Final Project Report

Group 5

Wang Yiwen A0284967E
Tang Haodi A0284966H
Zhang Xinming A0284869A
Min Jixin A0285324A
Ma Ziheng A0284884J
Tian Tan A0285069R

ME5413 Autonomous Mobile Robotics

Semester 2 | AY 2023/2024
08 Apr 2024

Introduction

In this project, given a mini-factory environment including 3 target areas and 1 restricted area in Gazebo (Fig 1), mapping, object recognition and path planning are required to complete the entire navigation process.

With the self-designed robot navigation software stack, the "Jackal" robot is able to move to assembly line 1 & 2 and destinate itself to the random box area to search for box 2 and then navigate itself to the delivery vehicle 1, 2 & 3.

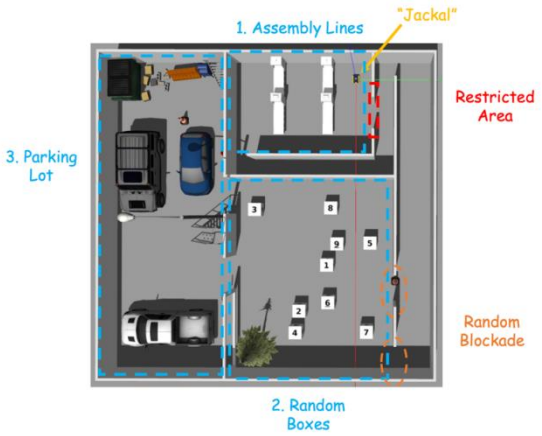


Fig 1. Mini-factory environment

Task 1: Mapping

A high-quality occupancy grid map is essential for route planning and robot navigation. In this project, typical 2D and 3D Simultaneous Localization and Mapping (SLAM) algorithms are implemented to generate a suitable 2D grid map of the mini-factory environment.

I) Mapping with *Gmapping*

Gmapping is the default algorithm for mapping in the provided 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 in reflecting features that are higher or lower than the installed LIDAR height or the glass wall which is transparent and shows a bit drifting in closing the loop. It can be seen from the resulted map (Fig 2) that bricks and the ladder in the parking lot area are also missing, which might potentially cause collision. Therefore, 3D SLAM algorithms will be tested and compared using A-LOAM and FAST-LIO to obtain sufficient 3D information.

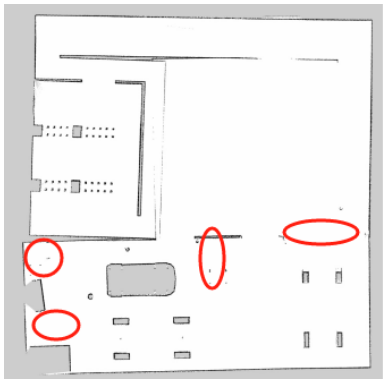


Fig 2. 2D mapping result using gmapping

II) Mapping with ALOAM

A-LOAM is an advanced implementation of LOAM (LOAM: Lidar Odometry and Mapping in Real-time), which uses Eigen and Ceres Solver to simplify code structure. It is clean and simple without complicated mathematical derivation and redundant operations. To build this SLAM and test on the environment:

(1) Clone the *ALOAM* repo to *ME5413_Final_Project/src* as a package.

(2) Edit *"scanRegistration.cpp"* to correctly subscribe pointcloud message *"/mid/points"*.

(3) Build the package and launch world and mapping nodes using command *"roslaunch me5413_world world.launch"* and *"roslaunch me5413_world mapping_aloam.launch"* and then use keyboard to control Jackal to move around the environment to build a map (Fig 3).

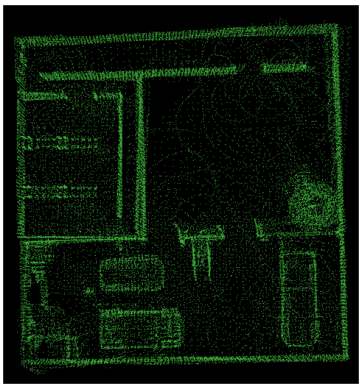


Fig 3. Point cloud map by A-LOAM

The 3D point cloud map generated by A-LOAM lacks intense points to reflect the details of the environment. This problem might be solved by increasing the LIDAR lines to 32 or more, increasing angular resolution (samples). Though this problem might not affect later performances, A-LOAM as a pure LIDAR SLAM, is considered not better as multi-sensor fusion SLAM algorithms to reflect the accurate mapping.

III) Mapping with FAST-LIO

FAST-LIO (Fast LiDAR-Inertial Odometry), combines lidar landmarks with IMU data using tightly linked, repeatedly extended Kalman filters for reliable navigation in chaotic, noisy, or fast-moving settings. In this project, building FAST-LIO in the workspace follows:

(1) Clone the FAST-LIO repo to *ME5413_Final_Project/src* as a package.

(2) Modify *"velodyne.yaml"* according to published imu and point cloud topics:

lid_topic: "/mid/points"
imu_topic: "/imu/data"
scan_line: 16

(3) Build the package and launch world and mapping nodes using command *"roslaunch me5413_world world.launch"* and *"roslaunch me5413_world mapping_fastlio.launch"* and then use keyboard to control Jackal to move around the environment to build a map.

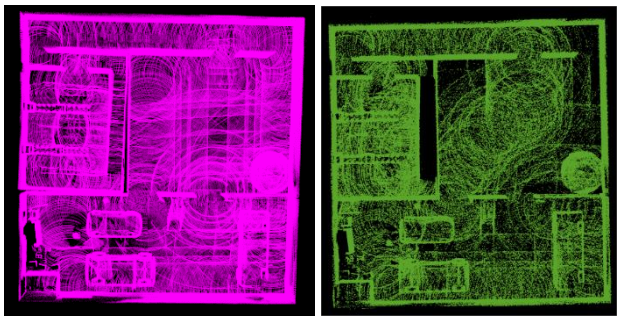


Fig 4. Point cloud map by FAST-LIO

During the process of mapping, the almost closed-off assembly line area shows duplicate walls in the generated map (Fig 4 Left), particularly noticeable in the very narrow corridor and the wall connecting the assembly line area with the parking lot. To address this problem, the scanning frequency of the LIDAR is increased to 50Hz and selecting an optimal mapping path and use IMU filters significantly mitigates the issue (Fig 4 right). It was determined that traversing only through the assembly line area, while avoiding the interior of the narrow corridor and instead passing by its entrances and exits, effectively reduced ground noise in the point cloud data of this region and minimized the occurrence of duplicate walls, and at the same time keeps the environment information intact.

These adjustments have proven to be crucial in enhancing the quality of the generated map in challenging environments, ensuring a more accurate representation of the physical space.

IV) Convert from PCD to Grid Map

Point cloud data (PCD) maps are obtained from two 3D SLAM algorithms which contains much more information of the environment. However, the navigation stack provided by ROS is based on 2D grid map. Thus, it is necessary to convert the 3D point cloud format map to a 2D grid map. From the FAST-LIO PCD map above, it is obvious that many circular noise points on the ground plane. A package named *"pcd2pgm"* provides the ability to project the 3D points onto a 2D plane which is based on a pass-through filter and a radius filter. The pass-through filter can extract desired point cloud within or out of

a specific range which is able to remove most of the ground noise and the rest will be eliminated by the radius filter. After directly convert the PCD map into grid map using command *"roslaunch pcd2pgm run.launch"* with parameters:

<param name="thre_z_min" value= "0" />

<param name="thre_z_max" value= "3" />

<param name="flag_pass_through" value= "0" />

The result shows that the ground noise will significantly affect the quality of the resulted grid map (Fig 5).

Fig 5. Directly converted gird map

To minimize the effect the noise and keep the lower frame of the glass wall remain, better parameters for the lowest boundary of the pass-through filter are set to be:

<param name="thre_z_min" value= "0.55" />

The resulted grid map is suitable for route planning and navigation. However, the entire tree is projected onto the 2D plane as obstacle which will decrease the free space for box recognition in the random box area (Fig 6).

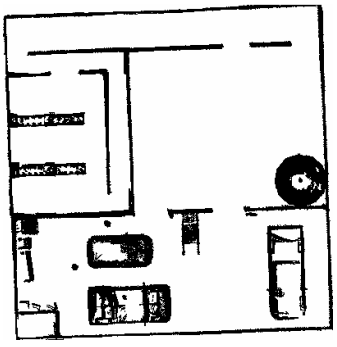


Fig 6. Grid map without ground noise

To solve this issue, the highest boundary for the pass-through filter is set to be:

<param name="thre_z_max" value= "0.9" />

Fig 7 shows the result of the PCD map after the pass-through filter and there remains some scattered noise points in the map.

