National University of Singapore

College of Design and Engineering

ME5413 Autonomous Mobile Robotics Final Project

Group 18

Shen Yixiu	A0263954U
Chen Yihui	A0263115N
Wang Renjie	A0263387U
Wang Longfei	A0263224M

https://github.com/Wang-Theo/ME5413_Final_Project

March 31, 2023

Table of contents

Project Description 3

Task 1 Mapping 3

I) Problem in 2D LiDAR SLAM algorithm 3
II) Mapping with FAST-LIO 3
III) Generate grid map with projection 6
IV) Evaluate FAST-LIO Performance 6

Task 2 Navigation 7

I) Localization 7
  a) AMCL 7
  b) Other localization 7
  c) Frame drifting problem 8
II) Planning 8
  a) Costmap 8
  b) Global Planning 9
  c) Local Planning 10
  d) Other problems 12
  e) Comparison and Discussion 12

Appendix 13

I) Appendix 1. Task 1 Mapping 13
II) Appendix 2. Task 2 Navigation 15

Reference 24

Project Description

With the rapid development of science and technology in recent years, mobile robot technology has been widely used in different areas including disaster rescue, indoor cleaning, driverless and other fields. To complete those navigation tasks, robots should be able to percept the surrounding environment and make reasonable decisions.

In this project, given a mini-factory environment including 3 accessible areas and 1 inaccessible area in Gazebo (Figure 1), mapping and path planning algorithms are tested to complete the navigation task. With the self-designed robot navigation software stack, the "Jackal" robot should move to the given pose within each area in the following sequence: Assembly Line 1, 2; Packaging Area 1, 2, 3, 4; Delivery Vehicle 1, 2, 3.

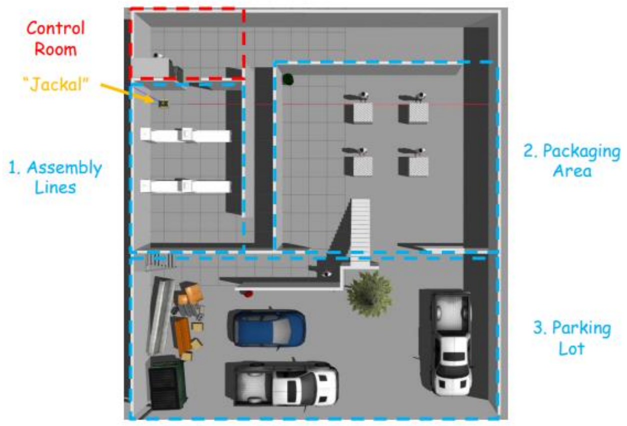


Figure 1.Shematic of mini-factory environment

Task 1 Mapping

Mapping is an essential step before navigation since it provides the robot with a representation of the surrounding environment, which allows the agent to plan its route, avoid obstacles and move efficiently towards its goal location. Commonly, SLAM (Simultaneous Localization and Mapping) algorithms are implemented to derive the map [1].

I) Problem in 2D LiDAR SLAM algorithm

Gmapping is the default algorithm for mapping in the original package and uses a particle filter to estimate the robot's pose and map the environment. Though the 2D mapping algorithm is indeed simpler, more intuitive and consumes fewer computing resources, it is limited and failed to handle some complex and special objects. The mapping result (Figure 2) shows that the glass wall cannot be scanned by lidar except the outline border, which may cause problems for future path planning. Also, only two ends of the stair can be mapped, and the robot may try to go under it and lead to collision.

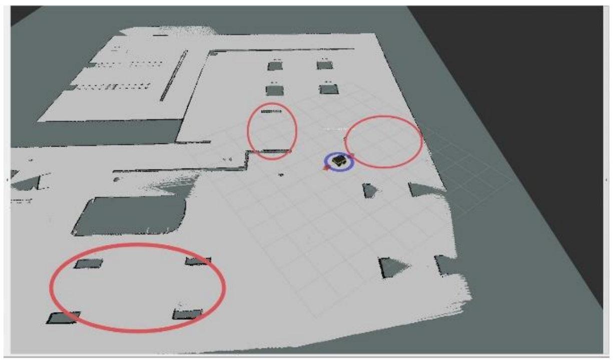


Figure 2. 2D mapping result using *gmapping*

Therefore, 2D SLAM algorithm is not capable to map the features of these special objects. Since 3D visual SLAM algorithms may obtain too much information and result in a large amount of computation cost, FAST-LIO 3D LiDAR SLAM algorithm is selected to map the environment [2, 3].

II) Mapping with FAST-LIO

The MARS Laboratory at the University of Hong Kong has suggested the Lidar-Imu Odometry approach known as FAST-LIO, which is reliable and computationally effective. It combines lidar landmarks with IMU data using tightly linked, repeatedly extended Kalman filters for reliable navigation in chaotic, noisy, or fast-moving settings [3]. The operation process to build the repo can be concluded as:

(1) After building the *"ME5413_Final_Project"* repo and FAST-LIO respectively, the FAST-LIO is moved into *"ME5413_Final_Project/src"* folder as a package in the repo whose name is *"FAST_LIO_"*.

(2) Use command *"roslaunch me5413_world world.launch"* to launch the world and use *"rostopic list"* to find the two topics required: pointcloud topic *"/mid/points"* for 3D LiDAR and sensor_msgs/Imu topic *"/imu/data"* for IMU.

```
Algorithm 1 RANSAC Algorithm
while $iter < iter_{max}$ do
    Step1: Randomly choose the smallest datset to estimate the model;
    Step2: Solve the model by the chosen dataset;
    Step3: Use all the other data to count the number of inliers;
    Step4: Compare the number of inliers of the current model and the best model, update the model;
    if the model is good enough then
        break;
    end if
end while
```

Figure 3. Algorithm Ransac Algorithm

(3) Set the LiDAR pointcloud topic name and IMU topic name in *"velodyne.yaml"* in config folder of *"FAST_LIO_"* which is shown in *velodyne.yaml* in Appendix 1 Figure 28.

(4) Create a new launch file *"fast_lio.launch"* in the launch folder of *"me5413_world"* package which is shown in *fast_lio.launch* in Appendix 1 Figure 27. *fast_lio.launch*

(5) Remake the repo using command *"catkin_make"*.

Then the mapping task can be done with command *"roslaunch me5413_world world.launch"* in first terminal and command *"roslaunch me5413_world fast_lio.launch"* in second terminal, and one pointcloud file *scans.pcd* will be saved.

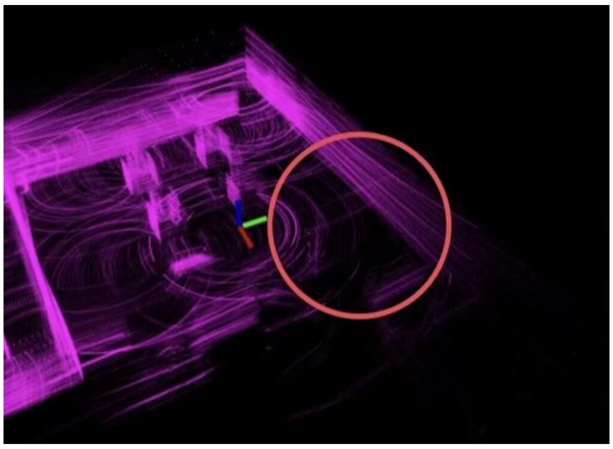


Figure 4. Failed to coincide with original pointclouds

In practice, the pointclouds are found sometimes fail to coincide with original one, like there will be two sets of pointclouds for one wall in different direction which is shown in Figure 4. After checking the callback function that publishes the point cloud as well as the synchronization of the algorithm, the problem is located at the 3D LiDAR itself and solved by increasing the scan rate from 10Hz to 50 Hz. The mapping result by FAST-LIO is shown in Figure 5.

However, the obtained point cloud still has large amounts of ground and noise points should be filtered. It is easy to find those ground cloud points are mostly on the same plane and can be filtered by constructing a simple RANdom SAmple Consensus (RANSAC) model, where all the ground points are regarded as inliers while the surrounding points as outliers [4].

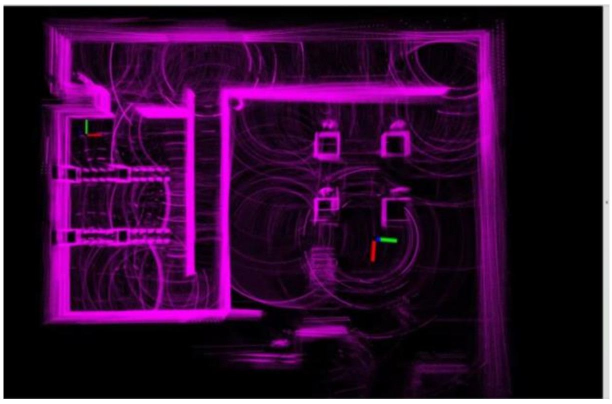


Figure 5. Point cloud mapped by FAST-LIO

The 2D plane model can be explained as,

$$\boldsymbol{aX + bY + cZ + d = 0} \tag{1}$$

which has 4 parameters and at least only 3 points are needed to solve it. As algorithm is shown in Figure 3, the data points are assumed to be inliers if its distance to the model plane is less than σ (Step 3). To accelerate the iteration, the maximum iteration number is also changed according to Eq(2):

$$\boldsymbol{iteration_{max} = \frac{log(1-P)}{log(1-t^n)}} \tag{2}$$

where P is the probability that require RANSAC algorithm to get the correct model, t is the ratio of inliers in the whole dataset, and n is the number of points. However, RANSAC algorithm performs badly when handling the non-planar ground (Figure 6), which in actual circumstances is typical. Certain low-lying items could be eliminated simultaneously since it will identify the whole plane model. Moreover, the approach may be computationally costly, and iteration count sensitive.

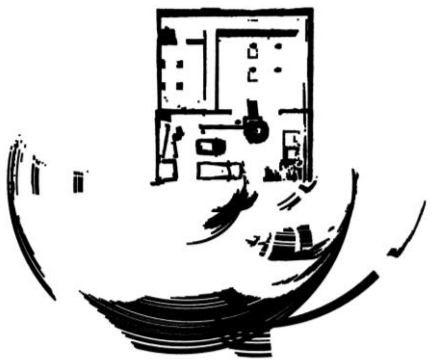


Figure 6. Ground segmentation by RANSAC

Another straightforward but efficient way is using a pass-through filter to clean the grid points since the end result is to generate a 2D grid map. All cloud points within the provided range will be eliminated up to the defined height. In implementation, the ground pointclouds are not published in RViz (modify callback function of “FAST_LIO_/LaserMapping.cpp” in Figure 29) and the result is obtained in Figure 7.

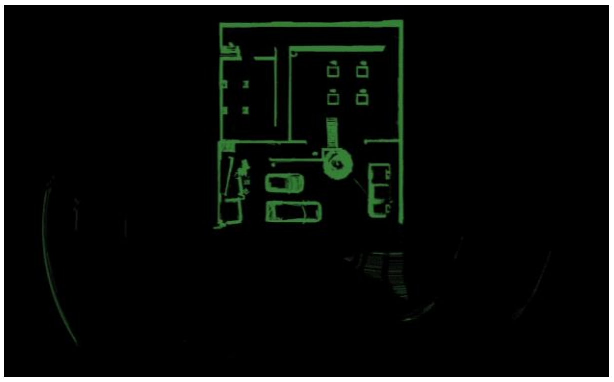


Figure 7.Ground segmentation via pass-through filter

As more ground point clouds are separated from the original one using the later filtering approach, it is evident from the above comparison that it performs better than the RANSAC algorithm. This is due to certain ground point clouds floating in the air due to the missing wall, which prevents them from fitting onto the plane. Nevertheless, certain ground points outside of the manufacturing area are still present in Figure 7, therefore the x-y axis range should also be constrained.

Moreover, a radius filter is employed to eliminate any remaining noise points following ground point cloud segmentation. The purpose underlying radius filtering is that it truly chooses locations based on the number of its neighbours are inside the spatial point's radius range. The algorithm calculates the number of points that fall inside the circle's centre for each iteration by first creating a circle in the point cloud data with a specific point at its centre. The point will be preserved if the number exceeds a predetermined value and erased if the number is less than the predetermined value.

Figure 8 illustrates the mapping performance after ground point cloud segmentation and outlier filtering. The treated point cloud is significantly "cleaner" and yet retains its structural information.

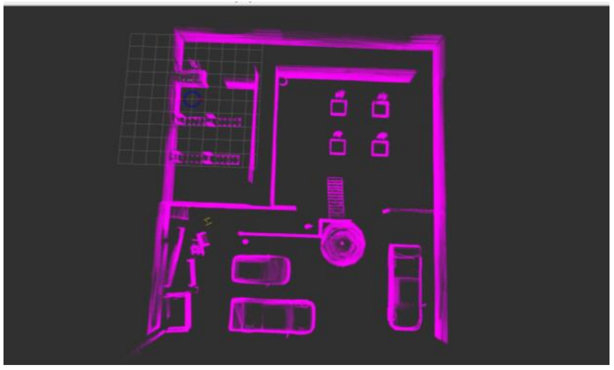


Figure 8. Resulted mapping performance after processing

FAST-LIO successfully depicts the structure of steps in comparison to those 2D mapping techniques, allowing the mobile robot to avoid traveling below them. Although the glass door cannot be scanned by the LiDAR, its perimeter can be produced and projected as a barrier. Cartography can only record the vehicle's four wheels, but the 3D SLAM system can map the entire body. Additionally, due to the file size limitation, the whole 3D mapping point cloud cannot be submitted into GitHub, one version can download in https://drive.google.com/file/d/19SF8SPli2iWed0avGKpOY63t1eXppWkJ/view?usp=share_link and view with the command as *pcl_viewer map_pointcloud.pcd*. However, the best original version is overwritten, the difference in this version is that the map is not fully scanned in the bottom right corner, but all subsequent maps are generated from perfect point cloud files, this can be considered as a reference version.

III) Generate grid map with projection

The map should be represented as an occupancy grid map, which is a discrete representation of the environment where each cell in the grid indicates the likelihood of occupancy at that point, in order to carry out the subsequent navigation assignment in ROS. As a consequence, the final point cloud file is projected onto a 2D grid map form, as seen in Figure 9 and Figure 10. According to these results, it demonstrates a fine map has been created with identifiable window and stair, and other obstacles both have a complete silhouette.

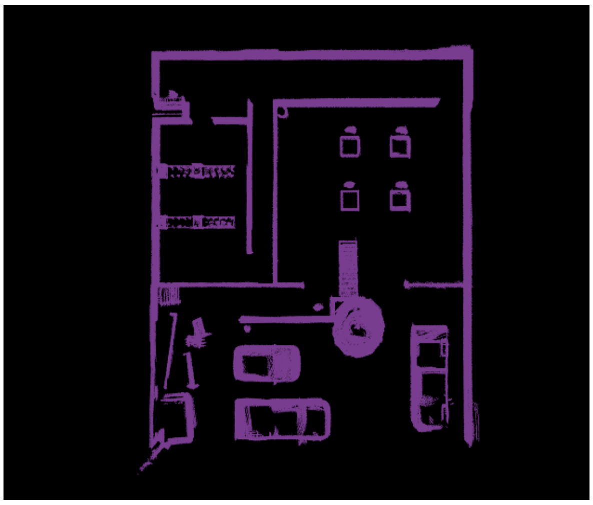


Figure 9.Schematic of mapped point cloud (Top view)

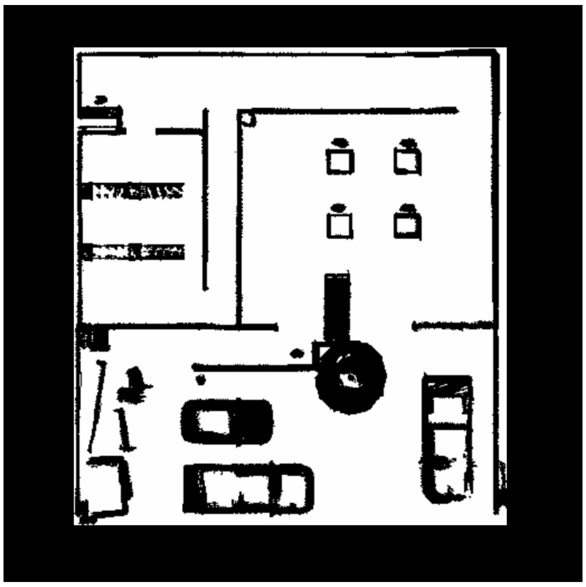


Figure 10. The resulted occupancy grid map

IV) Evaluate FAST-LIO Performance

Here the EVO tool is used to quantitatively evaluate the SLAM algorithm performance. By utilizing the command "*evo_ape*", the absolute pose error between the estimated and reference trajectory can be calculated to evaluate the algorithm performance. Moreover, the ground truth data has been stored under the topic /gazebo/ground_truth/state, and the trajectory is recorded (under topic odometry /Odometry) in a rosbag while mapping. Figure 11, Figure 12 displays the assessment performance outcomes directly. As quantifying results shown in Figure 26, the maximum alignment is 0.345405, while the mean alignment is 0.125986. The outcomes demonstrate the algorithm's accuracy performance maintains at a very good level.

