National University of Singapore

College of Design and Engineering

ME5413 Autonomous Mobile Robotics: Homework 2

Group 9:
Chen Yihui A0263115N
Wang Renjie A0263387U

Supervisor:
Prof. Marcelo H Ang Jr

Email Address: *e1010473@u.nus.edu*
e1010745@u.nus.edu

March 5, 2023

1 Task 1: ICP Algorithm for Matched Point Cloud

In Task 1, we assume the correspondence is known for the two point clouds and perform ICP algorithm to do point cloud registration. Firstly, the SVD-based ICP algorithm is reviewed.

Given the source point cloud $X = \{x_1, x_2, ..., x_m\}$ and target point cloud $Y = \{y_1, y_2, ..., y_m\}$, the task for point cloud registration is to find a rotation matrix R and a translation matrix t that can transform X to Y. In other words, we need to find R and t to minimize the sum of the squared distance between the corresponding points in the two point clouds:

$$\begin{aligned} E(R,t) &= \frac{1}{m}\sum_{i=1}^{m}\|y_i - Rx_i - t\|^2 \\ &= \frac{1}{m}\sum_{i=1}^{m}(\|y_i - u_y - R(x_i - u_x)\|^2 + \|u_y - Ru_x - t\|^2) \end{aligned} \tag{1}$$

where u_x and u_y are the centroid of the point set X and Y, $u_x = \frac{1}{m}\sum_{i=1}^{m} x_i$ and $u_y = \frac{1}{m}\sum_{i=1}^{m} y_i$. To simplify the problem, we divide Eq. 1 to:

$$E_1(R,t) = \frac{1}{m}\sum_{i=1}^{m}\|y_i - u_y - R(x_i - u_x)\|^2, \quad E_2(R,t) = \|u_y - Ru_x - t\|^2 \tag{2}$$

Therefore, we can first find the R to minimize $E_1(R,t)$ and then calculate t from $E_2(R,t)$. Suppose $x_i^{'} = x_i - u_x$, $y_i^{'} = y_i - u_y$, the problem turns to:

$$min\{E_1(R,t)\} = min\{\frac{1}{m}\sum_{i=1}^{m}(y_i^{'T}y_i^{'} + x_i^{'T}R^TRx_i^{'} - 2y_i^{'T}Rx_i^{'})\} = max\{\sum_{i=1}^{m} y_i^{'T}Rx_i^{'}\} \tag{3}$$

Using the properties of trace, we derive:

$$\sum_{i=1}^{m} y_i^{'T}Rx_i^{'} = \sum_{i=1}^{m} Trace(y_i^{'T}Rx_i^{'}) = Trace(\sum_{i=1}^{m} Rx_i^{'}y_i^{'T}) = Trace(RH) \tag{4}$$

where $H = \sum_{i=1}^{m} x_i^{'}y_i^{'T}$. According to Schwarz inequality, we can maximum Eq. 4 by finding a R that can convert RH to the form of AA^T. Therefore, we apply Singular Value Decomposition (SVD) on H and get $H = U\Sigma V^T$. Then we choose the rotation matrix:

$$R = VU^T \tag{5}$$

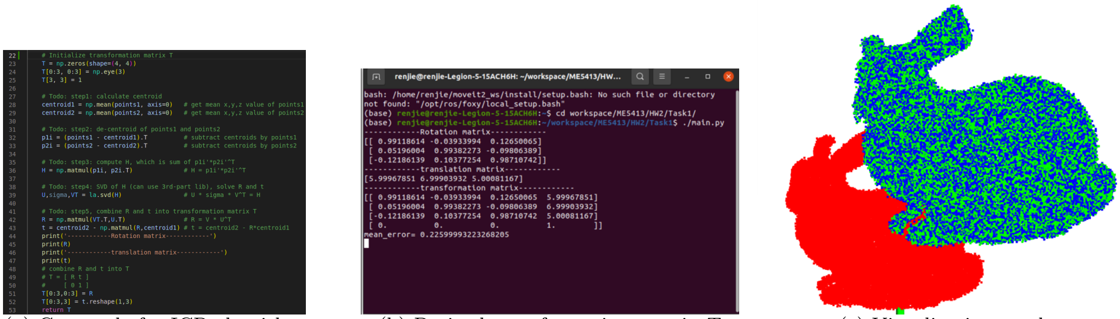
Then, $RH = VU^TU\Sigma V^T = V\Sigma V^T = V\Sigma^{1/2}(V\Sigma^{1/2})^T$, which is in the form of AA^T and we finally minimize $E_1(R,t)$. After obtaining R, we can solve the translation matrix t to minimize $E_2(R,t)$:

$$t = u_y - Ru_x \tag{6}$$

Since we have known the corresponding point pairs between source point cloud X and target Y, we can easily get the homogeneous transformation matrix $T \in SE(3)$ by combing the rotation matrix $R \in SO(3)$ and translation matrix $t \in \mathbb{R}^3$ computed in Eq. 5 and 6:

$$T = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \tag{7}$$

The core code of ICP algorithm with comment is shown in Fig. 1a and derived transformation matrix T is recorded in Fig. 1b. As is shown in Fig. 1c, the source point cloud (red) is transformed to the blue one, which is overlapped with the target point cloud (green). The final mean error between the blue and green point clouds is about 0.226.



(a) Core code for ICP algorithm (b) Derived transformation matrix T (c) Visualization result

Figure 1: Implementation of ICP algorithm for matched point cloud

2 Task 2: ICP Algorithm for Unmatched Point Cloud

In real cases, the correspondence between the source and target point cloud is usually unknown. A common method is to regard the nearest neighbor point as corresponding point and solve the problem iteratively. The ICP algorithm for unmatched point cloud is given in Alg. 1.

Algorithm 1 SVD-based ICP algorithm for unmatched point cloud

---

Initialize the accumulated transformation matrix $T\_accumulated$.
while $error > threshold$ **do**
  Step 1: Find the corresponding points from two point clouds.
  Step 2: Compute the transformation matrix T following the steps in Task 1.
  Step 3: Update accmulated matrix. $(T\_accumulated = T * T\_accumulated)$
  Step 4: Apply alignment. $(x_i^{'} = T * x_i)$
  Step 5: Update error.
end while

The *threshold* is set to be 0.1 and core code for the iterative ICP algorithm is shown in Fig. 2. To show the more and more overlapped rabits, we record the transformation matrix T and $T\_accumulated$ in iteration 7, 15 and 23, as well as visualizing the two point clouds and mean error between them (Fig. 3).

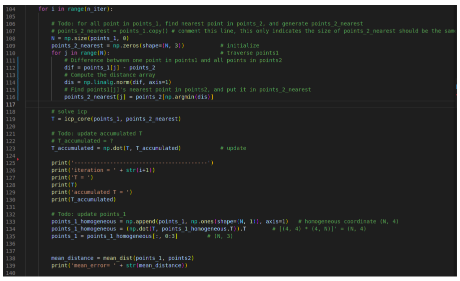
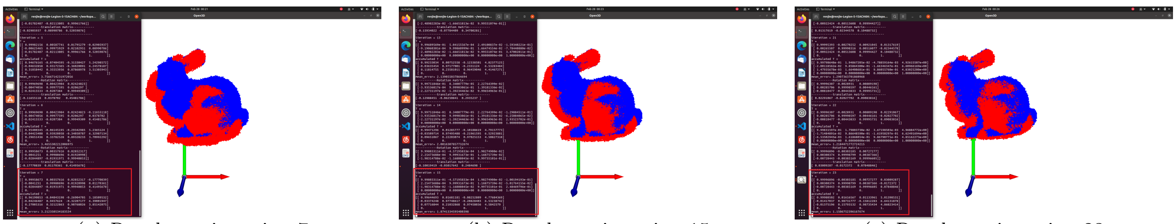


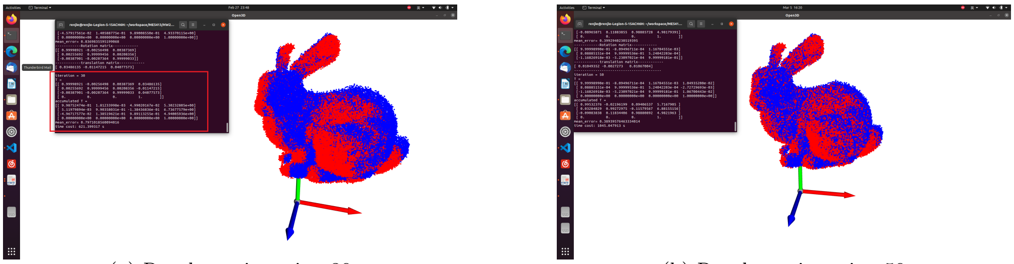
Figure 2: Code for ICP algorithm with unmatched point cloud



