Centralized Multi-agent Visual SLAM

Team 19 Final report for EECS568 / ROB530 Winter22

Github repository: Multi-Agent-Visual-SLAM

Rui Chen
University of Michigan
Ann Arbor, USA
ruiche@umich.edu

Goro Yeh
University of Michigan
Ann Arbor, USA
goroyeh@umich.edu

Hao Chen
University of Michigan
Ann Arbor, USA
haochern@umich.edu

Akshay Tondak
University of Michigan
Ann Arbor, USA
akshayt@umich.edu

Simeng Zhao
University of Michigan
Ann Arbor, USA
zsimeng@umich.edu

***Abstract*—Multi-agent SLAM has been an active area of study in the recent years with a lot of classical SLAM techniques saturating in the achievable localization accuracy. Through this paper, we implement a novel collaborative SLAM based on centralized communication network, with every robot running ORB_SLAM2. We construct modules from scratch for robot simulations and also make attempts towards improving the overall performance of the system. Finally we compare the running time between single robot and multi-robot mapping and propose a multi-agent robot system for a faster and more accurate environment mapping.**

***Index Terms*—Centralized collaboration frame, ORB_SLAM2, pose graph, point cloud, occupancy grid, semantic segmentation**

I. Introduction

The constant demand for improving accuracy and speed for SLAM systems has continuously motivated researchers to find new solutions. While the research on single agent SLAM has reached unprecedented heights, there are unavoidable drawbacks in dynamics and algorithms for single robot, and for certain scenarios that need cooperation. Cooperative SLAM using multiple robots can break through some of the bottlenecks that single agent SLAM might not be able to. It can shorten the time in performing global SLAM, it can improve loop closure accuracy to consequently improve the accuracy of SLAM, oftentimes inheriting the existing research results for single robot SLAM.

In this paper we describe our centralized multi-agent visual SLAM in ROS simulation framework. We discuss the performance and future improvements pertaining to this technique. In implementation, Each robot runs modified ORB_SLAM2, a monocular visual SLAM framework that takes in sensor image information and then applies g2o to do pose graph optimization.

II. Related Work and backgrounds

A. ORB_SLAM2

The front-end visual odometry is mainly used to estimate the relative motion parameters of the camera between frames, so as to indirectly infer the motion of the robot in this period of time and cumulatively estimate the structure of the environment. However, VO itself will naturally have cumulative errors, which makes back-end optimization and loop detection very important. The pose state information measured by the back-end VO and the loop detection information are used to process the noise generated in the SLAM process by using, for example, filtering or nonlinear optimization algorithms to achieve the purpose of optimization, so as to obtain the global map.

ORB_SLAM has produced three generations so far. [1] envisioned the first ORB SLAM which had the following novel features which also continued to the ORB_SLAM 2:

- Fast, real time operation using the covisibility graph.
- Allowed wide baseline and real time loop-closing.
- Performed Bundle Adjustment for accuracy.

It is, to the date, most popular baseline visual SLAM framework which motivated us to use its second version to implement the underlying SLAM for our setup. A monocular SLAM system based on orb feature recognition and running in real time naturally inherits the classic visual SLAM basic framework: reading sensor information (camera image information or LIDAR point cloud PCD) + visual odometry + optimization + loop closing + mapping.

ORB_SLAM3 [2] recently has further improved on the earlier versions in terms of speed of SLAM. The authors claim a speedup of 2 to 5 times and also utilize information from all the past ORB inputs and not just the last couple of frames. Since this is a fairly new research, we did not use it as our