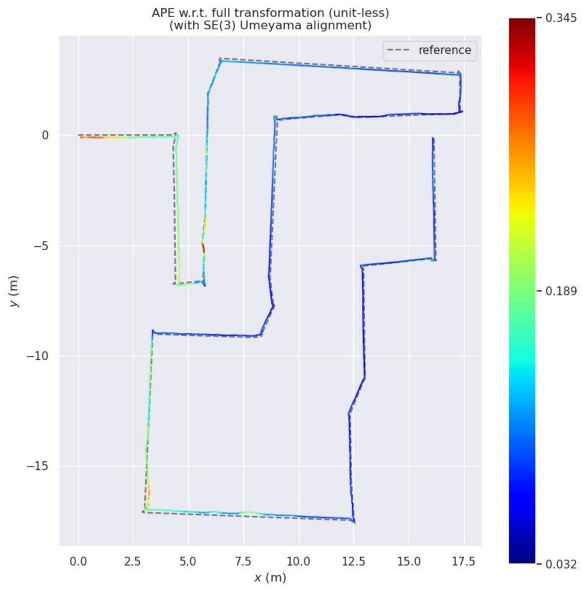


Figure 11. Evaluated trajectories

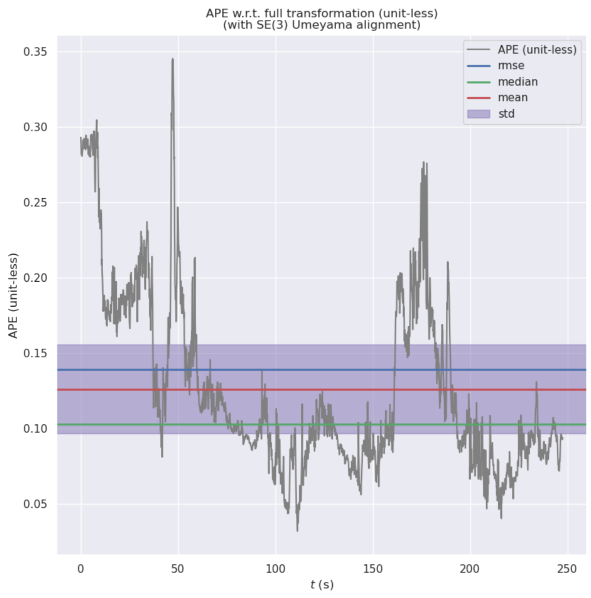


Figure 12. FAST-LIO accuracy performance using EVO

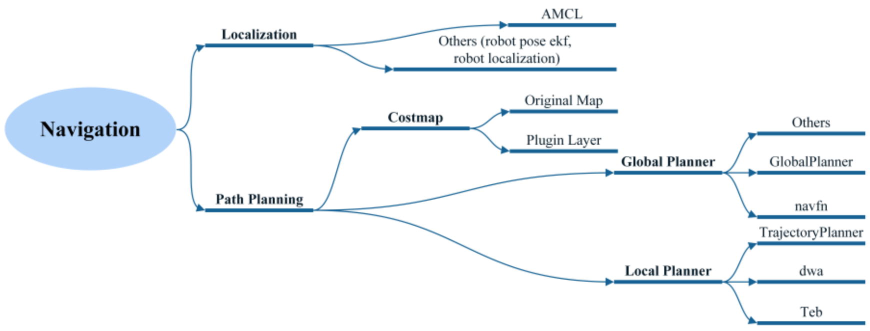


Figure 13. Schematic of navigation structure

Task 2 Navigation

The capacity of a robot to determine its own location and orientation inside the frame of reference is known as robot localization. In that it necessitates determining both the robot's present position and the position of the desired site inside the same frame of reference or coordinates, path planning is essentially an extension of localization [5]. Thus, navigation can be divided into localization and path planning with an existing map. Therefore, a specific structure tree can be utilized to describe task 2 assignment shown as Figure 13. The TF tree of this robot is shown in Appendix 2 Figure 44.

I) Localization

a) AMCL

A robot traveling in 2D can be located using the probabilistic localization system, called AMCL. It utilizes a particle filter to track a robot's attitude against a predetermined map as part of the adaptive Monte Carlo localization method (amcl) [6]. As the default locator used in ROS navigation, its parameters need to be adapted to the current task environment. The specific detailed parameters are shown in Figure 31. The maximum LIDAR beam is raised to boost the location accuracy (*3000*). Also, set the shortest and furthest LIDAR distances to the theoretical upper and lower bounds (parameter setting *-1*). Increase the minimum and maximum utilization limits (*4000, 6000*) for the particle filter parameters to enlarge the particle sample. Due to the usage of the "*diff-corrected*" model for the odometry parameters, the total "*odom alpha*" value is decreased. "*odom alpha3*" is increased (*0.3*) in order to enhance the distribution of noise estimates for straight-line runs. In order to discover the beginning position, the particle dispersion is raised by code to make it normal distributed. The code file contains the remaining adjustments.

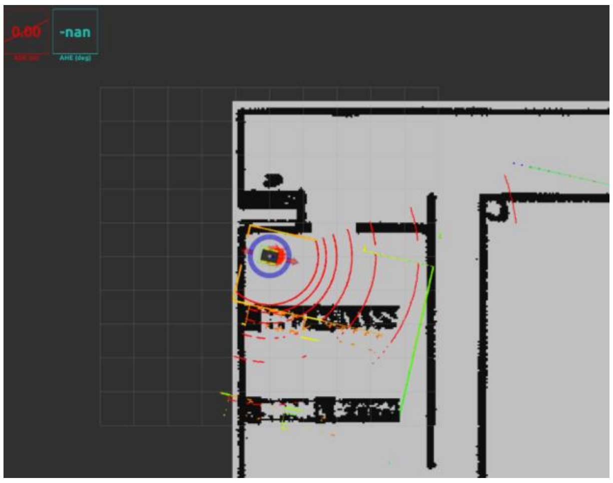


Figure 14.Schematic of frame drifting with default amcl

b) Other localization

Several localization approaches may be employed to solve this issue. Firstly, Cartographer is a SLAM method developed by Google and utilized in HW2 [7]. While having the ability to do complete localization, it is only appropriate for.pbstream maps produced by Cartographer Mapping. And it cannot be changed into.pgm.

AMCL is more suitable for the localization task with odometry, map and laser scan. And Extended Kalman Filter (EKF) method can accept and filter imu data. Robot localization is a standard package in ROS with EKF. There is a launch file named *ekf_template.launch* which can be configured in its *.yaml* in Appendix 2 Figure 32. When it is launched with AMCL together, the tf relation is conflicted and car joggles on the map. Another EKF package named *robot_pose_ekf* has been

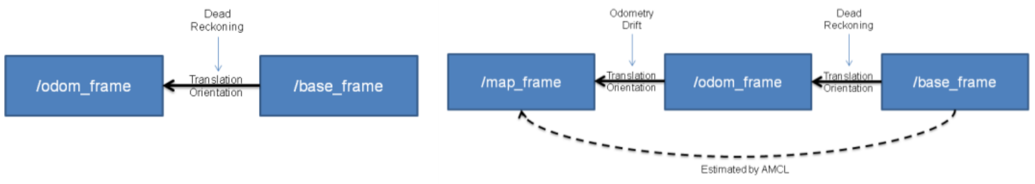


Figure 15. Odometry localization (Right) AMCL map localization (Left) [6]

installed [8]. After running it, the node graph is shown in Appendix 2 Figure 42. It indicates imu data has been received successfully.

c) Frame drifting problem

According to Figure 14, a default parameter setting amcl localization will exist a heavily frame drifting around stationary state. Laser scan frame shows that behaviour with a slowly rotation due to a bad localization. However, this problem persists when using the other positioning (*ekf*) aids mentioned above. By querying the *tf* structure, it demonstrates all these localizations are calling on the parameters of *ekf*, including odometry, which is also based on data published by *ekf* (Figure 15 [6], Figure 44). This issue is mostly spurred on by a change in the yaw angle deviation utilized for *ekf*, which is not adjusted for when at rest but is enhanced while in motion.

To solve this problem, three methods has been purposed as, (1) Tuning the *"update min"* parameters in amcl, in order to obtain a fast correction update rate. The disadvantage is a small rotation angle update can cause some stuttering when the robot operates later, due to excessive computation and redundant frame updates. (2) To prevent updating the odometer data to the frame when the speed and rotation angle are both zero, supply a node to control the odometer data subscription, modify the node of the move base, and execute this odom_control.py Figure 33. (3) Tunning other amcl parameters to achieve a slightly and slowly rotate behaviour.

II) Planning

a) Costmap

Costmap is to assign different cost to the map generated by mapping. It looks quite similar to the configuration space of robot that controls robot does not collide with obstacles. The parameters can be configured in corresponding costmap parameters files. There is another mission that the control area on the map is forbidden zone for car, which means car cannot move into that area. There are two methods can approach that target in this task.

1. Changing original map

The original grid map shows in Figure 10, in order to prohibit the robot from entering the control room, the original *pgm* file can be modified by using an editing tool such as a map editor or photoshop to modify the pixels in the restricted area to black, i.e. to indicate the presence of an obstacle. Subsequently, when costmap generation is performed, the generation value of the region is very high and no corresponding route planning is generated as Figure 16. Nevertheless, this approach modifies the map presentation , thus it is not advised.

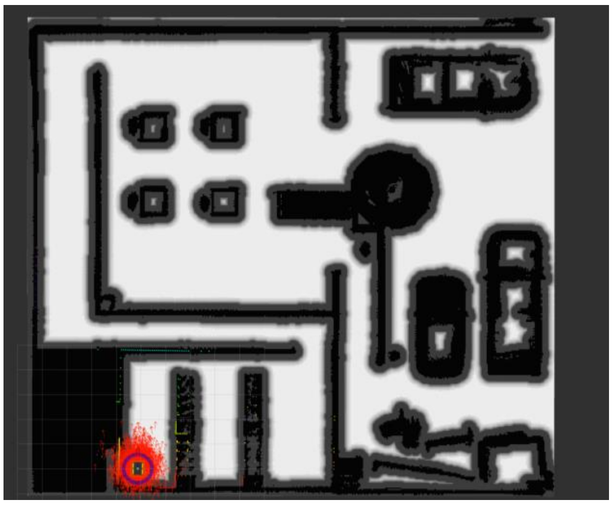


Figure 16. Costmap by changing original map

2. Plugin layer

move_base.launch will generate three layer in costmap. *StaticLayer* is mainly used to process static maps generated by *gmapping* or AMCL. *ObstacleLayer* mainly deals with obstacle inflation generated during movement.

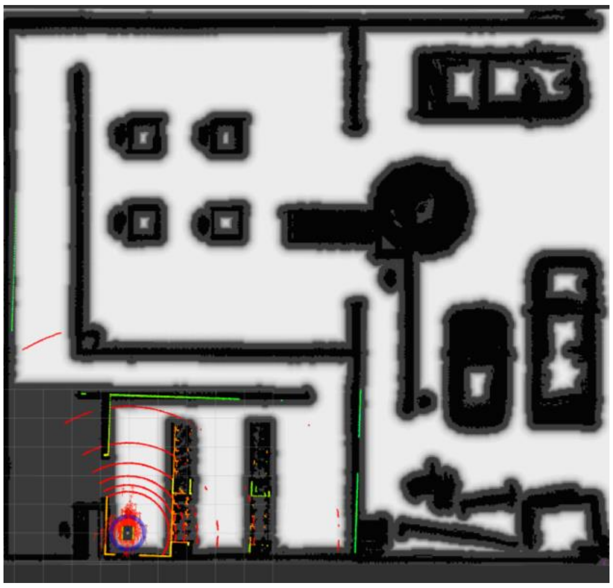


Figure 17. Costmap by plugin prohibition layer

InflationLayer mainly deals with the expansion of obstacle information on the robot's navigation ground (making the obstacles on the map larger than the actual obstacles), making the robot safer as much as possible. The second method is to add another layer, *ProhibitionLayer*, upon the *inflationlayer*. After install the package, add a *ProhibitionLayer* in *global_costmap_params.yaml* in Figure 34. And then, change the forbidden area coordinated in *prohibition_areas.yaml* to the coordinates measured before. Then in Rviz Global costmap, the corresponding area has become grey shown in Figure 17. The developed costmap would avoid this by designating the area within this coordinate to a high-cost route, satisfying the criterion. It was evident that on the original map, this area was not modified in pixel value.

b) Global Planning

1. navfn

Plans for a mobile base can be made utilizing navfn's quick interpolated navigation function, which is generated using Dijkstra's algorithm, however, the disadvantage is that it cannot handle negative weighted edges [9]. When implement this navfn in global planning, one issue that will be discovered is that the whole route cannot be created for direct setting of far distant target places, i.e. there is a route length limitaion in executation (shown in Figure 18). Thus, the vehicle will crash into an obstacle or fail to plan throughout the first half of the route when there is no path.

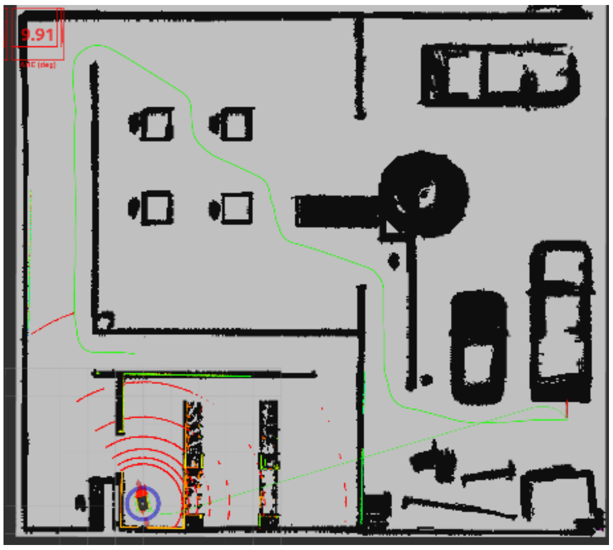


Figure 18. schematic of path shortage in navfn planner

2. Global Planner

Global Planner is another global planning algorithm which can use A* algorithm [10]. It can solve the problem contoured in *navfn* that the global planning path is not enough long to reach the goal destination, fine complete path shown in Figure 19 green line.

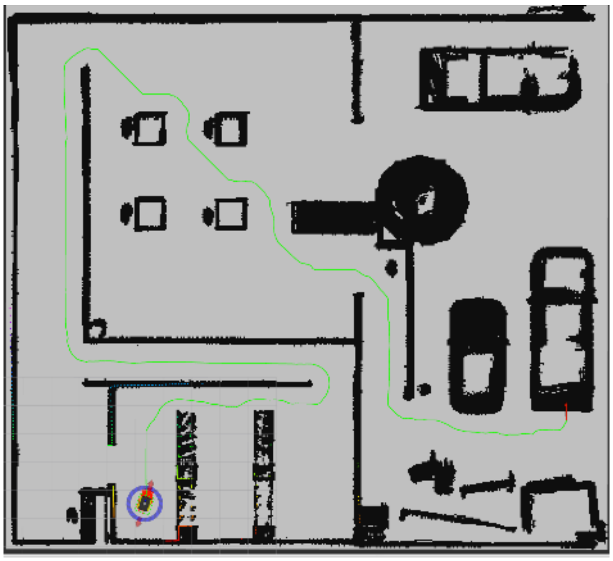


Figure 19. Complete path generated by GlobalPlanner

The parameters of global planning can be configured at *global_planner_params.yaml* in Figure 35. In order to switch the algorithm being used from Dijkstra and A*, *use_dijkstra* can be changed from true to false. Here is a problem encountered during project that path dose not shown in *Rviz*. It can be solved by changing global and local path topics in *navigation.rviz* in Appendix 2 Figure 36. Another problem is if *old_navfn_behavior* is set to be false, the path blinks in *Rviz* and sometimes car do not identify where to go, it is solved as it set to be true. The node graph of this method is shown in Figure 43.

