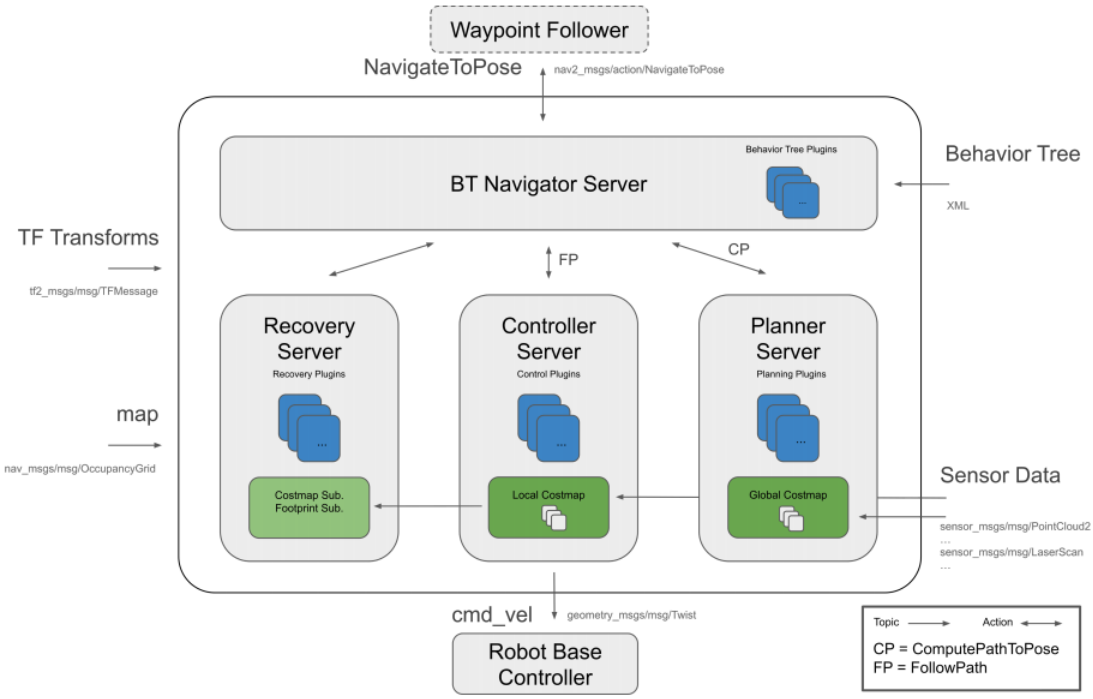
Navigation2 Navigation

1. Navigation2 Introduction

Navigation2 Overall Architecture



Navigation2 has the following tools:

- Tools for loading, providing and storing maps (Map Server)
- Tools for locating robots on maps (AMCL)
- Path planning tools for moving from point A to point B while avoiding obstacles (Nav2 Planner)
- Tools for controlling robots while following paths (Nav2 Controller)
- Tools for converting sensor data into cost map expressions in the robot world (Nav2 Costmap 2D)
- Tools for building complex robot behaviors using behavior trees (Nav2 Behavior Trees and BT Navigator)
- Tools for calculating recovery behaviors when failures occur (Nav2 Recoveries)
- Tools for following sequential waypoints (Nav2 Waypoint Follower)
- Tools and watchdogs for managing server lifecycles (Nav2 Lifecycle Manager)
- Plugins to enable user-defined algorithms and behaviors (Nav2 Core)

Navigation 2 (Nav 2) is the navigation framework included with ROS 2. Its purpose is to enable a mobile robot to move from point A to point B in a safe way. Therefore, Nav 2 can perform behaviors such as dynamic path planning, calculating motor speeds, avoiding obstacles, and recovering structures.

Nav 2 uses behavior trees (BT) to call modular servers to complete an action. Actions can be path calculations, control efforts, recovery, or other navigation-related actions. These actions are independent nodes that communicate with behavior trees (BT) through action servers.

Reference URL:

Navigation2 Document: https://navigation.ros.org/index.html

Navigation2 github: https://github.com/ros-planning/navigation2

Navigation2 Corresponding Paper: https://arxiv.org/pdf/2003.00368.pdf

Plugins provided by Navigation2: https://navigation.ros.org/plugins/index.html#plugins

2. Program Function Description

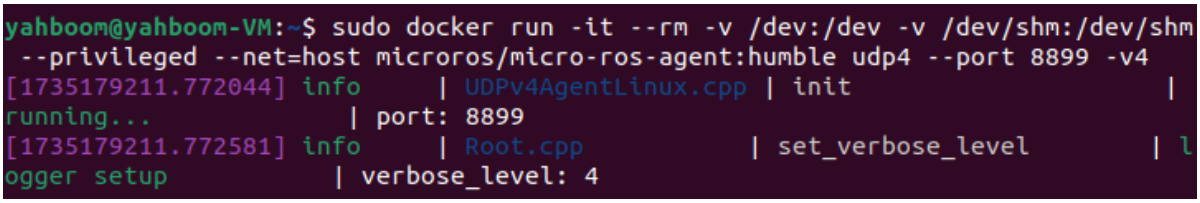
The car connects to the proxy, runs the program, and the map will be loaded in rviz. In the rviz interface, use the [2D Pose Estimate] tool to give the car an initial pose, and then use the [2D Goal Pose] tool to give the car a target point. The car will plan a path based on its own environment and move to the destination according to the planned path. If it encounters obstacles during the process, it will avoid obstacles by itself and stop after reaching the destination.