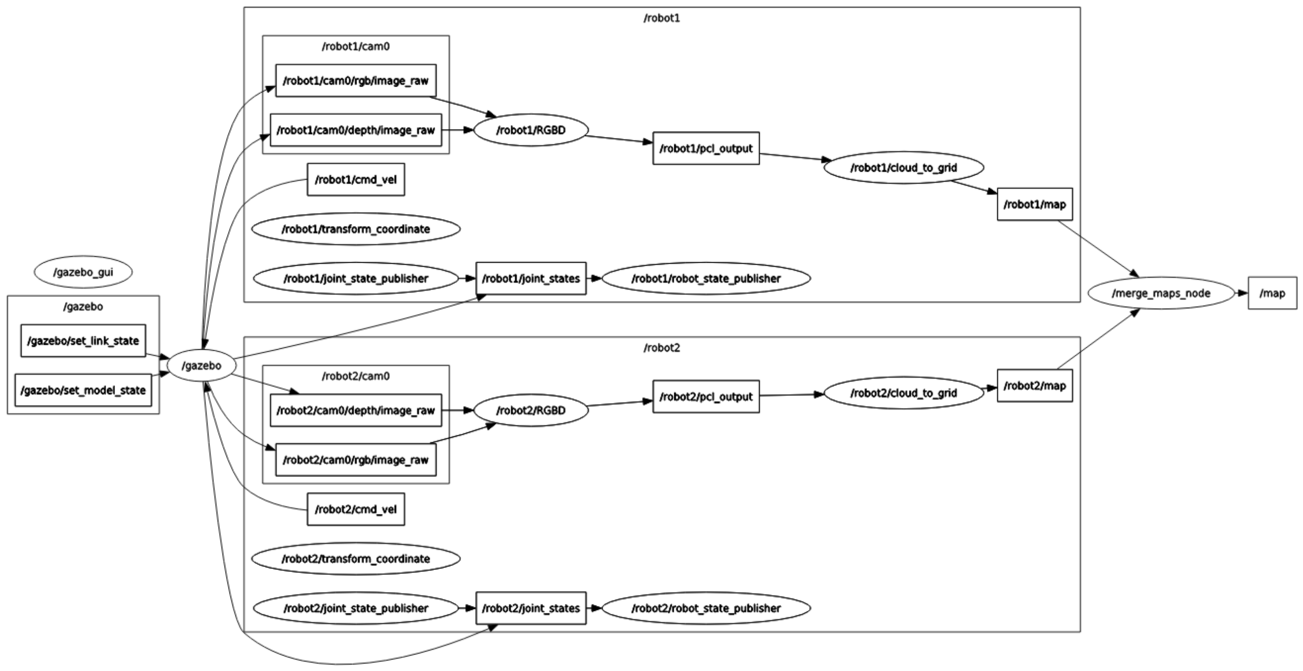


Fig. 1: ROS Node graph of the pipeline

method of doing SLAM but as future work, we can utilize the third version to examine if a higher speedup can be achieved.

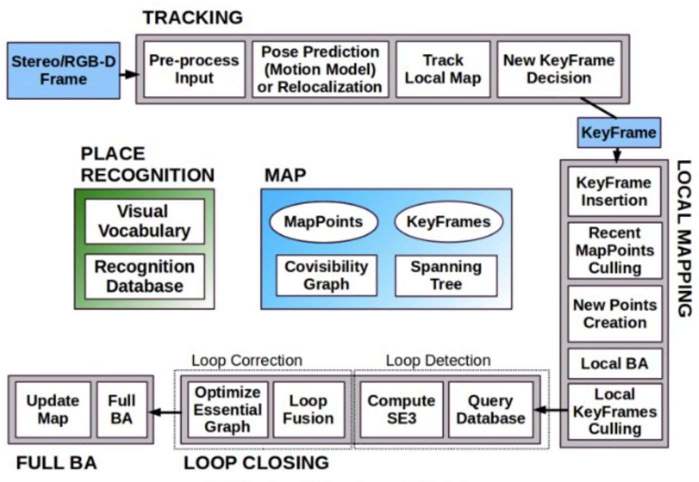
B. Collaborative SLAM solutions and communication network

Robots can improve the robustness of the system by establishing a reasonable communication mechanism to perform tasks that can not be performed by a single robot. However, it should be noted that although the structure of each robot in the multi robot system is different, and the sensing equipment may be different, the multi-machine SLAM is not the same as the multi-sensor fusion SLAM. The latter refers more to research based on single robot.

