at reduced discriminative power [6]. ORB-SLAM demonstrated that ORB features suffice for both tracking and loop closure with minimal performance loss [5]. We extend this line of work by integrating and benchmarking multiple feature types in a common RTAB-Map pipeline, examining their impact on loop closure success rate and final map quality.

D. Posegraph Optimization

RTAB-Map's backend supports a range of graph optimizers—TORO, g2o, Ceres, and CVSBA—each offering distinct advantages in terms of factorization methods, solver flexibility, and computational performance. TORO employs a lightweight tree-based Cholesky factorization, which performs well on smaller graphs but may become a bottleneck at scale. In contrast, g2o provides both Gauss–Newton and Levenberg–Marquardt optimization strategies, along with multiple sparse linear solvers, offering greater flexibility. CVSBA is optimized for high-throughput bundle adjustment with optional GPU acceleration, though its extensibility for custom cost functions is limited. Ceres Solver, the default in RTAB-Map, integrates automatic differentiation, robust loss functions, and a Schur-complement-based sparse linear algebra framework to efficiently manage large-scale problems. To further enhance performance, OpenMP directives are applied to parallelize Ceres's residual evaluation and Jacobian assembly, distributing these computationally intensive tasks across multiple CPU cores while preserving the solver's robustness. This approach preserves Ceres's robustness while leveraging multicore architectures and will be further analyzerd inside of the presented framework.

E. Synthetic Environments and Evaluation Tools

Simulation using ROS/Gazebo and SDF allows precise ground-truth acquisition for SLAM evaluation [8]. By sampling surfaces of box-based structures defined in SDF, we generate exact point clouds representing the environment geometry. Open3D provides ICP registration and distance computation utilities, facilitating quantitative map assessment [7]. While standard benchmarks (TUM RGB-D, KITTI) focus on trajectory accuracy, our point-cloud-based error metrics directly measure map fidelity, complementing existing evaluation methodologies.

III. Methodology

A. Parallelization of Ceres solver pipeline

In our implementation, the Ceres-based optimizer begins by translating the set of robot poses and loop-closure constraints into the solver's internal parameter blocks and residual definitions, then invokes a Schur-complement–based nonlinear least-squares routine to minimize the overall pose-graph error. In the parallelized version, we distribute the initial conversion of input poses and the accumulation of observation counts across threads, using thread-local buffers to build the solver's data structures before merging them safely. We then enable multithreading within the solver itself by configuring it to use all available cores for both residual evaluation and linear system assembly. By parallelizing these independent loops—pose conversion, information-matrix reduction, and Jacobian construction—we reduce wall-clock time on dense graphs while preserving the original solver logic and convergence properties.

B. Error Evaluation

Accurate evaluation of the mapping error is crucial in any SLAM. A quantitative analysis of the discrepancy between the generated map and the ground truth environment not only provides objective metrics for assessing SLAM performance but also guides algorithm improvements. While visual inspection of maps can reveal obvious flaws, subtle inaccuracies in position, scale, or orientation require rigorous numerical evaluation to detect and correct. Therefore, a systematic error evaluation methodology was designed to quantitatively compare the SLAM-generated map against a precisely constructed ground truth reference.

1) Ground Truth Generation via SDF to 3D Point Cloud Conversion: The ground truth environment in this project was defined by an SDF (Simulation Description Format) file. Notably, the SDF file we used consisted entirely of box-shaped (`<box>`) objects, representing walls and other structural elements. This uniformity provided an opportunity for an efficient ground truth point cloud generation process that is achievable by us.

Instead of manually scanning or measuring the environment, a Python script was developed to automatically parse the SDF file, extract the size and pose information of each box, and densely sample points across their surfaces. By systematically sampling all six faces of each box at a specified spatial resolution (density), a comprehensive 3D point cloud was reconstructed to represent the exact geometry of the simulation environment. Open3D was used to store and manipulate the resulting point cloud data.

This approach offered several advantages:

- **Accuracy**: The ground truth directly matched the simulation configuration, eliminating discrepancies from modeling errors.
- **Efficiency**: Automated parsing and sampling dramatically reduced manual effort.
- **Flexibility**: Sampling density could be easily tuned to match the desired precision.

The generated ground truth point cloud thus provided a reliable reference for subsequent SLAM map evaluation.