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

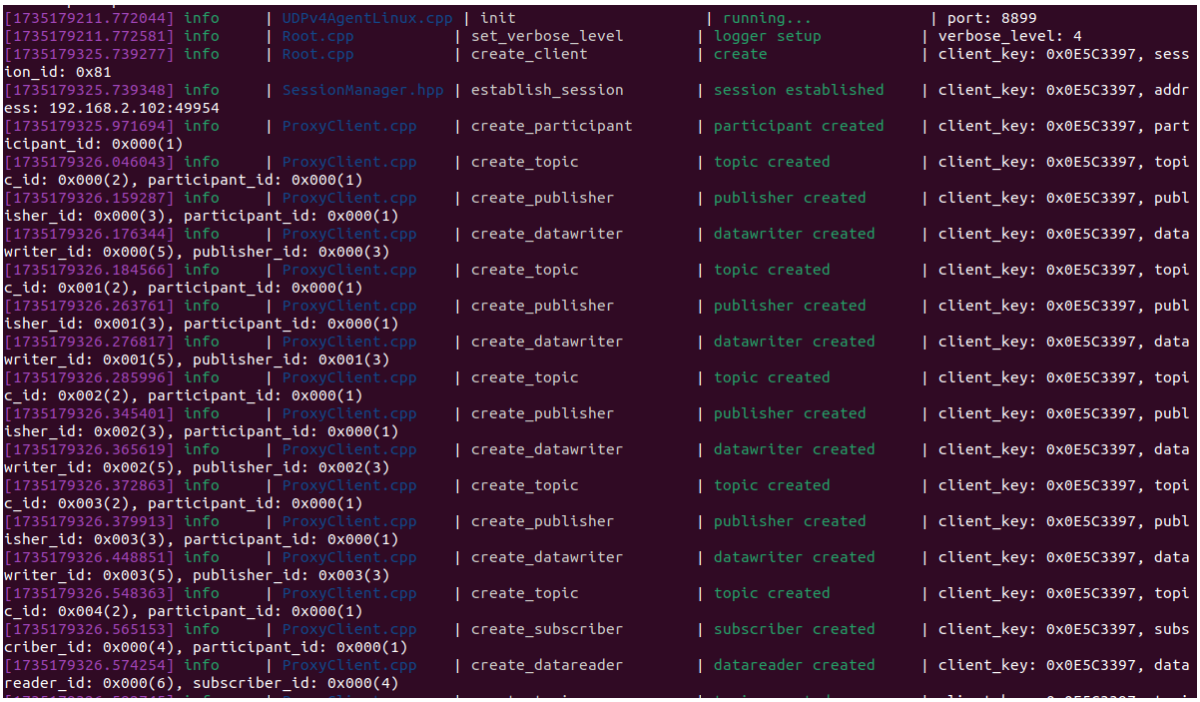
3. Start and connect the agent

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
```



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.



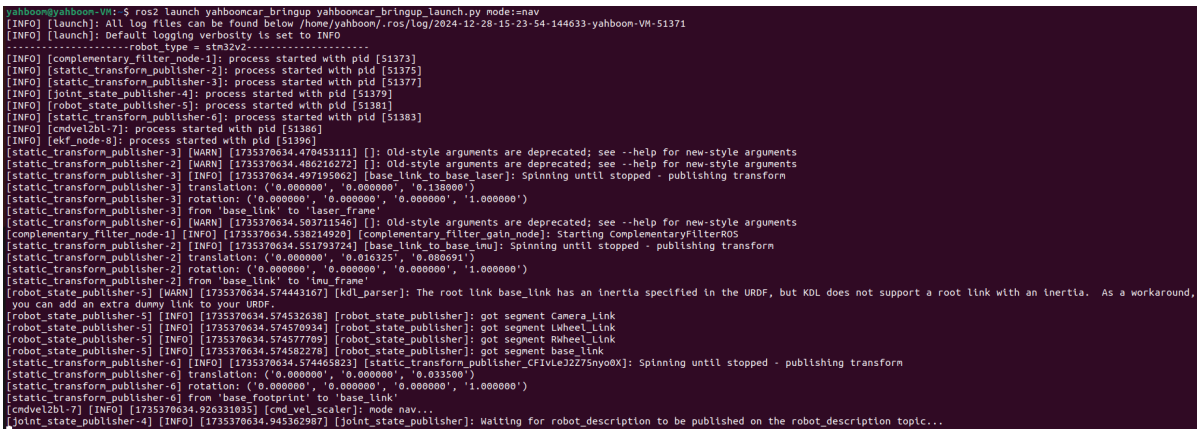
4. Start the program

4.1 Run the command

First, start the car to process the underlying data program. Take the matching virtual machine as an example. Enter the terminal,

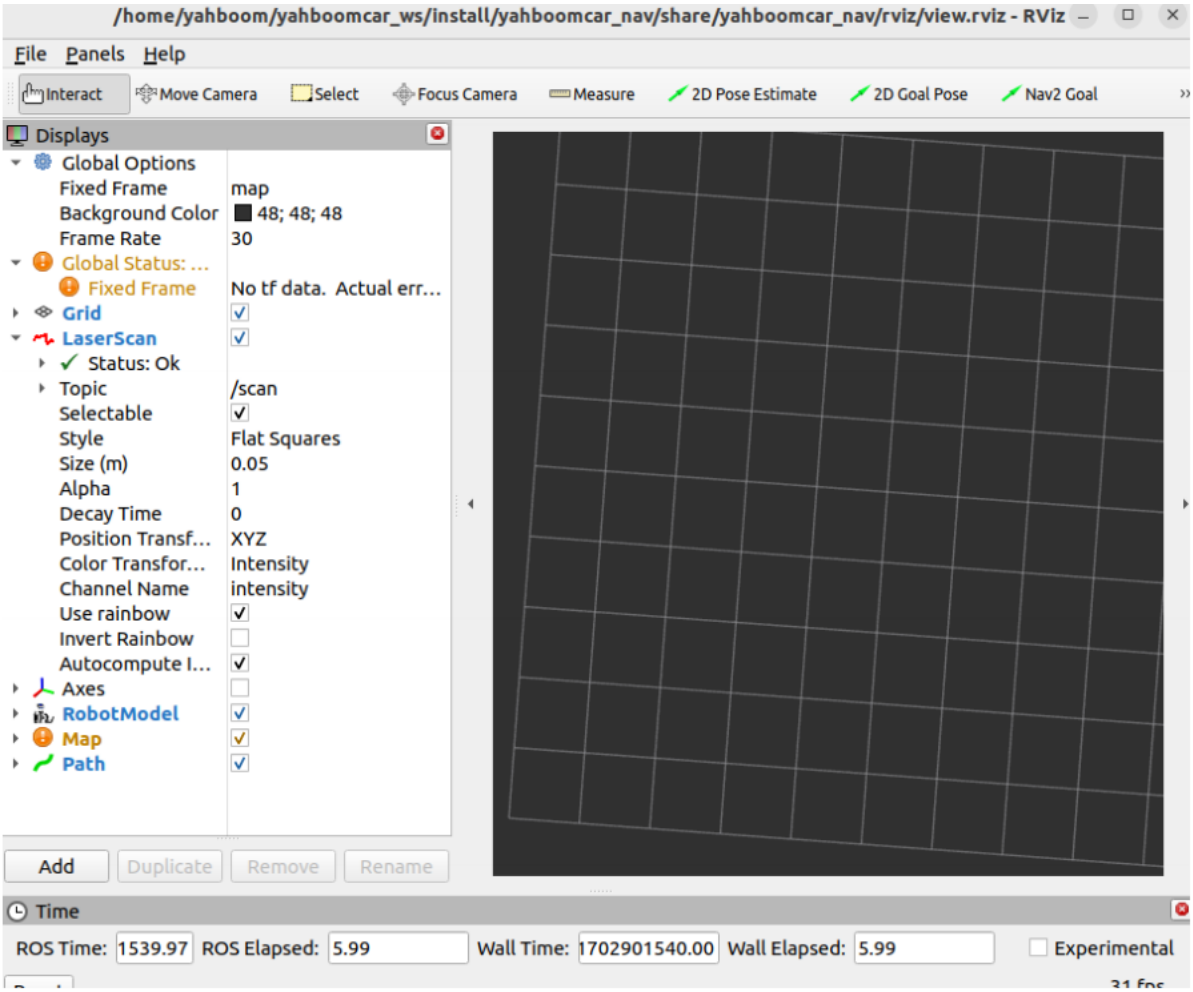
```
ros2 launch yahboomcar_bringup yahboomcar_bringup_launch.py mode:=nav
```