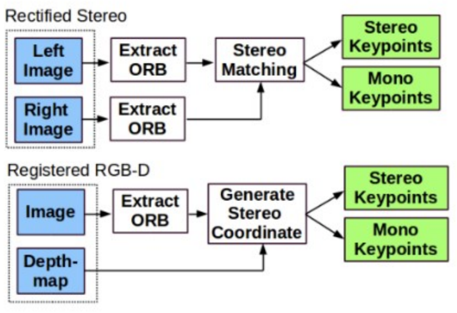
So far, the core problem of multi machine SLAM is still how to combine the information captured by multiple robot agents to build a single global map, and how to solve the accurate positioning and information transmission between agents in mobile. A natural idea is to allow agents to share information, especially the information collected by the agent's historical visiting environment, which is also the strategy adopted by many effective multi machine SLAM frameworks.

One of the earlier accepted collaborative SLAM technique termed C2TAM ([4]) differs from our approach in the way that it captures and sends the key-frames on the front-end robot and sends them to the central server for calculations. On the contrary, our approach runs active SLAM individually on the robots and sends their versions of the global map. MOARSLAM [5] is somewhat similar to our approach in the sense that it also runs a whole visual SLAM technique individually on the robots and sends back the local maps. As for the communication relationship between robots in multi machine SLAM, the most basic classification can be divided into centralized and fully distributed. CVI-SLAM ([6]) is the most recent and accepted approach to centralized Visual SLAM which performs loop closures and map fusion in the backend. Centralized means that an agent plays the role of server, this kind of system either relies on centralized or offline processing, or tries to keep most robots within the communication range all the time. And fully distributed means that each agent shares information with each other and has the same status. here sensor data required by SLAM is distributed on the robot, and the original communication data is difficult to be transmitted in the way we want due to bandwidth constraints or limited communication range.Now there is a communication mode combining two ideas, namely hybrid. Any exchange of key frame and map point position information between server and agent uses relative coordinates instead of absolute coordinates, because the exchange of absolute attitude may lead to the inconsistency between the newly transmitted key frame and the optimized map in the optimization process. When the global BA is running, pose info sometimes cannot be passed into the server map, because the data in the input buffer will not be corrected through the optimization steps, the absolute coordinates of the data will no longer match the back-end optimized map.

In terms of the relationship between agents in multi machine cooperation, whether distributed or centralized, many scientific research teams are trying corresponding countermeasures. Generally speaking, the implementation of centralized is rela-



(a) System Threads and Modules.



(b) Input pre-processing

Fig. 2: ORB_SLAM2 system pipeline (Image credits : [3])

tively less difficult and the results are richer than distributed. However, with the deepening of distributed optimization theory, the obvious advantages of distributed system also urge more researchers to make in-depth exploration. In addition, how to optimize communication is also a hot research field. How to reduce data transmission and how to ensure the stability of individual tasks in the case of communication disconnection in some areas? How to deal with the backlog of information after communication recovery... These problems are the key to multi machine collaborative SLAM and other collaborative tasks, but only a few systems can solve them.

III. Methodology and pipeline

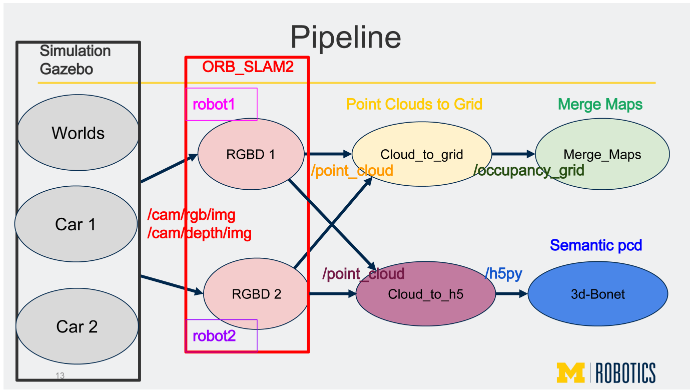


Fig. 3: Pipeline of the simulation

In our simulation, the RGBD images will be generated from gazebo indoor environment from the perspective of the robot, for different robots, we arrange different name space for them, so we can distinguish them while subscribing. As the robot's remote velocity controllers are sending linear and angular velocity command to the corresponding robot with same name space, the robot will move and the changing RGBD images will keep be subscribed by each robot's Modified ORB_SLAM2 system and be converted to point cloud data. Then here we will process the data in two different ways: in the first branch we compress the point cloud into occupancy grid map, which is preferred by mobile robotics; in the second branch, we do semantic segmentation to the point cloud.

