Cartographer Map Building

Note: The virtual machine needs to be in the same LAN as the car, and the ROS_DOMAIN_ID needs to be consistent. You can check [Must Read Before Use] to set the IP and ROS_DOMAIN_ID on the board.

1. Introduction to Cartographer

Cartographer is a 2D and 3D SLAM (simultaneous localization and mapping) library supported by the ROS system, which is open sourced by Google. The mapping algorithm is based on the method of graph optimization (multi-threaded backend optimization, problem optimization built by cere). It can combine data from multiple sensors (such as LIDAR, IMU and camera), synchronously calculate the position of the sensor and draw the environment around the sensor.

The source code of cartographer mainly includes three parts: cartographer, cartographer_ros and ceres-solver (backend optimization).

Cartographer uses the mainstream SLAM framework, which is a three-stage feature extraction, closed-loop detection, and backend optimization. A certain number of LaserScans form a submap, and a series of submaps form a global map. The short-term cumulative error of using LaserScan to build submaps is not large, but the long-term process of using submaps to build a global map will have a large cumulative error, so closed-loop detection is needed to correct the position of these submaps. The basic unit of closed-loop detection is submap, and closed-loop detection uses scan_match strategy.

The focus of cartographer is the creation of submaps that integrate multi-sensor data (odometry, IMU, LaserScan, etc.) and the implementation of scan_match strategy for closed-loop detection.

cartographer_ros runs under ROS and can receive various sensor data in the form of ROS messages. After processing, it is published in the form of messages for easy debugging and visualization.

2. Program function description

The car connects to the agent and runs the program. The map building interface will be displayed in rviz. Use the keyboard or handle to control the movement of the car until the map is built. Then run the command to save the map.

```
Note: Before running the program, the car needs to be restarted in a stable
standing posture to ensure that all sensors are reset
```

3. Start and connect the agent

Take the supporting virtual machine as an example, enter the following command to start the agent,

```
sudo docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privileged --
net=host microros/micro-ros-agent:humble udp4 --port 8899 -v4
```

```
yahboom@yahboom-VM:~$ sudo docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm
 --privileged --net=host microros/micro-ros-agent:humble udp4 --port 8899 -v4
[1735179211.772044] info     | UDPv4AgentLinux.cpp | init                     |
running...              | port: 8899
[1735179211.772581] info     | Root.cpp           | set_verbose_level        | l
ogger setup             | verbose_level: 4
```

Then, turn on the car switch and wait for the car to connect to the proxy. The connection is successful as shown in the figure below.

```
[1735179211.772044] info     | UDPv4AgentLinux.cpp | init                     | running...             | port: 8899
[1735179211.772581] info     | Root.cpp           | set_verbose_level        | logger setup           | verbose_level: 4
[1735179325.739277] info     | Root.cpp           | create_client            | create                 | client_key: 0x0E5C3397, sess
ion_id: 0x81
[1735179325.739348] info     | SessionManager.hpp | establish_session        | session established    | client_key: 0x0E5C3397, addr
ess: 192.168.2.102:49954
[1735179325.971694] info     | ProxyClient.cpp    | create_participant       | participant created    | client_key: 0x0E5C3397, part
icipant_id: 0x000(1)
[1735179326.046043] info     | ProxyClient.cpp    | create_topic             | topic created          | client_key: 0x0E5C3397, topi
c_id: 0x000(2), participant_id: 0x000(1)
[1735179326.159287] info     | ProxyClient.cpp    | create_publisher         | publisher created      | client_key: 0x0E5C3397, publ
isher_id: 0x000(3), participant_id: 0x000(1)
[1735179326.176344] info     | ProxyClient.cpp    | create_datawriter        | datawriter created     | client_key: 0x0E5C3397, data
writer_id: 0x000(5), publisher_id: 0x000(3)
[1735179326.184566] info     | ProxyClient.cpp    | create_topic             | topic created          | client_key: 0x0E5C3397, topi
c_id: 0x001(2), participant_id: 0x000(1)
[1735179326.263761] info     | ProxyClient.cpp    | create_publisher         | publisher created      | client_key: 0x0E5C3397, publ
isher_id: 0x001(3), participant_id: 0x000(1)
[1735179326.276817] info     | ProxyClient.cpp    | create_datawriter        | datawriter created     | client_key: 0x0E5C3397, data
writer_id: 0x001(5), publisher_id: 0x001(3)
[1735179326.285996] info     | ProxyClient.cpp    | create_topic             | topic created          | client_key: 0x0E5C3397, topi
c_id: 0x002(2), participant_id: 0x000(1)
[1735179326.345401] info     | ProxyClient.cpp    | create_publisher         | publisher created      | client_key: 0x0E5C3397, publ
isher_id: 0x002(3), participant_id: 0x000(1)
[1735179326.365619] info     | ProxyClient.cpp    | create_datawriter        | datawriter created     | client_key: 0x0E5C3397, data
writer_id: 0x002(5), publisher_id: 0x002(3)
[1735179326.372863] info     | ProxyClient.cpp    | create_topic             | topic created          | client_key: 0x0E5C3397, topi
c_id: 0x003(2), participant_id: 0x000(1)
[1735179326.379913] info     | ProxyClient.cpp    | create_publisher         | publisher created      | client_key: 0x0E5C3397, publ
isher_id: 0x003(3), participant_id: 0x000(1)
[1735179326.448851] info     | ProxyClient.cpp    | create_datawriter        | datawriter created     | client_key: 0x0E5C3397, data
writer_id: 0x003(5), publisher_id: 0x003(3)
[1735179326.548363] info     | ProxyClient.cpp    | create_topic             | topic created          | client_key: 0x0E5C3397, topi
c_id: 0x004(2), participant_id: 0x000(1)
[1735179326.565153] info     | ProxyClient.cpp    | create_subscriber        | subscriber created     | client_key: 0x0E5C3397, subs
criber_id: 0x000(4), participant_id: 0x000(1)
[1735179326.574254] info     | ProxyClient.cpp    | create_datareader        | datareader created     | client_key: 0x0E5C3397, data
reader_id: 0x000(6), subscriber_id: 0x000(4)
```