```
#Parameter description, the speed of the car will be adjusted, this is the navigation mode
mode:=nav
```



Then, start rviz, visualize navigation, and enter in the terminal

```
ros2 launch yahboomcar_nav display_launch.py
```

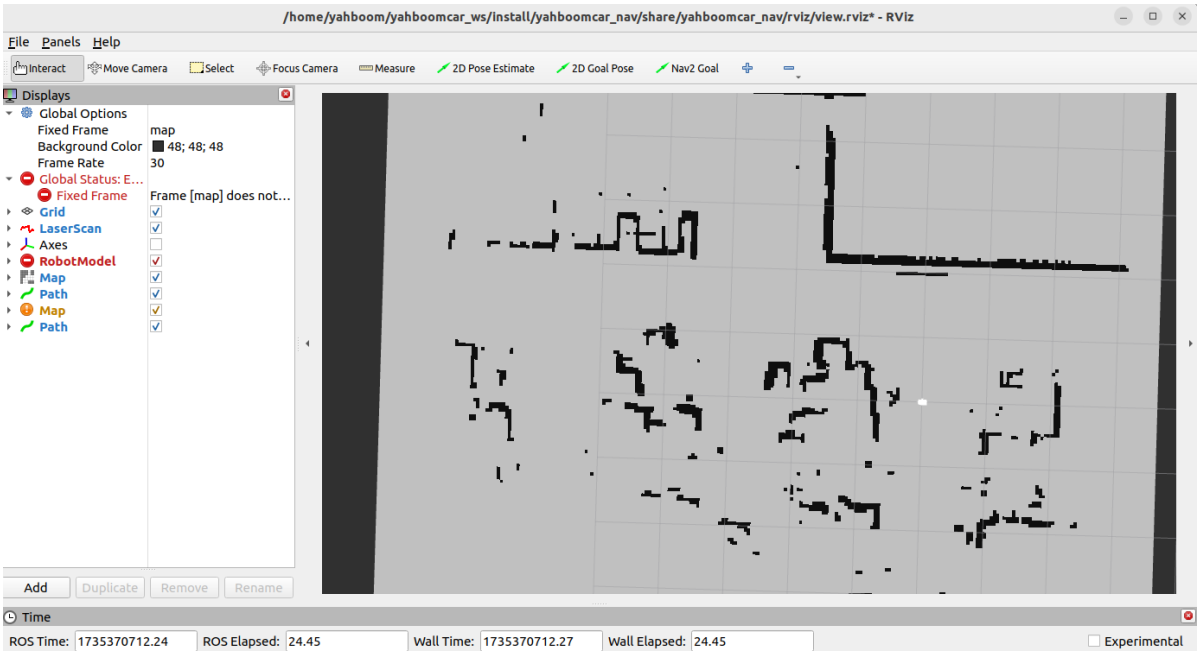


At this time, the map loading is not displayed, because the navigation program has not been started, so there is no map loading. Next, run the navigation node,

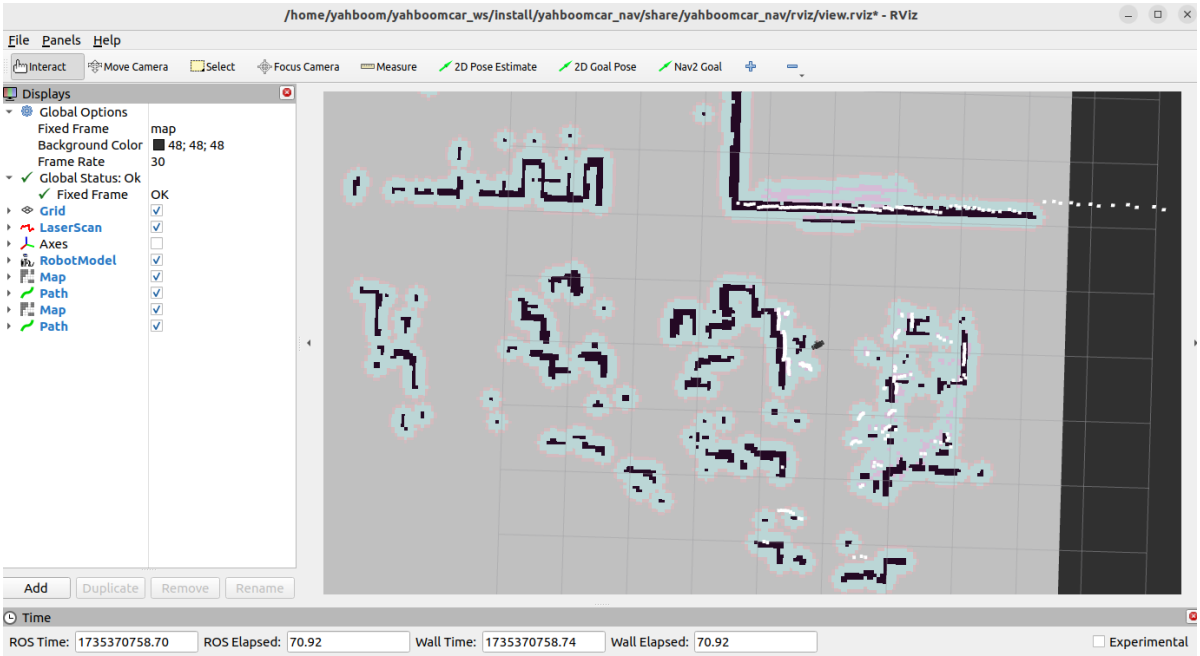
Take the virtual machine as an example, terminal input,