3. Other

The global planning algorithm can also be written by the developer. According to the blog [11], the work space can be created in terminal. After changing the corresponding line in CMakeLists.txt and implement A* algorithm in *astar.cpp* and *astar.h* into *src* document. The project can be *catkin_make* again. Then create *Astar_planner.xml* and change *package.xml*, the origin port *nav_core::BaseGlobalPlanner* can be changed to *Astar_planner::AstarPlannerROS*. Then the heuristic function in A* can be settled by user and other algorithms can also be published like this.

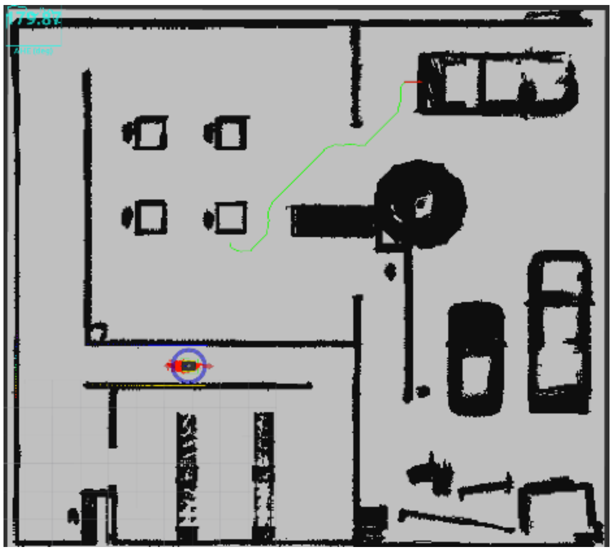


Figure 20. Path generated by *Segment_global_planner*

An easier way to use other algorithm is to download others form *github* like plug-in of the project. *Segment_global_planner* is for segments tracking [12]. This plugin allows user to set several segments as one global plan with *Rviz* as shown in Figure 20. And parameters can be configured in *segment_global_planner_params.yaml* in Figure 37. *A_better_Star* plugin is a modification of A* global planner from official *global_planner* package [13]. It utilizes approximate Euclidean distance as heuristics and make it faster than the precise value to compute. And it also updates the open list of A* for attempting to get a more optimal path. The parameters of this one is almost same with official one while it can change the *fast_euclidean_resolution* in Figure 38. The final path is figured in Figure 21. Another method is *rrt-global-planner* [14], it generates the path by rapidly exploring random tree. However, in this project, it is hard to generate global path may because of the complicated environment.

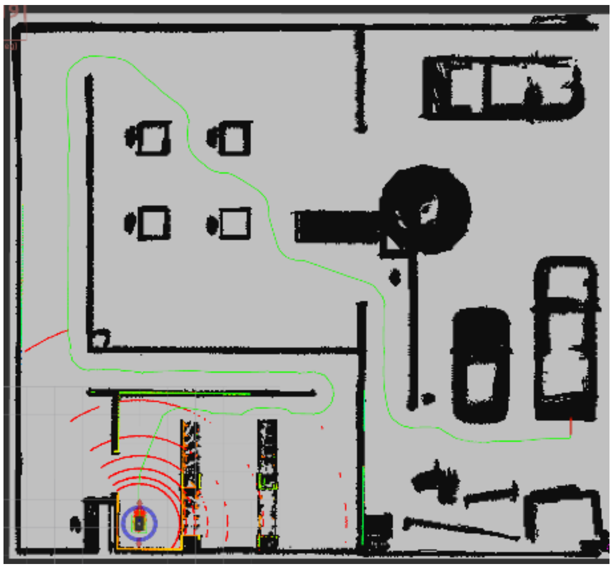


Figure 21. Path generated by *a_better_star*

All the above-mentioned methods can be switched in *move_base.launch* in Figure 40 through uncommenting other corresponding topics in it.

c) Local Planning

1. TrajectoryPlanner

A default local planner used in ros is Trajectory Rollout. Throughout the forward simulation cycle, TrajectoryPlanner determines velocities from a set of feasible velocities given the robot's maximum acceleration which has larger sample space than dwa [15]. Although it works well for straight sections, it is found that in the specific environment, the planner would leads to undesired errors at corners or near corners conditions. As shown in Figure 22, at 180-degree corners, collisions with obstacles can occur due to excessive turning speeds or small planning radius. Additionally, failure to recognise some obstacles when an early turn occurs at a straight-line distance and after a local destination conflict with the global plan, leading to serious drift and positioning errors (Figure 23). Obviously, the local planner conflicts with the global path and cannot coordinate the effect.

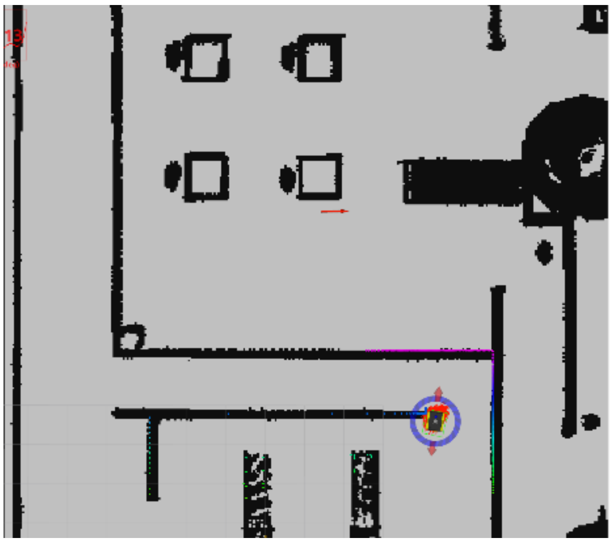


Figure 22. Schematic of corner turning behavior

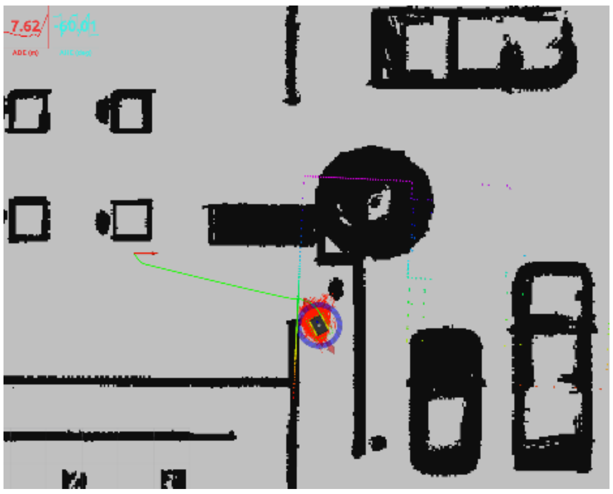


Figure 23. Schematic of collision with an obstacle

2. DWA

To score these orbital trajectories, the DWA algorithm (dynamic window approach) evaluates a lot of velocities (v, w), and the best speed is selected and delivered to the lower computer [16]. The parameters are shown in *dwa_local_planner_params.yaml* as Figure 39. The *sim_time* can be longer to do more forward simulation and set *occdist_scale* bigger to weigh more for obstacle avoidance. The velocity of robot cannot be set too high in order to prevent the path cannot be generated in time. Then it can solve the problem encountered in *base_local_planner* which shown in Figure 24. It performs a better and complete local path planning which satisfies the requirements.

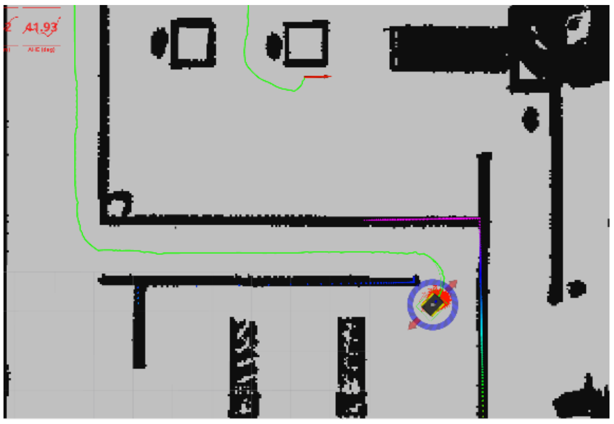


Figure 24. Fine voiding obstacles path by DWA.

3. TEB

The robot's trajectory is locally optimized by the underlying Timed Elastic Band technique in terms of trajectory execution time, separation from obstacles, and runtime kinodynamic compliance [17]. The specific parameters utilizing in teb shown in Figure 41. Worth mentioned, the footprint is required to set same as the robot as "*Polygonal*" ([-0.21, -0.165], [-0.21, 0.165], [0.21, 0.165], [0.21, -0.165]) to achieve a better identified shape in turning and obstacle avoidance.

When using TEB reaches the target position, it normally is towards the target without spinning after adjusting its orientation while in travel. In contrast, dwa initially arrives at the target coordinate point before rotating in situ to the desired orientation. Teb has a two-wheeled differential chassis that causes it to alter its orientation as it is moving, which causes an uneven course of motion and needless backtracking when it first starts moving shortly before it reaches the goal location. Although it can accomplish the aim and avoid collisions in this assignment, the robot's rear position obstacle avoidance may not always successful, according to the front laser scan sensor be applied only. Nevertheless, this does not exclude the application of other techniques or the addition of other sensors.

Table 1.Errors of different global local planner combination

	Global	navfn			GlobalPlanner			A_better_star		
Local	Error type	Three positions (131)			Three positions (131)			Three positions (131)		
DWA	Position	0.18	0.06	0.08	0.24	0.08	0.19	0.16	0.21	0.02
	Heading	-6.28	3.71	4.44	4.94	0.72	2.51	-4.58	1.11	-4.29
TEB	Position	0.26	0.06	0.18	0.04	0.16	0.16	0.13	0.11	0.12
	Heading	-2.78	3.67	-2.75	1.22	1.9	0.98	0.25	3.26	0.18

d) Other problems

Another concern is that, after the automobile has already arrived at Packing Area 3, the planned formation from Packaging Area 3 to Vehicle 1 is not very successful. As a result of the original labelled Vehicle 1 obtains high point pixel value after costmap development and its location within the obstacle, no workable path plan can be produced (shown as Figure 25).

The two possible solutions are changing the footprint padding in costmap to smaller value 0.05 or moving the Vehicle 1 position outer from the vehicle by 0.1m (within the acceptable range), this path can be generated fluently.

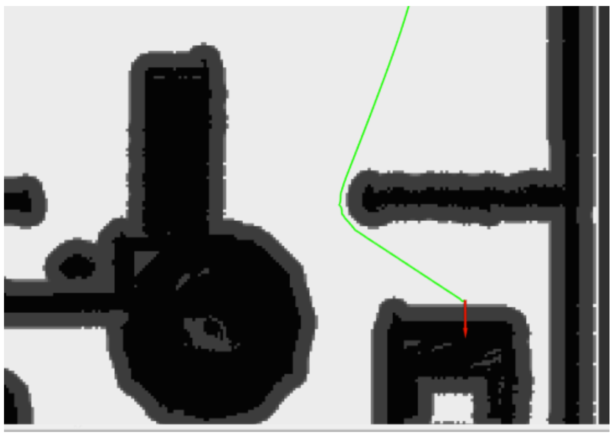


Figure 25. Schematic of vehicle 1 point position

e) Comparison and Discussion

Not as intuitive as SLAM, measures of navigation performance need to be defined on a case-by-case basis, with traditional long path comparisons including a range of measures such as path length, curvature, time cost and energy consumption [18, 19]. The most significant navigational indications in this task, however, may be classified into two categories: qualitative and quantitative. Many approaches are examined and tested in practice for the qualitative analysis as previously described, and a number of combinations that enabled flawless obstacle avoidance and arrival at the goal are chosen. Absolute location and heading errors can be employed in the quantitative analysis.

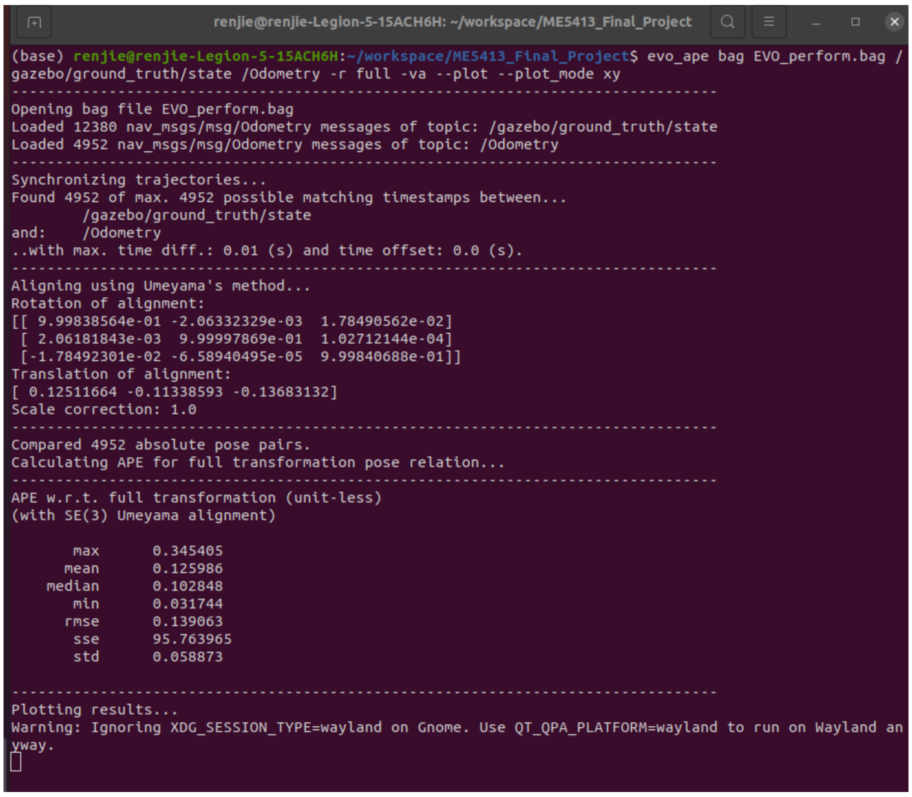
The specific value in absolute position error and heading errors are recorded in Table 1. The global planning algorithms are selected from *navfn*, *GlobalPlanner* and *a_better_star*. And the local planning algorithms are selected from DWA and TEB. As it indicates from table, the worst one is *navfn* while *GlobalPlanner* and *a_better_star* have smallest error in some points. And TEB seems have a better performance than DWA. And according to experiments, the localization method tuning has a huge influence on the error. Three methods are introduced which are *base_local_planner* (TrajectoryPlanner), *dwa_local_planner and teb_local_planner*. TrajectoryPlanner is not so advanced compared to other two methods that sometime collides with the wall. *dwa_local_planner is with a high calculation complexity with a good performance. Teb_local_planner* is more forward-looking; the car is basically kept in the global paths. With the consideration evaluated above, the final video is recorded using amcl, GlobalPlanner and dwa, and it contribute to a good and compliant performance.