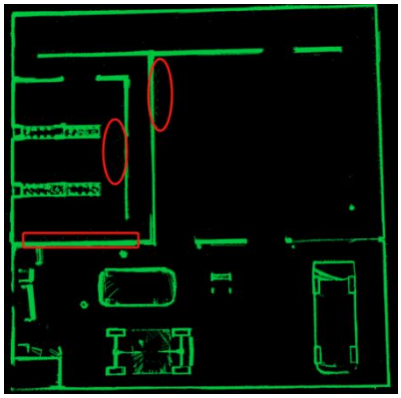


Fig 7. PCD map with scattered noise points

The radius filter is then configured with parameters:

<param name="thre_radius" value= "0.5" />

<param name="thres_point_count" value= "10" />

After going through this radius filter, the PCD map is clear enough to convert into 2D grid map. The final PCD map and 2D grid map are shown in Fig. 8, and the entire launch file of *"pcd2pgm"* is in the appendix. However, some environment information will be lost under the selective parameters mentioned above such as the board under the ladder and some part of the vehicle 3 although the tree is mapped perfectly. This is a trade-off we made to maximize the free space of the box area and retain as much other environmental information as possible. Method to solve this issue will be discussed in the navigation section.

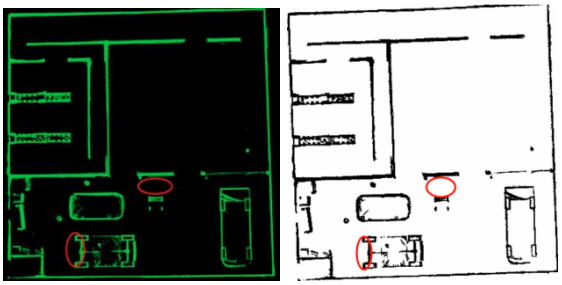


Fig 8. Final PCD map and grid map

V) Evaluate with EVO

EVO tool is used to evaluate the performance of FAST-LIO in this project. The ground truth of the trajectory is stored in the topic *"/gazebo/ground_truth/state"* and the SLAM odometry data is in the topic *"/Odometry"*. By recording these two topics in a rosbag while mapping using command *"rosbag record /gazebo/ground_truth/state /Odometry"* the EVO can be implemented with the command "*evo_ape bag fastlio.bag /gazebo/ground_truth/state /Odometry -r full -va --plot --plot_mode xy"*. From the result (Fig 9), the RMSE is 0.91 which indicates a good performance of FAST-LIO.

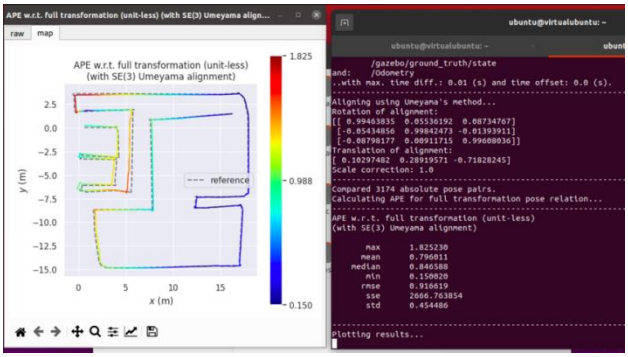


Fig 9. EVO result of FAST-LIO

Task 2: Navigation

The essence of navigation lies in the robot's ability to understand its surroundings, determine its position within it, and make intelligent decisions to move from one point to another while avoiding obstacles. In this project, the task is subdivided into three parts, localization, path planning and object recognition. Localization and path planning were implemented and improved upon ROS navigation stack. The underline logic of our navigation functional stack can be described using a schematic diagram (Fig 10). For the target recognition part, we developed our own package that enables autonomous recognition of "Box 2" in the random-box area.

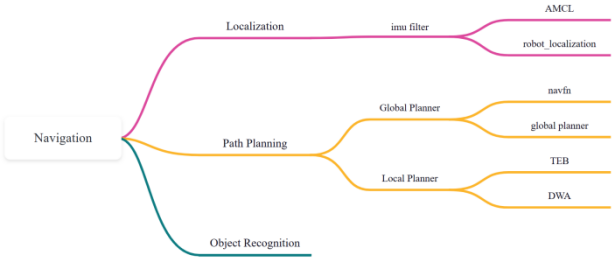


Fig 10. Navigation package structure

I) Localization

a. AMCL

AMCL is a probabilistic localization system used for a robot moving in 2D, allowing the robot to determine its position on a map and is particularly suited for environments where the robot's initial position is unknown, or the robot's sensors are subject to noise. AMCL is the official localization module in the ROS/ROS2 system and is the only specified localization algorithm in the navigation module. With respect to the preset environment, the detailed setting are in *"amcl.launch"*, maximum laser beams, minimum and maximum particles are all increased.

b. Robot localization

Robot_localization is a mature and widely used robot dynamic localization package based on Kalman filtering in ROS systems. The typical usage of robot_localization is to coordinate with the navigation module to realize the integration of various sensors and accurate route navigation. This package is integrated in the *"jackal_control"* package and is directly applied to the task setting. The schematic of the function of this package is shown in Fig 11.

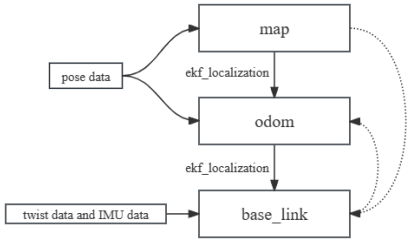


Fig 11. Functional description of robot_localization

The solid arrows between the three frames indicate

that their spatial relations can be directly solved by robot_localization while the dashed arrows indicate solutions indirectly through TF transformations. Pose data (including odom data) will be used for the alignment of the map coordinate system and the odom coordinate system, while Twist data and IMU data will be used for the alignment of the base_link, that is, the orientation of the robot body, which is the basis for converting the positioning accuracy.

c. Imu filter

After initiate Rviz, a slowly rotating LIDAR scan frame and loss of localization are observed (Fig 12)

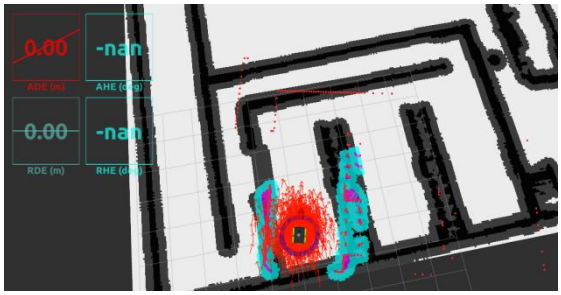


Fig 12. Drift of frame

This problem might be due to IMU noises and can be potentially improve by IMU filters. Imu_tools is a ROS provided stack for IMU filtering, in which a imu_complementary_filter which fuses angular velocities, accelerations, and (optionally) magnetic readings from a generic IMU device into a quaternion to represent the orientation of the device with respect to the global frame. Detailed implementation of this filter follows:

(1) Install imu_tools

(2) To avoid naming confusion and collision, edit the published imu topic name in *"jackal.gazebo"* to *"imu/data_raw"* .

(3) Activate complementary filter node in the *"navigation.launch"*.

The performance of this filter (Fig 13) indicates barely seen drifts. Node relations related to localization is partly displayed in Fig 14. Filtered IMU data *"/imu/data"* would be used for ekf localization.

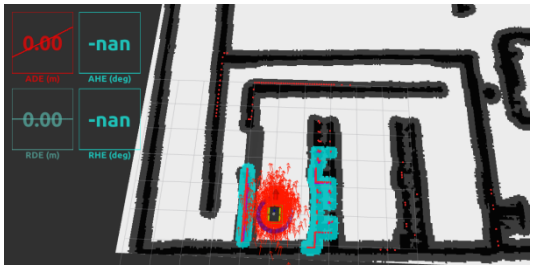


Fig 13. Initial pose after applying IMU filter

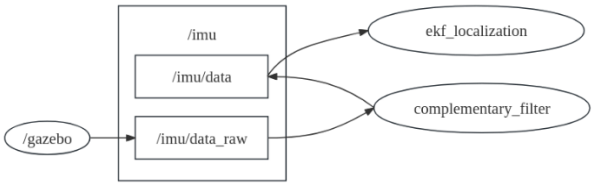


Fig 14. Node relations in localization (partial)

II) Path Planning

a. Costmap

Costmap is a key component used to represent the environment around a robot in terms of costs associated with traversability and obstacles. In ROS navigation stack, the costmap usually contains two parts, the global costmap and the local costmap to provide the ability of obstacle avoidance for the robot. The costmap usually contains three layers:

(1) Static layer: static layer represents a largely unchanging portion of the costmap which is the map provided for the navigation.

(2) Obstacle layer: obstacle layer tracks the obstacles as read by the sensor data.