```
ros2 launch yahboomcar_nav navigation_dwb_launch.py
maps:=/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/maps/testaa.yaml
```

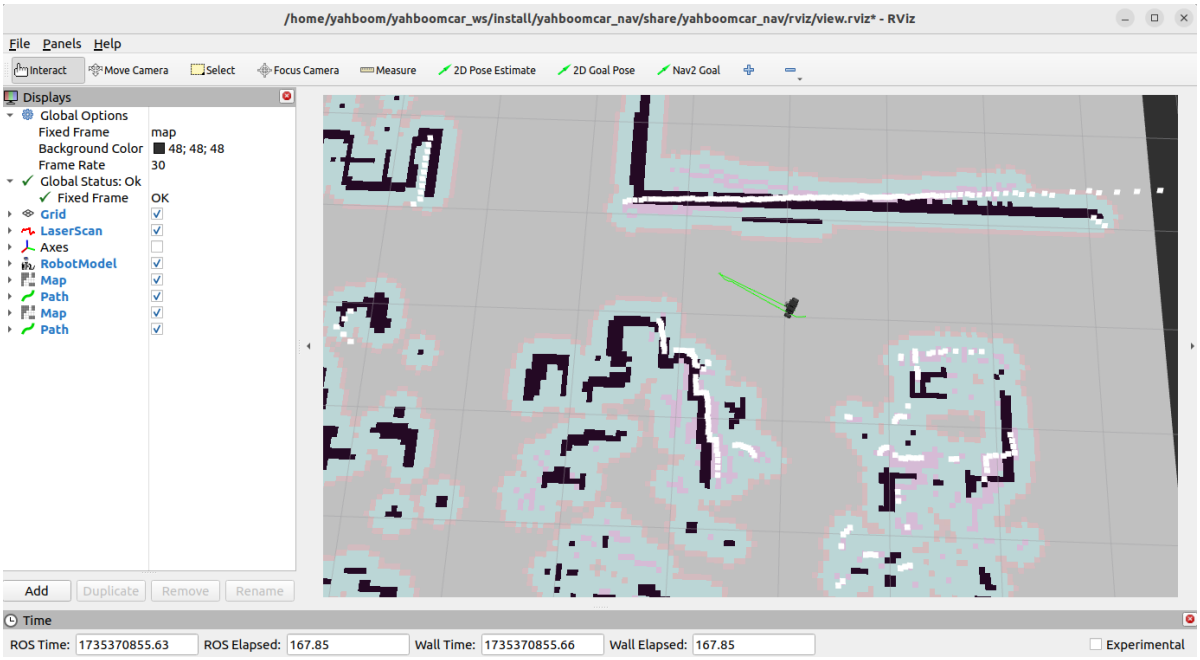
```
#Load map parameters: (can replace the target map)
maps:=/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/maps/testaa.yaml
#.pgm file must also be in the same path as .yaml
```



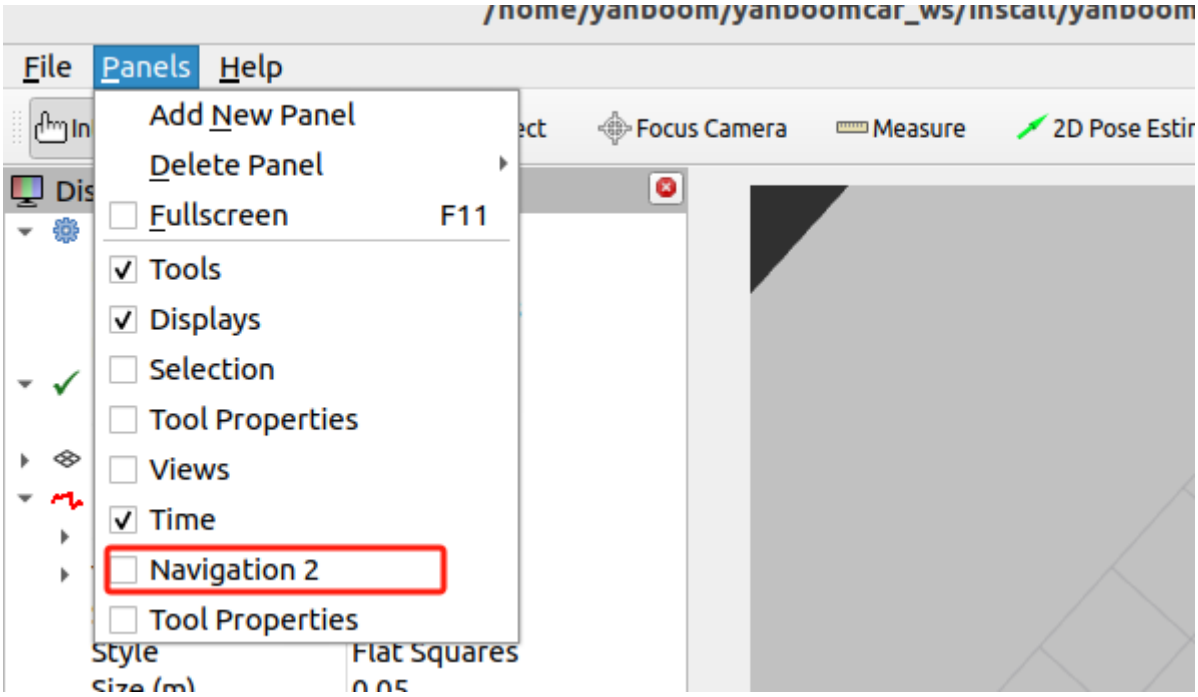
At this time, you can see that the map is loaded, and then we click [2D Pose Estimate】, set the initial pose for the car, and click and drag the mouse in rviz according to the position of the car in the actual environment, and the car model moves to the position we set. As shown in the figure below, if the area scanned by the radar roughly coincides with the actual obstacle, it means that the pose is accurate.



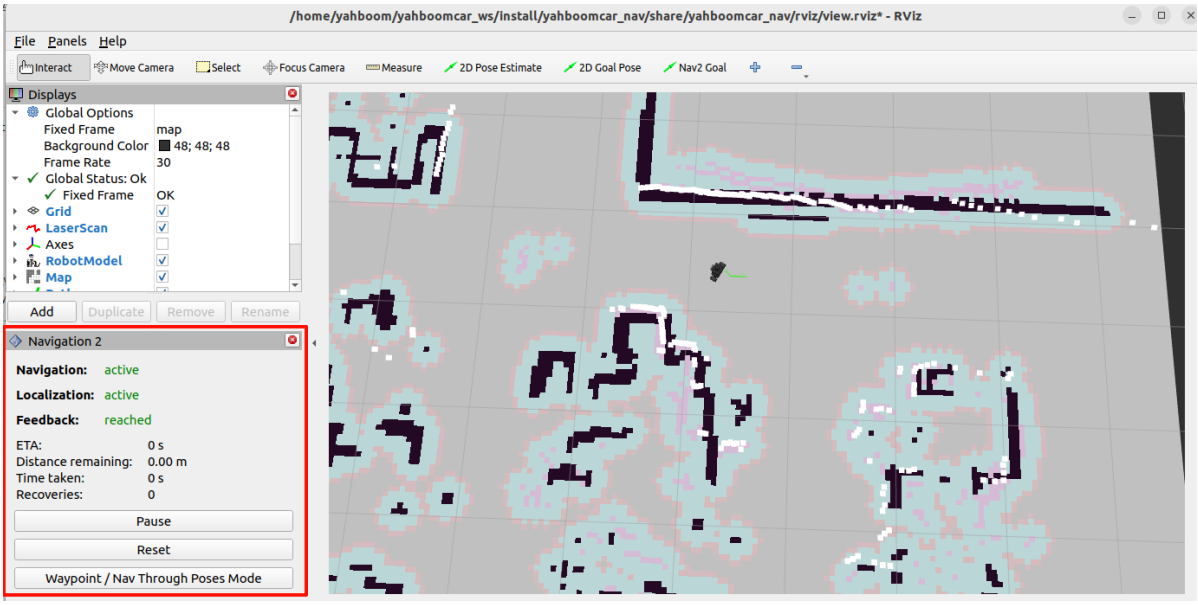