All files in GitHub repo link:

https://github.com/Wang-Theo/ME5413_Final_Project

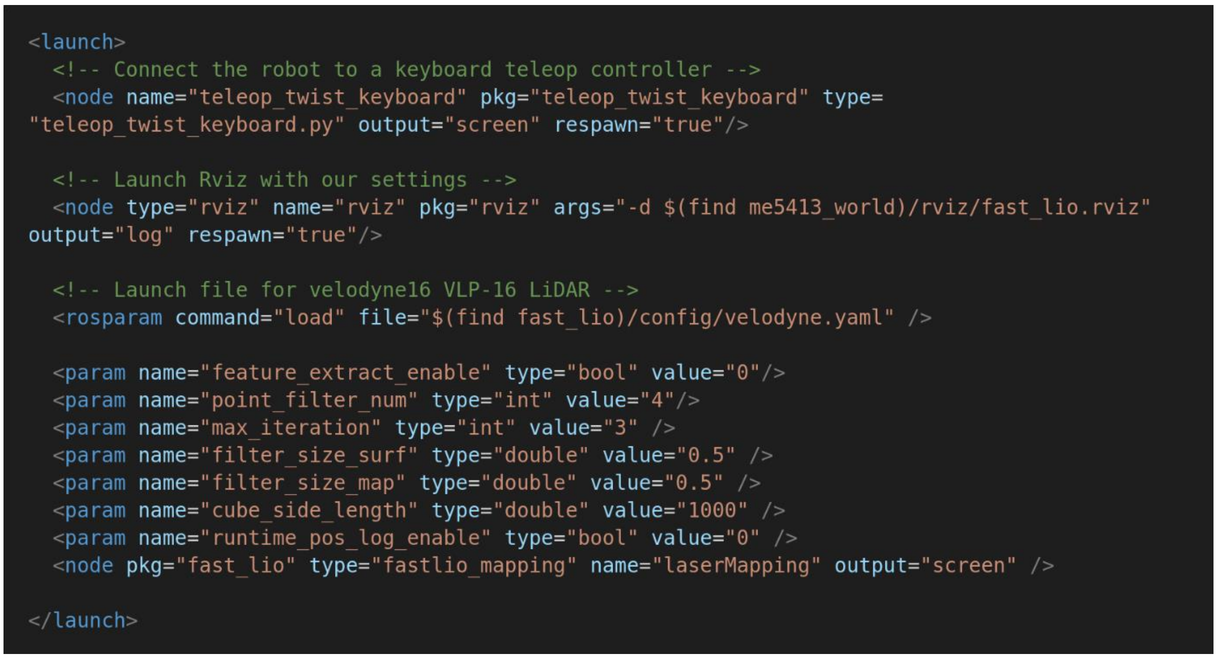
Appendix

I) Appendix 1. Task 1 Mapping


renjie@renjie-Legion-5-15ACH6H: ~/workspace/ME5413_Final_Project

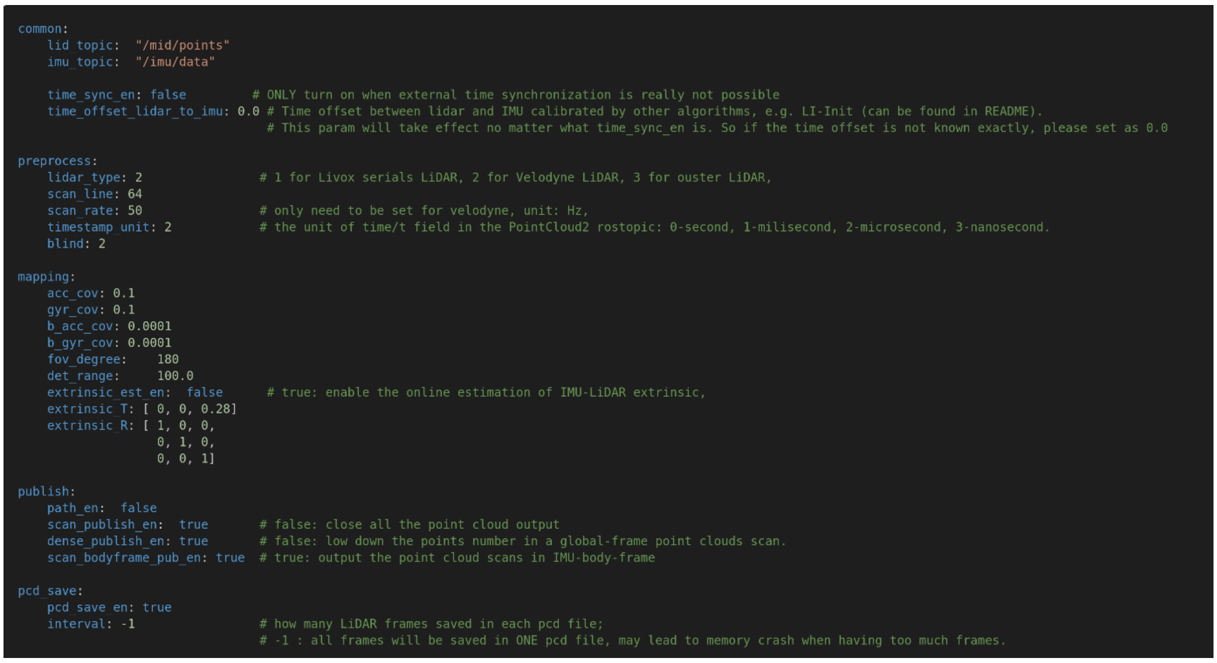
```
(base) renjie@renjie-Legion-5-15ACH6H:~/workspace/ME5413_Final_Project$ evo_ape bag EVO_perform.bag /
gazebo/ground_truth/state /Odometry -r full -va --plot --plot_mode xy
--------------------------------------------------------------------------------
Opening bag file EVO_perform.bag
Loaded 12380 nav_msgs/msg/Odometry messages of topic: /gazebo/ground_truth/state
Loaded 4952 nav_msgs/msg/Odometry messages of topic: /Odometry
--------------------------------------------------------------------------------
Synchronizing trajectories...
Found 4952 of max. 4952 possible matching timestamps between...
        /gazebo/ground_truth/state
and:    /Odometry
..with max. time diff.: 0.01 (s) and time offset: 0.0 (s).
--------------------------------------------------------------------------------
Aligning using Umeyama's method...
Rotation of alignment:
[[ 9.99838564e-01 -2.06332329e-03  1.78490562e-02]
 [ 2.06181843e-03  9.99997869e-01  1.02712144e-04]
 [-1.78492301e-02 -6.58940495e-05  9.99840688e-01]]
Translation of alignment:
[ 0.12511664 -0.11338593 -0.13683132]
Scale correction: 1.0
--------------------------------------------------------------------------------
Compared 4952 absolute pose pairs.
Calculating APE for full transformation pose relation...
--------------------------------------------------------------------------------
APE w.r.t. full transformation (unit-less)
(with SE(3) Umeyama alignment)

       max      0.345405
      mean      0.125986
    median      0.102848
       min      0.031744
      rmse      0.139063
       sse      95.763965
       std      0.058873

--------------------------------------------------------------------------------
Plotting results...
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland an
yway.
```

Figure 26. Evaluation performance results



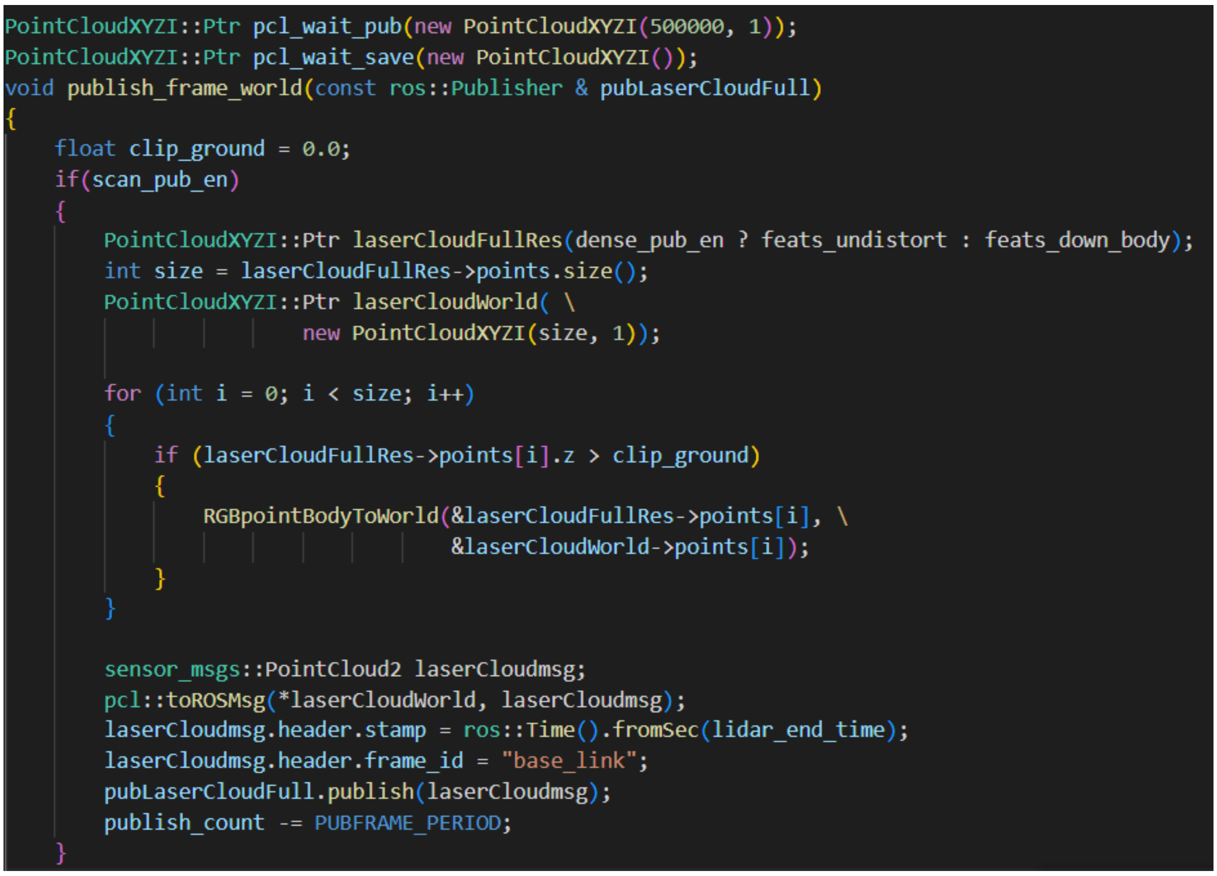
```
<launch>
  <!-- Connect the robot to a keyboard teleop controller -->
  <node name="teleop_twist_keyboard" pkg="teleop_twist_keyboard" type=
"teleop_twist_keyboard.py" output="screen" respawn="true"/>

  <!-- Launch Rviz with our settings -->
  <node type="rviz" name="rviz" pkg="rviz" args="-d $(find me5413_world)/rviz/fast_lio.rviz"
output="log" respawn="true"/>

  <!-- Launch file for velodyne16 VLP-16 LiDAR -->
  <rosparam command="load" file="$(find fast_lio)/config/velodyne.yaml" />

  <param name="feature_extract_enable" type="bool" value="0"/>
  <param name="point_filter_num" type="int" value="4"/>
  <param name="max_iteration" type="int" value="3" />
  <param name="filter_size_surf" type="double" value="0.5" />
  <param name="filter_size_map" type="double" value="0.5" />
  <param name="cube_side_length" type="double" value="1000" />
  <param name="runtime_pos_log_enable" type="bool" value="0" />
  <node pkg="fast_lio" type="fastlio_mapping" name="laserMapping" output="screen" />

</launch>
```

Figure 27. *fast_lio.launch*



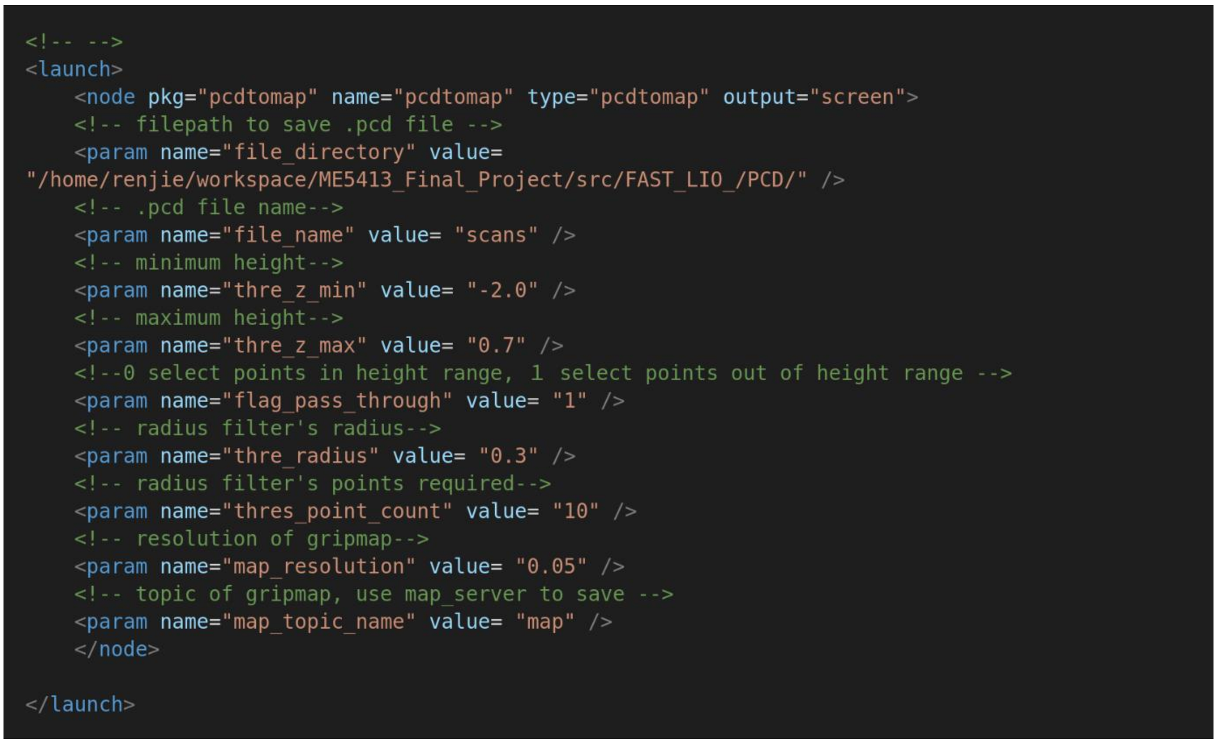
```yaml
common:
    lid_topic:  "/mid/points"
    imu_topic:  "/imu/data"

    time_sync_en: false         # ONLY turn on when external time synchronization is really not possible
    time_offset_lidar_to_imu: 0.0 # Time offset between lidar and IMU calibrated by other algorithms, e.g. LI-Init (can be found in README).
                                  # This param will take effect no matter what time_sync_en is. So if the time offset is not known exactly, please set as 0.0

preprocess:
    lidar_type: 2                # 1 for Livox serials LiDAR, 2 for Velodyne LiDAR, 3 for ouster LiDAR,
    scan_line: 64
    scan_rate: 50                # only need to be set for velodyne, unit: Hz,
    timestamp_unit: 2            # the unit of time/t field in the PointCloud2 rostopic: 0-second, 1-milisecond, 2-microsecond, 3-nanosecond.
    blind: 2

mapping:
    acc_cov: 0.1
    gyr_cov: 0.1
    b_acc_cov: 0.0001
    b_gyr_cov: 0.0001
    fov_degree:    180
    det_range:     100.0
    extrinsic_est_en:  false      # true: enable the online estimation of IMU-LiDAR extrinsic,
    extrinsic_T: [ 0, 0, 0.28]
    extrinsic_R: [ 1, 0, 0,
                   0, 1, 0,
                   0, 0, 1]

publish:
    path_en:  false
    scan_publish_en:  true       # false: close all the point cloud output
    dense_publish_en: true       # false: low down the points number in a global-frame point clouds scan.
    scan_bodyframe_pub_en: true  # true: output the point cloud scans in IMU-body-frame

pcd_save:
    pcd_save_en: true
    interval: -1                 # how many LiDAR frames saved in each pcd file;
                                 # -1 : all frames will be saved in ONE pcd file, may lead to memory crash when having too much frames.
```

Figure 28. *velodyne.yaml*



```cpp
PointCloudXYZI::Ptr pcl_wait_pub(new PointCloudXYZI(500000, 1));
PointCloudXYZI::Ptr pcl_wait_save(new PointCloudXYZI());
void publish_frame_world(const ros::Publisher & pubLaserCloudFull)
{
    float clip_ground = 0.0;
    if(scan_pub_en)
    {
        PointCloudXYZI::Ptr laserCloudFullRes(dense_pub_en ? feats_undistort : feats_down_body);
        int size = laserCloudFullRes->points.size();
        PointCloudXYZI::Ptr laserCloudWorld( \
                        new PointCloudXYZI(size, 1));

        for (int i = 0; i < size; i++)
        {
            if (laserCloudFullRes->points[i].z > clip_ground)
            {
                RGBpointBodyToWorld(&laserCloudFullRes->points[i], \
                                    &laserCloudWorld->points[i]);
            }
        }

        sensor_msgs::PointCloud2 laserCloudmsg;
        pcl::toROSMsg(*laserCloudWorld, laserCloudmsg);
        laserCloudmsg.header.stamp = ros::Time().fromSec(lidar_end_time);
        laserCloudmsg.header.frame_id = "base_link";
        pubLaserCloudFull.publish(laserCloudmsg);
        publish_count -= PUBFRAME_PERIOD;
    }
```