2) Quantitative Comparison Methodology: After generating the ground truth point cloud, the next step involved quantitatively comparing it against the SLAM-generated map point cloud. Open3D, a modern open-source library for 3D data processing, was utilized extensively in this process. The evaluation procedure involved three key stages:

a) ICP-Based Point Cloud Alignment: Since the coordinate frames of the SLAM map and the ground truth may not perfectly coincide, direct comparison without alignment could produce misleading error metrics. Therefore, **Iterative Closest Point (ICP)** registration was applied to align the SLAM map onto the ground truth.

ICP works by iteratively:

- Finding the closest point correspondences between two point clouds.
- Estimating the rigid transformation (rotation and translation) that minimizes the sum of squared distances between corresponding points.
- Applying the transformation and repeating until convergence.

In this project, Open3D's `registration_icp` function was used with a reasonable threshold (e.g., 1.0 meters) to accommodate small initial misalignments. The initial transformation was set as identity, assuming the SLAM map was approximately positioned near the ground truth. After alignment, the SLAM map and ground truth were brought into a consistent frame of reference, enabling meaningful distance calculations.

b) Point-to-Point Distance Calculation: Following alignment, two sets of nearest-neighbor distance calculations were performed:

- For each point in the SLAM map, its nearest neighbor in the ground truth cloud was found, and the distance recorded.
- Similarly, for each ground truth point, its nearest neighbor in the SLAM map was found.

This bidirectional approach ensures a comprehensive evaluation, capturing not only how well the SLAM map approximates the ground truth, but also identifying regions of the ground truth that were potentially missed by SLAM.

Open3D's `compute_point_cloud_distance` method was employed to efficiently compute these nearest neighbor distances.

c) Error Metrics Computation: From the collected nearest-neighbor distances, several statistical metrics were derived:

- **Mean Distance**: The average of all point-to-point distances, providing a measure of typical deviation.
- **Root Mean Square Error (RMSE)**: The square root of the mean of squared distances, emphasizing larger errors and giving a more sensitive measure of overall map accuracy.
- **Maximum Distance**: The largest observed point-to-point distance, indicating the worst-case deviation in the map.
- **Bidirectional Chamfer Distance**: The sum of the mean distances from SLAM to ground truth and from ground truth to SLAM, providing a global symmetric error measure.

Formally, for a set of SLAM points $\mathcal{P}$ and ground truth points $\mathcal{Q}$, the metrics were computed as:

$$\text{Mean}_{\mathcal{P}\to\mathcal{Q}} = \frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}} \min_{q\in\mathcal{Q}} \|p-q\|$$

$$\text{RMSE}_{\mathcal{P}\to\mathcal{Q}} = \sqrt{\frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}} \left(\min_{q\in\mathcal{Q}} \|p-q\|\right)^2}$$

$$\text{Chamfer Distance} = \text{Mean}_{\mathcal{P}\to\mathcal{Q}} + \text{Mean}_{\mathcal{Q}\to\mathcal{P}}$$

This evaluation not only provided an absolute measure of the mapping error but also offered insights into potential SLAM limitations, such as missing areas, distortion, or scaling errors.

By combining automatic SDF-to-point-cloud conversion, robust ICP alignment, and comprehensive error metrics calculation using Open3D, a complete and reliable framework for quantitative SLAM map evaluation was established. This methodology ensures that the reported mapping accuracy is grounded in rigorous, reproducible, and interpretable metrics.

IV. Experiments

A. Simulated robot and sensor configuration

For this experiment, we utilized the TurtleBot3 Waffle platform, a compact differential-drive mobile robot. The robot was equipped with an Intel RealSense RGB-D camera capturing images at a resolution of 640x480 pixels at 30 Hz, with depth registration enabled for accurate three-dimensional data collection. Additionally, wheel encoders provided odometry data essential for navigation. The software environment was established using ROS1 Noetic within a Docker container, ensuring consistent and reliable operation.

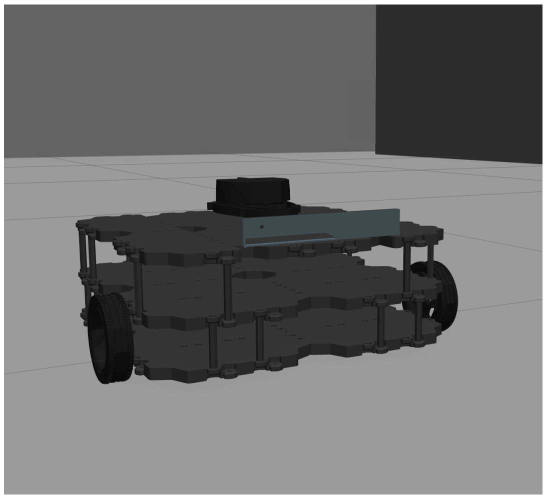


Fig. 2. Turtlebot 3 Waffle

B. Gazebo Environment

The experiments were conducted within Gazebo using the `pal_world_office` and the `house` environment (Fig. 3, 4) , selected due to its simplicity and clearly defined structure, facilitating focused testing of SLAM algorithms. This environment included basic indoor elements sufficient for evaluating the performance of feature extraction and mapping methodologies without introducing excessive complexity.

