Creating Better Environment Awareness For Racing Cars through SLAM

Siddharth Saha
Halıcıoğlu Data Science Institute
University of California, San Diego
La Jolla, CA, 92093
sisaha@ucsd.edu

Jay Chong
Halıcıoğlu Data Science Institute
University of California, San Diego
La Jolla, CA, 92093
jac269@ucsd.edu

Youngseo Do
Halıcıoğlu Data Science Institute
University of California, San Diego
La Jolla, CA, 92093
y1do@ucsd.edu

Abstract

The focus of this paper is the localization of a car in a racetrack. What this means is giving cars the ability to understand their environment much like how we humans do. We will be testing out the following SLAM (Simultaneous Localization and Mapping) open source package: RTABMAP in order to determine which package enables our car to learn the most about the environment in a reasonable period of time. It will be judged on 3 criterias:

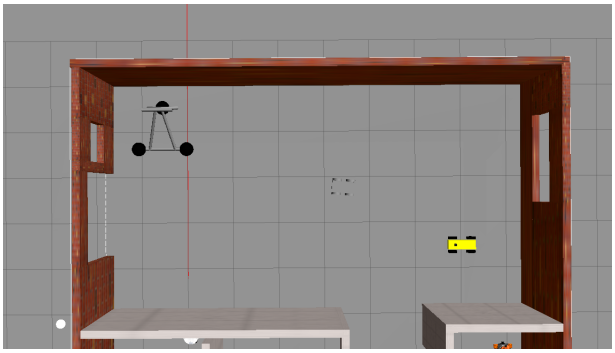
- The visual of the map generated and how close it is to the path traversed
- The reliability of the algorithm in how close it estimate it's location to the ground truth/actual position

1 Introduction

Creating maps is one of the most important tools a car has at it's disposal during autonomous driving. If I tell a car to move from point A to point B without giving it any other information a lot of things can go wrong. The car may waste time running into dead ends and going back to previous points to find the right path. If the car is moving at high enough speeds (such as in the racing track problem we are dealing with) it could very well crash before realizing this was not the path it should take. But what if the robot already knew the layout? It knows where point A is, where point B is and what the environment looks like in the area between A and B. This makes the situation a whole lot safer. The robot now knows what are the possible paths. It can even plan a path based off a heuristic. In fact, one of the most popular path planning algorithms, Pure Pursuit draws points on this map and makes the car move along those points to reach point B. Another popular algorithm, Rapidly Exploring Random Tree randomly draws paths in the map until it finds the shortest path to it's goal given the obstacles in the current map. All this is possible only if the robot has an idea of the environment around it. One of the most popular approaches in creating a map and giving the car an environment awareness is SLAM. SLAM stands for Simultaneous Localization and Mapping. The robot uses current sensor input to draw maps and once it draws maps it figures out what is its position in reference to this map. This has important use cases even outside the racing track as it can be used in the real world with pre-established maps to navigate between places

2 Dataset

For this project we will be generating our own data. This will be done by controlling a car and having it drive around a simulated environment. We use a simulated environment as opposed to a real world scenario to control the costs and risks associated with the real world scenario (for example if our car crashed then that could possibly damage some wiring or sensor). We carry out this simulation with the help of ROS Gazebo. You can see an example of a simulation environment below



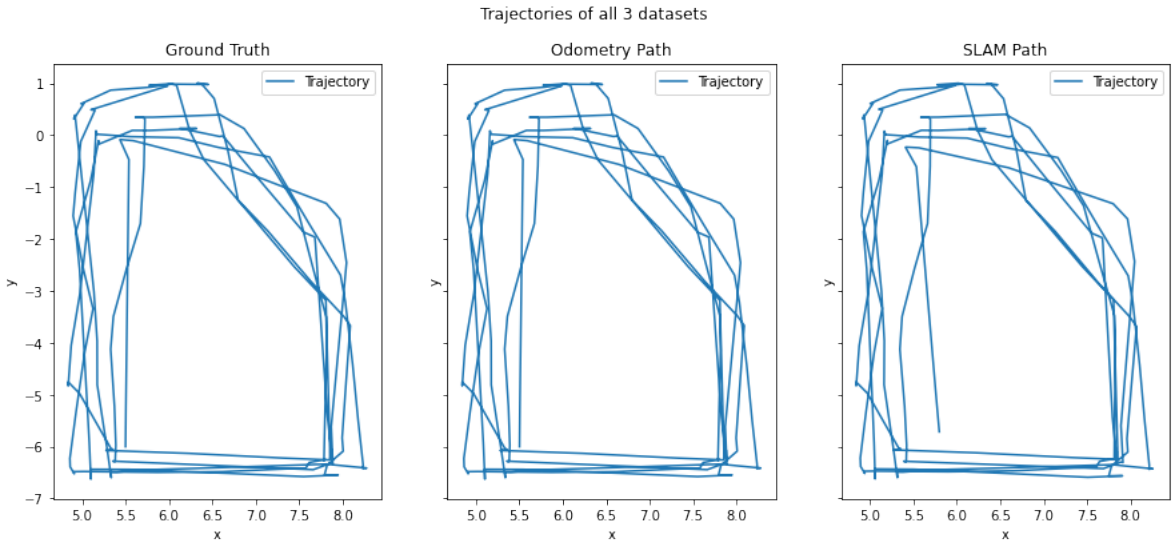
Another advantage of using a simulation environment is the presence of a ground truth position. The ground truth position is the true position of the robot at any given point in time. This is extremely useful in SLAM algorithms because this allows us to judge how reliable our own SLAM algorithm is in estimating it's location