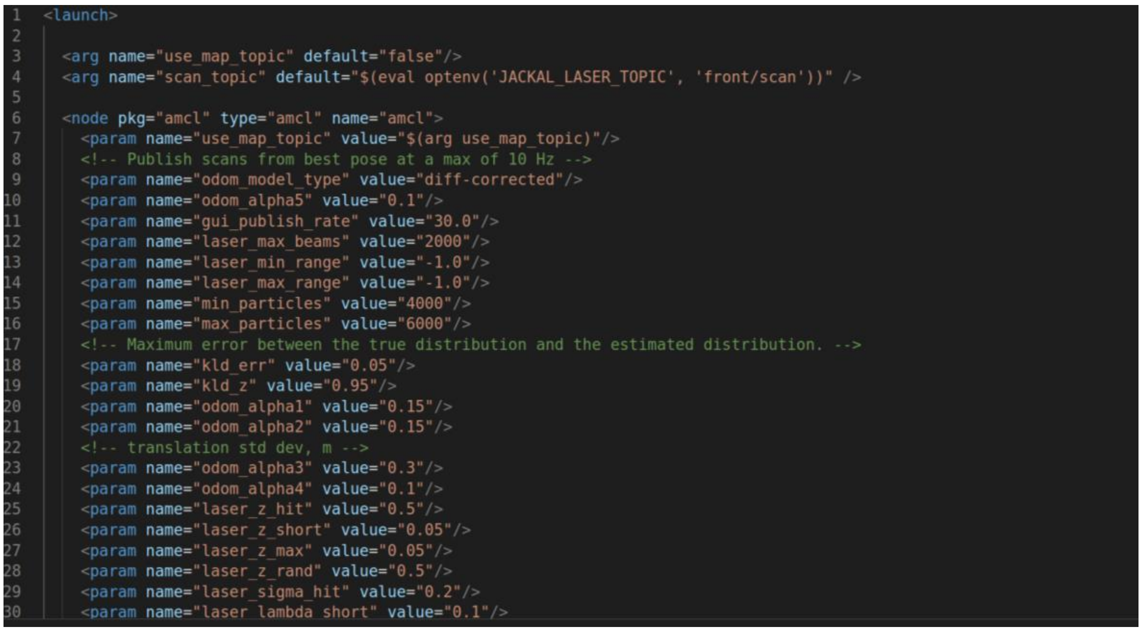
Figure 29. pointcloud callback function in *LaserMapping.cpp*



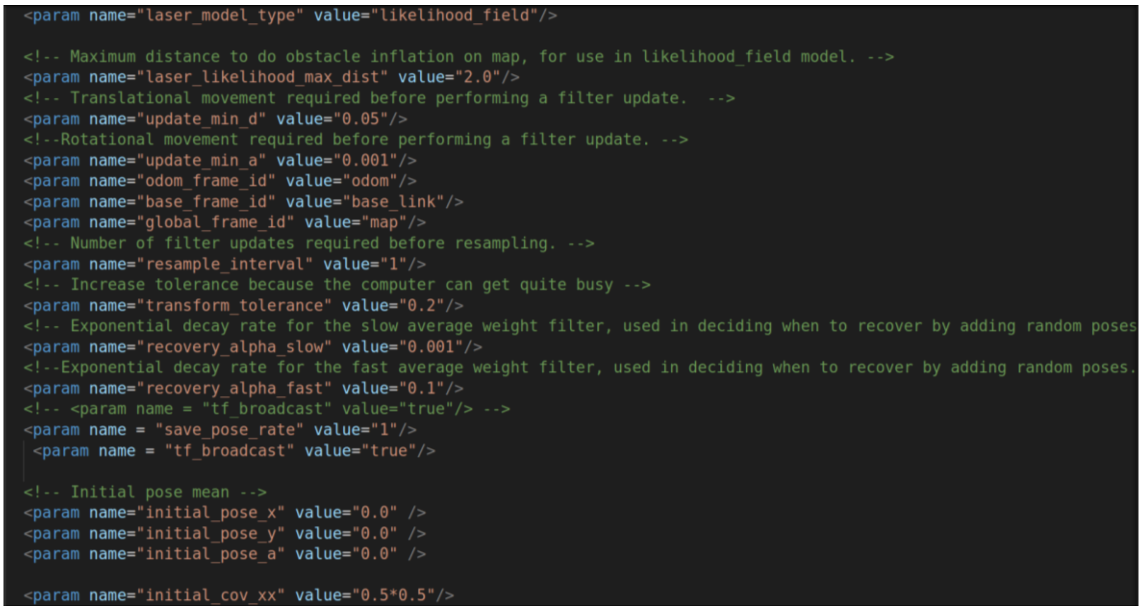
```xml
<!-- -->
<launch>
    <node pkg="pcdtomap" name="pcdtomap" type="pcdtomap" output="screen">
    <!-- filepath to save .pcd file -->
    <param name="file_directory" value=
"/home/renjie/workspace/ME5413_Final_Project/src/FAST_LIO_/PCD/" />
    <!-- .pcd file name-->
    <param name="file_name" value= "scans" />
    <!-- minimum height-->
    <param name="thre_z_min" value= "-2.0" />
    <!-- maximum height-->
    <param name="thre_z_max" value= "0.7" />
    <!--0 select points in height range, 1 select points out of height range -->
    <param name="flag_pass_through" value= "1" />
    <!-- radius filter's radius-->
    <param name="thre_radius" value= "0.3" />
    <!-- radius filter's points required-->
    <param name="thres_point_count" value= "10" />
    <!-- resolution of gripmap-->
    <param name="map_resolution" value= "0.05" />
    <!-- topic of gripmap, use map_server to save -->
    <param name="map_topic_name" value= "map" />
    </node>

</launch>
```

Figure 30. *pcdtomap.launch*

II) Appendix 2. Task 2 Navigation

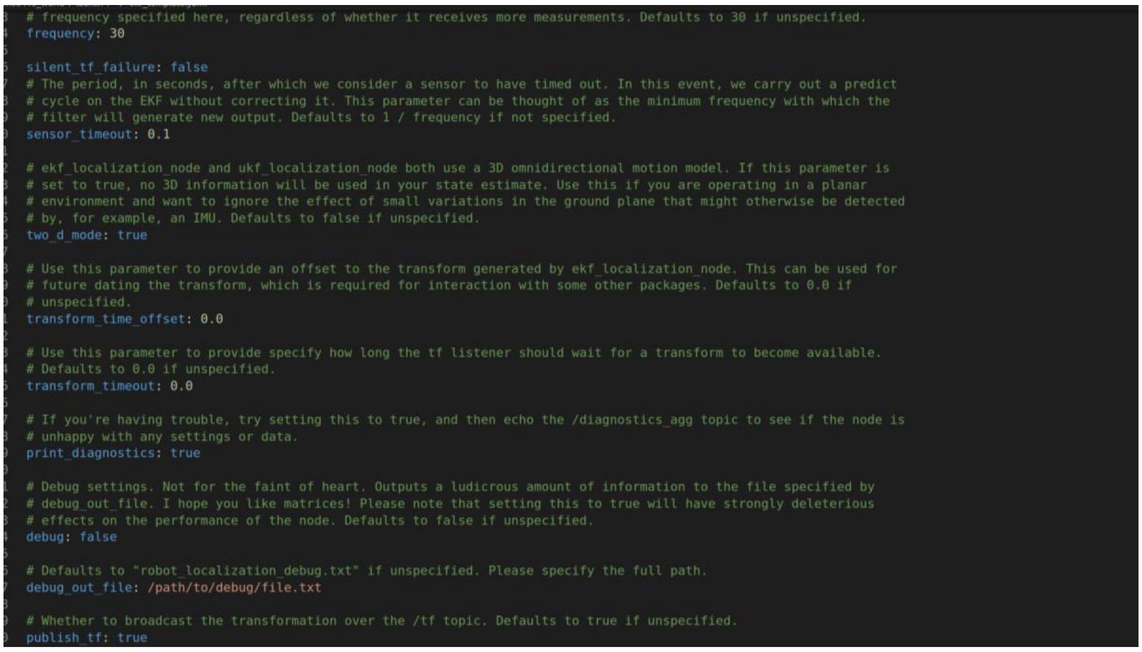


```xml
<launch>

  <arg name="use_map_topic" default="false"/>
  <arg name="scan_topic" default="$(eval optenv('JACKAL_LASER_TOPIC', 'front/scan'))" />

  <node pkg="amcl" type="amcl" name="amcl">
    <param name="use_map_topic" value="$(arg use_map_topic)"/>
    <!-- Publish scans from best pose at a max of 10 Hz -->
    <param name="odom_model_type" value="diff-corrected"/>
    <param name="odom_alpha5" value="0.1"/>
    <param name="gui_publish_rate" value="30.0"/>
    <param name="laser_max_beams" value="2000"/>
    <param name="laser_min_range" value="-1.0"/>
    <param name="laser_max_range" value="-1.0"/>
    <param name="min_particles" value="4000"/>
    <param name="max_particles" value="6000"/>
    <!-- Maximum error between the true distribution and the estimated distribution. -->
    <param name="kld_err" value="0.05"/>
    <param name="kld_z" value="0.95"/>
    <param name="odom_alpha1" value="0.15"/>
    <param name="odom_alpha2" value="0.15"/>
    <!-- translation std dev, m -->
    <param name="odom_alpha3" value="0.3"/>
    <param name="odom_alpha4" value="0.1"/>
    <param name="laser_z_hit" value="0.5"/>
    <param name="laser_z_short" value="0.05"/>
    <param name="laser_z_max" value="0.05"/>
    <param name="laser_z_rand" value="0.5"/>
    <param name="laser_sigma_hit" value="0.2"/>
    <param name="laser_lambda_short" value="0.1"/>
```



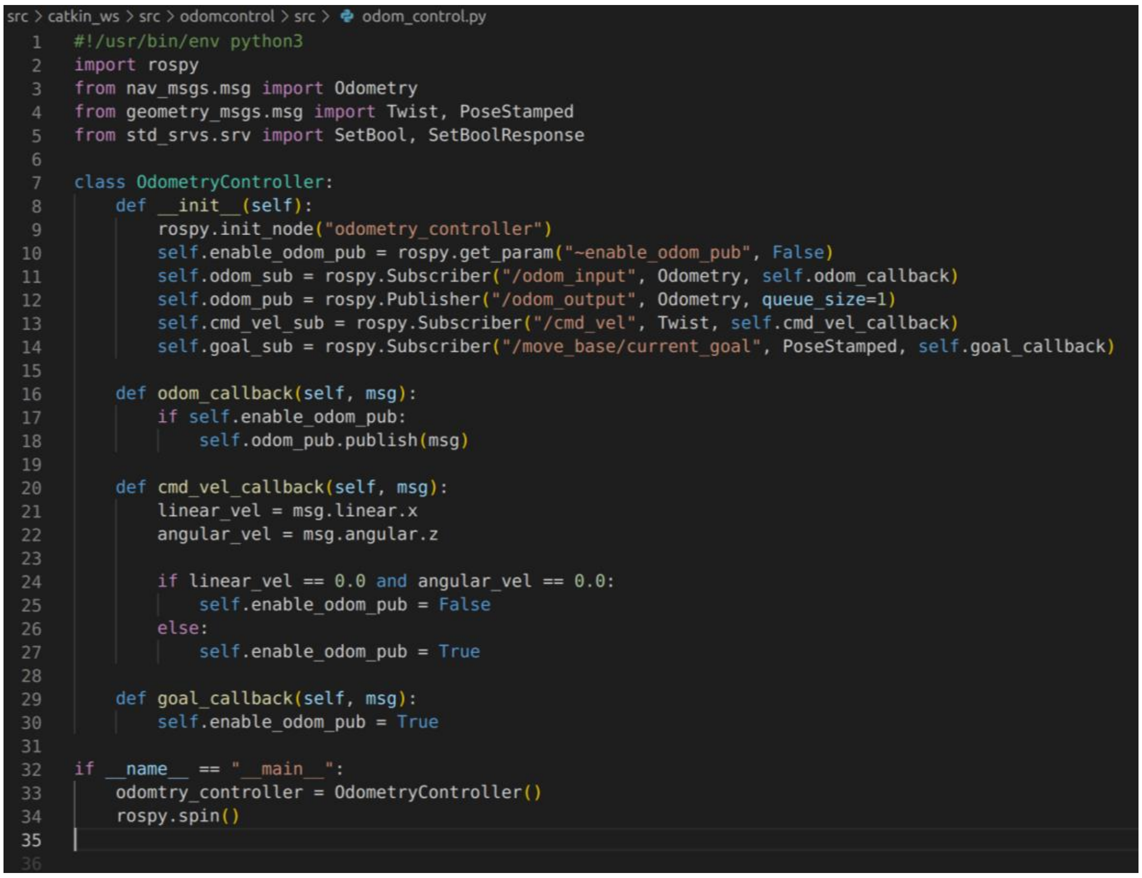
```
<param name="laser_model_type" value="likelihood_field"/>

<!-- Maximum distance to do obstacle inflation on map, for use in likelihood_field model. -->
<param name="laser_likelihood_max_dist" value="2.0"/>
<!-- Translational movement required before performing a filter update.  -->
<param name="update_min_d" value="0.05"/>
<!--Rotational movement required before performing a filter update. -->
<param name="update_min_a" value="0.001"/>
<param name="odom_frame_id" value="odom"/>
<param name="base_frame_id" value="base_link"/>
<param name="global_frame_id" value="map"/>
<!-- Number of filter updates required before resampling. -->
<param name="resample_interval" value="1"/>
<!-- Increase tolerance because the computer can get quite busy -->
<param name="transform_tolerance" value="0.2"/>
<!-- Exponential decay rate for the slow average weight filter, used in deciding when to recover by adding random poses
<param name="recovery_alpha_slow" value="0.001"/>
<!--Exponential decay rate for the fast average weight filter, used in deciding when to recover by adding random poses.
<param name="recovery_alpha_fast" value="0.1"/>
<!-- <param name = "tf_broadcast" value="true"/> -->
<param name = "save_pose_rate" value="1"/>
 <param name = "tf_broadcast" value="true"/>

<!-- Initial pose mean -->
<param name="initial_pose_x" value="0.0" />
<param name="initial_pose_y" value="0.0" />
<param name="initial_pose_a" value="0.0" />

<param name="initial_cov_xx" value="0.5*0.5"/>
```

Figure 31. *amcl params.yaml*



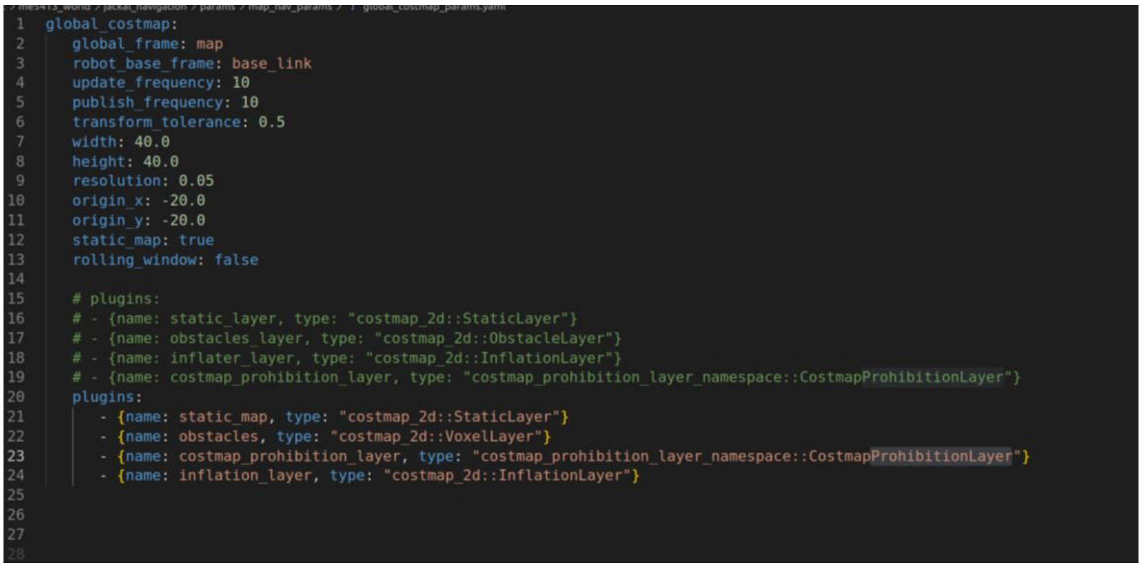
```
# frequency specified here, regardless of whether it receives more measurements. Defaults to 30 if unspecified.
frequency: 30

silent_tf_failure: false
# The period, in seconds, after which we consider a sensor to have timed out. In this event, we carry out a predict
# cycle on the EKF without correcting it. This parameter can be thought of as the minimum frequency with which the
# filter will generate new output. Defaults to 1 / frequency if not specified.
sensor_timeout: 0.1

# ekf_localization_node and ukf_localization_node both use a 3D omnidirectional motion model. If this parameter is
# set to true, no 3D information will be used in your state estimate. Use this if you are operating in a planar
# environment and want to ignore the effect of small variations in the ground plane that might otherwise be detected
# by, for example, an IMU. Defaults to false if unspecified.
two_d_mode: true

# Use this parameter to provide an offset to the transform generated by ekf_localization_node. This can be used for
# future dating the transform, which is required for interaction with some other packages. Defaults to 0.0 if
# unspecified.
transform_time_offset: 0.0

# Use this parameter to provide specify how long the tf listener should wait for a transform to become available.
# Defaults to 0.0 if unspecified.
transform_timeout: 0.0

# If you're having trouble, try setting this to true, and then echo the /diagnostics_agg topic to see if the node is
# unhappy with any settings or data.
print_diagnostics: true

# Debug settings. Not for the faint of heart. Outputs a ludicrous amount of information to the file specified by
# debug_out_file. I hope you like matrices! Please note that setting this to true will have strongly deleterious
# effects on the performance of the node. Defaults to false if unspecified.
debug: false

# Defaults to "robot_localization_debug.txt" if unspecified. Please specify the full path.
debug_out_file: /path/to/debug/file.txt

# Whether to broadcast the transformation over the /tf topic. Defaults to true if unspecified.
publish_tf: true
```