IV. Implementation

A. Simulation configuration

- Indoor environment: a museum model from solidworks
- Car model: include kinematics and urdf
- separate remote velocity controller for every robot
- April-tags are placed inside the museum which will help robots to localize and confirm their relative pose when they detect the tags whose positions are unknown to them.

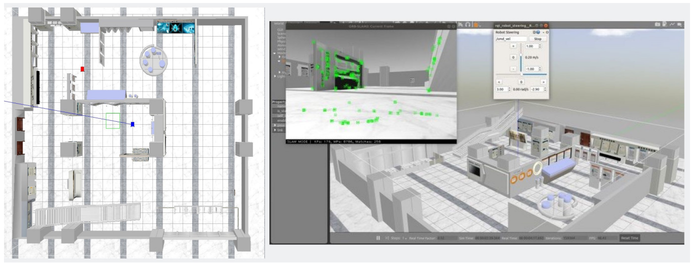


Fig. 4: indoor environment car model and controller in ROS

B. Modyfication on ORB_SLAM2

The major modification on ORB_SLAM2 is about real-time dense 3d point cloud, the original ORB_SLAM2 only have sparse point cloud from map points and some stable ORB features, we adopt 3d reconstruction algorithm to make and store 3d dense point cloud in real time. ORB_SLAM2 is divided into three threads: map construction, tracking and closed-loop detection. The three threads are carried out separately. The tracking thread first extracts the orb features of the input continuous images, estimates the detailed pose information, and then optimizes the pose, and uses the adjacent points to find more feature matching to optimize the pose and select the key frame. Map construction is to update the image information by adding key frames, generate the newly added map points after verification, and make necessary local adjustments when necessary. The main function of close loop detection thread is to select similar frames and detect the closed-loop by judging whether the RGBD info from the camera is highly similar to the previous historical records.

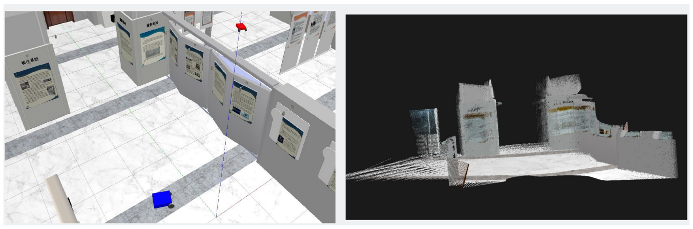


Fig. 5: real time dense 3d point cloud

Besides, we also make various modification on the interface and configuration to make the ORB_SLAM2 receive the RGB-D from corresponding robot.

C. Point cloud to Occupancy Grid Map

Actually when the mobile robotics want to to planning, they usually prefer 2D map, so we also create a ROS node to subscribe the dense PCD data in real time and compress or convert them into occupancy grid, then the occupancy grid map will be published under the same namespace for certain robot. For example if we run SLAM with 3 robots do the collaboration, 3 separate dense pcd will be converted to 3 separate occupancy grid, so in rostopic's list, we will have 3 RGBD image topics, 3 point cloud topics, and 3 matrices.

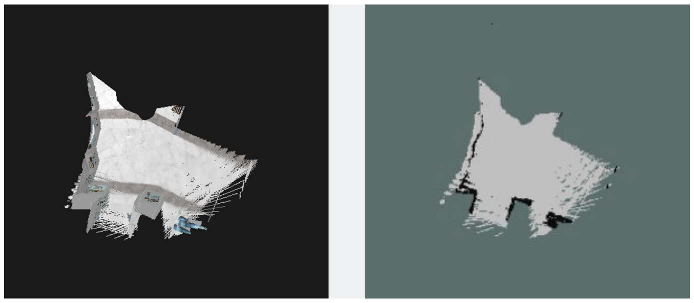


Fig. 6: converting PCD to occupancy grid in real time

And the algorithm to compress pcd into 2d data is simple: we know for occupancy grid map, the value of each grid usually represents the likelihood of being occupied, with 0.5 to be the initial value for unknown status, we will calculate the number of points within the grid in xy plane, then find those points which are within valid range of heights because for mobile robotic with certain height, those points with high z coordinates will not be the obstacles, only the points with lower z value will be counted into potential barrier. Then according to the number, we set threshold below which the grid will be considered as free or unknown, depending on the number of all points within the grid from top perspective. It should be noted that because the point cloud is growing in real time, the occupancy grid generation algorithm can be seen as incremental solution rather than batch solution.