Through the simulation environment we generate 3 sets of data. The first is the ground truth data which is recorded by Gazebo. The second is the estimated SLAM trajectory found by ROS. Lastly we have the estimated Odometry from the RTAB data. All 3 datasets follow the below format:

Column Name	Meaning
timestamp	Time stamp at which the position was recorded
tx	Translational X component position
ty	Translational Y component position
tz	Translational Z component position
rx	Imaginary X component of Quaternion or pitch
ry	Imaginary Y component of Quaternion or roll
rz	Imaginary Z component of Quaternion or angle of wheels
rw	Real W component of QUaternion or yaw

These 3 datasets are invaluable in judging the reliability of the RTABMAP SlAM algorithm. All 3 datasets have the exact same timestamps in their timestamp column. This means for any given timestamp we have 3 points of information on the location of the robot: the ground truth or true position, the estimated SLAM position and the estimated Odometry information. Since we know the true position we can use it to evaluate the estimated SLAM position and the estimated Odometry.

Since all the timestamps are already aligned and our dataset has no missing elements we did not need to do any preprocessing with it. We do a basic sanity test on the data by plotting the positions in each dataset and seeing if the plotted trajectory is similar to the one we would follow in the environment. An example is shown below



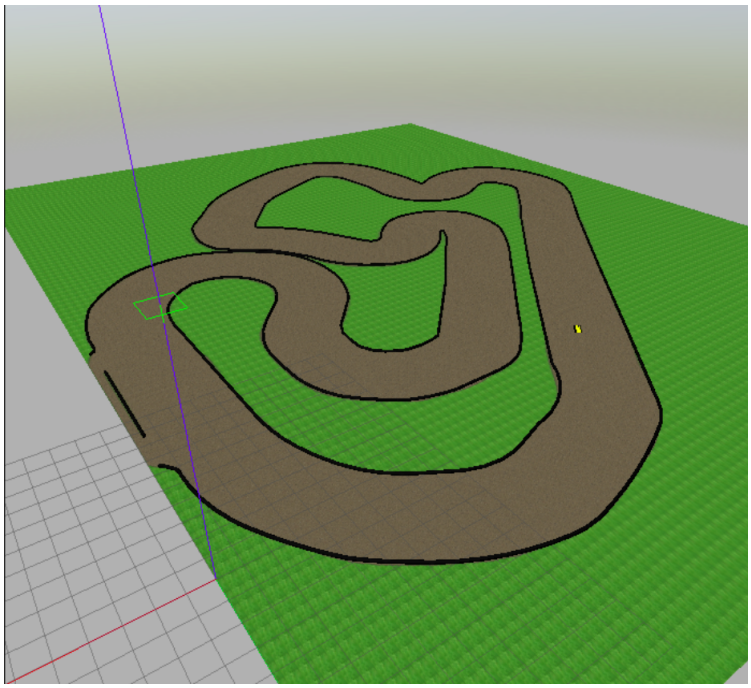
As we can see this seems to be a reasonable trajectory to follow in our simulation environment. It seems to be tiled 90 degrees but as long as all the trajectories follow this similar tilt it shouldn't affect our evaluation of the algorithm as they have a similar trajectory

3 Simulation Environment

Given the advantages of a simulation environment above we decided to invest some time in an appropriate simulation environment. Our track had unique conditions that made us choose the RTABMAP package. Here is the track we were aiming to race on

In a nutshell it's a gray road with green grass. Simple right? Not really. This posed a problem for our 2D Lidar since it had no information to bounce off of and detect. This is why we decided to use a stereo camera approach which RTABMAP does (more on this later). To properly test out the RTABMAP we needed a simulation environment that could match this. This posed to be an issue as most research in this has been done in indoor setups (probably due to ease). Even the resources on Gazebo mainly seem to point to indoor environment setups. Some like the FormulaPi Track had ancient file setups that were not in line with current infrastructure