4. Start the program

4.1 Run the command

Take the matching virtual machine as an example, input in the terminal,

```
ros2 launch yahboomcar_bringup yahboomcar_bringup_launch.py mode:=slam
```

```
#Parameter description, the speed of the car will be adjusted, this is the map building mode
mode:=slam
```

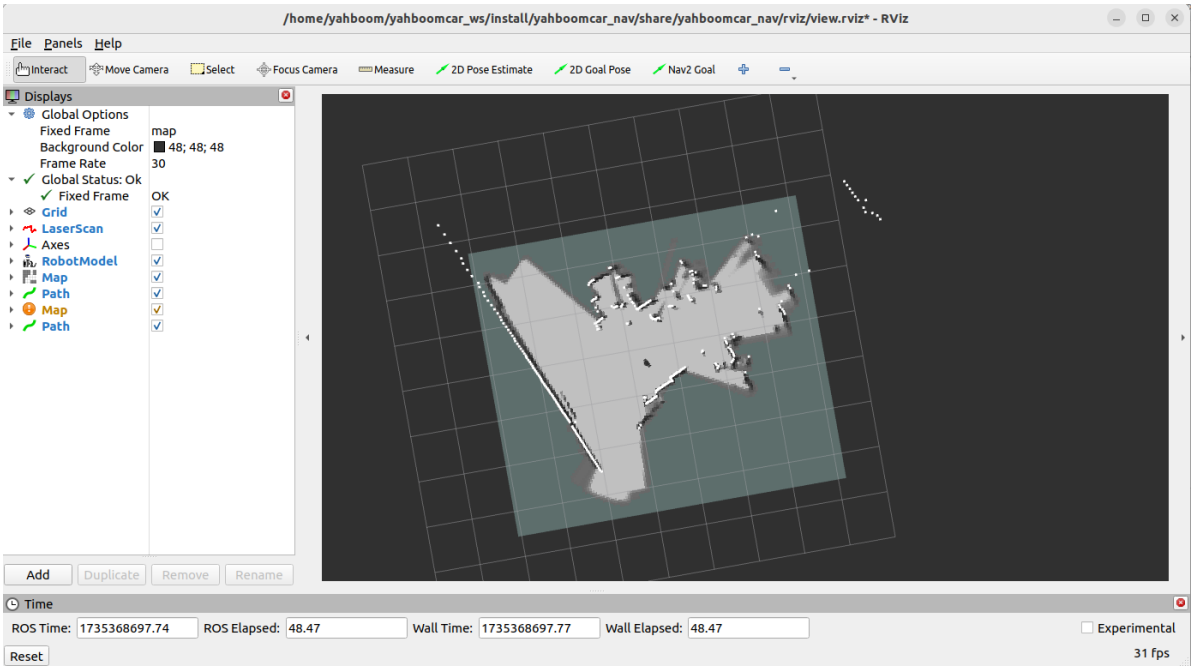
First, start the car to process the underlying data program

```
yahboom@yahboom-VM: $ ros2 launch yahboomcar_bringup yahboomcar_bringup_launch.py mode:=slam
[INFO] [launch]: All log files can be found below /home/yahboom/.ros/log/2024-12-28-14-23-18-418155-yahboom-VM-37886
[INFO] [launch]: Default logging verbosity is set to INFO
--------------------robot_type = stm32v2--------------------
[INFO] [complementary_filter_node-1]: process started with pid [37891]
[INFO] [static_transform_publisher-2]: process started with pid [37893]
[INFO] [static_transform_publisher-3]: process started with pid [37895]
[INFO] [joint_state_publisher-4]: process started with pid [37897]
[INFO] [robot_state_publisher-5]: process started with pid [37899]
[INFO] [static_transform_publisher-6]: process started with pid [37901]
[INFO] [cmdvel2bl-7]: process started with pid [37919]
[INFO] [ekf_node-8]: process started with pid [37921]
[static_transform_publisher-2] [WARN] [1735366998.708266419] []: Old-style arguments are deprecated; see --help for new-style arguments
[static_transform_publisher-3] [WARN] [1735366998.708218131] []: Old-style arguments are deprecated; see --help for new-style arguments
[static_transform_publisher-3] [INFO] [1735366998.736749065] [base_link_to_base_laser]: Spinning until stopped - publishing transform
[static_transform_publisher-3] translation: ('0.000000', '0.000000', '0.138000')
[static_transform_publisher-3] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-3] from 'base_link' to 'laser_frame'
[static_transform_publisher-2] [INFO] [1735366998.755642812] [base_link_to_base_imu]: Spinning until stopped - publishing transform