(3) Inflation layer: inflation layer adds new values around lethal obstacles in order to make the costmap represent the configuration space of the robot.

1. Global Costmap

The global costmap is usually a static map contains all known obstacles and inflates those obstacles with a desired inflated radius to guarantee the route generated by global path planner will avoid the obstacles. In this project, all the objects in the mini factory except the random boxes and cone can be treated as obstacles that already exist, and the global costmap is established based on these obstacles.

Noting that there is a gate near the start position

which is restricted to pass. This restricted area should be included in the global costmap so that the route generated by the global path planner will avoid this area. *"costmap_prohibition_layer"* is the ROS package that accounts for setting an additional customized prohibited area layer in the global costmap. By setting the coordination of desired area, global cost map will treat the area as obstacles. In addition, the missing information of the board under the ladder will cause path planning error to allow the Jackal robot to go through the bottom of the ladder. By adding another small prohibition area, this issue can be perfectly solved. The coordinates of the prohibition areas are stored in the *"prohibition_layer.yaml"*. The result global costmap is shown in Fig 15 with inflation radius is 0.21m.

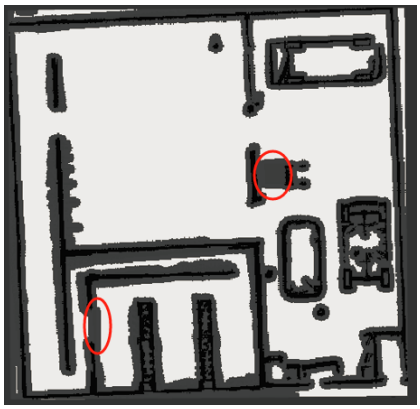


Fig 15. Final global costmap

For all parameters about global costmap, please refer to the file *"global_costmap_params.yaml"* and *"costmap_common_params.yaml"* in the appendix.

2. Local Costmap

Local costmap is another type of costmap that focuses on the unknown and dynamic objects that are not in the global costmap. In this project, the size of local costmap is set to be 8m×8m and the inflated radius of the local costmap is set to be 0.19m to avoid the random boxes and cone. All other parameters are provided in the *"local_costmap_params.yaml"* and *"costmap_common_params.yaml"*.

b. Global Planning

In ROS navigation stack, *"navfn/NavfnROS"* and *"global_planner/GlobalPlanner"* are two provided global path planner plugins which implement both Dijkstra and A* algorithms. *global_planner* is used in this project for global path planning since it is an improved version of *navfn*. Set the parameter: *"use_dijkstra: false"* to implement A* algorithm, otherwise, the planner will be based on Dijkstra algorithm.The planned route from start position to vehicle 3 by Dijkstra and A* algorithm are shown in Fig 16 and Fig 17 Compare with Dijkstra algorithm, A* performs a smoother path which can reduce the rotation of Jackal for less IMU and odometry drift error.

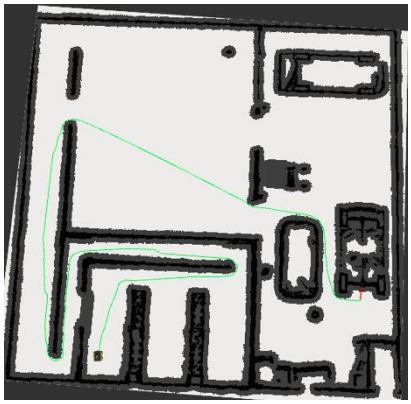


Fig 16. Planning route by Dijkstra

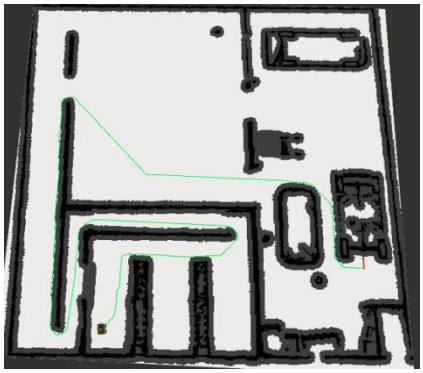


Fig 17. Planning route by A*

c. Local Planning

1. DWA Local Planner

Dynamic windows approach (DWA) is a typical local path planning method which controls the robot with the best velocity (v, w) after going through lots of sampling process. In order to have a more comprehensive output velocity, the number of samples for different directions is set to be:

"vx_samples: 10"

"vy_samples: 1"

"vtheta_samples: 20"

Other parameters of DWA planner is provided in *"dwa_local_planner_params.yaml"*. However, DWA local planner only focus on the planning for next step, the shortcoming of DWA is particularly pronounced when the Jackal is facing a "V" form obstacles formed by two boxes (Fig 18). It will be blocked due to the inflated area of the box and cannot find a path out.

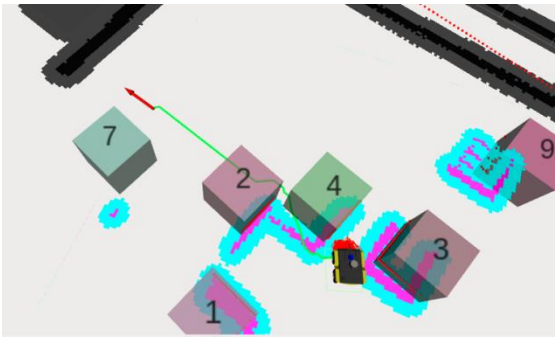


Fig 18. Issue with DWA local planner

2. TEB Local Planner

Timed elastic band is another local planner provided by navigation stack which is more forward-looking than DWA. It is possible for TEB local planner to optimize the local trajectory to move from the left side of box 2 or the right side of box 4 to get to the destination. The footprint parameter for TEB planner is set to be: ([-0.21, -0.165], [-0.21, 0.165], [0.21, 0.165], [0.21, -0.165]) to achieve a better performance of obstacle avoidance and turning.

III) Random box navigation

In the random box area, nine boxes are randomly generated, and in order to find specific blocks, the robot needs to be given the function of target recognition. After that, we need to design a particular path to navigate to the block.

a. Object Detection

1. Template Matching

Template Matching is a technique for finding parts of a larger image that match a template image and is widely used in image processing and computer vision. The core idea of this method is to slide the template image in the target image and at each position some measure of similarity between the template image and the target image is computed. By comparing these similarity metric values, the position of the template image in the target image can be determined. This project only needs to recognize fixed numbers, for example, the task of this group is to recognize the number "2". And the number templates are predefined and fixed, so there is no need for the algorithms to have generalization ability, only need to accurately recognize the determined templates. With such a task goal, template matching is a simple and effective method. In addition, since this method is sensitive to the features of the template image, such as resolution, contrast, and noise in detection, template images at different distances, angles, and lighting conditions are configured for better image recognition.

In the process of image processing, we first convert the image information to OpenCV format, then perform preprocessing (Gaussian blur) to reduce noise and performs template matching. If a high match score is achieved, a bounding box will be drawn around the number, which means the target is found.

2. Tesseract OCR

Tesseract OCR (Optical Character Recognition) is an open-source OCR engine that supports a wide range of image formats and can handle multi-column text, non-standard fonts and other complex text layouts.

In the process of image recognition using the Tesseract OCR, we found that the detection result is unstable. When the numbers on the block are very close, OCR will mistakenly recognize several numbers as a whole number. In addition, the recognition of numbers cannot be achieved using the automatically set threshold function. When I manually lower the threshold, then it can recognize numbers, but different thresholds need to be used in different scenarios. If the distance between the position of the robot and the square changes, the threshold value should also change accordingly. It is

obvious that manually adjusting the threshold during robot movement is not realistic. Therefore, we choose to use the template matching method for image recognition.

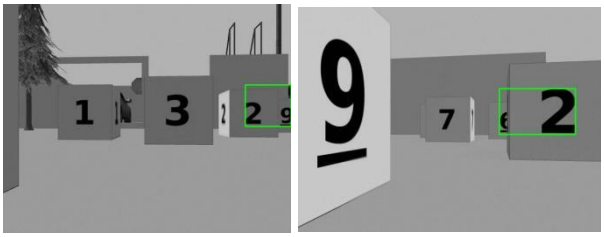


Fig 19. Detection failures of OCR

b. Target Navigation

When the robot has the ability of recognizing the target number, the next step is to navigate to the destination. A proper way of achieving this is to calculate the coordinates of the block and then use the previous navigation methods. In this part, we use camera ranging technique and design a special path to find the target.