(a) Results at iteration 7 (b) Results at iteration 15 (c) Results at iteration 23

Figure 3: Implementation of ICP algorithm for unmatched point cloud

We can see the mean error is reduced after each iteration (3.212 to 1.874 to 1.151), and it declines more and more slowly as iteration number increases. The final result at 30th iteration is shown in Fig. 4, where the mean error is about 0.797 and still higher than the mannually set threshold. We increase the maximum iteration number to 50 and the result is shown in Fig. 4b. The mean error is only 0.389 now and the accumulated transformation matrix *T_accumulated* is very close to that we derived in Task1. However, the cost time is rather high (1045s) because of the time-consuming corresponding points search procedure. In practice, kd-tree can be used to search for the nearest points more efficiently. Moreover, some learning-based methods have been popular for solving the point cloud registration problem.



(a) Results at iteration 30 (b) Results at iteration 50

Figure 4: Implementation of ICP algorithm for unmatched point cloud

3 Task 3: Running SLAM Algorithms

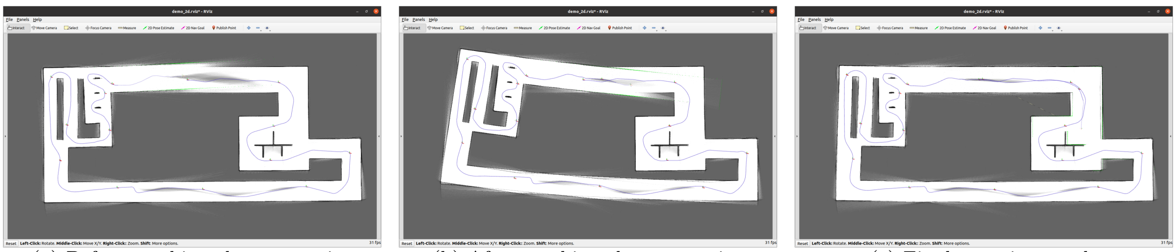
3.1 Cartographer for 2D LiDAR SLAM

Cartographer is an open-source, real-time, simultaneous localization and mapping (SLAM) system developed by Google. It is designed to create high-resolution maps of indoor and outdoor environments using data collected from a variety of sensors, including LiDAR, IMU, and cameras. Here we apply the algorithm to do 2D mapping using the collected data in the given rosbag (*'<2dlidar.bag >'*).

After compiling the Cartographer on ROS Noetic, we need to configure the *.lua* and *.lauch* file to run the algorithm on our own rosbag. First we check the topics in the given rosbag and use *"rqt_graph"* and *"rqt_tf_tree"* to plot the node diagram and tf structure. Then we need to change some important parameters:

- We set *published_frame* to *"odom"* and set the flag *use_odometry* as *"true"*.
- Since we only have one lidar, we set *num_laser_scans = 1, num_multi_echo_laser_scans = 0.*
- We also need to set *TRAJECTORY_BUILDER_2D.use_imu_data = false* since we haven't applied IMU.

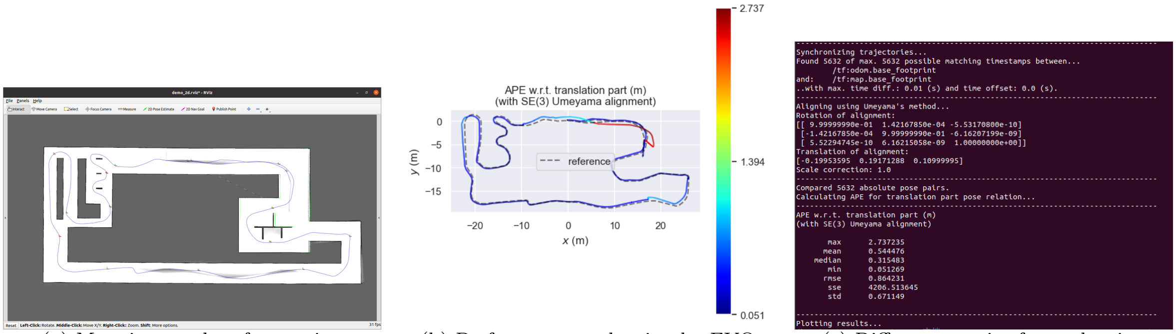
As for the .launch file, we start recording a new rosbag before the algorithm starts for evaluating the performance afterwards. We include our *.lua* configuration and start rviz for visualization. The whole *.lua* and *.launch* file is included in the appendix. The mapping process visualized by rviz is shown in Fig. 5, where we can see the loop closure can adjust the map to achieve better results.