D. Semantic segmentation

3D-Bonet is an efficient segmentation algorithm for point cloud instances based on bounding box regression. The approximate bounding box regression is realized by minimizing the correlation cost function, and the final instance segmentation is realized by point mask prediction. 3D-Bonet achieves the effect of state of the art on ScanNet and s3dis data sets, which can be said to be a lightweight neural network for point cloud semantic segmentation. We just replicate the network here to do semantic segmentation so we won't dive into the details here. The steps of installing virtual environment can be seen in github link. Here is the visualization of dense PCD. Because the input format of 3D-Bonet is h5 so we also create a scripts transforming the PCD here to h5 within ROS.

We also have made some attempts in using the results from

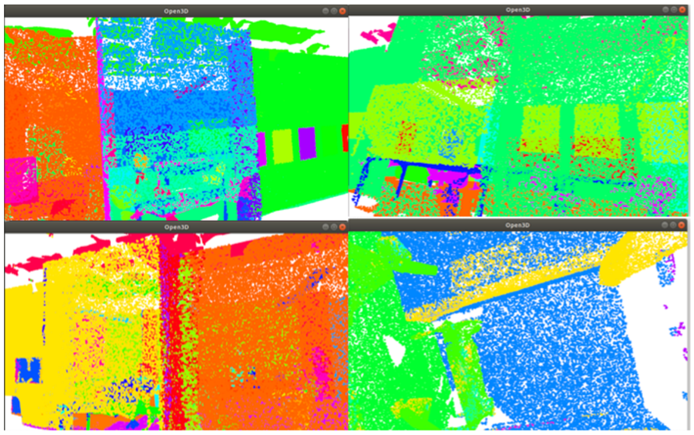


Fig. 7: semantic segmentation based on 3d-Bonet

semantics segmentation to help merge the data.

E. Estimate the Transformation

Although the master node knows the exact initial location of all robots, each robot doesn't know the relative pose of other robots, otherwise the project will be euivalent to single agent SLAM, therefore except for the accuracy in SLAM for each robot, estimating the translation between two or more agent will also be essential to accurate map merging. One of our solutions is first we get the point clouds of exploring same area, so the two point cloud should be aligned given correct translation. So we use RANSAC to exclude the outliers and then implement ICP to find the matrix.

As illustrated in Figure 14, by translation, the second robot's Occupancy Grid Mapping (OGM) is superimposed on the first robot's OGM, even though the local maps obtained by the two robots have significant errors. The x-axis and y-axis represent the local coordinate in robot1's frame.

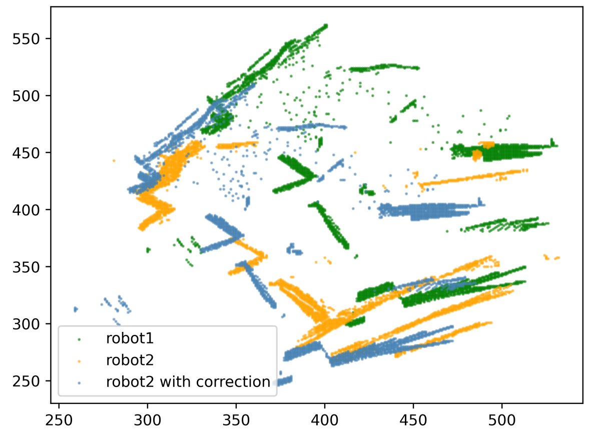


Fig. 8: RANSAC_Translation Algorithm

Algorithm 1 RANSAC_Translation Algorithm

```
**Input:** ArrayWithSize(N,2) $XY, XY'$ // $[x_i, y_i]$, $[x'_i, y'_i]$
**Output:** MatrixWithSize(3,3) $H$
$H_{best}, count_{best} = None, -1$
**for** $trial$ in range(NUM_TRIALS) **do**
    draw $[x, y]$, $[x', y']$ from $XY, XY'$
    $H = \begin{pmatrix} 1 & 0 & x-x' \\ 0 & 1 & y-y' \\ 0 & 0 & 1 \end{pmatrix}$
    $XY_{proj} = homography\_transform(XY, H)$
    $count_{temp} = sum\left(\|XY' - XY_{proj}\|_2 < eps\right)$
    **if** $count_{temp}$ $count_{best}$ **then**
        $H_{best}, count_{best}$ ←H, count$_{temp}$
    **end if**
**end for**
```