Single-point navigation, click the [2D Goal Pose] tool, and then select a target point in rviz. The car plans a path based on the surrounding situation and moves to the target point along the path.



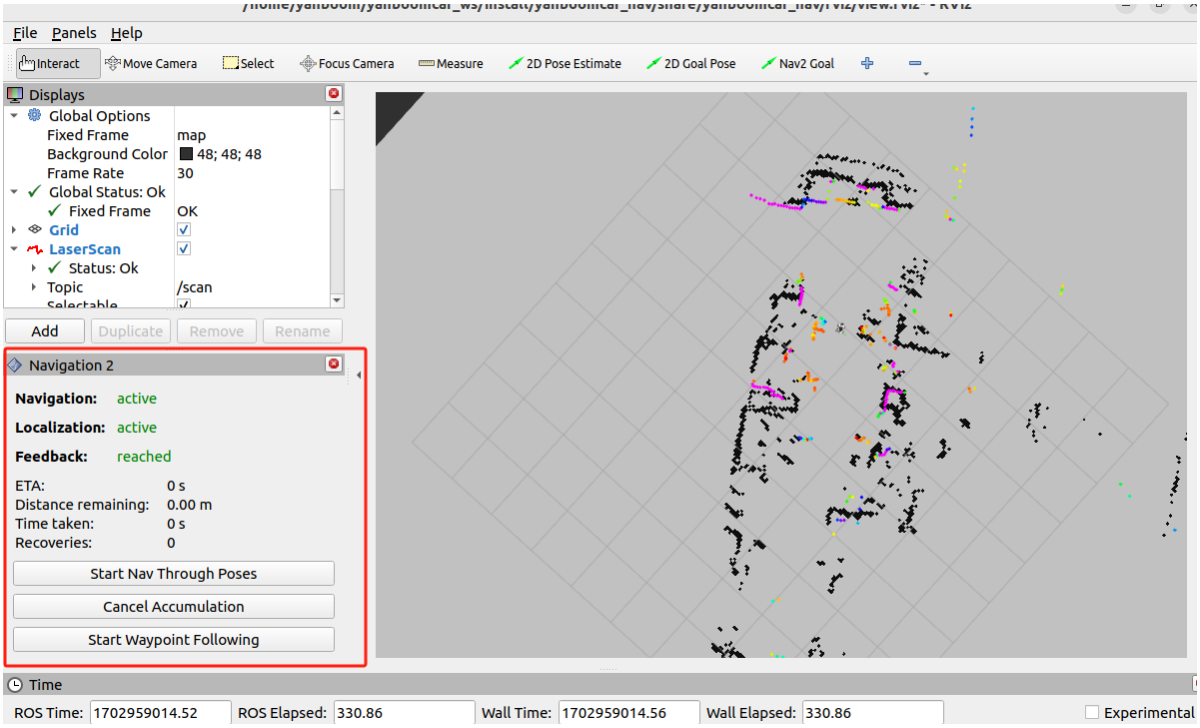
For multi-point navigation, you need to add the nav2 plug-in.



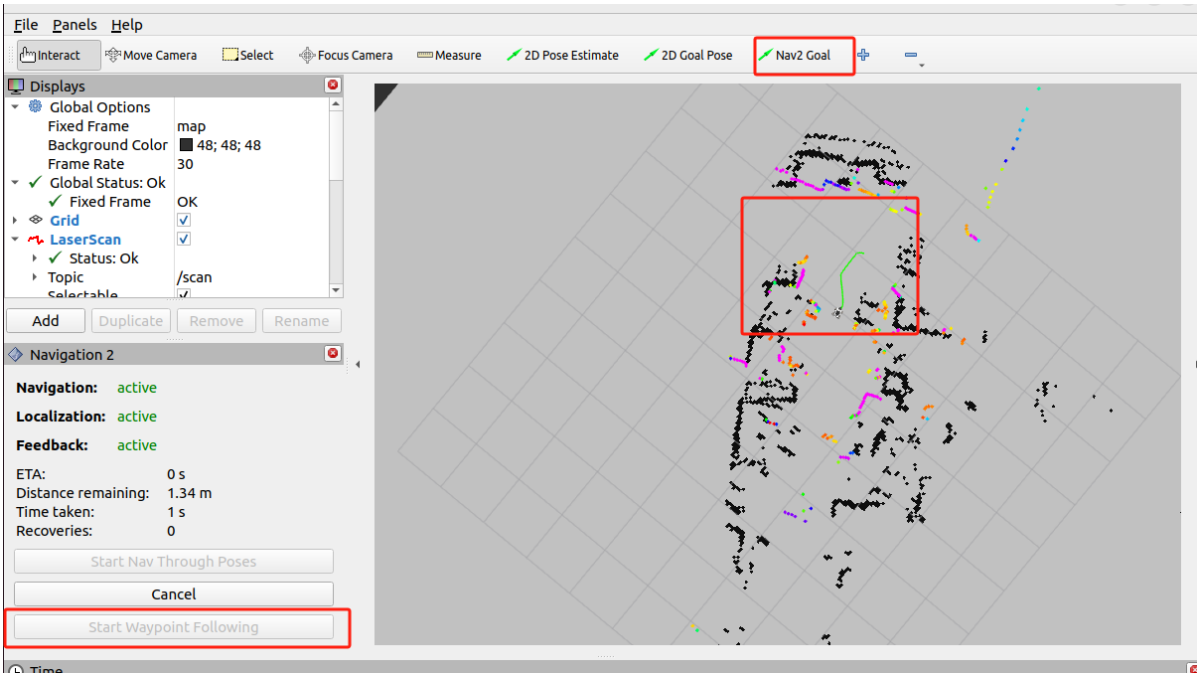
After adding, rviz displays as follows.



Then click [Waypoint/Nav Through Poses Mode],



Use the [Nav2 Goal】Give multiple arbitrary target points, then click 【Start Waypoint Following】 to start planning path navigation. The car will automatically go to the next point after reaching the target point according to the order of the selected points, without any operation. After reaching the last point, the car stops and waits for the next instruction.

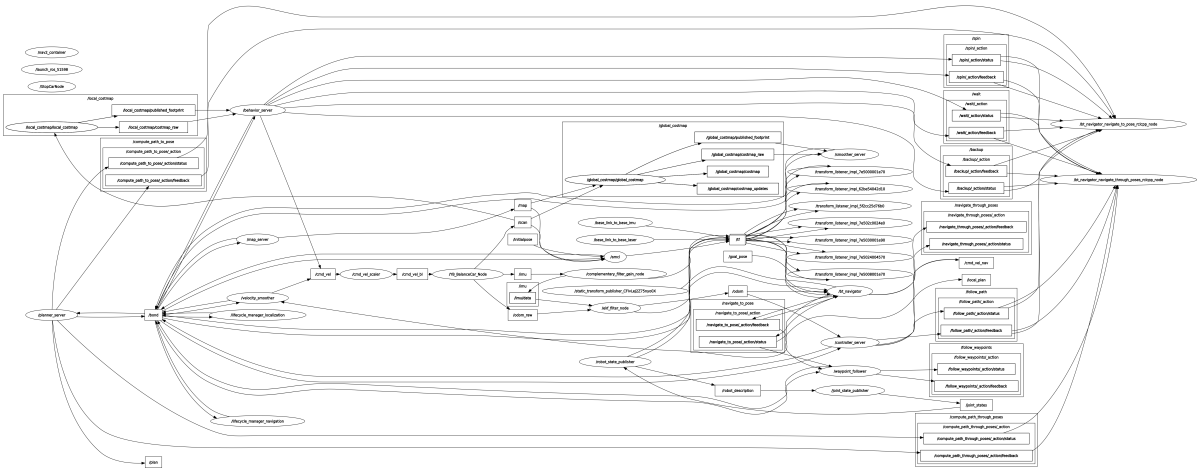


5. View the node communication graph

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.



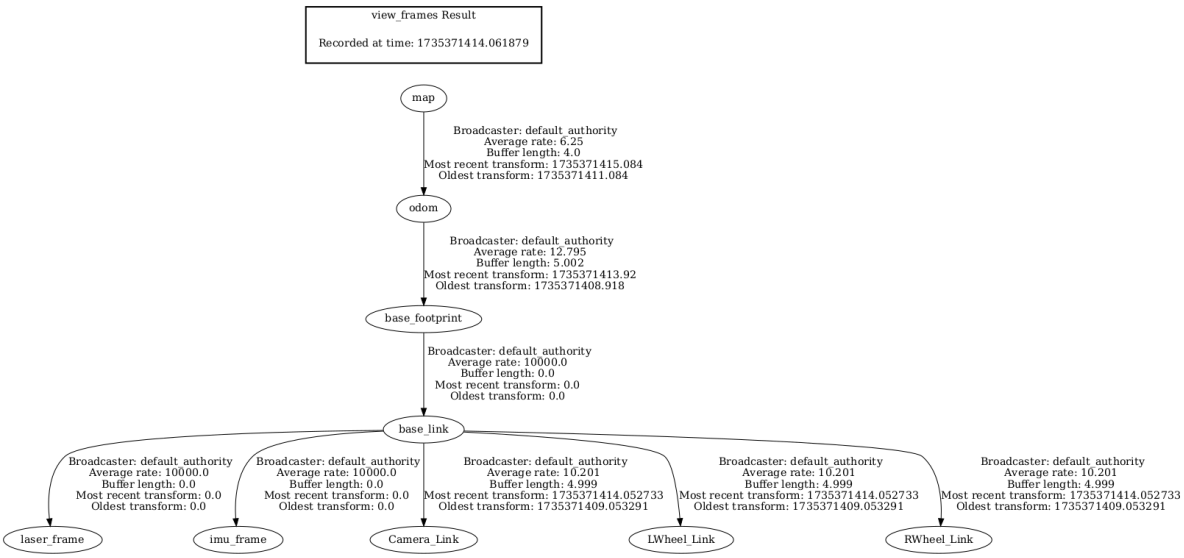
6. View TF tree

Terminal input,