(a) Before reaching the start point (b) After reaching the start point (c) Final mapping result

Figure 5: 2D LiDAR SLAM by Cartographer algorithm

We can see the map rotation is obvious since we have no IMU, we should give a lower rotation weight for *ceres_scan_matcher*. We also set minimum and maximum lidar range to be 0.3 and 100 respectively, and voxel filter size is set to be 0.02. Moreover, we find the loop closure is not stable, so we increase the *min_score* of *constraint_builder* to 0.7. The mapping results after tuning is shown in Fig. 6a, where the performance looks slightly better than the former case. By the EVO tool, we plot the evaluation results in Fig. 6b. We can see the maximum drift (2.737) appears after the robot reaches the start point, and the Absolute RMSE is 0.864 (Fig. 6c), which is relatively good.



(a) Mapping results after tuning (b) Performance evaluation by EVO (c) Different metrics for evaluation

Figure 6: Performance evaluation by EVO

One of the main disadvantages of Cartographer algorithm is that it requires a lot of computational resources, which can limit its use on low-power devices. Therefore, we may modify more parameters to achieve a lower latency. In addition, it can be challenging to configure and fine-tune so many hyper-parameters in Cartographer, and the "best" combination should be selected according to specific applications.

3.2 A-LOAM for 3D LiDAR SLAM

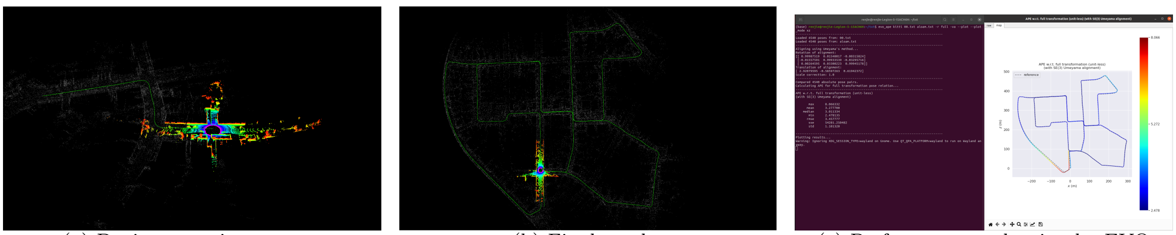
A-LOAM (Advanced Lidar Odometry And Mapping) is an improved framework from LOAM, using Eigen and Ceres Solvers to simplify code structure. A-LOAM use two parallel algorithms: a low-precision high-speed odometer and a low-speed high-precision odometer, combining the two odometers to obtain real-time map updates. The feature matching method adopted improves the accuracy and operation efficiency of the algorithm.

In this task, we refer this repo[1] to help do the task and directly generate odometry file for EVO evaluation. The process of algorithm implementation is: 1) Develop the Aloam from repo[1] with modifications needed. 2) Create a folder named *'txt'* with *'00.txt'* and empty *'aloam.txt'*. 3) Launch the Aloam and then meanwhile play *'<3dlidar.bag >'*. 4) Evaluate the performance of the algorithm with EVO.

Notice that in step 1, there are some modifications we did which the README.md do not mention.

- Firstly, uncomment the *"//generate the KITTI format trajectory result"* part in the *laserMapping.cpp*, so that the Aloam lidar odometry result will be stored in *'aloam.txt'*.
- Secondly, the *'ceres::LocalParameterization'* and *'ceres::EigenQuaternionParameterization'* are deprecated in the latest release of Ceres Solver (v 2.1.0) and need to be replaced to *'ceres::Manifold'* and *'ceres::QuaternionManifold()'*.