Finally we came across GeorgiaTech AutoRally[2]. The simulation environment it provided us was perfect for our needs. It provided a Brown Track with Green grass. It had a small black wall around it but if we remained faithful and didn't use the Lidar it would not matter as it still offers a contrast (arguably harder one)



For the car model we borrowed from Andres Garcia's work[3] which already incorporated a yellow car with a Kinect sensor. The Kinect sensor was modified to simulate the Intel D455 Realsense Camera due to the lack of plugins for it currently

We Dockerized the entire simulation environment to control for dependencies and allow it to be run on other machines. The mapping process was made easy with a .yaml file used for editing configurations and maintaining proper versioning

4 Methods

The study will be carried out by analyzing the datasets generated. We will generate datasets for the RTABMAP SLAM Package. This process will be repeated for multiple hyperparameter combinations in each package giving us multiple trios of data. Each trio will then be evaluated based off 2 criteria

- The visual of the map generated and how close it is to the path traversed
- The reliability of the algorithm in how close it estimate it's location to the ground truth/actual position

The visual of the map generated is fairly easy to explain. We will see if the map generated matches our simulation environment. This visual will serve as a qualification round. If any of the datasets fail to generate a good enough visual we will not consider it in the reliability round

The reliability of the algorithm is more tricky. Currently there are 2 popular metrics for judging SLAM Algorithms

- Absolute Trajectory Error (ATE)
- Relative Pose Error (RPE)

Absolute trajectory error requires that absolute ground truth poses are available as it directly measures the difference between points of the true and estimated trajectory (SLAM trajectory). Relative pose error is more useful when there is only sparse, relative relations are available as the ground truth. It measures the error between all pairs of timestamp pairs in the estimated trajectory (odometry trajectory). The formulas for both are displayed below.

Absolute Trajectory Error[1]

The first step is to find the error matrix by projecting the estimated trajectory P_i over the ground truth Q_i where i symbolizes the timestamp. This is done with the help of a rigid body transformation S calculated using the method of Horn

$$E_i := Q_i^{-1} S P_i$$

The ATE is defined as the root mean squared error from error matrices:

$$ATE = \frac{1}{n} \sum_{i=1}^{n} ||trans(E_i)||^2$$

Relative Pose Error[1]

Over here F_i calculates the relative pose error / drift between 2 timestamps

$$F_i^{\Delta} = (Q_i^{-1} Q_{i+\Delta})^{-1} (P_i^{-1} P_{i+\Delta})$$

Similar to absolute trajectory error, we can calculate root mean squared error over all timestamp indices i.

$$RPE = \frac{1}{n} \sum_{i=1}^{n} ||trans(F_i)||^2$$

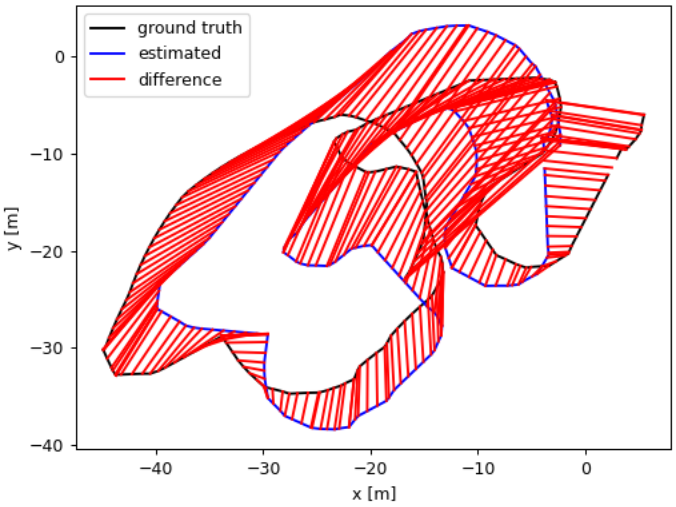
The lower these metrics the better. We will calculate these metrics on all the hyperparameter combinations possible. Whichever produces the lowest error is what we will choose as our final mapping package to be executed in the real world

5 What is RTABMAP?

RTAB-Map (Real Time Appearance Based Mapping) is a RGB-D, Stereo and Lidar Graph-based SLAM approach on a global loop closure detector. The main approach that it uses to determine the likelihood of a new image coming from any location is bag-of-words. The memory management approach limits the number of locations used for loop closure detection and graph optimization, accounting for constraints which are created by loop closure hypothesis. The RTAB-Map approach can be used by itself using a handheld Kinect or stereo camera, or on a robot that is equipped with a laser rangefinder for 3DoF mapping[4][5]. The rtabmap_ ros is the software package used for learning how to use the RTAB-Map approach with ROS. The RTAB-Map approach is significantly influenced by the adjustment (tuning) of parameters, as the robot's performance in localization can be improved by experimenting with several types of parameters such as visual odometry and mapping.