[static_transform_publisher-2] translation: ('0.000000', '0.016325', '0.080691')
[static_transform_publisher-2] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-2] from 'base_link' to 'imu_frame'
[static_transform_publisher-6] [WARN] [1735366998.763312566] []: Old-style arguments are deprecated; see --help for new-style arguments
[complementary_filter_node-1] [INFO] [1735366998.768034422] [complementary_filter_gain_node]: Starting ComplementaryFilterROS
[static_transform_publisher-6] [INFO] [1735366998.839170830] [static_transform_publisher_9AYR1PTo0c5wx4eI]: Spinning until stopped - publishing transform
[static_transform_publisher-6] translation: ('0.000000', '0.000000', '0.033500')
[static_transform_publisher-6] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-6] from 'base_footprint' to 'base_link'
[robot_state_publisher-5] [WARN] [1735366998.852874409] [kdl_parser]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia.  As a workaround,
 you can add an extra dummy link to your URDF.
[robot_state_publisher-5] [INFO] [1735366998.852986860] [robot_state_publisher]: got segment Camera_Link
[robot_state_publisher-5] [INFO] [1735366998.853068472] [robot_state_publisher]: got segment LWheel_Link
[robot_state_publisher-5] [INFO] [1735366998.853077800] [robot_state_publisher]: got segment RWheel_Link
[robot_state_publisher-5] [INFO] [1735366998.853083051] [robot_state_publisher]: got segment base_link
[cmdvel2bl-7] [INFO] [1735366999.117587672] [cmd_vel_scaler]: mode slam...
[joint_state_publisher-4] [INFO] [1735366999.176700145] [joint_state_publisher]: Waiting for robot_description to be published on the robot_description topic...
```

Next, run the map building node, input the terminal, and wait for rviz to open automatically,

```
ros2 launch yahboomcar_nav map_cartographer_launch.py use_rviz:=true
```

```
#Parameter description, select whether to open rviz, true is open, false is not open
use_rviz:=true
```