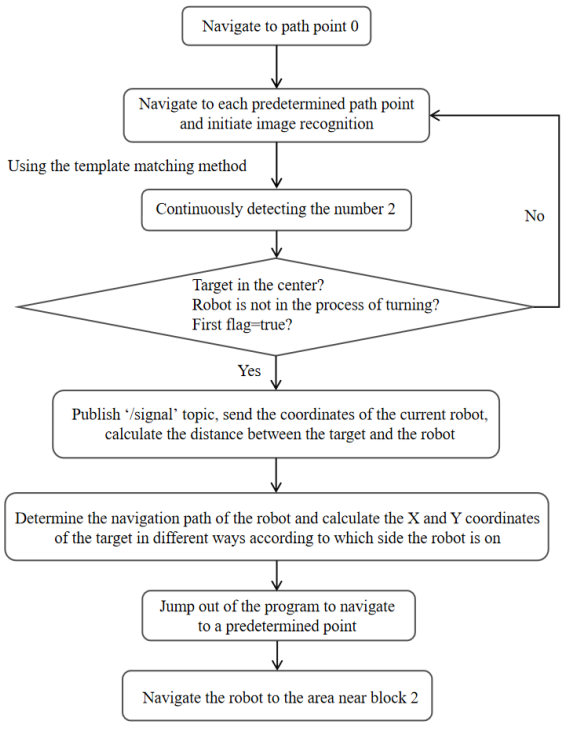


Fig 20. Flow chart of the random box navigation

1. Camera Ranging

The program adopts a distance measurement method based on camera imaging principle and similar triangle principle. The pixel width of the object on the imaging sensor can be obtained by analyzing the picture, we already know the focal length and the actual object size, and the distance from the camera to the object can be calculated by the formula below.

$$L = \frac{f \times D}{P}$$

('L' is the distance from the camera to the object, 'f' is the focal length, 'D' is the actual size of the object, and 'P' is the pixel width of the object on the imaging sensor.)

In the 'image_process' program, 'ratio_w', 'ratio_h': are the scale factors used to calculate the pixel width and height of the cube, 'FOCAL_LENGTH' is the focal length of the camera in pixels, 'BLOCK_WIDTH' is the actual width of the cube in meters. With this information, the distance between the camera and the robot can be measured.

2. Path Planning

We have adjusted the position of the camera to the left side of the robot, with the aim of enabling real-time observation of the position of the blocks while the robot is moving. We have set several path points at the edge of the random box area (Fig 21), allowing the robot to circle the area along a straight path to find square '2'. When we detect the number '2' and the bounding box is in the middle of the screen, the program retrieves the current position of the robot from the odometer and calculates the distance between the car and the block. Through this information, we can obtain the coordinates of the block and navigate to the vicinity of the specified block.

When the robot reaches the scheduled point 0, the image processing sub-process is started. If square 2 is detected during any of the path points, a stop signal is received and the script stops to continue navigating to the next point. After the system calculates the coordinates of the block, the robot receives instructions to navigate directly in front of the block and faces the camera directly towards the block.

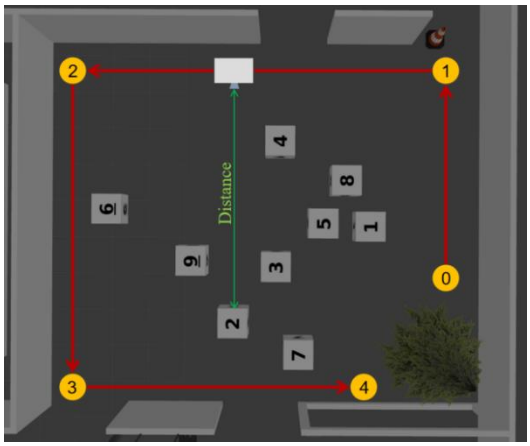


Fig 21. The detection routes

3. Performance Analysis

The random box task consists of two parts, target detection and navigation, starting with the target detection performance. Template matching is performed on the full image of the camera, and when the number "2" is in the middle of the camera image, the distance from the robot to the target is calculated, and the program runs with the following results.

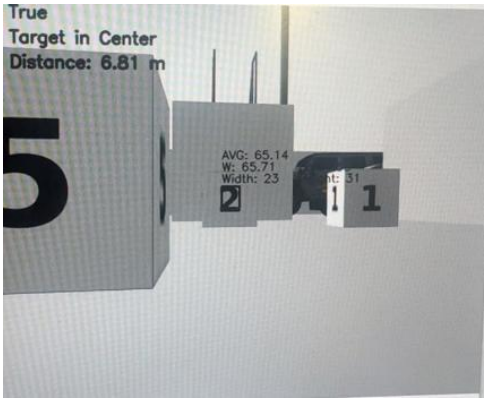


Fig 22. Target detection results

The robust and relatively stable performance of the target detection algorithm has been tested, and the position of the target is well transmitted to the navigation program through the '/signal' theme.

Fig 23. Signal topic

Subsequently, the navigation program calculates the target location based on the information in the '/signal' topic and navigates to the vicinity.

Fig 24. Target navigation

Besides, there are four indicators in the interface of rviz that mark the error of robot navigation in real-time, which are ADE, AHE, RDE, RHE. ADE and RDE represent average and relative displacement error, while AHE and RHE represent average and relative heading error. During the process of moving to the target location, these parameters are constantly changing, indicating that the robot is constantly adjusting its path according to the surrounding environment. When heading to a fixed location, ADE and RDE are generally not very large. When the robot travels through the random box area, due to the complex environment, the robot will continuously adjust the path, resulting in a different path from the predicted one, which will increase the displacement error and heading error.

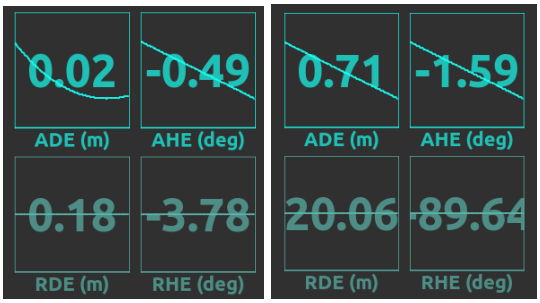


Fig 25. Evaluation metrics

4. Difficulties and Solutions

First, in the testing, there were problems with false detections when the robot turns (Fig 26). It is not convenient to calculate coordinates when turning, so we hope to only perform detection on straight roads. As the vehicle comes around a corner and turns to the next path, the camera rotates and the process may detect image '2', incorrectly outputting the computed location and navigating it. We tried several methods,

including periodic sleep of image processing during the turning and using angular acceleration judgment, but these methods did not yield good results. Finally, we chose to use the current position orientation quaternion to compute the vehicle's 'yaw' information. According to the value of 'yaw', we can judge whether the robot is travelling on a predefined straight-line path or making a turn. If it is in the process of making a turn, the robot will not execute the program of image detection, and it won't send the information to the planning function. It ensures that the robot will be in a predefined straight-line path when it detects the target.

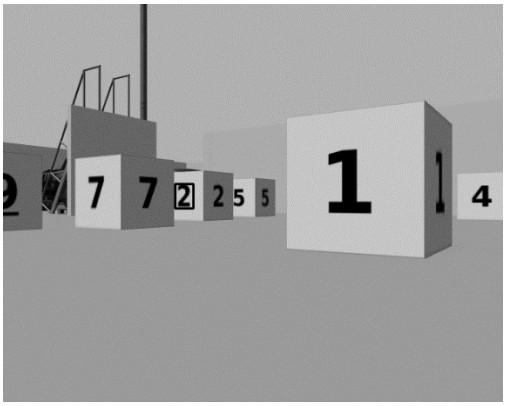


Fig 26. False detection during the turning

Second, in the initial tests, sometimes even if the robot can find the block, the car cannot stop facing the block in the end, resulting in a deviation between the calculated coordinates and the actual coordinates. We found that this is because the odometer has accumulated errors due to offset issues. Therefore, we choose to first obtain the coordinates of the odometer at the starting point of the path and compare them with the coordinates we set, and then add the offset error to the calculated block coordinates. After adjustment, the robot can navigate directly to the front of the block.

Third, upon detection of the target, continuous output of information to the topic occurs when the target is positioned at the center of the image. Moreover, subtle disturbances in position affect the calculation of coordinates and distances. The proposed solution involves setting a flag to transmit only the initial data acquisition.

Fourth, the issue arises when the data output is occasionally not received by the planning functionality due to the disappearance of the topic after publication. The proposed solution is to continuously transmit the detected coordinates and distances at a certain frequency until the planning program can accurately receive and output the coordinates for navigation.

Ultimately, an issue emerged concerning the integration of the user interface. The initial approach entailed leveraging a system command to automatically execute the "rosrun" program within the terminal, facilitating robotic planning. However, this method precipitated performance degradation within the rviz interface, manifesting as stuttering and, in some instances, abrupt terminations. The underlying issue was identified as the "sub system" command executing directly in the active terminal session, leading to resource contention with rviz's operational processes. The resolution involved the adoption of an alternative command, delineated within a C++ (CPP) file, designed to instantiate a new terminal session. Within this newly created session, the "rosrun" command was executed, effectively mitigating the identified integration and resource contention issues.