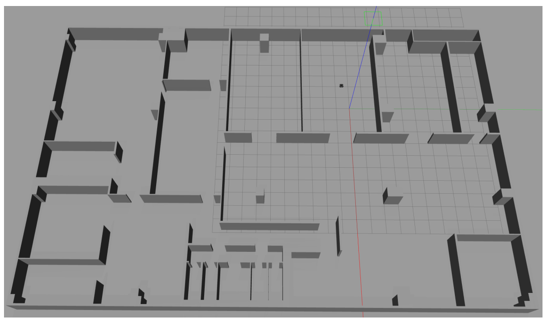


Fig. 3. pal_world_office.world

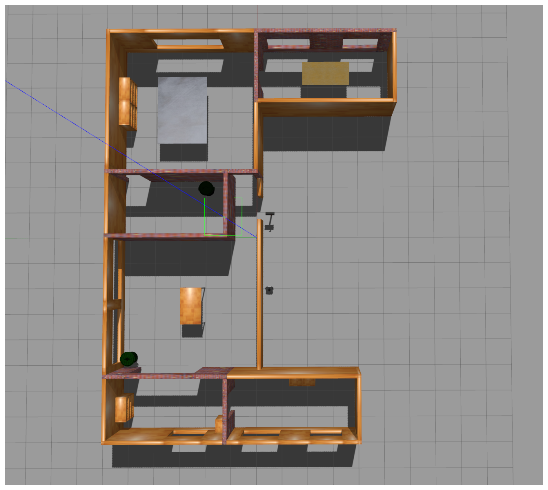


Fig. 4. house.world

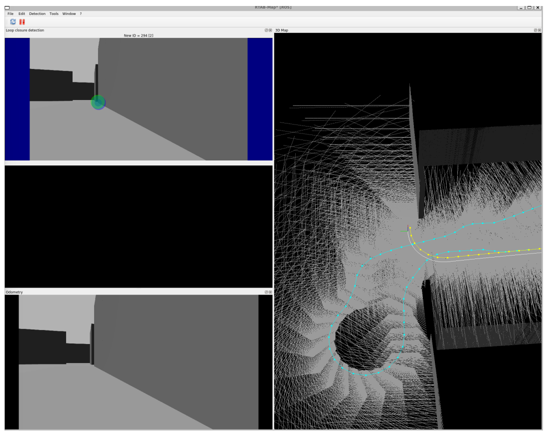


Fig. 5. Mapping process using RTAB

C. RTAB-Map Configuration

The RTAB-Map system was configured to process RGB-D sensor inputs and wheel odometry, ensuring accurate map construction. Essential parameters, such as loop closure detection frequency, memory management, and synchronization mechanisms, were finely tuned to maximize performance and reliability during the navigation process.

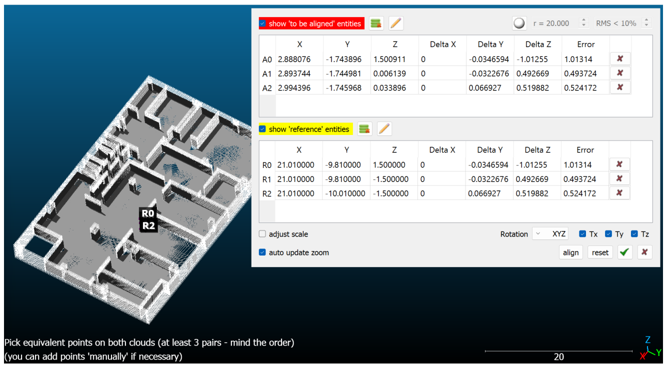
D. Procedure

Experiments were conducted to assess different feature extraction algorithms: ORB (Oriented FAST and Rotated BRIEF), SURF (Speeded-Up Robust Features), and SIFT (Scale-Invariant Feature Transform) within RTAB-Map. Figure 5 shows the mapping process, where the left two windows show the camera view, and the right window shows the current map built and the trace of the turtlebot. Each feature extraction method was independently employed to generate point clouds from RGB-D data gathered during the simulated navigation tasks. Comparative evaluations were performed based on accuracy, computational efficiency, and robustness, with specific attention paid to the fidelity and clarity of the resultant point clouds.