F. Merge Occupancy Grid Maps

We created a ROS node to convert two occupancy grid maps generated from the two robots individually to a single merged final map. The algorithm we designed is show in algorithm 2

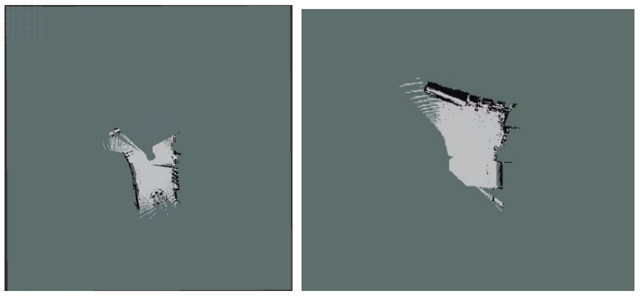
In order to merge two maps to a global final map, we need to transform each map to the global coordinate frame, which requires the transformation matrices from different robots. The mathematical equation of transforming a grid in robot1's coordinate frame to the global coordinate frame is in equation 1. x, y is a grid in robot1's frame and x', y' is the grid location in the world frame. x0, y0, theta is the initial pose of robot1 relative to the world frame. Note that to transform a point from robot1's frame back to the global frame, we need to multiply the grid in robot1's frame by the inverse of the transformation matrix.

The merge_maps process can be referred to figure 9, 10, and 11

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & x_0 \\ \sin\theta & \cos\theta & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \quad (1)$$

Algorithm 2 Merge Maps Algorithm

```
**Input:** OccupancyGrid $grid1, grid2$ // Maps from two robots
**Output:** OccupancyGrid $grid$ // Final merged map
$grid.header \leftarrow grid1.header$
$grid.info \leftarrow grid1.info$
**for** $i \leftarrow 0$ to $grid1.data.size$ **do**
    **if** $grid1.data[i]$ is not 0.5 and $grid2.data[i]$ is 0.5 **then**
        $grid.data[i] \leftarrow grid1.data[i]$
    **else if** $grid1.data[i]$ is 0.5 and $grid2.data[i]$ is not 0.5 **then**
        $grid.data[i] \leftarrow grid2.data[i]$
    **else if** $grid1.data[i]$ is 0.5 and $grid2.data[i]$ is 0.5 **then**
        $grid.data[i] \leftarrow 0.5$
    **else**
        $grid.data[i] \leftarrow (grid1.data[i] + grid2.data[i])/2$
    **end if**
**end for**
```



(a) /robot1/map (b) /robot2/map

Fig. 9: Maps created by two robots

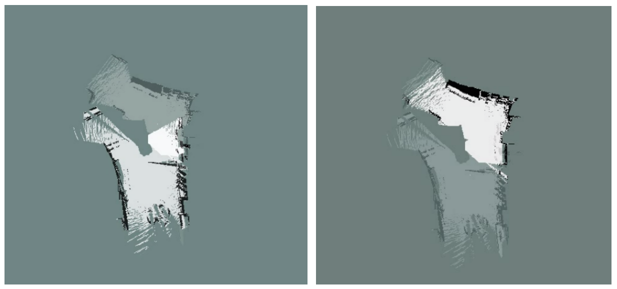
.