Figure 32. *ekf_template_params.yaml*


src > catkin_ws > src > odomcontrol > src > odom_control.py

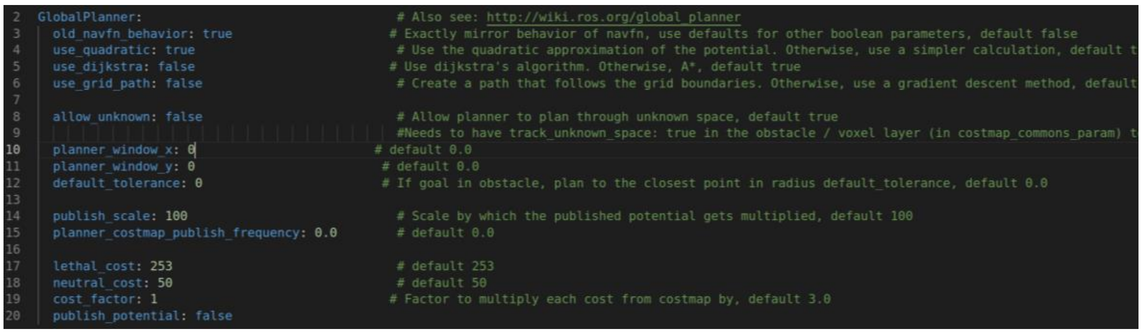
```python
#!/usr/bin/env python3
import rospy
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist, PoseStamped
from std_srvs.srv import SetBool, SetBoolResponse

class OdometryController:
    def __init__(self):
        rospy.init_node("odometry_controller")
        self.enable_odom_pub = rospy.get_param("~enable_odom_pub", False)
        self.odom_sub = rospy.Subscriber("/odom_input", Odometry, self.odom_callback)
        self.odom_pub = rospy.Publisher("/odom_output", Odometry, queue_size=1)
        self.cmd_vel_sub = rospy.Subscriber("/cmd_vel", Twist, self.cmd_vel_callback)
        self.goal_sub = rospy.Subscriber("/move_base/current_goal", PoseStamped, self.goal_callback)

    def odom_callback(self, msg):
        if self.enable_odom_pub:
            self.odom_pub.publish(msg)

    def cmd_vel_callback(self, msg):
        linear_vel = msg.linear.x
        angular_vel = msg.angular.z

        if linear_vel == 0.0 and angular_vel == 0.0:
            self.enable_odom_pub = False
        else:
            self.enable_odom_pub = True

    def goal_callback(self, msg):
        self.enable_odom_pub = True

if __name__ == "__main__":
    odomtry_controller = OdometryController()
    rospy.spin()
```

Figure 33. odom_control.py


me5413_world > jackal_navigation > params > map_nav_params > global_costmap_params.yaml

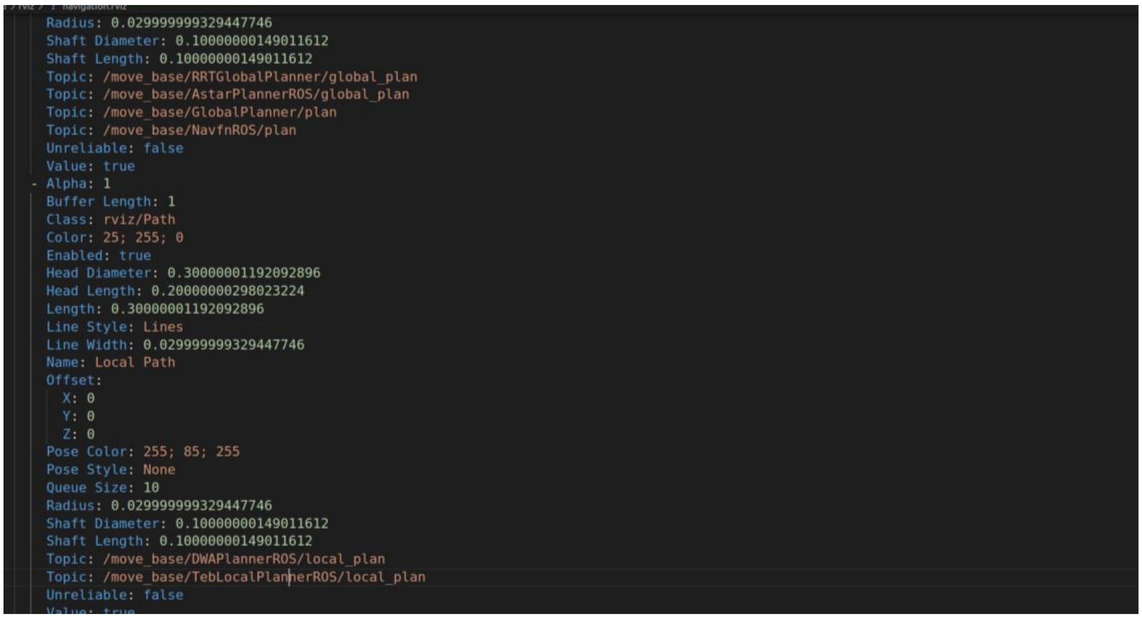
```yaml
global_costmap:
  global_frame: map
  robot_base_frame: base_link
  update_frequency: 10
  publish_frequency: 10
  transform_tolerance: 0.5
  width: 40.0
  height: 40.0
  resolution: 0.05
  origin_x: -20.0
  origin_y: -20.0
  static_map: true
  rolling_window: false

  # plugins:
  # - {name: static_layer, type: "costmap_2d::StaticLayer"}
  # - {name: obstacles_layer, type: "costmap_2d::ObstacleLayer"}
  # - {name: inflater_layer, type: "costmap_2d::InflationLayer"}
  # - {name: costmap_prohibition_layer, type: "costmap_prohibition_layer_namespace::CostmapProhibitionLayer"}
  plugins:
    - {name: static_map, type: "costmap_2d::StaticLayer"}
    - {name: obstacles, type: "costmap_2d::VoxelLayer"}
    - {name: costmap_prohibition_layer, type: "costmap_prohibition_layer_namespace::CostmapProhibitionLayer"}
    - {name: inflation_layer, type: "costmap_2d::InflationLayer"}
```

Figure 34. *global_costmap_params.yaml*



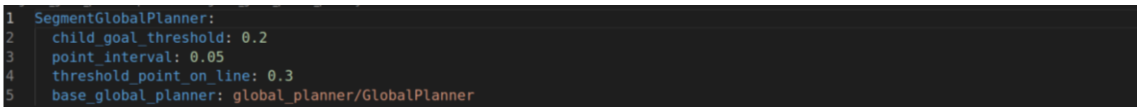
```yaml
GlobalPlanner:                                  # Also see: http://wiki.ros.org/global_planner
  old_navfn_behavior: true                     # Exactly mirror behavior of navfn, use defaults for other boolean parameters, default false
  use_quadratic: true                           # Use the quadratic approximation of the potential. Otherwise, use a simpler calculation, default t
  use_dijkstra: false                          # Use dijkstra's algorithm. Otherwise, A*, default true
  use_grid_path: false                          # Create a path that follows the grid boundaries. Otherwise, use a gradient descent method, default

  allow_unknown: false                          # Allow planner to plan through unknown space, default true
                                                #Needs to have track_unknown_space: true in the obstacle / voxel layer (in costmap_commons_param) t
  planner_window_x: 0                         # default 0.0
  planner_window_y: 0                          # default 0.0
  default_tolerance: 0                         # If goal in obstacle, plan to the closest point in radius default_tolerance, default 0.0

  publish_scale: 100                            # Scale by which the published potential gets multiplied, default 100
  planner_costmap_publish_frequency: 0.0        # default 0.0

  lethal_cost: 253                              # default 253
  neutral_cost: 50                              # default 50
  cost_factor: 1                               # Factor to multiply each cost from costmap by, default 3.0
  publish_potential: false
```

Figure 35. *global_planner_params.yaml*



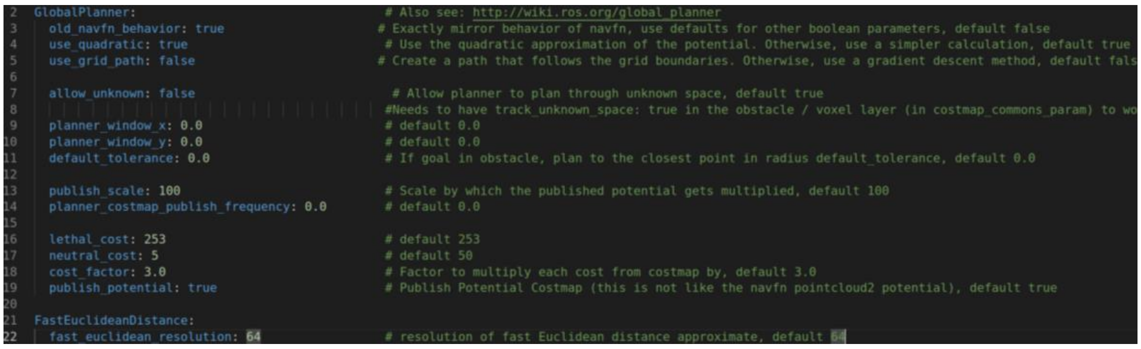
```
    Radius: 0.029999999329447746
    Shaft Diameter: 0.10000000149011612
    Shaft Length: 0.10000000149011612
    Topic: /move_base/RRTGlobalPlanner/global_plan
    Topic: /move_base/AstarPlannerROS/global_plan
    Topic: /move_base/GlobalPlanner/plan
    Topic: /move_base/NavfnROS/plan
    Unreliable: false
    Value: true
  - Alpha: 1
    Buffer Length: 1
    Class: rviz/Path
    Color: 25; 255; 0
    Enabled: true
    Head Diameter: 0.30000001192092896
    Head Length: 0.20000000298023224
    Length: 0.30000001192092896
    Line Style: Lines
    Line Width: 0.029999999329447746
    Name: Local Path
    Offset:
      X: 0
      Y: 0
      Z: 0
    Pose Color: 255; 85; 255
    Pose Style: None
    Queue Size: 10
    Radius: 0.029999999329447746
    Shaft Diameter: 0.10000000149011612
    Shaft Length: 0.10000000149011612
    Topic: /move_base/DWAPlannerROS/local_plan
    Topic: /move_base/TebLocalPlannerROS/local_plan
    Unreliable: false
    Value: true
```

Figure 36. *navigation.rviz*



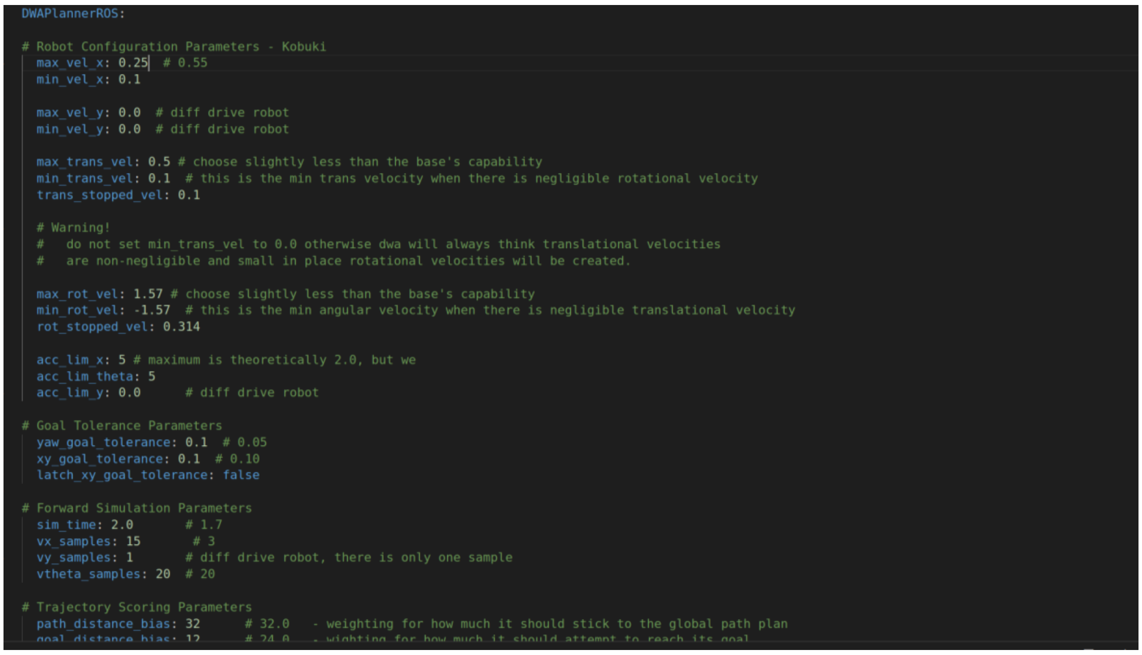
```yaml
SegmentGlobalPlanner:
  child_goal_threshold: 0.2
  point_interval: 0.05
  threshold_point_on_line: 0.3
  base_global_planner: global_planner/GlobalPlanner
```

Figure 37. *segment_global_planner_params.yaml*

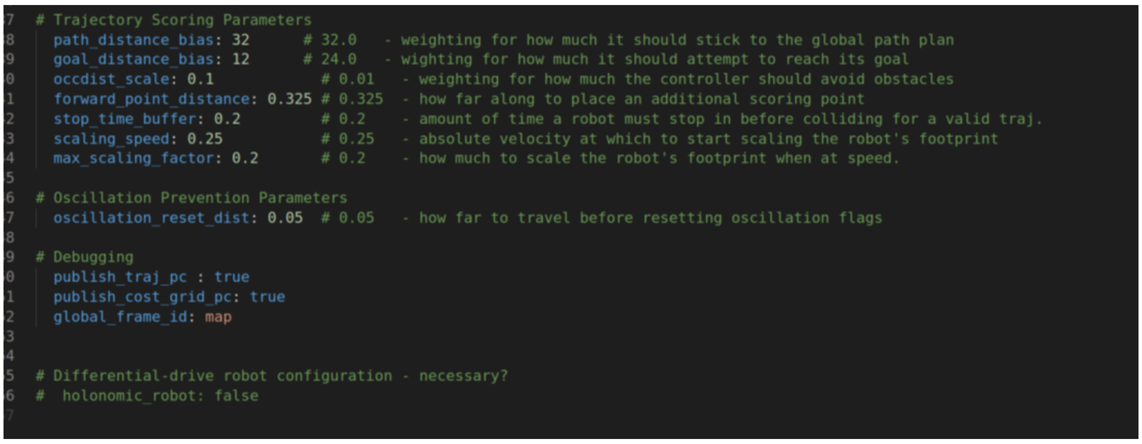


```yaml
GlobalPlanner:                                 # Also see: http://wiki.ros.org/global_planner
  old_navfn_behavior: true                    # Exactly mirror behavior of navfn, use defaults for other boolean parameters, default false
  use_quadratic: true                          # Use the quadratic approximation of the potential. Otherwise, use a simpler calculation, default true
  use_grid_path: false                        # Create a path that follows the grid boundaries. Otherwise, use a gradient descent method, default fals

  allow_unknown: false                          # Allow planner to plan through unknown space, default true
                                               #Needs to have track_unknown_space: true in the obstacle / voxel layer (in costmap_commons_param) to wo
  planner_window_x: 0.0                        # default 0.0
  planner_window_y: 0.0                        # default 0.0
  default_tolerance: 0.0                       # If goal in obstacle, plan to the closest point in radius default_tolerance, default 0.0

  publish_scale: 100                           # Scale by which the published potential gets multiplied, default 100
  planner_costmap_publish_frequency: 0.0       # default 0.0

  lethal_cost: 253                             # default 253
  neutral_cost: 5                              # default 50
  cost_factor: 3.0                             # Factor to multiply each cost from costmap by, default 3.0
  publish_potential: true                      # Publish Potential Costmap (this is not like the navfn pointcloud2 potential), default true

FastEuclideanDistance:
  fast_euclidean_resolution: 64                # resolution of fast Euclidean distance approximate, default 64
```