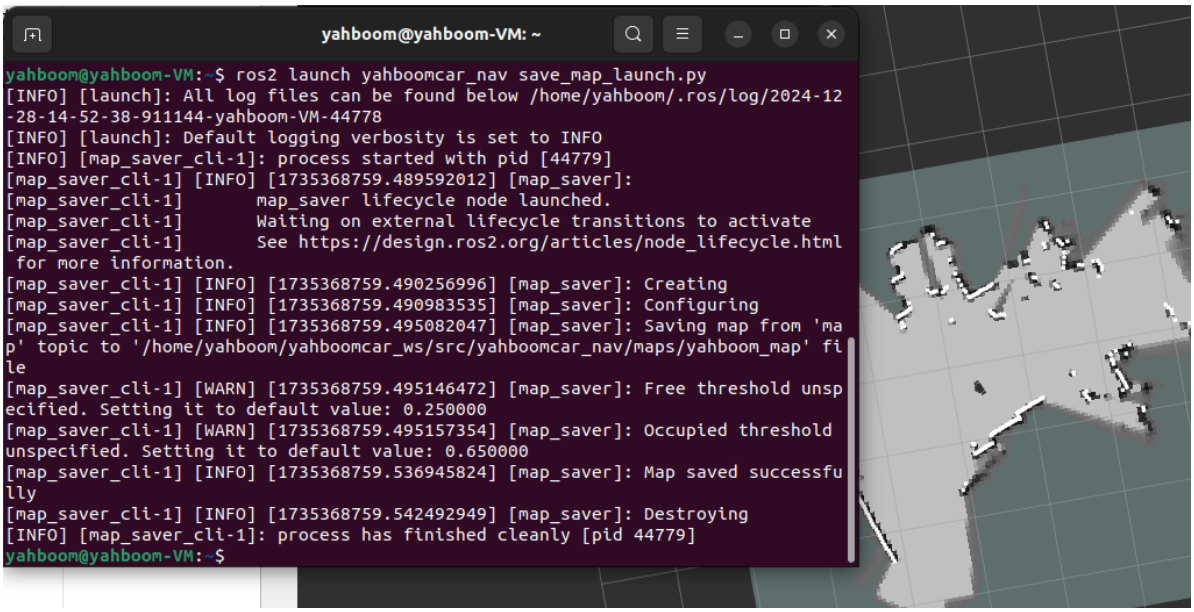
Then run the handle control or keyboard control, choose one, terminal input,

```
#Keyboard control
ros2 run yahboomcar_ctrl yahboom_keyboard
#Handle control
ros2 launch yahboomcar_ctrl yahboomcar_joy_launch.py
```

Then control the car and slowly walk through the area where the map needs to be built. After the map is built, enter the following command to save the map, terminal input,

4.1.1 Save yaml map

```
ros2 launch yahboomcar_nav save_map_launch.py
```



A map named yahboom_map will be saved. This map is saved in,

Take the matching virtual machine as an example code path:

```
/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/maps
```

Two files will be generated, one is yahboom_map.pgm and the other is yahboom_map.yaml. Check the contents of yaml,

```
image: yahboom_map.pgm
mode: trinary
resolution: 0.05
origin: [-10, -10, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

- image: the image representing the map, i.e. yahboom_map.pgm
- mode: this property can be one of trinary, scale or raw, depending on the selected mode, trinary mode is the default mode
- resolution: the resolution of the map, meters/pixels
- 2D pose (x,y,yaw) of the lower left corner of the map, where yaw is rotated counterclockwise (yaw=0 means no rotation). Currently many parts of the system ignore yaw values.
- negate: whether to invert the meaning of white/black and free/occupied (the interpretation of the threshold is not affected)
- occupied_thresh: pixels with an occupied probability greater than this threshold are considered fully occupied.
- free_thresh: pixels with an occupied probability less than this threshold are considered fully free.

4.1.2 Save pbstream map

Then enter the following command to stop building the map,

```
ros2 service call /finish_trajectory cartographer_ros_msgs/srv/FinishTrajectory
"{trajectory_id: 0}"
```

Then enter the following command to save the pbstream file

```
ros2 service call /write_state cartographer_ros_msgs/srv/WriteState "{filename:
'/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/maps/yahboom_map.pbstream'}"
```

The path of the filename parameter is the path where the pbstream file of the map is saved.

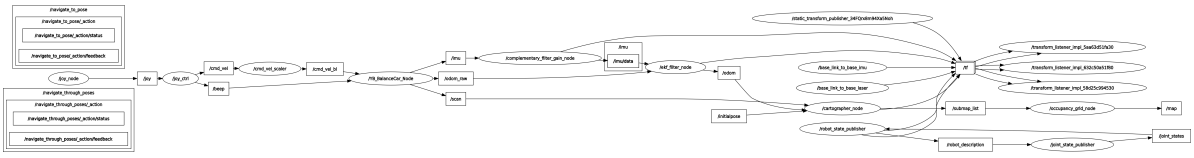
Finally, enter the following command to convert the pbstream file to a pgm file.