V. Results

The link for presentation and video demo are as follows: https://www.youtube.com/watch?v=w1nkEfZI4GE

A. Evaluation for SLAM with different number of engaged agents

In order to evaluate the performance of our multi-agent SLAM system, we first ran the single robot SLAM of the



(a) /robot1/map (b) /robot2/map

Fig. 10: Maps created by two robots(After transformation to the same world frame

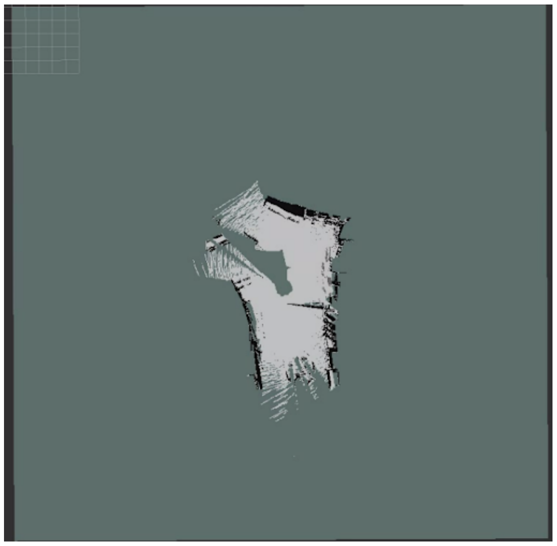


Fig. 11: The final map after merging two individual maps

whole map and recorded the time consumed. The, we ran our multi-robot SLAM system and recorded the time used. The table I below shown the runtime in different scenarios.

TABLE I: Running Time of different tests

	Running Time(sec)			
Scenarios	***1st***	***2nd***	***3rd***	***Avg***
Single Robot	407	409	406	407
Two Robots	211	208	207	208
Three Robots	167	172	154	160

But on the other hand, due to the cpu and graphics limitation, the runtime here won't be accurate, theoretically speaking, the runtime of N agents' collaborative SLAM should only be 1/N times the single agent runime, if we ignore the extra runtime for the overlap area, so the bias here make sense.

B. Evaluation for merged maps and point clouds

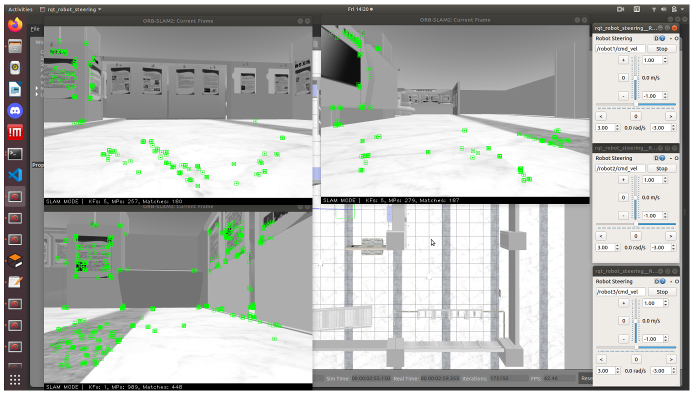


Fig. 12: Test run with 3 robots with merged grid map

Here we run a collaborative SLAM with 3 robots starting from different origins, we launch 3 different remote velocity controller to control 3 robots, and 3 RGB-D images with different name space will be subscribed by 3 corresponding ORB_SLAM2. Just as the figures shows above we published the real trajectories of each robot to rviz along with their merged grid map, then we use the transformation to go back and merge the point cloud data.