Future Improvements

a. The quality of original PCD map can be improved by modifying the parameters of SLAM algorithms.
b. Drifting issues of odometry and IMU could be calibrated using a well-designed correction algorithm to improve the distance calculation accuracy and navigation accuracy.
c. Better path planning algorithms is recommended to handle the "V" form obstacles.
d. Learning based method can be implemented for recognition and distance calculation.

Appendix

```
<launch>
<node pkg="pcd2pgm" name="pcd2pgm" type="pcd2pgm" output="screen">
<!-- Path for pcd file-->
<param name="file_directory" value= "/home/tang/" />
<!-- pcd file name-->
<param name="file_name" value= "my_map_fastlio" />
<!-- Minimum chosen height-->
<param name="thre_z_min" value= "0.55" />
<!-- Maximum chosen height-->
<param name="thre_z_max" value= "0.9" />
<!--0 Chosen within range, 1 chosen outside range-->
<param name="flag_pass_through" value= "0" />
<!-- Radius of radius filter-->
<param name="thre_radius" value= "0.1"/>
<!-- Point required by radius filter-->
<param name="thres_point_count" value= "10" />
<!-- Grid map resolution-->
<param name="map_resolution" value= "0.05"/>
<!-- Transformed grid map topic, default 'map', or to use map_server-->
<param name="map_topic_name" value= "map" />
</node>

</launch>
```

Fig 27. pcd2pgm/launch/run.launch

```
<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 GMapping -->
  <include file="$(find jackal_navigation)/launch/include/gmapping.launch" />

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

</launch>
```

Fig 28. mapping.launch

```
<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"/>

    <param name="scan_line" type="int" value="16" />

    <!-- if 1, do mapping 10 Hz, if 2, do mapping 5 Hz. Suggest to use 1, it will adjust frequence automaticlly -->
    <param name="mapping_skip_frame" type="int" value="1" />

    <!-- remove too closed points -->
    <param name="minimum_range" type="double" value="0.3"/>

    <remap from ="/velodyne_points" to ="/mid/points" />

    <param name="mapping_line_resolution" type="double" value="0.2"/>
    <param name="mapping_plane_resolution" type="double" value="0.4"/>

    <node pkg="aloam_velodyne" type="ascanRegistration" name="ascanRegistration" output="screen" />

    <node pkg="aloam_velodyne" type="alaserOdometry" name="alaserOdometry" output="screen" />

    <node pkg="aloam_velodyne" type="alaserMapping" name="alaserMapping" output="screen" />

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

</launch>
```

Fig 29. mapping_aloam.launch

```xml
<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"/>

    <arg name="rviz" default="true" />

    <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="/use_sim_time" value="false" />
    <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" />

    <!--group if="$(arg rviz)">
    <node launch-prefix="nice" pkg="rviz" type="rviz" name="rviz" args="-d $(find fast_lio)/rviz_cfg/loam_livox.rviz" />
    </group-->

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

</launch>
```

Fig 30. mapping_fast_lio.launch

```xml
 <launch>

   <!-- Run the map server -->
   <arg name="map_file" default="$(find me5413_world)/maps/map.yaml"/>
   <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

   <!-- Launch the AMCL Localizer -->
   <include file="$(find me5413_world)/launch/amcl.launch" />

   <!-- Launch Move Base -->
   <include file="$(find me5413_world)/launch/move_base.launch" />

   <node pkg="imu_complementary_filter" type="complementary_filter_node"
       name="complementary_filter_gain_node" output="screen">
     <param name="do_bias_estimation" value="true"/>
     <param name="do_adaptive_gain" value="true"/>
     <param name="use_mag" value="false"/>
     <param name="gain_acc" value="0.01"/>
     <param name="gain_mag" value="0.01"/>
   </node>

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

   <node ns="me5413_world" pkg="me5413_world" type="goal_publisher_node" name="goal_publisher_node" output="screen" />

 </launch>
```

Fig 31. navigation.launch

```
<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="resample_interval" value="1"/>
    <param name="transform_tolerance" value="0.3"/>
    <param name="gui_publish_rate" value="30.0"/>
    <param name="laser_max_beams" value="720"/>
    <param name="laser_min_range" value="0.0"/>
    <param name="laser_max_range" value="10.0"/>
    <param name="min_particles" value="1000"/>
    <param name="max_particles" value="5000"/>
    <!-- Maximum error between the true distribution and the estimated distribution. -->
    <param name="kld_err" value="0.05"/>
    <param name="kld_z" value="0.99"/>
    <param name="odom_alpha1" value="0.2"/>
    <param name="odom_alpha2" value="0.2"/>
    <!-- translation std dev, m -->
    <param name="odom_alpha3" value="0.2"/>
    <param name="odom_alpha4" value="0.2"/>
    <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. A good value might be
    <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. A good value might be 0
    <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"/>
    <param name="initial_cov_yy" value="0.5*0.5"/>
    <param name="initial_cov_aa" value="(π/12)*(π/12)"/>

    <!-- When set to true, AMCL will subscribe to the map topic rather than making a service call to receive its map.-->
    <param name="receive_map_topic" value="true"/>
    <!--  When set to true, AMCL will only use the first map it subscribes to, rather than updating each time a new one is received. -->
    <param name="first_map_only" value="false"/>
    <remap from="scan" to="$(arg scan_topic)"/>
  </node>

</launch>
```

Fig 32. amcl.launch

```xml
<launch>

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

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

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

    <rosparam file="$(find me5413_world)/params/prohibition_areas.yaml" command="load" ns="global_costmap/costmap_prohibition_layer" />

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


    <rosparam file="$(find me5413_world)/params/global_planner_params.yaml" command="load" />
    <rosparam file="$(find me5413_world)/params/move_base_params.yaml" command="load" />

    <!-- <param name="base_global_planner" type="string" value="navfn/NavfnROS" /> -->
    <param name="base_global_planner" value="global_planner/GlobalPlanner"/>


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

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




</launch>
```

Fig 33. move_base.launch

```yaml
me5413_world:
  # the frame of all the poses below
  frame_id: 'world'

  # robot starting point
  start:
    x: 0.0
    y: 0.0
    yaw: 0.0

  # assembly line poses
  assembly_line_1:
    x: 4.05
    y: -1.7
    yaw: -1.57
  assembly_line_2:
    x: 4.05
    y: -4.7
    yaw: -1.57

  # packing area poses
  # packing_area_1:
  #   x: 13.0
  #   y: -0.6
  #   yaw: -1.57
  # packing_area_2:
  #   x: 10.0
  #   y: -0.6
  #   yaw: -1.57
  # packing_area_3:
  #   x: 10.0
  #   y: -3.5
  #   yaw: -1.57
  # packing_area_4:
  #   x: 13.0
  #   y: -3.5
  #   yaw: -1.57

  #area 2
  area_2:
    x: 17
    y: 1.5
    yaw: -1.57
  # delivery vehicle poses
  vehicle_1:
    x: 15.5
    y: -8.2
    yaw: -1.57
  vehicle_2:
    x: 2.9
    y: -9.9
    yaw: 0.0
  vehicle_3:
    x: 2.9
    y: -13.1
    yaw: 0.0
```

Fig 34. config.yaml

```
68        // Box buttons
69        void simplePanel::on_button_2_1_clicked()
70        {
71            ROS_INFO_STREAM("Setting Box 1 as the GOAL.");
72            ui_->label_status->setText("Heading to Box 1");
73            this->goal_name_msg_.data = "/area_2";
74            this->pub_goal_.publish(this->goal_name_msg_);
75        }
76        void simplePanel::on_button_2_2_clicked()
77        {
78            ROS_INFO_STREAM("Button 2 clicked.");
79            ui_->label_status->setText("Target detection box 2");
80            system("gnome-terminal -- rosrun image_cv planning.py");
81        }
```

Fig 35. rviz_panal.cpp

```
22      <gazebo>
23        <plugin name="imu_controller" filename="libhector_gazebo_ros_imu.so">
24          <robotNamespace>/</robotNamespace>
25          <updateRate>50.0</updateRate>
26          <bodyName>imu_link</bodyName>
27          <topicName>imu/data_raw</topicName>
28          <accelDrift>0.005 0.005 0.005</accelDrift>
29          <accelGaussianNoise>0.005 0.005 0.005</accelGaussianNoise>
30          <rateDrift>0.005 0.005 0.005 </rateDrift>
31          <rateGaussianNoise>0.005 0.005 0.005 </rateGaussianNoise>
32          <headingDrift>0.005</headingDrift>
33          <headingGaussianNoise>0.005</headingGaussianNoise>
34        </plugin>
35      </gazebo>
```