```
ros2 run cartographer_ros cartographer_pbstream_to_ros_map -
map_filestem=/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/maps/yahboom_map -
pbstream_filename=/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/maps/yahboom_ma
p.pbstream -resolution=0.05
```

5. View the node communication graph

Terminal input,

```
ros2 run rqt_graph rqt_graph
```

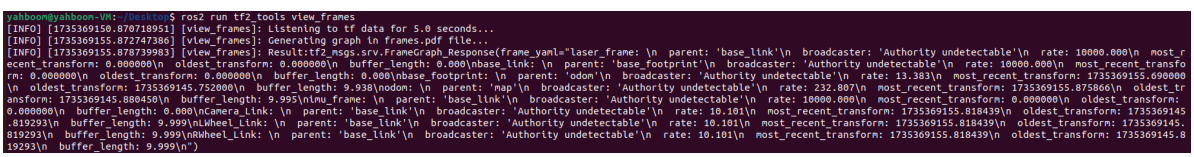


If it is not displayed at the beginning, select [Nodes/Topics(all)], and then click the refresh button in the upper left corner.

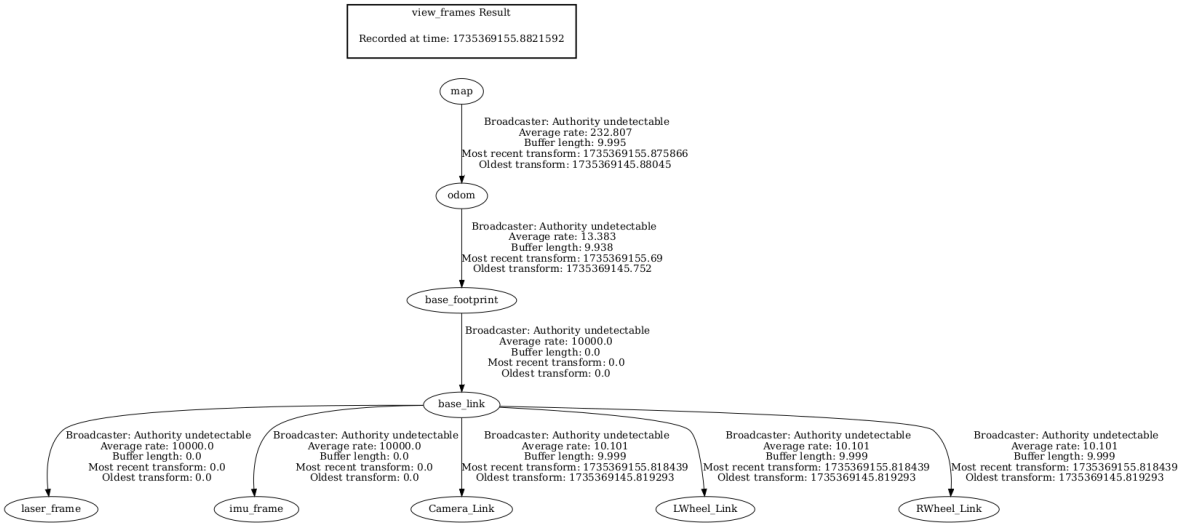
6. View TF tree

Terminal input,

```
ros2 run tf2_tools view_frames
```



After running, two files will be generated in the terminal directory, namely .gv and .pdf files, and the pdf file is the TF tree.



7. Code analysis

Taking the virtual machine as an example, here we only explain the map_cartographer_launch.py file for building the map. The file path is,

```
/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/launch
```

map_cartographer_launch.py