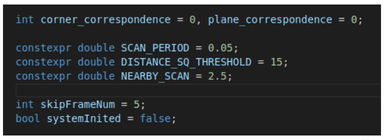
The result of the EVO evaluation of this 3D SLAM task and the screenshot of algorithm running in rviz is shown in Fig. 7. From the result, it can be seen that the max error value is 8.066, the mean error value is 3.278 and the min error value is 2.478. Thus, the performance of this algorithm is not bad.



(a) During mapping process (b) Final results (c) Performance evaluation by EVO

Figure 7: 3D SLAM algorithm running in rviz and performance

To further improve the performance, we tried tuning some parameters ('DISTANCE_SQ_THRESHOLD' from 25 to 15 and 'SCAN_PERIOD' from 0.1 to 0.05) in *LaserOdometry.cpp* and successfully reduced the maximum error from 8.066 to 7.748 as shown in Fig. 8.



```
int corner_correspondence = 0, plane_correspondence = 0;

constexpr double SCAN_PERIOD = 0.05;
constexpr double DISTANCE_SQ_THRESHOLD = 15;
constexpr double NEARBY_SCAN = 2.5;

int skipFrameNum = 5;
bool systemInited = false;
```

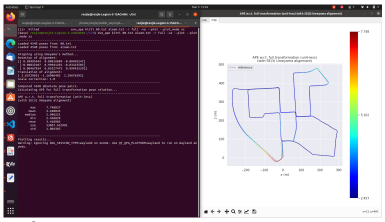


Figure 8: 3D SLAM task result after tuning

The disadvantage of A-LOAM is that it does not do back-end closed-loop detection, so perhaps we can seek to correct motion estimation drift through closed-loop detection. Alternatively, the output can be added to a filter, such as a Kalman filter, to further reduce motion estimation drift.