```
ros2 run tf2_tools view_frames
```

```
yahboom@yahboom-VM:~/Desktop$ ros2 run tf2_tools view_frames
[INFO] [1735371409.047055920] [view_frames]: Listening to tf data for 5.0 seconds...
[INFO] [1735371414.054443006] [view_frames]: Generating graph in frames.pdf file...
[INFO] [1735371414.057701646] [view_frames]: Result:tf2_msgs.srv.FrameGraph_Response(frame_yaml="base_footprint: \n  parent: 'odom'\n  broadcaster: 'default_authority'\n  rate: 12.795\n  most_recent_transform: 1735371415.084000\n  oldest_transform: 1735371411.084000\n  buffer_length: 4.000\nlaser_frame: \n  parent: 'base_link'\n  broadcaster: 'default_authority'\n  rate: 10000.000\n  most_recent_transform: 0.000000\n  oldest_transform: 0.000000\n  buffer_length: 0.000\nbase_link: \n  parent: 'base_footprint'\n  broadcaster: 'default_authority'\n  rate: 10000.000\n  most_recent_transform: 0.000000\n  oldest_transform: 0.000000\n  buffer_length: 0.000\nimu_frame: \n  parent: 'base_link'\n  broadcaster: 'default_authority'\n  rate: 10000.000\n  most_recent_transform: 0.000000\n  oldest_transform: 0.000000\n  buffer_length: 0.000\nCamera_Link: \n  parent: 'base_link'\n  broadcaster: 'default_authority'\n  rate: 10.201\n  most_recent_transform: 1735371414.052733\n  oldest_transform: 1735371409.053291\n  buffer_length: 4.999\nLWheel_Link: \n  parent: 'base_link'\n  broadcaster: 'default_authority'\n  rate: 10.201\n  most_recent_transform: 1735371414.052733\n  oldest_transform: 1735371409.053291\n  buffer_length: 4.999\nRWheel_Link: \n  parent: 'base_link'\n  broadcaster: 'default_authority'\n  rate: 10.201\n  most_recent_transform: 1735371414.052733\n  oldest_transform: 1735371409.053291\n  buffer_length: 4.999\n")
```

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

Here we only explain navigation_dwb_launch.py of navigation,

Take the virtual machine as an example, the file path is,

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

navigation_dwb_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.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node

def generate_launch_description():
    package_path = get_package_share_directory('yahboomcar_nav')
    nav2_bringup_dir = get_package_share_directory('nav2_bringup')

    use_sim_time = LaunchConfiguration('use_sim_time', default='false')
    namespece = LaunchConfiguration('namespece', default='')
    map_yaml_path = LaunchConfiguration(
        'maps',
default=os.path.join('/home/yahboom/yahboomcar_ws/src/yahboomcar_nav', 'maps',
'yahboom_map.yaml'))
    nav2_param_path = LaunchConfiguration('params_file', default=os.path.join(
        package_path, 'params', 'dwb_nav_params.yaml'))

    return LaunchDescription([
        DeclareLaunchArgument('use_sim_time', default_value=use_sim_time,
                              description='Use simulation (Gazebo) clock if
true'),
        DeclareLaunchArgument('namespece', default_value=namespece,
                              description='Use simulation (Gazebo) clock if
true'),
```

```
        DeclareLaunchArgument('maps', default_value=map_yaml_path,
                              description='Full path to map file to load'),
        DeclareLaunchArgument('params_file', default_value=nav2_param_path,
                              description='Full path to param file to load'),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
                [nav2_bringup_dir, '/launch', '/bringup_launch.py']),
            launch_arguments={
                'map': map_yaml_path,
                'use_sim_time': use_sim_time,
                'namespece': namespece,
                'params_file': nav2_param_path}.items(),
        ),
        Node(
            package='yahboomcar_nav',
            executable='stop_car'
        ) ,
        Node(
     package='tf2_ros',
     executable='static_transform_publisher',
     name='base_link_to_base_laser',
     arguments=['-0.0046412', '0' ,
'0.094079','0','0','0','base_link','laser_frame']
    )
    ])
```

The following nodes are started here:

- base_link_to_base_laser: publish static TF transformation;
- stop_car: parking node, after ctrl c exits the program, the parking speed will be published to the car;
- bringup_launch.py: launch file for navigation, the file is located at, `/opt/ros/humble/share/nav2_bringup/launch`

In addition, a navigation parameter configuration file dwb_nav_params.yaml and a map file yahboom_map.yaml are loaded.