Figure 38. global_planner_params.yaml (a_better_star)

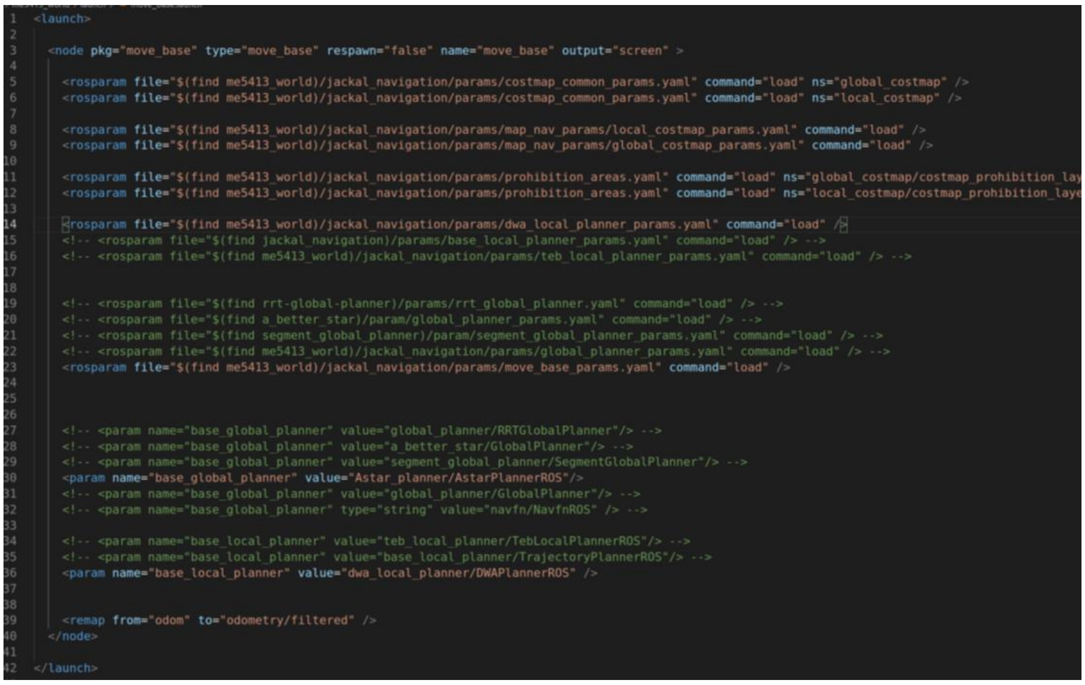


```yaml
DWAPlannerROS:

# Robot Configuration Parameters - Kobuki
  max_vel_x: 0.25  # 0.55
  min_vel_x: 0.1

  max_vel_y: 0.0  # diff drive robot
  min_vel_y: 0.0  # diff drive robot

  max_trans_vel: 0.5 # choose slightly less than the base's capability
  min_trans_vel: 0.1  # this is the min trans velocity when there is negligible rotational velocity
  trans_stopped_vel: 0.1

  # Warning!
  #   do not set min_trans_vel to 0.0 otherwise dwa will always think translational velocities
  #   are non-negligible and small in place rotational velocities will be created.

  max_rot_vel: 1.57 # choose slightly less than the base's capability
  min_rot_vel: -1.57  # this is the min angular velocity when there is negligible translational velocity
  rot_stopped_vel: 0.314

  acc_lim_x: 5 # maximum is theoretically 2.0, but we
  acc_lim_theta: 5
  acc_lim_y: 0.0      # diff drive robot

# Goal Tolerance Parameters
  yaw_goal_tolerance: 0.1  # 0.05
  xy_goal_tolerance: 0.1  # 0.10
  latch_xy_goal_tolerance: false

# Forward Simulation Parameters
  sim_time: 2.0       # 1.7
  vx_samples: 15       # 3
  vy_samples: 1       # diff drive robot, there is only one sample
  vtheta_samples: 20  # 20

# Trajectory Scoring Parameters
  path_distance_bias: 32      # 32.0   - weighting for how much it should stick to the global path plan
  goal_distance_bias: 12      # 24.0   - wighting for how much it should attempt to reach its goal
```



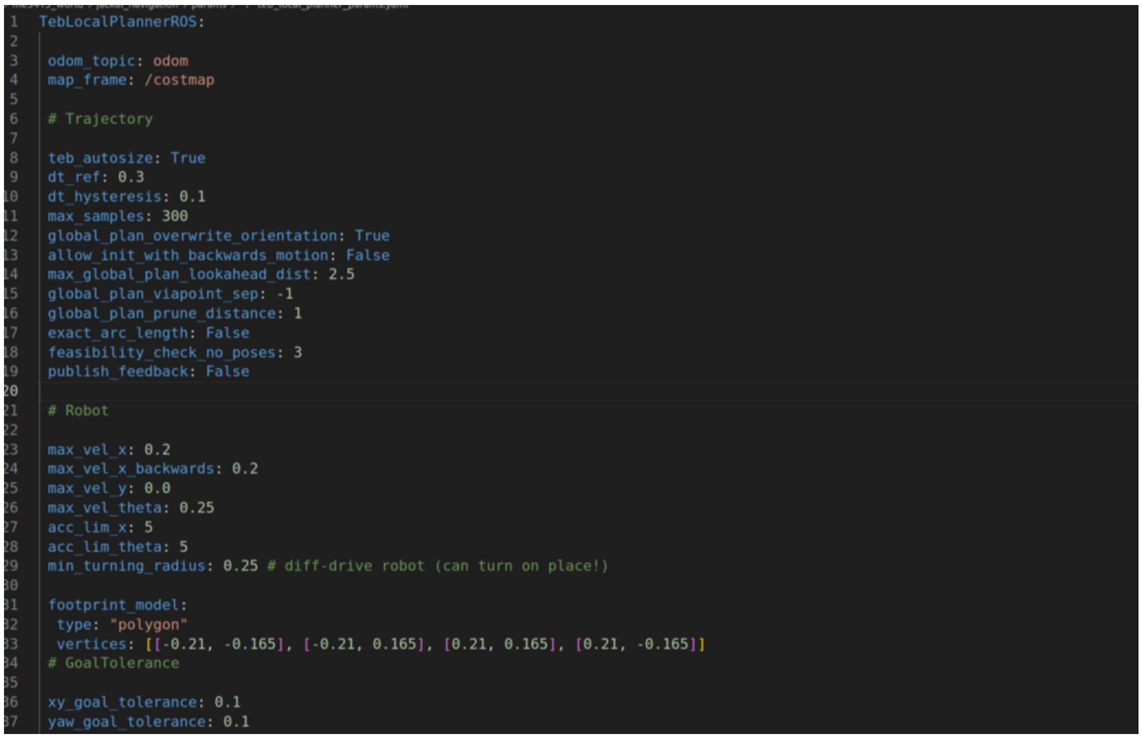
```yaml
# Trajectory Scoring Parameters
  path_distance_bias: 32      # 32.0   - weighting for how much it should stick to the global path plan
  goal_distance_bias: 12      # 24.0   - wighting for how much it should attempt to reach its goal
  occdist_scale: 0.1            # 0.01   - weighting for how much the controller should avoid obstacles
  forward_point_distance: 0.325 # 0.325  - how far along to place an additional scoring point
  stop_time_buffer: 0.2         # 0.2    - amount of time a robot must stop in before colliding for a valid traj.
  scaling_speed: 0.25           # 0.25   - absolute velocity at which to start scaling the robot's footprint
  max_scaling_factor: 0.2       # 0.2    - how much to scale the robot's footprint when at speed.

# Oscillation Prevention Parameters
  oscillation_reset_dist: 0.05  # 0.05   - how far to travel before resetting oscillation flags

# Debugging
  publish_traj_pc : true
  publish_cost_grid_pc: true
  global_frame_id: map


# Differential-drive robot configuration - necessary?
#  holonomic_robot: false
```

Figure 39. dwa_local_planner_params.yaml

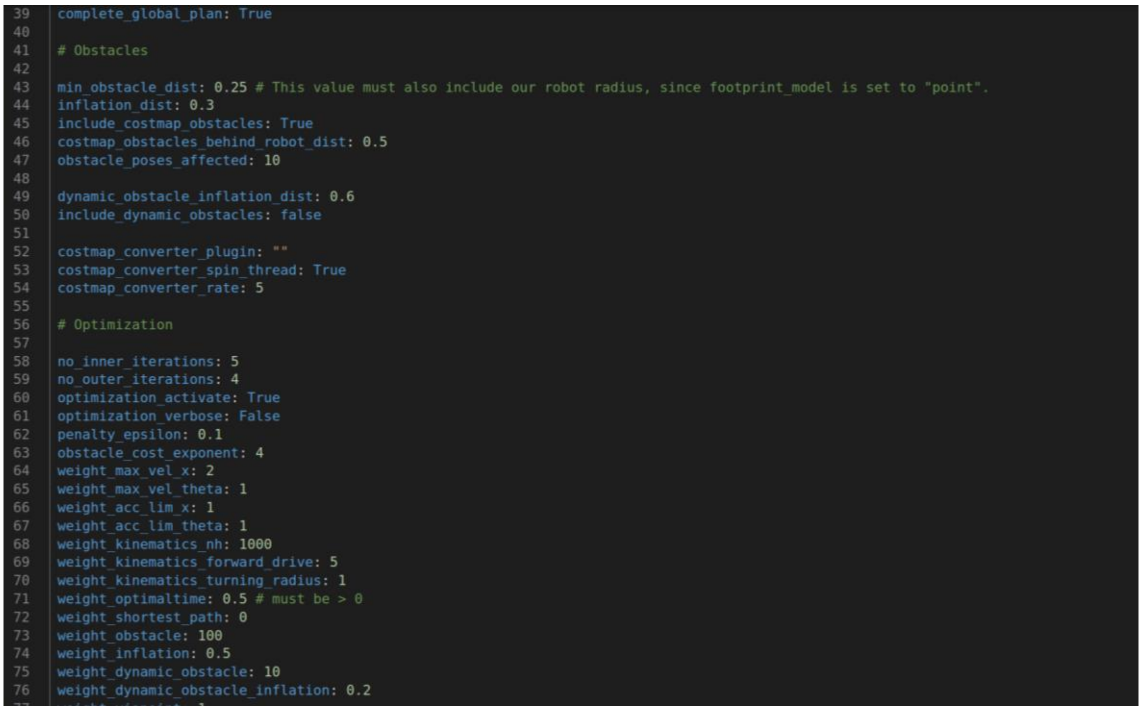


```xml
<launch>

  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen" >

    <rosparam file="$(find me5413_world)/jackal_navigation/params/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find me5413_world)/jackal_navigation/params/costmap_common_params.yaml" command="load" ns="local_costmap" />

    <rosparam file="$(find me5413_world)/jackal_navigation/params/map_nav_params/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find me5413_world)/jackal_navigation/params/map_nav_params/global_costmap_params.yaml" command="load" />

    <rosparam file="$(find me5413_world)/jackal_navigation/params/prohibition_areas.yaml" command="load" ns="global_costmap/costmap_prohibition_lay
    <rosparam file="$(find me5413_world)/jackal_navigation/params/prohibition_areas.yaml" command="load" ns="local_costmap/costmap_prohibition_laye

    <rosparam file="$(find me5413_world)/jackal_navigation/params/dwa_local_planner_params.yaml" command="load" />
    <!-- <rosparam file="$(find jackal_navigation)/params/base_local_planner_params.yaml" command="load" /> -->
    <!-- <rosparam file="$(find me5413_world)/jackal_navigation/params/teb_local_planner_params.yaml" command="load" /> -->


    <!-- <rosparam file="$(find rrt-global-planner)/params/rrt_global_planner.yaml" command="load" /> -->
    <!-- <rosparam file="$(find a_better_star)/param/global_planner_params.yaml" command="load" /> -->
    <!-- <rosparam file="$(find segment_global_planner)/param/segment_global_planner_params.yaml" command="load" /> -->
    <!-- <rosparam file="$(find me5413_world)/jackal_navigation/params/global_planner_params.yaml" command="load" /> -->
    <rosparam file="$(find me5413_world)/jackal_navigation/params/move_base_params.yaml" command="load" />



    <!-- <param name="base_global_planner" value="global_planner/RRTGlobalPlanner"/> -->
    <!-- <param name="base_global_planner" value="a_better_star/GlobalPlanner"/> -->
    <!-- <param name="base_global_planner" value="segment_global_planner/SegmentGlobalPlanner"/> -->
    <param name="base_global_planner" value="Astar_planner/AstarPlannerROS"/>
    <!-- <param name="base_global_planner" value="global_planner/GlobalPlanner"/> -->
    <!-- <param name="base_global_planner" type="string" value="navfn/NavfnROS" /> -->

    <!-- <param name="base_local_planner" value="teb_local_planner/TebLocalPlannerROS"/> -->
    <!-- <param name="base_local_planner" value="base_local_planner/TrajectoryPlannerROS"/> -->
    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />


    <remap from="odom" to="odometry/filtered" />
  </node>

</launch>
```

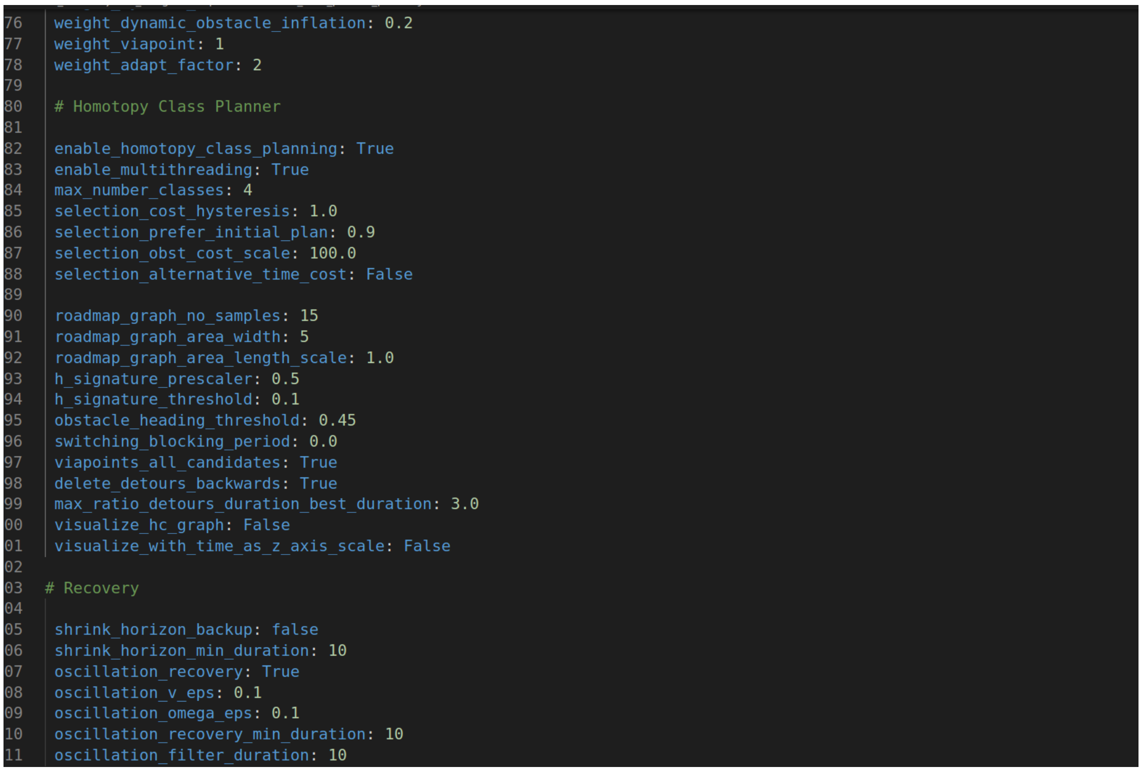
Figure 40. *move_base.launch*



```yaml
TebLocalPlannerROS:

 odom_topic: odom
 map_frame: /costmap

 # Trajectory

 teb_autosize: True
 dt_ref: 0.3
 dt_hysteresis: 0.1
 max_samples: 300
 global_plan_overwrite_orientation: True
 allow_init_with_backwards_motion: False
 max_global_plan_lookahead_dist: 2.5
 global_plan_viapoint_sep: -1
 global_plan_prune_distance: 1
 exact_arc_length: False
 feasibility_check_no_poses: 3
 publish_feedback: False

 # Robot

 max_vel_x: 0.2
 max_vel_x_backwards: 0.2
 max_vel_y: 0.0
 max_vel_theta: 0.25
 acc_lim_x: 5
 acc_lim_theta: 5
 min_turning_radius: 0.25 # diff-drive robot (can turn on place!)

 footprint_model:
  type: "polygon"
  vertices: [[-0.21, -0.165], [-0.21, 0.165], [0.21, 0.165], [0.21, -0.165]]
 # GoalTolerance

 xy_goal_tolerance: 0.1
 yaw_goal_tolerance: 0.1
```



```yaml
  complete_global_plan: True

  # Obstacles

  min_obstacle_dist: 0.25 # This value must also include our robot radius, since footprint_model is set to "point".
  inflation_dist: 0.3
  include_costmap_obstacles: True
  costmap_obstacles_behind_robot_dist: 0.5
  obstacle_poses_affected: 10

  dynamic_obstacle_inflation_dist: 0.6
  include_dynamic_obstacles: false

  costmap_converter_plugin: ""
  costmap_converter_spin_thread: True
  costmap_converter_rate: 5

  # Optimization

  no_inner_iterations: 5
  no_outer_iterations: 4
  optimization_activate: True
  optimization_verbose: False
  penalty_epsilon: 0.1
  obstacle_cost_exponent: 4
  weight_max_vel_x: 2
  weight_max_vel_theta: 1
  weight_acc_lim_x: 1
  weight_acc_lim_theta: 1
  weight_kinematics_nh: 1000
  weight_kinematics_forward_drive: 5
  weight_kinematics_turning_radius: 1
  weight_optimaltime: 0.5 # must be > 0
  weight_shortest_path: 0
  weight_obstacle: 100
  weight_inflation: 0.5
  weight_dynamic_obstacle: 10
  weight_dynamic_obstacle_inflation: 0.2
```



```yaml
  weight_dynamic_obstacle_inflation: 0.2
  weight_viapoint: 1
  weight_adapt_factor: 2

  # Homotopy Class Planner

  enable_homotopy_class_planning: True
  enable_multithreading: True
  max_number_classes: 4
  selection_cost_hysteresis: 1.0
  selection_prefer_initial_plan: 0.9
  selection_obst_cost_scale: 100.0
  selection_alternative_time_cost: False

  roadmap_graph_no_samples: 15
  roadmap_graph_area_width: 5
  roadmap_graph_area_length_scale: 1.0
  h_signature_prescaler: 0.5
  h_signature_threshold: 0.1
  obstacle_heading_threshold: 0.45
  switching_blocking_period: 0.0
  viapoints_all_candidates: True
  delete_detours_backwards: True
  max_ratio_detours_duration_best_duration: 3.0
  visualize_hc_graph: False
  visualize_with_time_as_z_axis_scale: False

# Recovery

  shrink_horizon_backup: false
  shrink_horizon_min_duration: 10
  oscillation_recovery: True
  oscillation_v_eps: 0.1
  oscillation_omega_eps: 0.1
  oscillation_recovery_min_duration: 10
  oscillation_filter_duration: 10
```