6 Hyper-Tuning of Parameters

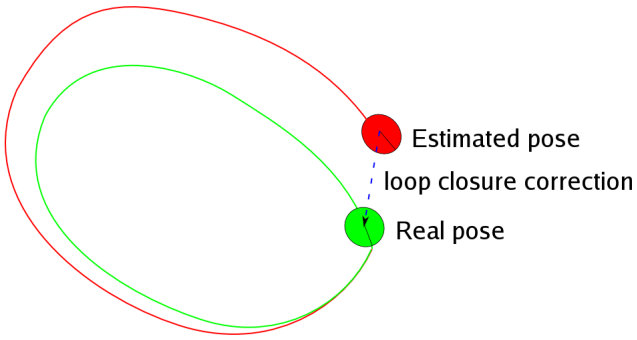
We will be using the **Absolute Trajectory Error (ATE)** as our main metric as we have the data for the absolute ground truth to accurately evaluate our SLAM algorithm.



In our baseline test, we achieved an ATE of **9.119737**. The above map is a visual representation of the ATE. This path is created through driving around a simulated track twice. The first lap is to lay the ground truth and create the mapping and the second lap is to test the localization of the mapping.

6.1 Loop Closure Detection Parameters

Loop Closure Detection detects whether or not the current location has previously been visited or not. A good Loop Closure Detection Algorithm is critical to SLAM as the map size increase, the more computationally expensive it gets. The figure below is a visual representation of what the loop closure is doing. When the robot returns to a past location, the algorithm will be correcting its position towards the real pose.



There are 2 parameters in the RTABMAP library which deal with the Loop Closure Detection. The first is **Keypoint Memory (KP)** and the second is **Visual Registration (VIS)**. The KP parameter determines which algorithm will be used to assign keypoints to the map created. The VIS parameter determines which algorithm will be used to process the visual data being created.

We compared 3 popular loop closure algorithms: **SURF, SIFT, ORB**

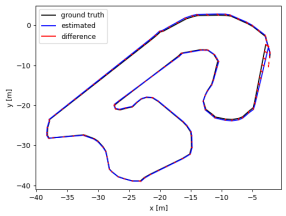


Figure 1: SURF ATE: 0.246418

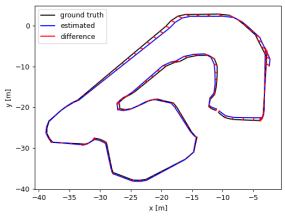


Figure 2: SIFT ATE: 0.531504

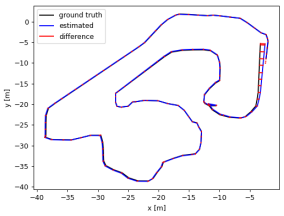


Figure 3: ORB ATE: 0.104784

The figures above show the ATE of each algorithm. These results were surprising because we initially thought that SURF and SIFT algorithms would do much better than ORB. This is because the two algorithms use very discriminative features. We thought that the more detailed the data is, the better the performance. However, we found that since we were mapping a racetrack with no landmarks, the visual data that is being received will be identical. In a way, both SURF and SIFT are over fitting. Whereas ORB does not use as discriminative features as the other two algorithms, thus resulting in a lower ATE.

We wanted to see if we could improve the ATE further by changing the parameter: **LoopClosureReextractFeatures (LCRF)**. This parameter can help to improve the localization of the robot in tight, narrow areas.

I used the SIFT algorithm's ATE as a baseline, as this LCRF parameter's default is false.

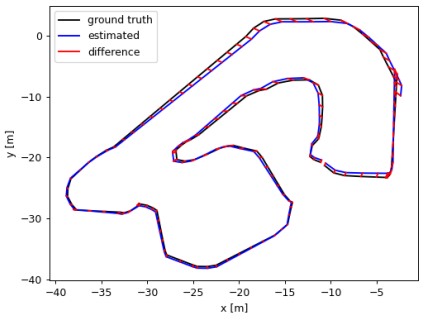


Figure 4: False ATE: 0.531504

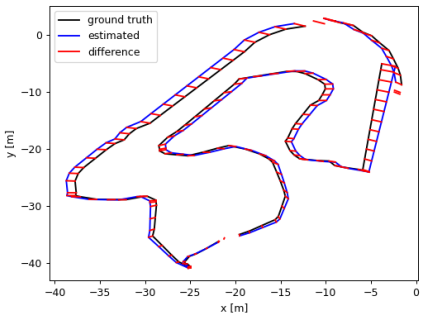


Figure 5: True ATE: 0.842040

The figure above shows the difference when LCRF is at its default value, False, and when the it is set to True. By setting the LCRF parameter to true, we also had to increase the value of the MinInliers parameter as the default value was too low for the loop closure algorithm to function properly. We increased it from 15 to 30 to make sure that the loop closure is properly executed. When we ran the simulation, we actually had a worse ATE than the baseline. This may be due to the over fitting issue we faced when comparing loop closure detection algorithms. It seems that through out testing, the more visual data there is, the ATE will begin to do worse.