The map file is located at,

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

Take the virtual machine as an example, the navigation parameter table is located at,

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

dwb_nav_params.yaml,

```
amcl:
  ros__parameters:
    use_sim_time: False
    alpha1: 0.5
    alpha2: 0.2
    alpha3: 0.2
```

```
    alpha4: 0.4
    alpha5: 0.2
    base_frame_id: "base_footprint"
    beam_skip_distance: 0.5
    beam_skip_error_threshold: 0.9
    beam_skip_threshold: 0.3
    do_beamskip: false
    global_frame_id: "map"
    lambda_short: 0.1
    laser_likelihood_max_dist: 2.0
    laser_max_range: 100.0
    laser_min_range: -1.0
    laser_model_type: "likelihood_field"
    max_beams: 60
    max_particles: 2000
    min_particles: 500
    odom_frame_id: "odom"
    pf_err: 0.05
    pf_z: 0.99
    recovery_alpha_fast: 0.0
    recovery_alpha_slow: 0.0
    resample_interval: 1
    robot_model_type: "nav2_amcl::DifferentialMotionModel"
    save_pose_rate: 0.5
    sigma_hit: 0.02
    tf_broadcast: true
    transform_tolerance: 1.25
    update_min_a: 0.06
    update_min_d: 0.025
    z_hit: 0.5
    z_max: 0.05
    z_rand: 0.5
    z_short: 0.05
    scan_topic: scan

bt_navigator:
  ros__parameters:
    use_sim_time: False
    global_frame: map
    robot_base_frame: base_link
    odom_topic: /odom
    bt_loop_duration: 10
    default_server_timeout: 20
    default_bt_xml_filename: "navigate_to_pose_w_replanning_and_recovery.xml"
    # 'default_nav_through_poses_bt_xml' and 'default_nav_to_pose_bt_xml' are use
defaults:
    # nav2_bt_navigator/navigate_to_pose_w_replanning_and_recovery.xml
    # nav2_bt_navigator/navigate_through_poses_w_replanning_and_recovery.xml
    # They can be set here or via a RewrittenYaml remap from a parent launch file
to Nav2.
    plugin_lib_names:
      - nav2_compute_path_to_pose_action_bt_node
      - nav2_compute_path_through_poses_action_bt_node
      - nav2_smooth_path_action_bt_node
      - nav2_follow_path_action_bt_node
      - nav2_spin_action_bt_node
```

```
      - nav2_wait_action_bt_node
      - nav2_assisted_teleop_action_bt_node
      - nav2_back_up_action_bt_node
      - nav2_drive_on_heading_bt_node
      - nav2_clear_costmap_service_bt_node
      - nav2_is_stuck_condition_bt_node
      - nav2_goal_reached_condition_bt_node
      - nav2_goal_updated_condition_bt_node
      - nav2_globally_updated_goal_condition_bt_node
      - nav2_is_path_valid_condition_bt_node
      - nav2_initial_pose_received_condition_bt_node
      - nav2_reinitialize_global_localization_service_bt_node
      - nav2_rate_controller_bt_node
      - nav2_distance_controller_bt_node
      - nav2_speed_controller_bt_node
      - nav2_truncate_path_action_bt_node
      - nav2_truncate_path_local_action_bt_node
      - nav2_goal_updater_node_bt_node
      - nav2_recovery_node_bt_node
      - nav2_pipeline_sequence_bt_node
      - nav2_round_robin_node_bt_node
      - nav2_transform_available_condition_bt_node
      - nav2_time_expired_condition_bt_node
      - nav2_path_expiring_timer_condition
      - nav2_distance_traveled_condition_bt_node
      - nav2_single_trigger_bt_node
      - nav2_goal_updated_controller_bt_node
      - nav2_is_battery_low_condition_bt_node
      - nav2_navigate_through_poses_action_bt_node
      - nav2_navigate_to_pose_action_bt_node
      - nav2_remove_passed_goals_action_bt_node
      - nav2_planner_selector_bt_node
      - nav2_controller_selector_bt_node
      - nav2_goal_checker_selector_bt_node
      - nav2_controller_cancel_bt_node
      - nav2_path_longer_on_approach_bt_node
      - nav2_wait_cancel_bt_node
      - nav2_spin_cancel_bt_node
      - nav2_back_up_cancel_bt_node
      - nav2_assisted_teleop_cancel_bt_node
      - nav2_drive_on_heading_cancel_bt_node
      - nav2_is_battery_charging_condition_bt_node

bt_navigator_navigate_through_poses_rclcpp_node:
  ros__parameters:
    use_sim_time: False

bt_navigator_navigate_to_pose_rclcpp_node:
  ros__parameters:
    use_sim_time: False

controller_server:
  ros__parameters:
    use_sim_time: False
    controller_frequency: 20.0
    min_x_velocity_threshold: 0.001
```

```
    min_y_velocity_threshold: 0.5
    min_theta_velocity_threshold: 0.001 #0.01
    max_theta_velocity_threshold: 5.0 #0.5
    failure_tolerance: 0.3
    progress_checker_plugin: "progress_checker"
    goal_checker_plugins: ["general_goal_checker"]
    controller_plugins: ["FollowPath"]

    # Progress checker parameters
    progress_checker:
      plugin: "nav2_controller::SimpleProgressChecker"
      required_movement_radius: 0.5
      movement_time_allowance: 10.0
    # Goal checker parameters
    precise_goal_checker:
     plugin: "nav2_controller::SimpleGoalChecker"
     xy_goal_tolerance: 0.25 #0.25
     yaw_goal_tolerance: 0.15 #0.15
     stateful: True
    general_goal_checker:
      stateful: True
      plugin: "nav2_controller::SimpleGoalChecker"
      xy_goal_tolerance: 0.25 #0.25
      yaw_goal_tolerance: 0.15 #0.15
    # DWB parameters
    FollowPath:
      plugin: "dwb_core::DWBLocalPlanner"
      debug_trajectory_details: True
      min_vel_x: -0.50
      min_vel_y: 0.0
      max_vel_x: 0.55
      max_vel_y: 0.0
      max_vel_theta: 10.0
      min_speed_xy: -0.50
      max_speed_xy: 0.55
      min_speed_theta: -10.0
      # Add high threshold velocity for turtlebot 3 issue.
      # https://github.com/ROBOTIS-GIT/turtlebot3_simulations/issues/75
      acc_lim_x: 2.5
      acc_lim_y: 0.0
      acc_lim_theta: 5.2 #4.5
      decel_lim_x: -2.5
      decel_lim_y: 0.0
      decel_lim_theta: -5.2 #-4.5
      vx_samples: 20
      vy_samples: 5