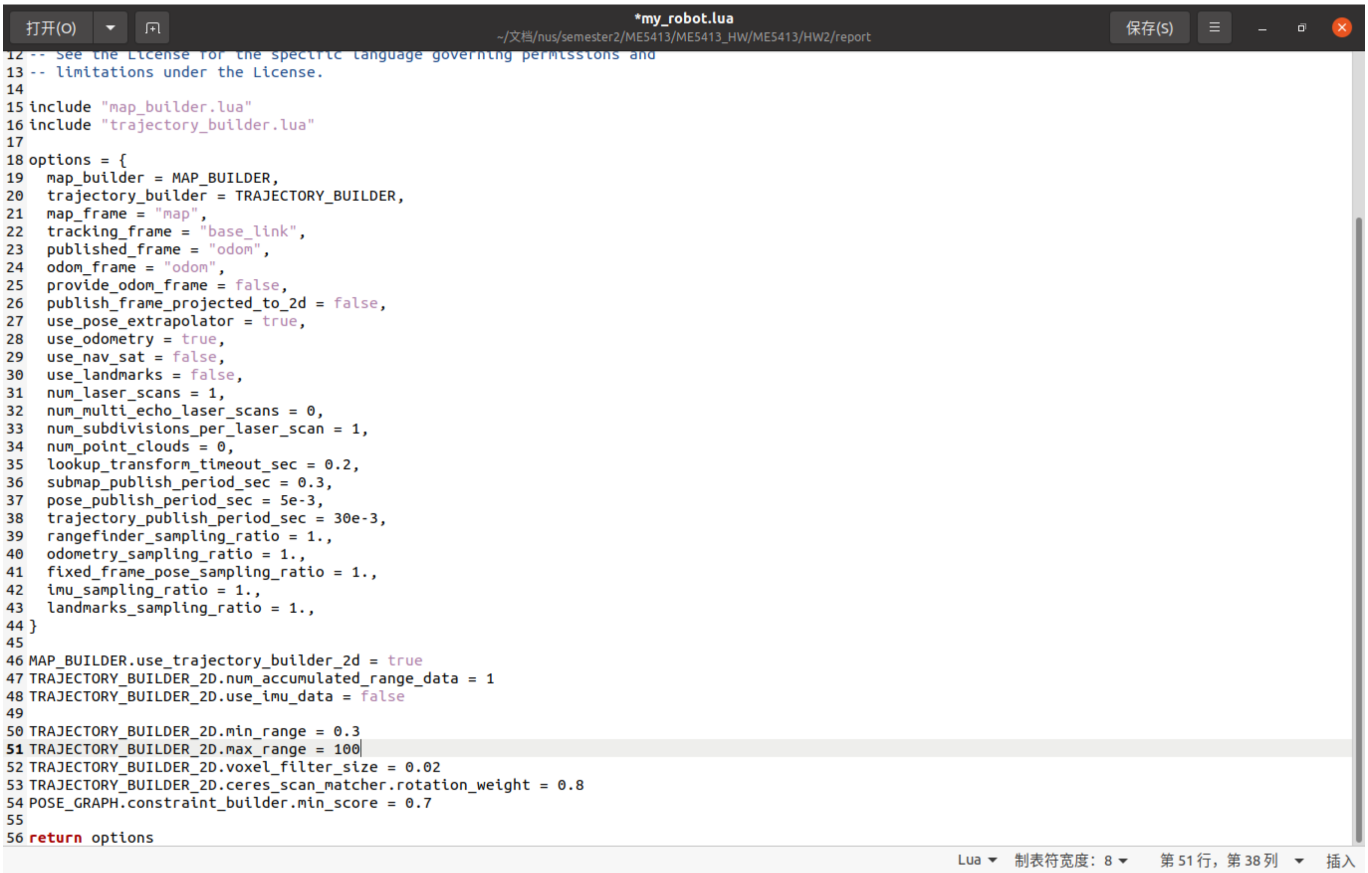
3.3 Bonus task

References

[1]nuslde (2023) aloam_lidar_odom_result_generate [Source Code]

A Appendix

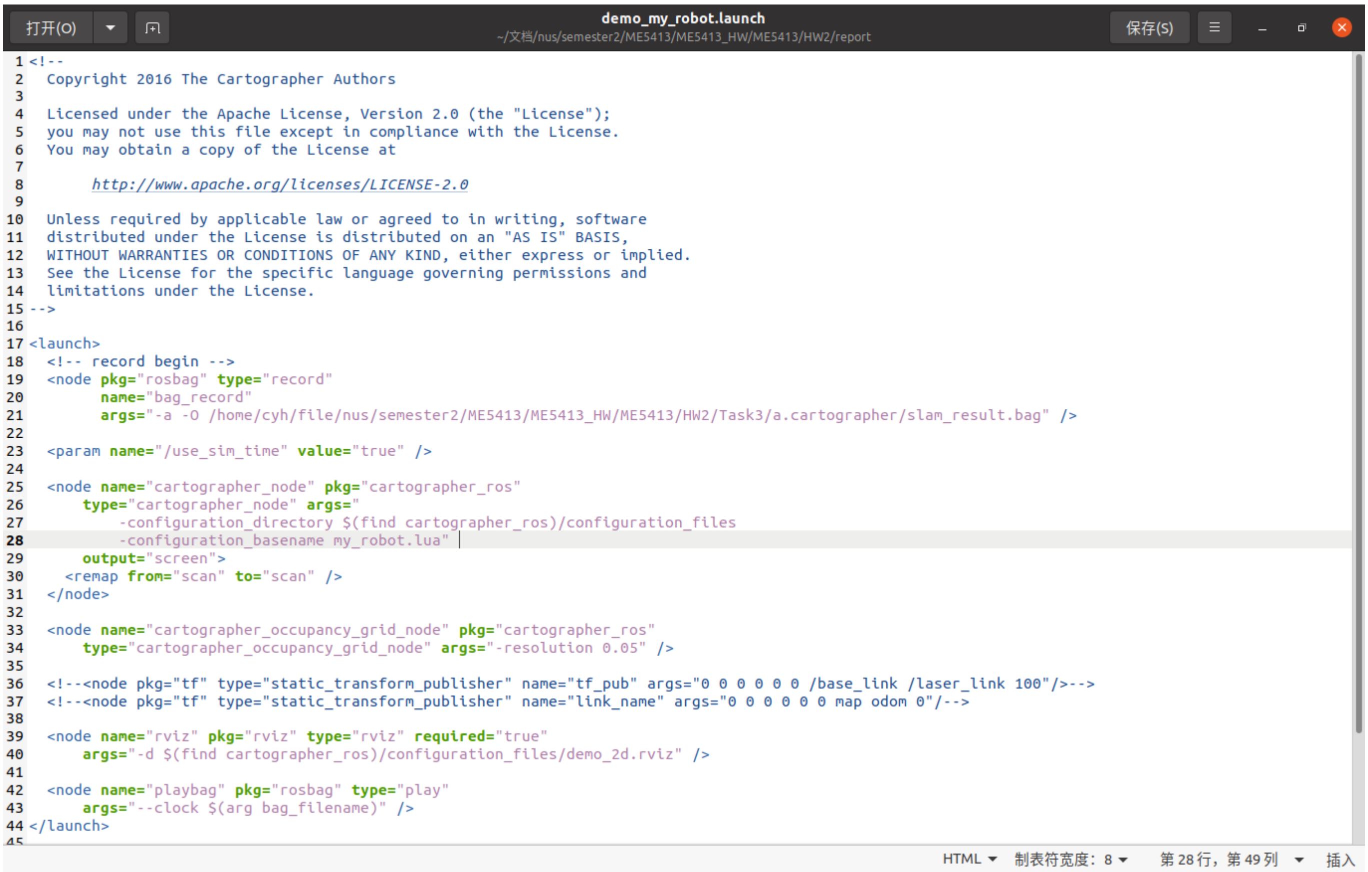
my_robot.lua



```lua
-- See the License for the specific language governing permissions and
-- limitations under the License.

include "map_builder.lua"
include "trajectory_builder.lua"

options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
  tracking_frame = "base_link",
  published_frame = "odom",
  odom_frame = "odom",
  provide_odom_frame = false,
  publish_frame_projected_to_2d = false,
  use_pose_extrapolator = true,
  use_odometry = true,
  use_nav_sat = false,
  use_landmarks = false,
  num_laser_scans = 1,
  num_multi_echo_laser_scans = 0,
  num_subdivisions_per_laser_scan = 1,
  num_point_clouds = 0,
  lookup_transform_timeout_sec = 0.2,
  submap_publish_period_sec = 0.3,
  pose_publish_period_sec = 5e-3,
  trajectory_publish_period_sec = 30e-3,
  rangefinder_sampling_ratio = 1.,
  odometry_sampling_ratio = 1.,
  fixed_frame_pose_sampling_ratio = 1.,
  imu_sampling_ratio = 1.,
  landmarks_sampling_ratio = 1.,
}

MAP_BUILDER.use_trajectory_builder_2d = true
TRAJECTORY_BUILDER_2D.num_accumulated_range_data = 1
TRAJECTORY_BUILDER_2D.use_imu_data = false

TRAJECTORY_BUILDER_2D.min_range = 0.3
TRAJECTORY_BUILDER_2D.max_range = 100
TRAJECTORY_BUILDER_2D.voxel_filter_size = 0.02
TRAJECTORY_BUILDER_2D.ceres_scan_matcher.rotation_weight = 0.8
POSE_GRAPH.constraint_builder.min_score = 0.7

return options
```