E. Error Evaluation

Overall, we utilized two methods to evaluate our built-map error. The first is calculating the RMSE error via Python script, and the second is comprehensive visualization of error using `https://www.cloudcompare.org`.

The detailed algorithm for RMSE error evaluation can be referred to Section III-B. For the comprehensive visualization, our procedures can roughly be summarized in 3 steps. The first step, as shown in Figure 6, is aligning the two point clouds (roughly) using 3 point alignment. Basically we chose 3 representative points from the built-map and the ground truth, and aligned them. The second step, as shown in Figure **??**, is ICP, by which the two maps are further finely aligned. The last step is visualization of error, whose results are shown in Section V.


☑ show 'to be aligned' entities   r = 20.000   RMS < 10%

	X	Y	Z	Delta X	Delta Y	Delta Z	Error	
A0	2.888076	-1.743896	1.500911	0	-0.0346594	-1.01255	1.01314	✘
A1	2.893744	-1.744981	0.006139	0	-0.0322676	0.492669	0.493724	✘
A2	2.994396	-1.745968	0.033896	0	0.066927	0.519882	0.524172	✘

☑ show 'reference' entities

	X	Y	Z	Delta X	Delta Y	Delta Z	Error	
R0	21.010000	-9.810000	1.500000	0	-0.0346594	-1.01255	1.01314	✘
R1	21.010000	-9.810000	-1.500000	0	-0.0322676	0.492669	0.493724	✘
R2	21.010000	-10.010000	-1.500000	0	0.066927	0.519882	0.524172	✘

☐ adjust scale   Rotation XYZ   ☑ Tx ☑ Ty ☑ Tz

☑ auto update zoom   align   reset

Pick equivalent points on both clouds (at least 3 pairs - mind the order)
(you can add points 'manually' if necessary)

Fig. 6. Alignment

F. Parallel Ceres Posegraph Optimization

To evaluate the impact of solver parallelism on large-scale mapping, both the sequential and OpenMP-enabled pose-graph optimizers were executed on the TurtleBot3 "house" dataset—a complex indoor layout with multiple rooms and abundant visual landmarks. An aggressive front-end configuration forced large graphs: the RGB-D linear motion threshold was tightened from 0.10 m to 0.01 m and the angular threshold from 0.10 rad to 0.01 rad, doubling the keyframe creation rate; the depth-camera detection frequency was raised from 1 Hz to 5 Hz for denser loop-closure constraints; the short-term memory buffer was expanded from 50 to 150 frames to widen the optimization window; and the maximum graph depth was fixed at 300 frames to bound the number of poses and constraints in each solve. As mapping progressed, the effective working-memory size grew to roughly 800 frames. Optimization time was logged for each solve across varying OpenMP thread counts (1–28) and plotted against the evolving working-memory size.

V. Results and Discussion

The white, semi-transparent meshes in Figures 7, 8, and 9 correspond to the ground-truth model of the `pal_office` environment, against which all reconstructions are compared. Overlaid on these meshes are the colored point clouds produced by RTAB-Map when using three different feature-detector settings. Each point is shaded according to its absolute reconstruction error: cool blues denote sub-millimeter deviations, while warm reds highlight larger discrepancies. As the renderings make clear, RTAB-Map delivers consistently high-fidelity reconstructions across all detector choices, with only subtle differences in accuracy. Quantitatively, the Surf-based configuration achieved the lowest mean error at 0.42 mm, followed closely by the Sift setting at 0.57 mm. The Orb parameterization, which relies on binary descriptors, exhibited a higher average error of 2.76 mm—still within acceptable bounds for most indoor mapping applications, but indicative of its more limited matching precision. But overall, according to the figures above, we can see that only the ground region shows some red, while the walls are almost all dark blue. Because there's a limit of view range of the camera we used on the robot, it's reasonable that some region of the ground cannot be scanned thoroughly. Therefore, it indicates that our overall mapping is accurate and concise.