Fig 36. jackal.gazebo

```
1   #!/usr/bin/env python
2
3   import rospy
4   from std_msgs.msg import String
5
6   def callback(data):
7       rospy.loginfo(rospy.get_caller_id() + 'I heard %s', data.data)
8
9   def listener():
10
11      # In ROS, nodes are uniquely named. If two nodes with the same
12      # name are launched, the previous one is kicked off. The
13      # anonymous=True flag means that rospy will choose a unique
14      # name for our 'listener' node so that multiple listeners can
15      # run simultaneously.
16      rospy.init_node('listener', anonymous=True)
17
18      rospy.Subscriber('chatter', String, callback)
19
20      # spin() simply keeps python from exiting until this node is stopped
21      rospy.spin()
22
23  if __name__ == '__main__':
24      listener()
```

Fig 37. listener.py

```python
#!/usr/bin/env python3

import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2
import numpy as np
import os
from std_msgs.msg import String
from nav_msgs.msg import Odometry
from sensor_msgs.msg import Imu
from tf.transformations import euler_from_quaternion
from geometry_msgs.msg import Pose
from geometry_msgs.msg import Point

ratio_w = 0.35  # 0.32
ratio_h = 0.48  # 0.45

# Camera focal length(pixel)
FOCAL_LENGTH = 554.254691191187

# Box width(m)
BLOCK_WIDTH = 0.8

Flag = 'False'
First_flag = 'False'
current_pose = None  # Current pose
initial_pose = None  # Initial pose
initial_flag = 'False'
initial_point = {'x': 17, 'y': -4.5, 'yaw': 1.57}
bias_pose = None

angular_velocity_z = None  # Angular velocity in z
orientation_quaternion = None  # Pose quaternion
yaw = None

current_dir = os.path.dirname(os.path.realpath(__file__))
template_images = []
for i in range(1, 11):
    template_images.append(cv2.imread(os.path.join(current_dir, f'template_2/2_{i}.png'), cv2.IMREAD_GRAYSCALE))

def odometry_callback(msg):
    global current_pose, initial_flag, initial_pose, bias_pose
    current_pose = msg.pose.pose
    if initial_flag == 'False':
        initial_pose = current_pose
        initial_pose_info = f"Initial Position: x={initial_pose.position.x:.2f}, y={initial_pose.position.y:.2f}, z={initial_pose.position.z:.2f}"
        rospy.loginfo(initial_pose_info)
        bias_pose = Pose()
        bias_pose.position.x = initial_pose.position.x - initial_point['x']
        bias_pose.position.y = initial_pose.position.y - initial_point['y']
        rospy.loginfo(f"Bias Pose: x={bias_pose.position.x:.2f}, y={bias_pose.position.y:.2f}, z={bias_pose.position.z:.2f}")
        initial_flag = 'True'


def imu_callback(msg):
    global angular_velocity_z, orientation_quaternion, yaw
    angular_velocity_z = msg.angular_velocity.z
    orientation_quaternion = msg.orientation


    euler_angles = euler_from_quaternion([orientation_quaternion.x, orientation_quaternion.y, orientation_quaternion.z, orientation_quaternion.w])
    yaw = euler_angles[2]  # Yaw angle
    #rospy.loginfo(f"Yaw angle: {yaw:.2f} radians")  # Output to terminal



def talker(distance_str):
    global Flag, current_pose, bias_pose
    if Flag == 'True' and current_pose is not None:
        pub = rospy.Publisher('signal', String, queue_size=10)
        position = current_pose.position

        position_info = f"Robot Position: x={position.x:.2f}, y={position.y:.2f}, z={position.z:.2f}"


        new_position = Point()
        new_position.x = position.x - bias_pose.position.x
        new_position.y = position.y - bias_pose.position.y
        new_position.z = position.z

        new_position_info = f"Robot New Position: x={new_position.x:.2f}, y={new_position.y:.2f}, z={new_position.z:.2f}"

        rate = rospy.Rate(10)
        while not rospy.is_shutdown():
            rospy.loginfo(distance_str)
            rospy.loginfo(position_info)
            rospy.loginfo(new_position_info)
            pub.publish(f"{distance_str}, {new_position_info}")
            Flag = 'False'  # Set Flag to 'False'
            rate.sleep()


def image_callback(msg):
```

```
    global Flag
    global First_flag, yaw
    try:
        bridge = CvBridge()
        cv_image = bridge.imgmsg_to_cv2(msg, "mono8")
        img_height, img_width = cv_image.shape
        target_gray = cv2.GaussianBlur(cv_image, (5, 5), 0)

        max_similarity = -1
        best_match_loc = None
        best_template_index = None

        for idx, template_image in enumerate(template_images):
            res = cv2.matchTemplate(target_gray, template_image, cv2.TM_CCOEFF_NORMED)
            _, max_val, _, max_loc = cv2.minMaxLoc(res)

            if max_val > max_similarity:
                max_similarity = max_val
                best_match_loc = max_loc
                best_template_index = idx

        threshold = 0.7
        if max_similarity >= threshold:
            template_image_shape = template_images[best_template_index].shape
            top_left = best_match_loc
            bottom_right = (top_left[0] + template_image_shape[1], top_left[1] + template_image_shape[0])
            cv2.rectangle(cv_image, top_left, bottom_right, (0, 0, 255), 2)

            width = template_image_shape[1]
            height = template_image_shape[0]

            W = round(width / ratio_w, 2)
            H = round(height / ratio_h, 2)

            AVG = round((W + H) / 2, 2)

            yaw_threshold = 0.5 # 0.5

            if abs((bottom_right[0] + top_left[0]) / 2 - img_width / 2) < img_width * 0.05 and First_flag == 'False' and (((1.57 - yaw_threshold) < yaw < (1.57 +
                yaw_threshold))  or ((-1.57 - yaw_threshold) < yaw < (-1.57 + yaw_threshold)) or ((0 - yaw_threshold) < yaw < (0 + yaw_threshold))) or ((-3.14) < yaw <
                (-3.14 + yaw_threshold)) or ((3.14- yaw_threshold) < yaw < (3.14)):    # or ((-3.14 - yaw_threshold) < yaw < (-3.14 + yaw_threshold))
                distance = (BLOCK_WIDTH * FOCAL_LENGTH) / AVG
                distance_str = f"Distance: {distance:.2f} m"
                Flag = 'True'
                First_flag = 'True'

                talker(distance_str)
```

```
                # Output on image
                cv2.putText(cv_image, "Target in Center", (20, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2, cv2.LINE_AA)
                cv2.putText(cv_image, distance_str, (20, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2, cv2.LINE_AA)
            else:
                Flag = 'False'

        # cv2.putText(cv_image, Flag, (20, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2, cv2.LINE_AA)

        cv2.imshow("Image Window", cv_image)
        cv2.waitKey(1)
    except Exception as e:
        print(e)

def main():
    rospy.init_node('image_subscriber', anonymous=True)

    rospy.Subscriber("/odometry/filtered", Odometry, odometry_callback)  # Subscribe odom
    rospy.Subscriber("/imu/data", Imu, imu_callback)  # Subscribe IMU data
    rospy.Subscriber("/front/image_raw", Image, image_callback)

    rospy.spin()

if __name__ == '__main__':
    main()
```