Figure 9: my_robot.lua

demo_my_robot.launch



```
<!--
  Copyright 2016 The Cartographer Authors

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->

<launch>
  <!-- record begin -->
  <node pkg="rosbag" type="record"
        name="bag_record"
        args="-a -O /home/cyh/file/nus/semester2/ME5413/ME5413_HW/ME5413/HW2/Task3/a.cartographer/slam_result.bag" />

  <param name="/use_sim_time" value="true" />

  <node name="cartographer_node" pkg="cartographer_ros"
      type="cartographer_node" args="
          -configuration_directory $(find cartographer_ros)/configuration_files
          -configuration_basename my_robot.lua"
      output="screen">
    <remap from="scan" to="scan" />
  </node>

  <node name="cartographer_occupancy_grid_node" pkg="cartographer_ros"
      type="cartographer_occupancy_grid_node" args="-resolution 0.05" />

  <!--<node pkg="tf" type="static_transform_publisher" name="tf_pub" args="0 0 0 0 0 0 /base_link /laser_link 100"/>-->
  <!--<node pkg="tf" type="static_transform_publisher" name="link_name" args="0 0 0 0 0 0 map odom 0"/-->

  <node name="rviz" pkg="rviz" type="rviz" required="true"
      args="-d $(find cartographer_ros)/configuration_files/demo_2d.rviz" />

  <node name="playbag" pkg="rosbag" type="play"
      args="--clock $(arg bag_filename)" />
</launch>
```


Figure 10: demo_my_robot.launch