Figure 41. *teb_local_planner_params.yaml*

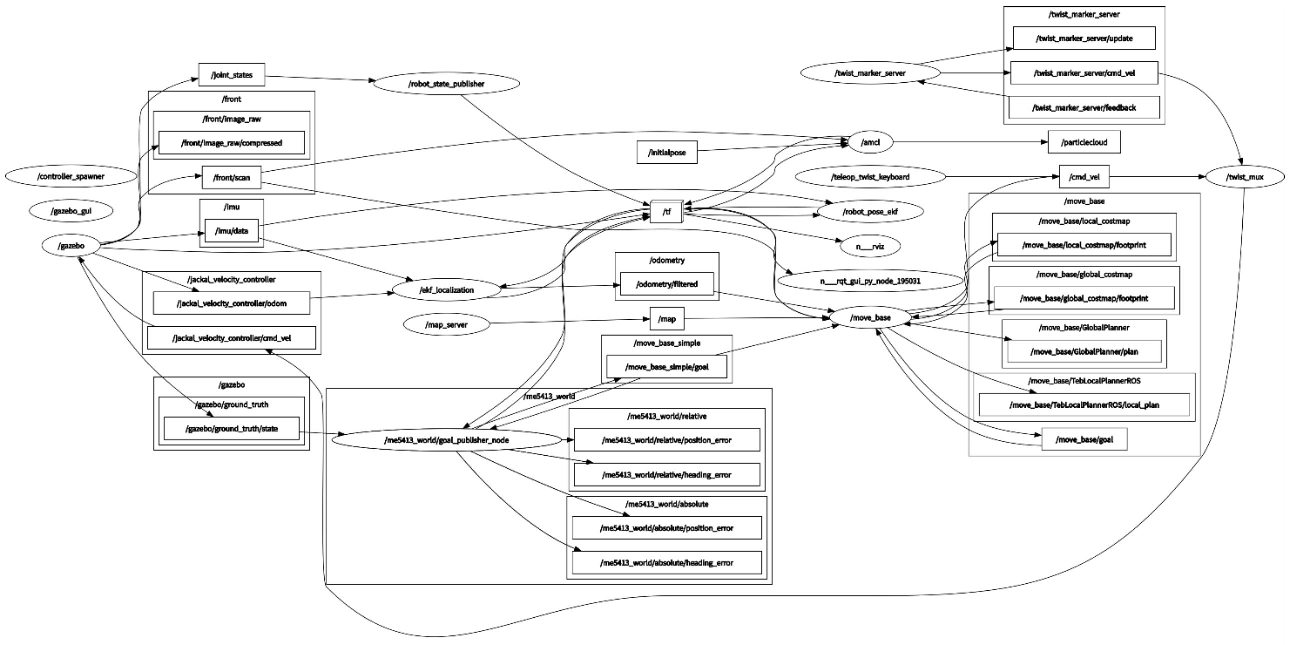

/twist_marker_server
/twist_marker_server/update
/twist_marker_server/cmd_vel
/twist_marker_server/feedback
/joint_states
/robot_state_publisher
/front
/front/image_raw
/front/image_raw/compressed
/front/scan
/amcl
/particlecloud
/initialpose
/controller_spawner
/gazebo_gui
/gazebo
/imu
/imu/data
/tf
/robot_pose_ekf
/teleop_twist_keyboard
/cmd_vel
/twist_mux
/move_base
/move_base/local_costmap
/move_base/local_costmap/footprint
n___rviz
n___rqt_gui_py_node_195031
/move_base/global_costmap
/move_base/global_costmap/footprint
/jackal_velocity_controller
/jackal_velocity_controller/odom
/jackal_velocity_controller/cmd_vel
/ekf_localization
/odometry
/odometry/filtered
/map_server
/map
/move_base/GlobalPlanner
/move_base/GlobalPlanner/plan
/move_base_simple
/move_base_simple/goal
/move_base/TebLocalPlannerROS
/move_base/TebLocalPlannerROS/local_plan
/gazebo/ground_truth
/gazebo/ground_truth/state
/me5413_world
/me5413_world/goal_publisher_node
/me5413_world/relative
/me5413_world/relative/position_error
/me5413_world/relative/heading_error
/move_base/goal
/me5413_world/absolute
/me5413_world/absolute/position_error
/me5413_world/absolute/heading_error

Figure 42. Schematic of robot_pose_ekf node graph

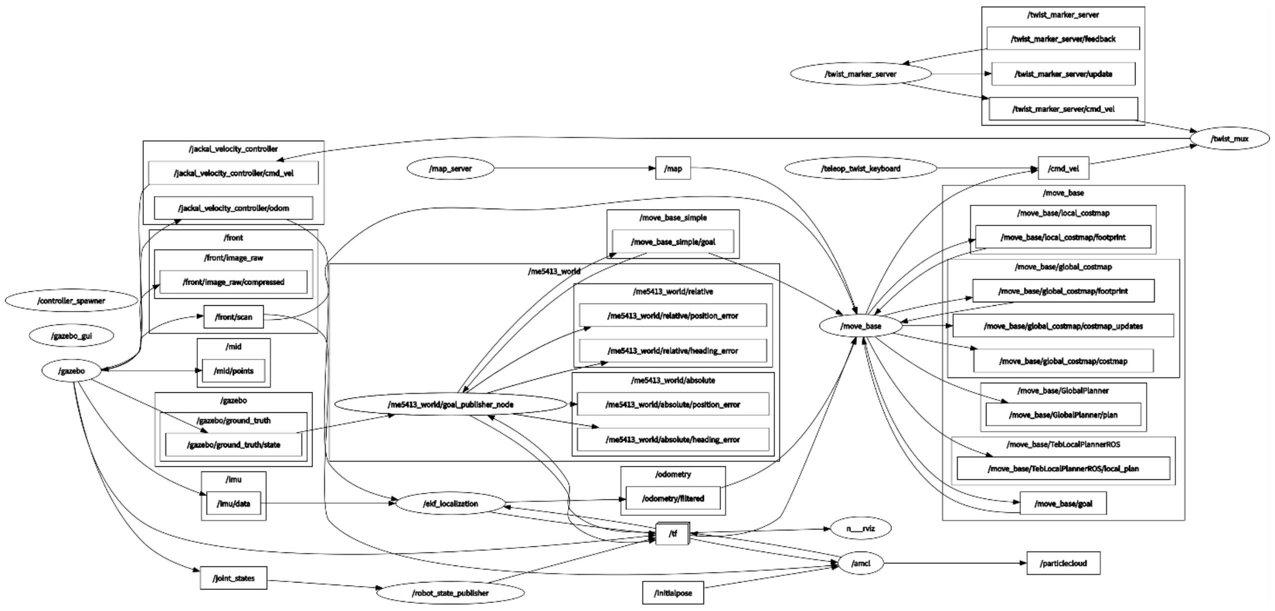

/twist_marker_server
/twist_marker_server/feedback
/twist_marker_server/update
/twist_marker_server/cmd_vel
/twist_mux
/jackal_velocity_controller
/jackal_velocity_controller/cmd_vel
/jackal_velocity_controller/odom
/map_server
/map
/teleop_twist_keyboard
/cmd_vel
/move_base
/move_base/local_costmap
/move_base/local_costmap/footprint
/front
/front/image_raw
/front/image_raw/compressed
/front/scan
/move_base_simple
/move_base_simple/goal
/me5413_world
/me5413_world/relative
/me5413_world/relative/position_error
/me5413_world/relative/heading_error
/move_base/global_costmap
/move_base/global_costmap/footprint
/move_base/global_costmap/costmap_updates
/move_base/global_costmap/costmap
/controller_spawner
/gazebo_gui
/gazebo
/mid
/mid/points
/me5413_world/absolute
/me5413_world/absolute/position_error
/me5413_world/absolute/heading_error
/me5413_world/goal_publisher_node
/move_base/GlobalPlanner
/move_base/GlobalPlanner/plan
/gazebo/ground_truth
/gazebo/ground_truth/state
/move_base/TebLocalPlannerROS
/move_base/TebLocalPlannerROS/local_plan
/imu
/imu/data
/ekf_localization
/odometry
/odometry/filtered
/move_base/goal
/tf
n___rviz
/amcl
/particlecloud
/joint_states
/robot_state_publisher
/initialpose

Figure 43. Schematic of node graph of GlobalPlanner

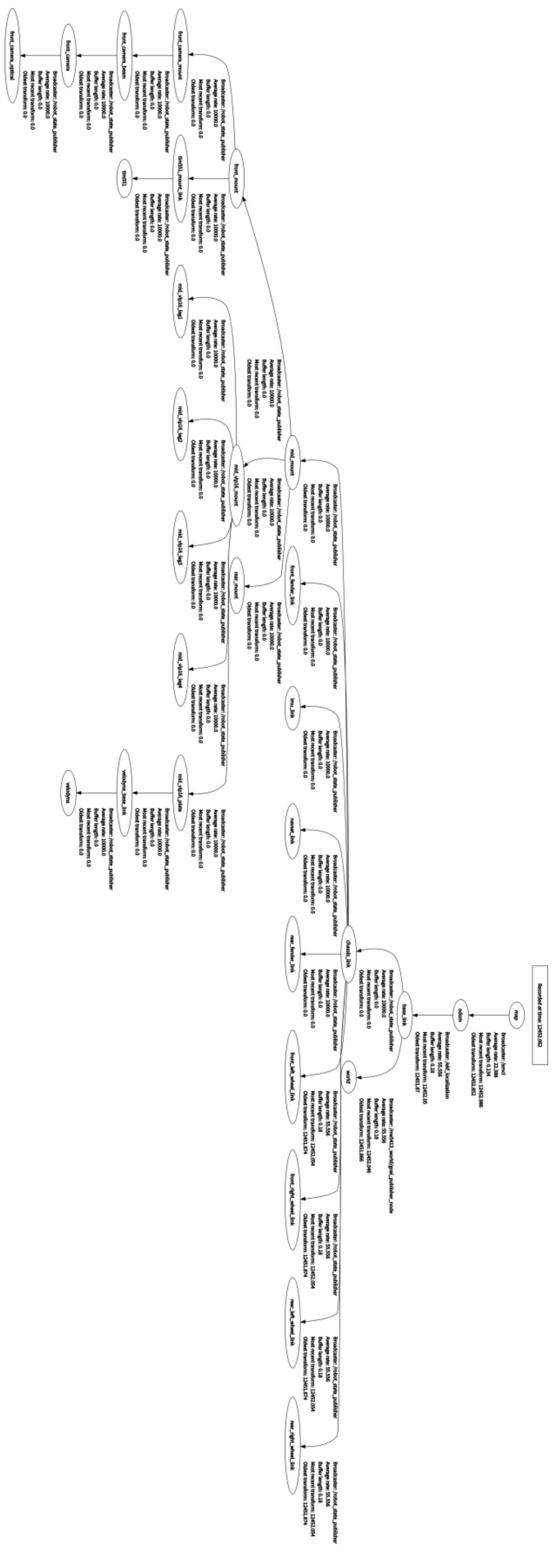


Figure 44. TF tree of robot

Reference

1. Bailey, T. and H. Durrant-Whyte, *Simultaneous localization and mapping (SLAM): Part II.* IEEE robotics & automation magazine, 2006. **13**(3): p. 108-117.
2. ziv-lin, *FAST_LIO*. 2022.
3. Xu, W. and F. Zhang, *Fast-lio: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter.* IEEE Robotics and Automation Letters, 2021. **6**(2): p. 3317-3324.
4. Derpanis, K.G., *Overview of the RANSAC Algorithm.* Image Rochester NY, 2010. **4**(1): p. 2-3.
5. Stachniss, C., *Robotic mapping and exploration*. Vol. 55. 2009: Springer.
6. ROSwiki. *AMCL*. 2021; Available from: http://wiki.ros.org/amcl.
7. Thakur, M.B., M. Schmid, and V.N. Krovi, *Benchmarking the Localization Accuracy of 2D SLAM Algorithms on Mobile Robotic Platforms*. 2020, SAE Technical Paper.
8. ROSWiki. *robot_pose_ekf* 2022; Available from: http://wiki.ros.org/robot_pose_ekf.
9. Zheng, K., *Ros navigation tuning guide.* Robot Operating System (ROS) The Complete Reference (Volume 6), 2021: p. 197-226.
10. ROSWiki. *global_planner*. 2021; Available from: http://wiki.ros.org/global_planner.
11. CSDN, *astar.cpp cpp file, application with ROS' A* algorithm global road strength planner*. 2022.
12. WLwind. *segment_global_planner*. 2021; Available from: https://github.com/WLwind/segment_global_planner.
13. WLwind, *a_better_star*. 2019: Github.
14. mech0ctopu, *rrt-global-planner*. 2022: Github.
15. Tian, Y., et al. *An Autonomous Behavior Switching Method for Indoor Mobile Service Robots*. in *2021 IEEE International Conference on Mechatronics and Automation (ICMA)*. 2021. IEEE.
16. Fox, D., W. Burgard, and S. Thrun, *The dynamic window approach to collision avoidance.* IEEE Robotics & Automation Magazine, 1997. **4**(1): p. 23-33.
17. ROSwiki. *teb_local_planner*. 2021; Available from: http://wiki.ros.org/teb_local_planner.
18. Fankhauser, P., et al. *Kinect v2 for mobile robot navigation: Evaluation and modeling*. in *2015 international conference on advanced robotics (ICAR)*. 2015. IEEE.
19. Muñoz, N., J. Valencia, and N. Londono. *Evaluation of navigation of an autonomous mobile robot*. in *Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems*. 2007.