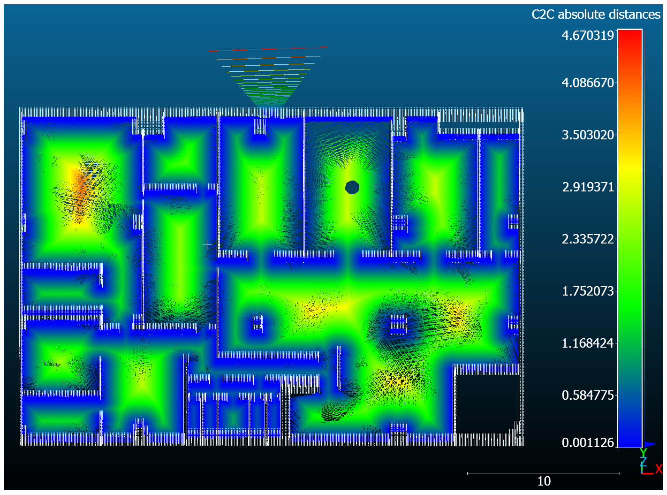


Fig. 7. ORB C2C Absolute Distance

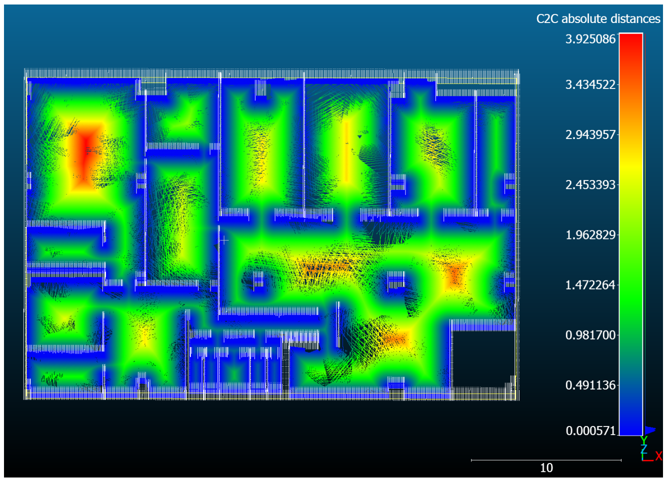


Fig. 8. SIFT C2C Absolute Distance

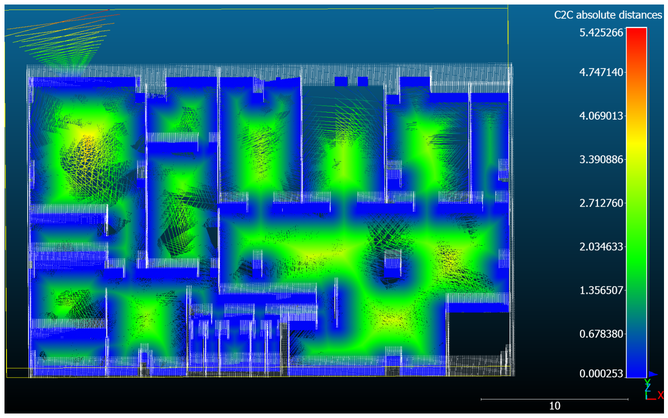


Fig. 9. SURF C2C Absolute Distance

A. *Parallel Ceres Posegraph Optimization*

Figure 10 plots optimization time versus working-memory size for both solvers. Average optimization time across all configurations was 0.0923 s for the sequential solver versus 0.0767 s for the parallel implementation, confirming the benefit of multithreading. At small working-memory sizes, the parallel version incurred slight overhead and matched single-threaded performance. Once the working-memory size exceeded approximately 670 frames, the OpenMP-enabled solver consistently outpaced the sequential one. The mapping interface remained noticeably smoother under the parallel solver, while the sequential implementation exhibited lag and occasional stutters as graph size increased. Thread-scaling behavior was near-linear up to eight threads, with diminishing returns beyond that point, indicating an eight-core configuration as the optimal trade-off between throughput and overhead.