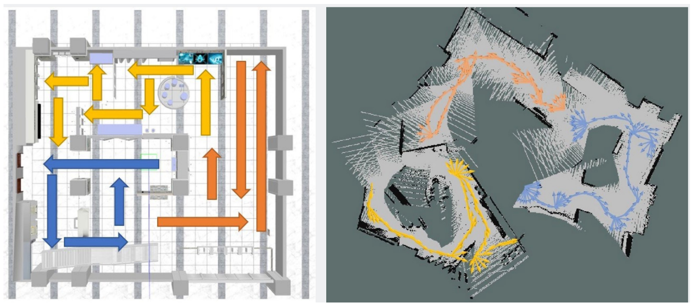


Fig. 13: Real and expected trajectories of 3 robots with merged grid map

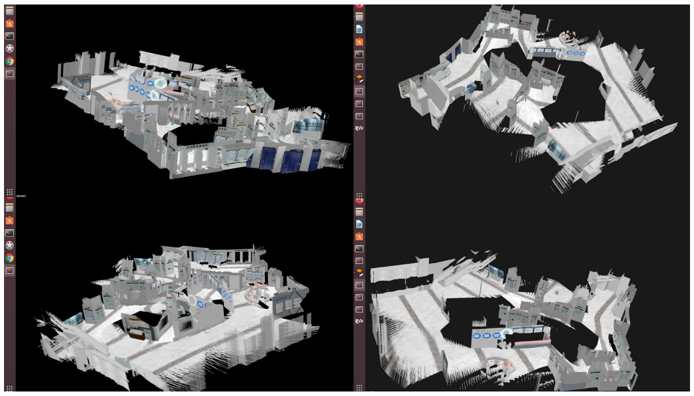


Fig. 14: Merged point cloud using same transformation

From the results above we can see both the occupancy grid map and the point cloud data are not satisfactory. That happens because the distribution of textures and features varies in the museum, so the ORB_SLAM performs terribly especially where the textures are not enough. Also in cloud to grid module, we can have error when count points within valid range, the boundary is set manually. For example, we want the area of stairs to be free in occupancy grid because the car can move freely under the stairs, but in the project, some points will still be considered as points in occupied area. And we try to get the depth information of the April tags in robot's horizon, the depth value is also not reliable enough.

VI. Discussions

The undesirable results and limited application scenarios of current approach are caused by the following reasons:

- The transformation matrix estimated using RANSAC assumes that the starting positions of the robots are relatively close to each other because too many outliers from a large amount of map data can cause incorrect fusion.

- Since the Occupancy grid mapping(OGM) is generated by real-time 3D point cloud projection, its point cloud information is inaccurate. Since the algorithm does not synchronize the point clouds updated by ORB_SLAM to OGM, a large number of necessary corrections are lost, which leads to less satisfactory results.
- In multi-robot collaborative mapping, an individual robot does not have much chance to perform loop explorations, so closed-loop detection is rarely triggered to optimize the recorded paths. Since the robots' pose-edge graphs are independent of each other, closed-loop detection across robots is difficult to achieve, resulting in an accumulation of errors and large offsets at the map joints.

VII. Future improvement

At present, limited by computing resources, only three schemes of robot exploration area have been tested at the same time. However, in our practical application scenarios, such as disaster rescue, space exploration, etc., more robots are often used for rapid map construction. Therefore, we hope that in the future, we will be able to implement collaborative SLAMs that include more robots, and even map construction based on robot swarms.

Next, our current approach to merging is overlapping maps based on the estimated initial relative location while the internal pose graphs are independent between robots, which leads to uncontrollable errors at map junctions. In the future, we hope to modify the pipeline of ORB_SLAM so that robots can identify, register, and synchronize common feature points among robots, and then use these feature points as edges to connect independent graphs between robots for loop closure detection and further online optimization.

In addition, the current data set of our project comes from the simulation environment, which is different from the data in the real world. At the same time, there are currently few datasets suitable for multi-robot SLAM, so in the future, we hope to use multi-robot field collection data to build a dataset, and continue to develop our project on this dataset.

Finally, we want to optimize the data storage method. At present, we also save some log information when producing maps, which makes the amount of data we store grow rapidly and makes it difficult for us to generate large-scale maps. Therefore, we want to optimize map information in the future. storage method to increase the scale of our stored data.

References

[1] R. Mur-Artal, J. Montiel, and J. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, pp. 1147 – 1163, 10 2015.

[2] C. Campos, R. Elvira, J. J. G. Rodriguez, J. M. M. Montiel, and J. D. Tardos, "Orb-slam3 an accurate open-source library for visual, visual inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, pp. 1874–1890, dec 2021.

[3] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-d cameras," *IEEE Transactions on Robotics*, vol. 33, pp. 1255–1262, oct 2017.

[4] L. Riazuelo, J. Civera, and J. Montiel, "C2tam: A cloud framework for cooperative tracking and mapping," *Robotics and Autonomous Systems*, vol. 62, no. 4, pp. 401–413, 2014.

[5] J. G. Morrison, D. Gálvez-López, and G. Sibley, "Moarslam: Multiple operator augmented rslam," in *DARS*, 2014.

[6] M. Karrer, P. Schmuck, and M. Chli, "Cvi-slam—collaborative visual-inertial slam," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2762–2769, 2018.