Fig 38. image_process.py

```python
#!/usr/bin/env python3

import rospy
import actionlib
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from geometry_msgs.msg import PoseStamped
import subprocess
import os
import signal

from std_msgs.msg import String
from nav_msgs.msg import Odometry
from tf.transformations import quaternion_from_euler

# Define trajectory coordinates and yaw angle
points = [
    {'x': 17, 'y': -4.5, 'yaw': 1.57},
    {'x': 17, 'y': 1.4, 'yaw': 1.57},
    {'x': 7.2, 'y': 1.4, 'yaw': -3.14},
    {'x': 7.2, 'y': -6.7, 'yaw': -1.57},
    {'x': 15.4, 'y': -6.7, 'yaw': 0.0}
]

# Index for path to be followed
paths = [(0, 1), (1, 2), (2, 3), (3, 4)]


# Initialize node
rospy.init_node('navigate_through_points')

# move_base action client
client = actionlib.SimpleActionClient('move_base', MoveBaseAction)
client.wait_for_server()

image_process_proc = None

# Flag for whether to stop at next point
stop_at_next_point = False
current_path_index = 0  # Track current path index
First_flag = 'False'

# Call back function, to stop receive signals
def stop_callback(msg):
    global stop_at_next_point, current_path_index, First_flag
    # Check whether message have distance message
    if 'Distance' in msg.data and First_flag == 'False':
        try:
            # Extract distance
            parts = msg.data.split(',')
            distance_str = parts[0].split(':')[1].strip()
            distance = float(distance_str.split()[0])
            # Extract curren location
            x_pos_str = parts[1].split('=')[1].strip()
            y_pos_str = parts[2].split('=')[1].strip()
            x_pos = float(x_pos_str.split()[0])
            y_pos = float(y_pos_str.split()[0])
            # Adjust coordinate according to relative position
            X = 0.4 #0.8
            # distance = abs(distance)

            if current_path_index == 0:  # 0-1 path
                target_x = x_pos - (distance - X)
                target_y = y_pos
                yaw = 1.57
            elif current_path_index == 1:  # 1-2 path
                target_x = x_pos
```

```
                    target_y = y_pos - (distance - X)
                    yaw = -3.14
                elif current_path_index == 2:  # 2-3 path
                    target_x = x_pos + (distance - X)
                    target_y = y_pos
                    yaw = -1.57
                elif current_path_index == 3:  # 3-4 path
                    target_x = x_pos
                    target_y = y_pos + (distance - X)
                    yaw = 0

                rospy.loginfo(f"Calculating block position to navigate: x={target_x}, y={target_y}, yaw={yaw}")

                # Navigate to calculated box position
                navigate_to_block(target_x, target_y, yaw)
                stop_at_next_point = True

                First_flag = 'True'

            except ValueError as e:
                rospy.logerr(f"Error parsing message: {msg.data} | Error: {e}")




def navigate_to_block(x, y, yaw):
    """Navigate to calculated box position, output log info"""
    quaternion = quaternion_from_euler(0, 0, yaw)
    goal_pose = PoseStamped()
    goal_pose.header.frame_id = 'map'
    goal_pose.pose.position.x = x
    goal_pose.pose.position.y = y
    goal_pose.pose.orientation.x = quaternion[0]
    goal_pose.pose.orientation.y = quaternion[1]
    goal_pose.pose.orientation.z = quaternion[2]
    goal_pose.pose.orientation.w = quaternion[3]
    goal = MoveBaseGoal()
```

```
    goal.target_pose = goal_pose
    client.send_goal(goal)
    client.wait_for_result()

    if client.get_state() == actionlib.GoalStatus.SUCCEEDED:
        rospy.loginfo("Successfully reached the block!")
    else:
        rospy.loginfo("Failed to reach the block.")

def navigate_to_point(x, y, yaw):
    """Navigate to point, output log info"""
    quaternion = quaternion_from_euler(0, 0, yaw)
    goal_pose = PoseStamped()
    goal_pose.header.frame_id = 'map'
    goal_pose.pose.position.x = x
    goal_pose.pose.position.y = y
    goal_pose.pose.orientation.x = quaternion[0]
    goal_pose.pose.orientation.y = quaternion[1]
    goal_pose.pose.orientation.z = quaternion[2]
    goal_pose.pose.orientation.w = quaternion[3]
    goal = MoveBaseGoal()
    goal.target_pose = goal_pose
    client.send_goal(goal)
    client.wait_for_result()

    if client.get_state() == actionlib.GoalStatus.SUCCEEDED:
        rospy.loginfo(f"Successfully reached point {x}, {y}")
    else:
        rospy.loginfo("Failed to reach the point.")


#def odometry_callback(msg):
    #global current_pose
    #current_pose = msg.pose.pose


# Stop subscriber
```

```
141  rospy.Subscriber('/signal', String, stop_callback)
142
143  navigate_to_point(points[0]['x'], points[0]['y'], points[0]['yaw'])
144  rospy.sleep(0.5)
145
146
147  #current_pose = None  # current pose
148  #rospy.Subscriber("/odometry/filtered", String, odom_callback)
149
150
151  # Define goal pose
152  for idx, (start, end) in enumerate(paths):
153      current_path_index = idx
154      if idx == 0 and image_process_proc is None:
155          image_process_path = os.path.join(os.path.dirname(__file__), 'image_process.py')
156          image_process_proc = subprocess.Popen(['python3', image_process_path])
157
158      # Navigate to point and output log info
159      rospy.sleep(0.5)
160      navigate_to_point(points[end]['x'], points[end]['y'], points[end]['yaw'])
161      rospy.sleep(0.5)
162      if stop_at_next_point:
163          rospy.loginfo("Detected object, stopped to calculate block position")
164          break
165
166
167  # Shutdown
168  def shutdown_hook():
169      rospy.loginfo("Shutting down...")
170      if image_process_proc:
171          image_process_proc.send_signal(signal.SIGINT)
172
173
174  rospy.on_shutdown(shutdown_hook)
175
176  rospy.spin()
```

Fig 39. planning.py

```
1   #!/usr/bin/env python
2
3   import rospy
4   from std_msgs.msg import String
5
6   def talker():
7       pub = rospy.Publisher('chatter', String, queue_size=10)
8       rospy.init_node('talker', anonymous=True)
9       rate = rospy.Rate(10) # 10hz
10      while not rospy.is_shutdown():
11          hello_str = "hello world %s" % rospy.get_time()
12          rospy.loginfo(hello_str)
13          pub.publish(hello_str)
14          rate.sleep()
15
16  if __name__ == '__main__':
17      try:
18          talker()
19      except rospy.ROSInterruptException:
20          pass
```

Fig 40. talker.py

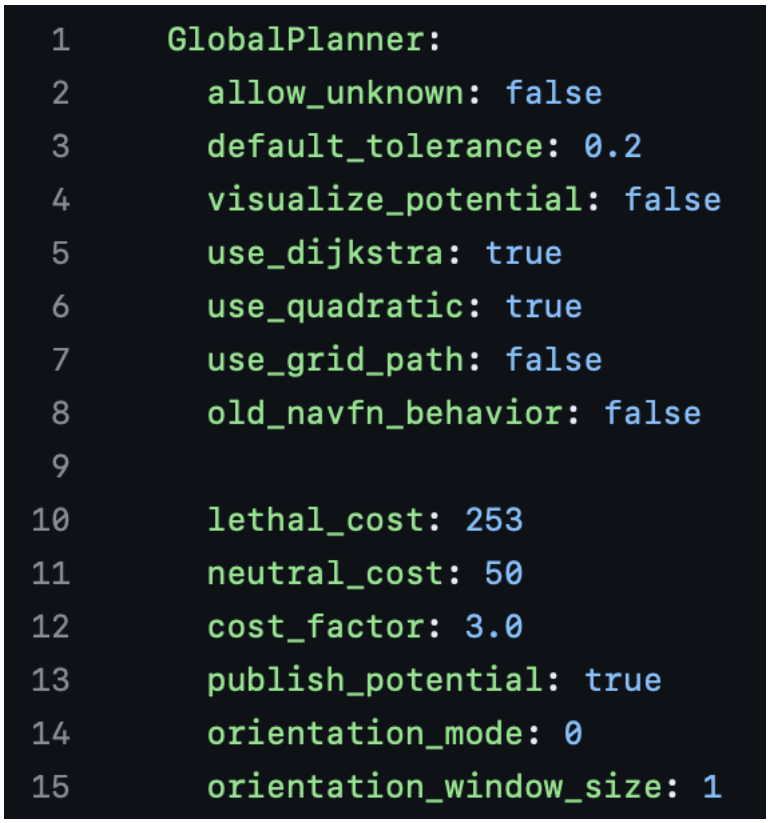
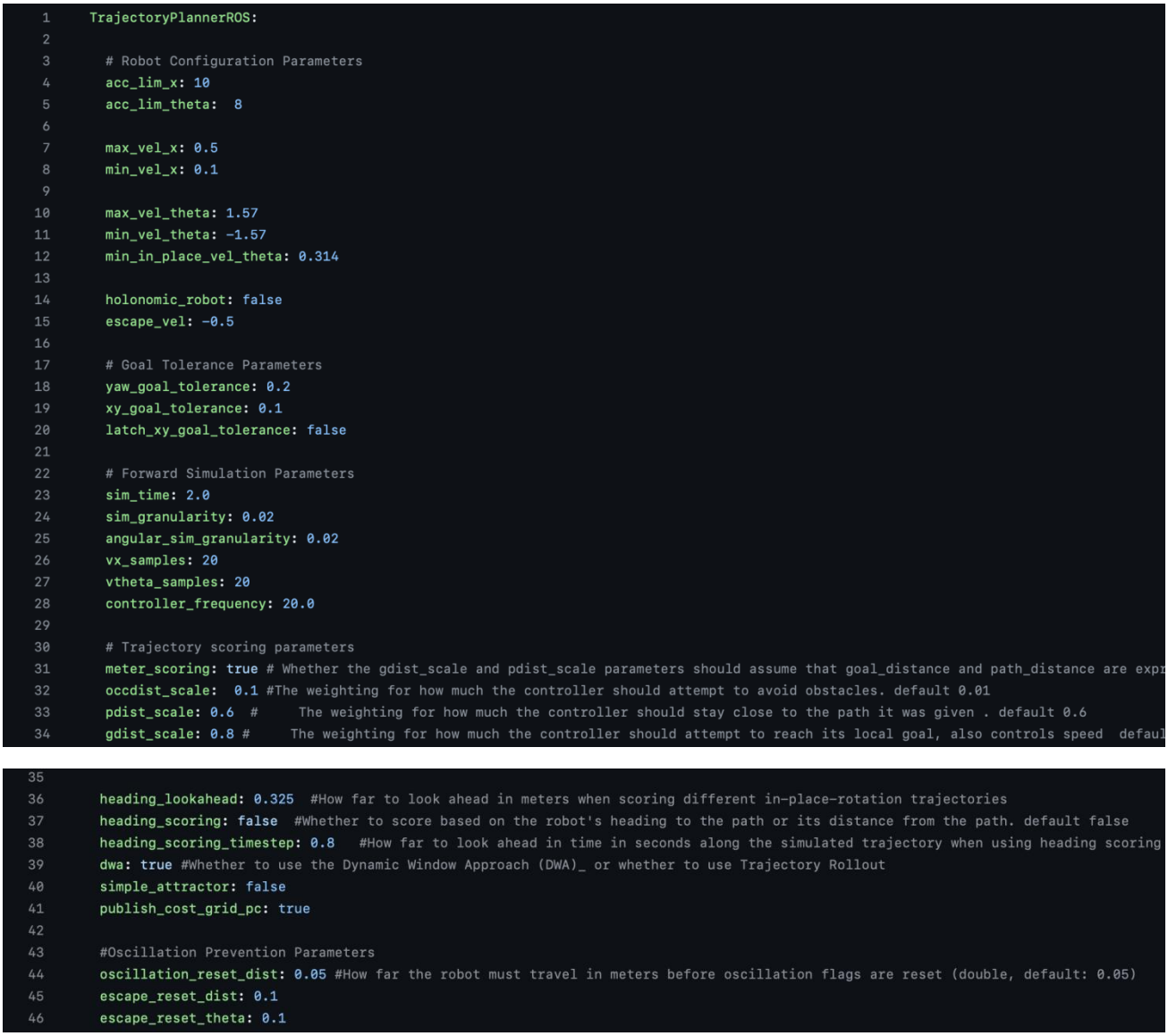