In addition to loop closure detection algorithms, visual data and its feature size play a critical role in further lowering the ATE. We decided to change the **MaxFeatures** parameter. This parameter affects the number of features that the keypoint mapping takes in when running the loop closure detection algorithmn.

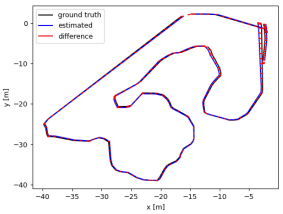


Figure 6: 600 ATE: 0.688109

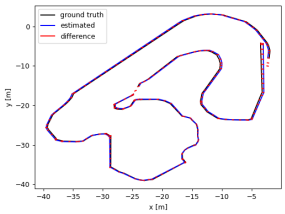


Figure 7: 200 ATE: 0.286628

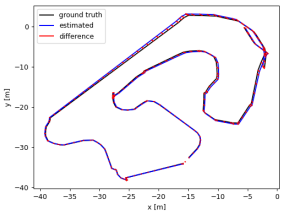


Figure 8: 50 ATE: 0.344276

This figures above are the ATE for each respective value. Our results align with our previous assumption during our loop closure detection algorithm test. The more features there are, the more likely it is to over fit in our racetrack simulation. However, if the parameter value is too low, then it begins to under fit.

6.2 Visual Odometry Parameters

Visual odometry parameters are what estimates the camera's motion of characteristic image points (image features). Image features are computed from image corners, which are image regions with high intensity gradients. Image features are used to look for matches between subsequent images to find correspondencies. The camera's motion is computed from a set of corresponding 3D points between two images.

We changed the following parameters: **Odometry Strategy, Frame-To-Map MaxSize, Visual Optical Flow Dependencies, and Visual MaxFeatures**

We tuned visual odometry parameters, in order to increase odometry frequency given the limited computing power of my PC. we changed the visual odometry strategy from frame-to-map (0) to frame-to-frame (1). Frame-to-frame setting has an advantage over frame-to-map in that the tracked features drop under a ratio of the previous frame that the robot had encountered. In other words, even when the robot is not moving, the key frame would not change and remember the feature present at that frame. This contributes to the flexibility of the robot to localize well at times where pausing of the robot is inevitably needed during driving. My other parameter, OdomF2M MaxSize maintains a local track map with a certain maximum number of words. we reduced this value from default 2000 to 1000, in order to speed up the odometry frequency. (faster and more accurate localization) For similar reasons, we also reduced Vis Maxfeatures (maximum features extracted by image) from 1000 to 600 and expected that this setting would lower down the ATE. We also changed the Vis CorType(optical flow correspondencies) to 1 since it only works with Odom Strategy: 1 and guarantees more features being matched despite its lack of robustness compared to 0. After direct comparison of ATE between strategy 0 and and CorType 0 with strategy 1 and CorType 1, the latter did better (6.37 vs. 7.89). Then we tested with Vis Maxfeatures: 300 and OdomF2M MaxSize: 500. It turned out that these were too few to increase the odometry speed, giving me a similar ATE in the 6 range.

6.3 Mapping Parameters

Mapping parameters are what adjusts the setting of the map environment in order to improve the localization abilities of the robot based on the landscape, occupancy grid, and awareness of projected obstacles around the track.

We changed the following parameters: **3DOF and Optimizer Slam 2D**

3DOF refers to the tracking of rotational motion (pitch, yaw, roll) and considers only three perpendicular axes of movement. This is in contrast to 6DOF, which also considers back, forward, up, and down motions. The track that we have is flat ground and our robot makes frequent angular turns around the curves, so we wanted to enforce visual odometry to track the vehicle in the context of 3DOF to improve the robustness of the mapping.

We also set the Optimizer Slam2D parameter to true because it would optimize the pose that consists of (x,y) translation and rotation, which the robot encounters. After one lap doing the drive with localization:= false, this submap will be considered for loop closure. Once a good match is found around this estimated pose, this match will be added to the loop closing constraint, which will then optimize the process by closing the loop every time that the location is revisited. The Slam2D is the appropriate parameter to execute this process. we compared and contrasted the performance of the robot before and after Force3DoF is set to true, and the result was similar results in ATE: (**6.82** before, **6.38** after)

We also changed the **Grid Size** parameter to see how the size of the occupancy map affects our ATE.