```
import os
from ament_index_python.packages import
get_package_share_directory,get_package_share_path
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription,DeclareLaunchArgument
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch_ros.actions import Node
from launch.substitutions import LaunchConfiguration

def generate_launch_description():
    package_path = get_package_share_path('yahboomcar_nav')
    package_launch_path
=os.path.join(get_package_share_directory('yahboomcar_nav'), 'launch')
    default_rviz_config_path = package_path / 'rviz/view.rviz'
    rviz_arg = DeclareLaunchArgument(name='rvizconfig',
default_value=str(default_rviz_config_path),
                                     description='Absolute path to rviz config
file')

    cartographer_launch =
IncludeLaunchDescription(PythonLaunchDescriptionSource(
        [package_launch_path, '/cartographer_launch.py'])
    )
    base_link_to_laser_tf_node = Node(
     package='tf2_ros',
     executable='static_transform_publisher',
     name='base_link_to_base_laser',
     arguments=['0.0', '0.0', '0.138', '0', '0', '0', 'base_link',
'laser_frame']
    )
```

```
    rviz_node = Node(
        package='rviz2',
        executable='rviz2',
        name='rviz2',
        output='screen',
        arguments=['-d', LaunchConfiguration('rvizconfig')],
    )

    return
LaunchDescription([rviz_arg,rviz_node,cartographer_launch,base_link_to_laser_tf_
node])
```

Here, a launch file - cartographer_launch and a node that publishes static transformations - base_link_to_laser_tf_node are run.

Taking the virtual machine as an example, take a closer look at cartographer_launch. The file is located at,

```
/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/launch
```

cartographer_launch.py,

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch_ros.actions import Node
from launch.substitutions import LaunchConfiguration
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir


def generate_launch_description():
    use_sim_time = LaunchConfiguration('use_sim_time', default='false')
    package_path = get_package_share_directory('yahboomcar_nav')
    configuration_directory = LaunchConfiguration('configuration_directory',
default=os.path.join(
                                                  package_path, 'params'))
    configuration_basename = LaunchConfiguration('configuration_basename',
default='lds_2d.lua')

    resolution = LaunchConfiguration('resolution', default='0.05')
    publish_period_sec = LaunchConfiguration(
        'publish_period_sec', default='1.0')

    return LaunchDescription([
        DeclareLaunchArgument(
            'configuration_directory',
            default_value=configuration_directory,
            description='Full path to config file to load'),
        DeclareLaunchArgument(
            'configuration_basename',
```

```
            default_value=configuration_basename,
            description='Name of lua file for cartographer'),
        DeclareLaunchArgument(
            'use_sim_time',
            default_value='false',
            description='Use simulation (Gazebo) clock if true'),

        Node(
            package='cartographer_ros',
            executable='cartographer_node',
            name='cartographer_node',
            output='screen',
            parameters=[{'use_sim_time': use_sim_time}],
            arguments=['-configuration_directory', configuration_directory,
                       '-configuration_basename', configuration_basename],
            remappings=[('/odom','/odom')]
                       ),

        DeclareLaunchArgument(
            'resolution',
            default_value=resolution,
            description='Resolution of a grid cell in the published occupancy
grid'),

        DeclareLaunchArgument(
            'publish_period_sec',
            default_value=publish_period_sec,
            description='OccupancyGrid publishing period'),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
                [ThisLaunchFileDir(), '/occupancy_grid_launch.py']),
            launch_arguments={'use_sim_time': use_sim_time, 'resolution':
resolution,
                              'publish_period_sec': publish_period_sec}.items(),
        ),
    ])
```

Here we mainly run the cartographer_node map building node and occupancy_grid_launch.py, and load the parameter configuration file.

The file is located in (taking the supporting virtual machine as an example),

```
/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/params
```

lds_2d.lua,

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

options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
```

```
  tracking_frame = "base_footprint",
  published_frame = "odom",
  odom_frame = "odom",
  provide_odom_frame = false,
  publish_frame_projected_to_2d = false,
  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.use_imu_data = false
TRAJECTORY_BUILDER_2D.min_range = 0.10
TRAJECTORY_BUILDER_2D.max_range = 3.5
TRAJECTORY_BUILDER_2D.missing_data_ray_length = 3.
TRAJECTORY_BUILDER_2D.use_online_correlative_scan_matching = true
TRAJECTORY_BUILDER_2D.motion_filter.max_angle_radians = math.rad(0.1)

POSE_GRAPH.constraint_builder.min_score = 0.65
POSE_GRAPH.constraint_builder.global_localization_min_score = 0.7

return options
```