Fig 41. base_global_planner_params.yaml



```yaml
TrajectoryPlannerROS:

  # Robot Configuration Parameters
  acc_lim_x: 10
  acc_lim_theta:  8

  max_vel_x: 0.5
  min_vel_x: 0.1

  max_vel_theta: 1.57
  min_vel_theta: -1.57
  min_in_place_vel_theta: 0.314

  holonomic_robot: false
  escape_vel: -0.5

  # Goal Tolerance Parameters
  yaw_goal_tolerance: 0.2
  xy_goal_tolerance: 0.1
  latch_xy_goal_tolerance: false

  # Forward Simulation Parameters
  sim_time: 2.0
  sim_granularity: 0.02
  angular_sim_granularity: 0.02
  vx_samples: 20
  vtheta_samples: 20
  controller_frequency: 20.0

  # Trajectory scoring parameters
  meter_scoring: true # Whether the gdist_scale and pdist_scale parameters should assume that goal_distance and path_distance are expr
  occdist_scale:  0.1 #The weighting for how much the controller should attempt to avoid obstacles. default 0.01
  pdist_scale: 0.6  #     The weighting for how much the controller should stay close to the path it was given . default 0.6
  gdist_scale: 0.8 #     The weighting for how much the controller should attempt to reach its local goal, also controls speed  defaul

  heading_lookahead: 0.325  #How far to look ahead in meters when scoring different in-place-rotation trajectories
  heading_scoring: false  #Whether to score based on the robot's heading to the path or its distance from the path. default false
  heading_scoring_timestep: 0.8   #How far to look ahead in time in seconds along the simulated trajectory when using heading scoring
  dwa: true #Whether to use the Dynamic Window Approach (DWA)_ or whether to use Trajectory Rollout
  simple_attractor: false
  publish_cost_grid_pc: true

  #Oscillation Prevention Parameters
  oscillation_reset_dist: 0.05 #How far the robot must travel in meters before oscillation flags are reset (double, default: 0.05)
  escape_reset_dist: 0.1
  escape_reset_theta: 0.1
```

Fig 42. base_local_planner_params.yaml

```yaml
map_type: costmap
origin_z: 0.0
z_resolution: 1
z_voxels: 2
resolution: 0.05
obstacle_range: 2.5 #2
raytrace_range: 3

publish_voxel_map: False #Flase
transform_tolerance: 0.5
meter_scoring: true

#Cost function parameters
inflation_radius: 0.21
cost_scaling_factor: 8

footprint: [[-0.21, -0.165], [-0.21, 0.165], [0.21, 0.165], [0.21, -0.165]]
footprint_padding: 0.1

plugins:
- {name: obstacles_layer, type: "costmap_2d::ObstacleLayer"}
- {name: inflater_layer, type: "costmap_2d::InflationLayer"}

obstacles_layer:
  observation_sources: scan
  scan: {sensor_frame: tim551, data_type: LaserScan, topic: front/scan, marking: true, clearing: true, min_obstacle_height: -2.0, max_obst
```

Fig 43. costmap_common_params.yaml

```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
  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
```

Fig 44. dwa_local_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 fals
  use_quadratic: true                          # Use the quadratic approximation of the potential. Otherwise, use a simpler calculation,
  use_grid_path: false                        # Create a path that follows the grid boundaries. Otherwise, use a gradient descent method,

  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
  planner_window_x: 0.0                        # default 0.0
  planner_window_y: 0.0                        # default 0.0
  default_tolerance: 2                        # 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 tr

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

Fig 45. global_planner_params.yaml

```yaml
  shutdown_costmaps: false

  controller_frequency: 20.0
  controller_patience: 15.0

  planner_frequency: 20.0
  planner_patience: 5.0

  oscillation_timeout: 0.0
  oscillation_distance: 0.5

  recovery_behavior_enabled: true
  clearing_rotation_allowed: true
```

Fig 46. move_base_params.yaml

```yaml
  prohibition_areas:
  # #定义一个禁止点
  #  - [17.09, -6.388]
  # # 定义一个禁止通行的线
  #  - [[8.33, 2.11],
  #     [8.26, 5.11]]
  # 定义一个禁止通行的区域
   - [[1, 0.75],
      [3, 0.75],
      [1, 0.85],
      [3,0.85]]

   - [[9.5, -8],
      [10.5, -8],
      [9.5, -9],
      [10.5,-9]]
```

Fig 47. prohibition_areas.yaml

```yaml
TebLocalPlannerROS:

 odom_topic: odom
 map_frame: odom

 # Trajectory

 teb_autosize: True
 dt_ref: 0.3
 dt_hysteresis: 0.1
 max_samples: 300
 global_plan_overwrite_orientation: False #False
 allow_init_with_backwards_motion: False
 max_global_plan_lookahead_dist: 3
 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: 0.8 #0.2
 max_vel_x_backwards: 0.2
 max_vel_y: 0.0
 max_vel_theta: 0.8 #0.25
 acc_lim_x: 5
 acc_lim_theta: 6
 min_turning_radius: 0 # diff-drive robot (can turn on place!)0.25

 footprint_model:
  type: "polygon"
  #type: "point"
  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
```

```
    free_goal_vel: False
    complete_global_plan: True

    # Obstacles

    min_obstacle_dist: 0.2 # This value must also include our robot radius, since footprint_model is set to "point". 0.25
    inflation_dist: 0.2 #0.3
    include_costmap_obstacles: True
    costmap_obstacles_behind_robot_dist: 0.5
    obstacle_poses_affected: 10

    dynamic_obstacle_inflation_dist: 0.2
    include_dynamic_obstacles: True

    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: 10
    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
```

Fig 48. teb_local_planner_params.yaml

```yaml
global_costmap:
   global_frame: map
   robot_base_frame: base_link
   update_frequency: 5.0
   publish_frequency: 0.0
   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: costmap_prohibition_layer, type: "costmap_prohibition_layer_namespace::CostmapProhibitionLayer"}
   - {name: inflater_layer, type: "costmap_2d::InflationLayer"}
```

Fig 49. global_costmap_paramas.yaml

```yaml
local_costmap:
   global_frame: map
   robot_base_frame: base_link
   update_frequency: 10.0
   publish_frequency: 2.0
   width: 8
   height: 8
   resolution: 0.05
   static_map: false
   rolling_window: true

plugins:
   #- {name: static_layer, type: "costmap_2d::StaticLayer"}
   - {name: obstacles_layer, type: "costmap_2d::ObstacleLayer"}
   - {name: inflater_layer, type: "costmap_2d::InflationLayer"}
inflater_layer:
   inflation_radius: 0.1
```

Fig 50. local_costmap_params.yaml