The grid_ size parameter is the minimum size of the occupancy map, which is what maps the track environment in an array of cells, each cell holding a probability value that the cell is occupied. If grid_ size is 0, the map grows as new data is added. we thought that, by setting grid_ size to 50, the robot would have early access (speed up the process of locating cells better than having data being added just as the robot reaches a particular cell) to the probability data and localize on track better, once the occupancy map calculates the probability of cells based off of mapping data created by the initial localization:= false single lap. =¿ expected that higher values of grid_ size would lead to a lower ATE. We tested this parameter with values 20, 40, and 60 as well, and they each produced an ATE of **5.68**, **4.12**, **3.81**. We thought that for grid_ size too, there is a maximum threshold to which the robot can take advantage of a larger number of grid_ size. We think this is because even with the largest grid_ size of the occupancy map implemented, the robot faces other challenges in localizing, particularly at consecutive curves. The role of loop closure is more important at curves because the occupancy map could potentially give false probability info of a cell to the robot if loop closure has not been done properly and the robot has a history of encountering that cell.

Another mapping parameter we investigated was: **Point Clouds**

A point cloud is a set of data points in shape, representing the 3D shape with three coordinates (x,y,z). They are generally produced by 3D scanners that measure the points on the external surfaces

(in our case, the track wall). The problem with point clouds is that if they are generated by 3D reconstruction techniques (typical generation way if ur not using a laser), they are much noisier (has many outliers) compared to when they are generated by lasers. In our case, we don't use LaserScan. So we felt the need to reduce both cloud_ noise_ filtering_ radius and cloud_ noise_ filtering_ min_ neighbors to filter out outliers. We initially tested with cloud_ noise_ filtering_ radius = 1 and cloud_ noise_ filtering_ min_ neighbors = 2 which gave me an ATE of **1.07**. We thought reducing cloud_ noise_ filtering_ radius to a really small level would be more impactful than modifying cloud_ noise_ filtering_ min_ neighbors because we don't know where the majority of point clouds are scattered within the surfaces, and what the radius of populated scatters will be. So we tested with different values: 0.1, 0.05, 0.01, and 0.05 gave me the lowest ATE at 0.316. We were able to analyze that a filtering radius of 0.01 was too small to properly cover the populated points and also not sufficient to rule out the outliers.

We changed the parameter: **Projected Occupancy Grid Characteristics**

Proj_ max_ height parameter refers to the mapping maximum height of points used for projection. we set this parameter value to 2.0 (height we picked as an initial guess) so that the occupancy grid (2D map) would extensively capture the slice of the 3D track. By setting the value of this parameter to a relatively small number, mobile robots can address the problem of generating maps (localization:= false) from noisy and uncertain measurement data (the robot making over- and under-estimated turns at a series of curves due to the different number of times we press 'l' or 'j' at different turns) and potentially lead to a lower value of ATE. After simulating with a relatively higher number (**5** -¿ ATE of **0.37**), the robot's performance wasn't as good as when it was **2** (ATE of **0.12**).

Another part of the projected occupancy grid characteristics is: **Proj_ max_ ground_ angle**

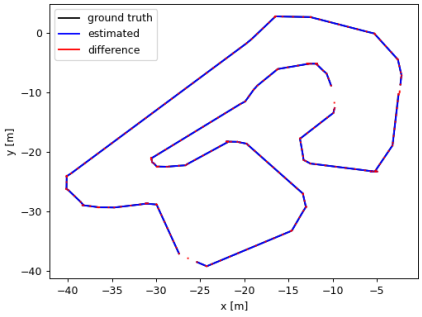


Figure 9: ground _ angle: 45 ATE: 0.008

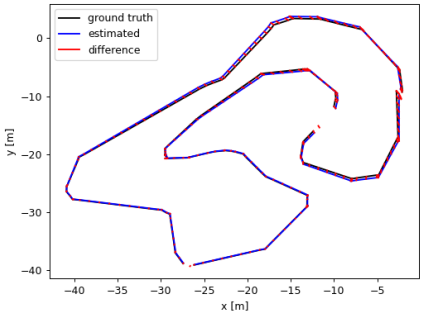


Figure 10: ground _ angle: 60 ATE: 0.008

Proj_ max_ ground_ angle is a parameter that sets the maximum angle between a certain point normal to the grounds normal (for example, a perpendicular angle formed between an obstacle and the ground), and proj_ max_ ground_ height is a parameter that rules out points that are under the designated threshold (the value of this parameter). The reason we chose to adjust proj_ max_ ground_ angle is that we expected them to lower down the ATE by setting it to a relatively high number and recognize the right or left walls of the track whenever driving through the curves. Specifically, the right wall of the track may be placed right in front of the robot's current route when there is a leftward curve ahead. The robot would be able to recognize the wall as an 'obstacle' and be directed to avoid it. we set proj_ max_ height to 0.1 because, by nature, the robot doesn't need to pay attention to tiny-height obstacles around the track (can be things like grass). If this parameter's value has a relatively big number, this parameter would be a bad one to use, because the robot would end up ruling out the objects that it should account for. We tested proj_ max_ ground_ angle with values 30, 45, 60 to see where the threshold could be. The ATE values returned were: **0.08**, **0.007**, and **0.005**.