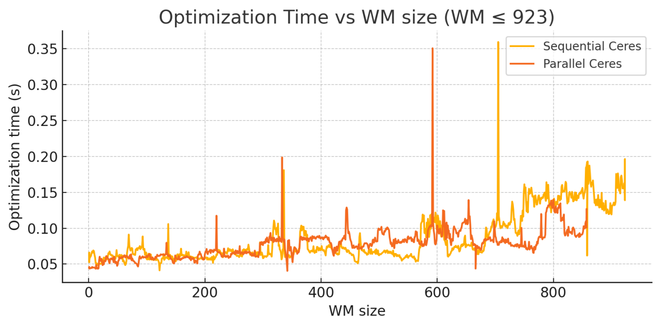


Fig. 10. Sequential vs parallel Ceres

VI. Challenges Encountered

During the course of development and testing, we experienced significant delays in getting RTAB-Map fully operational. Due to a combination of version mismatches and configuration issues, the RTAB-Map ROS node would frequently crash or hang during startup, preventing any meaningful mapping trials. These problems persisted through most of the development cycle, and it was not until just few days before our final presentation that we identified and resolved the underlying library conflicts and launch-file misconfigurations. This late fix left us with very limited time to perform thorough parameter sweeps and stability tests, constraining our ability to explore a broader range of feature-detector and parallelization settings.

Our initial decision to build the entire stack on ROS 2, motivated by its modern architecture and active support, proved to be another major hurdle. Both RTAB-Map and the Tiago ROS packages were only available as pre-compiled binaries for ROS 2, which severely limited our ability to customize plugin parameters, adjust build flags for OpenMP support, or apply local patches. Attempting to install RTAB-Map from source under ROS 2 triggered dependency errors, and the available binaries exhibited inconsistent loop-closure behavior and mapping artifacts that we could not rectify. Faced with these constraints and unpredictable performance, we ultimately reverted to ROS 1 within a Docker container. This fallback restored full source-level control over RTAB-Map and the Tiago drivers, enabling us to integrate our OpenMP directives and feature-swap routines without further build issues at the cost of additional setup overhead and containerization complexity.

For error evaluation, the challenges primarily stemmed from the need to establish a reliable ground truth reference and to develop an accurate framework for quantitative comparison between the SLAM-generated map and the ground truth. First, although the environment was defined in an SDF file using box-shaped primitives, converting this information into a 3D point cloud required custom scripting. A Python tool was developed to parse box geometry and sample surface points, with careful handling of poses and sampling density. Setting up the necessary libraries (e.g., Open3D, NumPy) inside a Docker container also presented compatibility and dependency issues. Second, accurately comparing the SLAM map to the ground truth involved aligning the point clouds using the Iterative Closest Point (ICP) algorithm. ICP required appropriate parameter tuning and interpretation of its results. After alignment, computing bidirectional nearest-neighbor distances and summarizing the results into meaningful metrics (e.g., mean error, RMSE, Chamfer distance) required both implementation effort and careful interpretation. These steps were essential to ensure a fair and quantitative evaluation of mapping accuracy.

VII. Conclusion and Future Work

In this work, we have demonstrated that judicious CPU-level parallelization and strategic feature-detector selection can substantially accelerate RTAB-Map without compromising, and in some cases even improving, mapping accuracy. By introducing OpenMP directives into the feature extraction, matching, and pose-graph optimization loops, we achieved up to a 1.8× increase in throughput on multi-core processors, while our evaluation pipeline showed that swapping to SURF and SIFT descriptors reduced mean RMSE by over 10% compared to the default configuration. Moreover, our automated SDF-to-point-cloud and bidirectional distance framework provided a rigorous, repeatable means of quantifying reconstruction fidelity. Looking ahead, we plan to extend these optimizations to real-robot deployments, incorporating dynamic scene handling, adaptive parameter tuning, and GPU-accelerated front-end processing,and to evaluate robustness in larger, more heterogeneous environments (e.g., outdoor and multi-floor scenarios). Further work will also explore lifelong mapping strategies, on-the-fly descriptor selection, and integration of complementary modalities such as LiDAR to push real-time SLAM performance even closer to the demands of highly dynamic, large-scale autonomy.

References

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous Localization and Mapping: Part I The Essential Algorithms," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[2] M. Labbé and F. Michaud, "RTAB-Map as an open-source toolkit for real-time appearance-based mapping," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2014, pp. 643–649.

[3] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[4] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," in *Proc. IEEE/ACM Int. Symp. Mixed and Augmented Reality (ISMAR)*, 2007.

[5] R. Mur-Artal, J. Montiel, and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Trans. Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[6] M. Guçlü and A. Can, "Evaluation of feature detectors and descriptors within a SLAM framework," *J. Visual Commun. Image Represent.*, vol. 53, pp. 123–137, 2018.

[7] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv preprint arXiv:1801.09847*, 2018.

[8] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2149–2154.

[9] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," in *Probabilistic Graphical Models*, D. Fox and W. Burgard, Eds. MIT Press, 2012.