      vtheta_samples: 20
      sim_time: 1.7
      linear_granularity: 0.05
      angular_granularity: 0.025
      transform_tolerance: 0.25
      xy_goal_tolerance: 0.2
      trans_stopped_velocity: 0.25
      short_circuit_trajectory_evaluation: True
      stateful: True
```

```
      critics: ["RotateToGoal", "Oscillation", "BaseObstacle", "GoalAlign",
"PathAlign", "PathDist", "GoalDist"]
      BaseObstacle.scale: 0.02
      PathAlign.scale: 32.0
      PathAlign.forward_point_distance: 0.1
      GoalAlign.scale: 24.0 #24.0
      GoalAlign.forward_point_distance: 0.1
      PathDist.scale: 32.0
      GoalDist.scale: 24.0
      RotateToGoal.scale: 48.0 #32.0
      RotateToGoal.slowing_factor: 0.5 #5.0
      RotateToGoal.lookahead_time: -1.0

local_costmap:
  local_costmap:
    ros__parameters:
      update_frequency: 5.0
      publish_frequency: 2.0
      global_frame: odom
      robot_base_frame: base_link
      use_sim_time: False
      rolling_window: true
      width: 3
      height: 3
      resolution: 0.05
      robot_radius: 0.15
      plugins: ["voxel_layer", "inflation_layer"]
      inflation_layer:
        plugin: "nav2_costmap_2d::InflationLayer"
        cost_scaling_factor: 3.0
        inflation_radius: 0.18
      voxel_layer:
        plugin: "nav2_costmap_2d::VoxelLayer"
        enabled: True
        publish_voxel_map: True
        origin_z: 0.0
        z_resolution: 0.05
        z_voxels: 16
        max_obstacle_height: 2.0
        mark_threshold: 0
        observation_sources: scan
        scan:
          topic: /scan
          max_obstacle_height: 2.0
          clearing: True
          marking: True
          data_type: "LaserScan"
          raytrace_max_range: 3.0
          raytrace_min_range: 0.0
          obstacle_max_range: 2.5
          obstacle_min_range: 0.0
      static_layer:
        plugin: "nav2_costmap_2d::StaticLayer"
        map_subscribe_transient_local: True
      always_send_full_costmap: True
```

```yaml
global_costmap:
  global_costmap:
    ros__parameters:
      update_frequency: 1.0
      publish_frequency: 1.0
      global_frame: map
      robot_base_frame: base_link
      use_sim_time: False
      robot_radius: 0.15
      resolution: 0.05
      track_unknown_space: true
      plugins: ["static_layer", "obstacle_layer", "inflation_layer"]
      obstacle_layer:
        plugin: "nav2_costmap_2d::ObstacleLayer"
        enabled: True
        observation_sources: scan
        scan:
          topic: /scan
          max_obstacle_height: 2.0
          clearing: True
          marking: True
          data_type: "LaserScan"
          raytrace_max_range: 3.0
          raytrace_min_range: 0.0
          obstacle_max_range: 2.5
          obstacle_min_range: 0.0
      static_layer:
        plugin: "nav2_costmap_2d::StaticLayer"
        map_subscribe_transient_local: True
      inflation_layer:
        plugin: "nav2_costmap_2d::InflationLayer"
        cost_scaling_factor: 3.0
        inflation_radius: 0.18
      always_send_full_costmap: True

map_server:
  ros__parameters:
    use_sim_time: False
    # Overridden in launch by the "map" launch configuration or provided default
value.
    # To use in yaml, remove the default "map" value in the
tb3_simulation_launch.py file & provide full path to map below.
    yaml_filename: ""

map_saver:
  ros__parameters:
    use_sim_time: False
    save_map_timeout: 5.0
    free_thresh_default: 0.25
    occupied_thresh_default: 0.65
    map_subscribe_transient_local: True

planner_server:
  ros__parameters:
    expected_planner_frequency: 20.0
    use_sim_time: False
```

```
    planner_plugins: ["GridBased"]
    GridBased:
      plugin: "nav2_navfn_planner/NavfnPlanner"
      tolerance: 0.5
      use_astar: false
      allow_unknown: true
smoother_server:
  ros__parameters:
    use_sim_time: False
    smoother_plugins: ["simple_smoother"]
    simple_smoother:
      plugin: "nav2_smoother::SimpleSmoother"
      tolerance: 1.0e-10
      max_its: 1000
      do_refinement: False

behavior_server:
  ros__parameters:
    costmap_topic: local_costmap/costmap_raw
    footprint_topic: local_costmap/published_footprint
    cycle_frequency: 10.0
    behavior_plugins: ["spin", "backup", "drive_on_heading", "assisted_teleop",
"wait"]
    spin:
      plugin: "nav2_behaviors/Spin"
    backup:
      plugin: "nav2_behaviors/BackUp"
    drive_on_heading:
      plugin: "nav2_behaviors/DriveOnHeading"
    wait:
      plugin: "nav2_behaviors/Wait"
    assisted_teleop:
      plugin: "nav2_behaviors/AssistedTeleop"
    global_frame: odom
    robot_base_frame: base_link
    transform_tolerance: 0.15
    use_sim_time: False
    simulate_ahead_time: 2.0
    max_rotational_vel: 5.0 #1.0
    min_rotational_vel: 4.0 #0.4
    rotational_acc_lim: 3.2    #3.2

robot_state_publisher:
  ros__parameters:
    use_sim_time: False

waypoint_follower:
  ros__parameters:
    use_sim_time: False
    loop_rate: 20
    stop_on_failure: false
    waypoint_task_executor_plugin: "wait_at_waypoint"
    wait_at_waypoint:
      plugin: "nav2_waypoint_follower::WaitAtWaypoint"
      enabled: True
      waypoint_pause_duration: 200
```

```yaml
velocity_smoother:
  ros__parameters:
    use_sim_time: False
    smoothing_frequency: 20.0
    scale_velocities: False
    feedback: "OPEN_LOOP"
    max_velocity: [0.35, 0.0, 3.0]
    min_velocity: [-0.35, 0.0, -3.0]
    max_accel: [2.5, 0.0, 3.2]
    max_decel: [-2.5, 0.0, -3.2]
    odom_topic: "odom"
    odom_duration: 0.1
    deadband_velocity: [0.0, 0.0, 0.0]
    velocity_timeout: 1.0
```

This parameter table configures the parameters required for each node started in the navigation launch file.