Lastly, we looked at the **Scale Invariant Feature Transformation (SIFT)** Parameter.

SIFT is a Scale Invariant Feature Transformation, which is a parameter that is responsible for feature detection in computer vision, to detect and describe local features in images. What the SIFT

ContrastThreshold parameter does is that it filters out the weak images (key points that are poorly localized on an edge).

We changed the parameter: **ContrastTreshold**

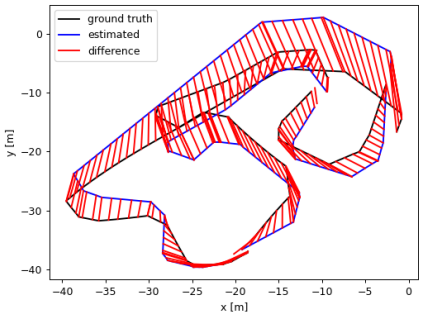


Figure 11: contrastThreshold: 0.03 ATE: 5.226

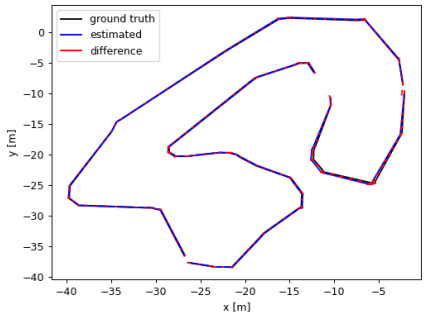


Figure 12: contrastThreshold: 0.05 ATE: 0.736

We thought that lowering down this number slightly to 0.03 would increase the number of features that are extracted out and enhance the robot's ability to differentiate between low contrast images (images that are similar) and therefore lower down the ATE, but the result was completely the opposite, returning **5.22**. So we tested the opposite case (bringing it up to **0.05**, since the default is **0.04**) to see if that would lower down the ATE. Although the ATE wasn't reduced from the point the last parameter was used, it gave me a similar ATE of **0.73**. What's interesting about this parameter is that there is no particular direct or inverse relationship between a robot's ability to distinguish between images based on the number of images that are filtered out =¿ it doesn't necessarily impact the robot's performance itself. This is because the robot relies more heavily on the mapping aspect (making a more robust map by loop closure, forcing 3DOF) to perform better, rather than learning from features filtered out from images.

6.4 Odometry ORB_ SLAM2 Parameter

This parameter measures the camera trajectory and 3D reconstruction of RGB-D cameras. It is responsible for detecting loops and re-localizing the camera in real-time, as the robot drives around the track.

We investigated the parameter: **FPS**

ORBSLAM2 Fps is a parameter indicating the camera Fps. we tried simulating with a minimized screen but did not see a significant improvement in Fps nor the ATE. What this parameter does is that it sets the rate at which my hardware system completes frames or a rolling average over many seconds. We tried manually updating this parameter at different values like 2.0, 5.0, 10.0 to test the robot's performance and see if the higher this parameter value, the smoother animations become and improve target tracking, which in our case is navigating through the track until the robot completes a lap. we thought it would have less distracting effects (and therefore lower down ATE) to angular turns (robot not turning quickly enough for left or right turns even after a keyboard command) or linear progress (the motion of robot going forward being lagged frequently and not properly reflecting the number of times I was pressed). After trying these cases, the ATE did not decrease any further from before: 2.0 $\longrightarrow$ **0.166549**, 5.0 $\longrightarrow$ **0.16784**, 10 $\longrightarrow$ **0.16687**, which showed that adjusting this parameter was not very significant. We believe this is because it did not directly impact the robot's ability to localize (the robot already has helpful features like odometry speed and mapping that brought down ATE to a significant level).

7 Results and Summary

Parameter File	ATE	RPE
Param1	0.126869	0.0014830374993572571
Param2	6.347690	0.0012715256291768412
Param3	0.688109	0.0010163356143470914
Param4	0.286628	0.0011323441858982354
Param5	12.110266	0.0014574414967737574
Param6	0.344276	0.0014484948848125106
Param7	0.793643	0.0013272938373127628
Param8	0.452278	0.0020178021331756753
Param9	0.475291	0.0019628173913403785
Param10	0.173608	0.002009954306641947
Param11	8.366418	0.0007821873813825113
Param12	6.375329	0.0008285912122776104
Param13	9.748151	0.0010936579406343558
Param14	6.388980	0.0010824339653301993
Param15	3.742185	0.0010536650137952539
Param16	0.316091	0.0009974357010700636
Param17	0.007340	0.0012492207615797163
Param18	0.124913	0.0010220859357666478
Param19	5.226961	0.0013603459086136346
Param20	0.166549	0.0013430826348192519
Param21	8.875549	0.0018356609167728504
Param22	0.194874	0.0019830555019465265
Param23	0.104784	0.0016119525304246392
Param24	0.693315	0.0019219784692654518
Param25	0.531504	0.0019161435984749954
Param26	0.842040	0.002801834447570765
Param27	0.246418	0.00221310896841887

Throughout our experiments, there were a lot of results that surprised us. How initial assumptions did not take account the environment the robocar was in. The tuning process is extremely tedious since we have to drive the car around the race track ourselves. Given the lack of computing power, the simulation runs extremely poorly. In future tuning experiments, it is better for us to focus on the loop closure detection algorithm parameter rather than fine tuning the other parameters.

8 Improvements

Overall we laid down the baseline for a streamlined experiment setup. We also maintained safety in the COVID-19 pandemic by keeping everything in the simulation environment. However this does have it's drawbacks. The simulation environment took a toll on some machines and ran smoothly in others. Since it was unreasonable for one person to parameter tune we had to use different machines. This led to different performances. We tried keeping it faithful as much as possible by following the same path but there are bound to be some minor adjustment issues here and there. If we did this on our line following robot in person it would have been more faithful but also more risky

Moreover the Dockerized container currently runs best with a Linux OS and Intel Graphics due to ease of running in regular docker instead of the Nvidia Docker + Xhost access. This poses a problem for those without such a configuration. Even the Virtual Machine Linux didn't work due to trouble in forwarding X11 to Docker container. David from our team spend 5 weeks trying to get the simulation to work in a myriad of ways before finally deciding to dual boot Ubuntu

9 Team Contributions

Siddharth Saha: Wrote sections Abstract, Introduction, Dataset, Simulation Environment, Methods and Improvements. Conducted research and designed research methodology, library for generation of plots and metrics, simulation environment and Dockerized container as well as

distributed and assigned tasks based off research methodology

Jay Chong: Simulated and generated test data (params 1-10 and 21-27) by tuning and testing various parameter as well as providing intuition for each of the combinations tested. Compared and constrasted the ATE (Average Trajectory Error) results using different values of certain parameters affecting localization as well as analyzed the plots generated. Assisted team in generating data for earlier assignments. Wrote section 6.1 and edited/provided guidance on what to edit and integrated remaining subsections of section 6. Completed EDA for 180A assignment given test data

Youngseo Do: Simulated and generated test data (params 11 20) using loop closure detection, odometry visual, mapping, SIFT (Scale Invariant Feature Transformation), and ORBSLAM2 parameters on the Gazebo simulator environment. Compared and contrasted the ATE (Average Trajectory Error) results using different values of certain parameters that are effective for tuning the robot's performance as well as analyzed the plots generated. Analyzed the results of different hyperparameter tuning experiments based on what the parameter's effect on ATE was, why I used the parameters and why I believed these effects were shown. Wrote the introductory paragraph about the concept of RTAB-Map and explained the significance of its parameters with regards to improving the robot's ability to localize on the track map and raise environmental awareness.

10 References

[1] A Benchmark for the Evaluation of RGB-D SLAM Systems. Jurgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, Daniel Cremers. http://ais.informatik.uni-freiburg.de/publications/papers/sturm12iros.pdf

[2] AutoRally: An Open Platform for Aggressive Autonomous Driving. Brian Goldfain, Paul Drews, Changxi You, Matthew Barulic, Orlin Velev, Panagiotis Tsiotras, James M. Rehg. Control Systems Magazine (CSM), 2019. [PDF][BibTex]

[3] Real Time Based Appearance Mapping, Andres Garcia https://github.com/AndresGarciaEscalante/Real-Time-Appearance-Based-Mapping

[4] M. Labbé and F. Michaud, "Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation," in IEEE Transactions on Robotics, vol. 29, no. 3, pp. 734-745, 2013. (IEEE Xplore)

[5] M. Labbé and F. Michaud, "Memory management for real-time appearance-based loop closure detection", in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011, pp. 1271–1276. (IEEE Xplore) Visit RTAB-Map's page on IntRoLab for detailed information on the loop closure detection approach and related datasets.

11 Appendix

For all parameter files you can look here