8、Navigation and obstacle avoidance

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

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

Navigation2 corresponds to the paper: https://arxiv.org/pdf/2003.00368.pdf

teb_local_planner： https://github.com/rst-tu-dortmund/teb_local_planner/tree/foxy-devel

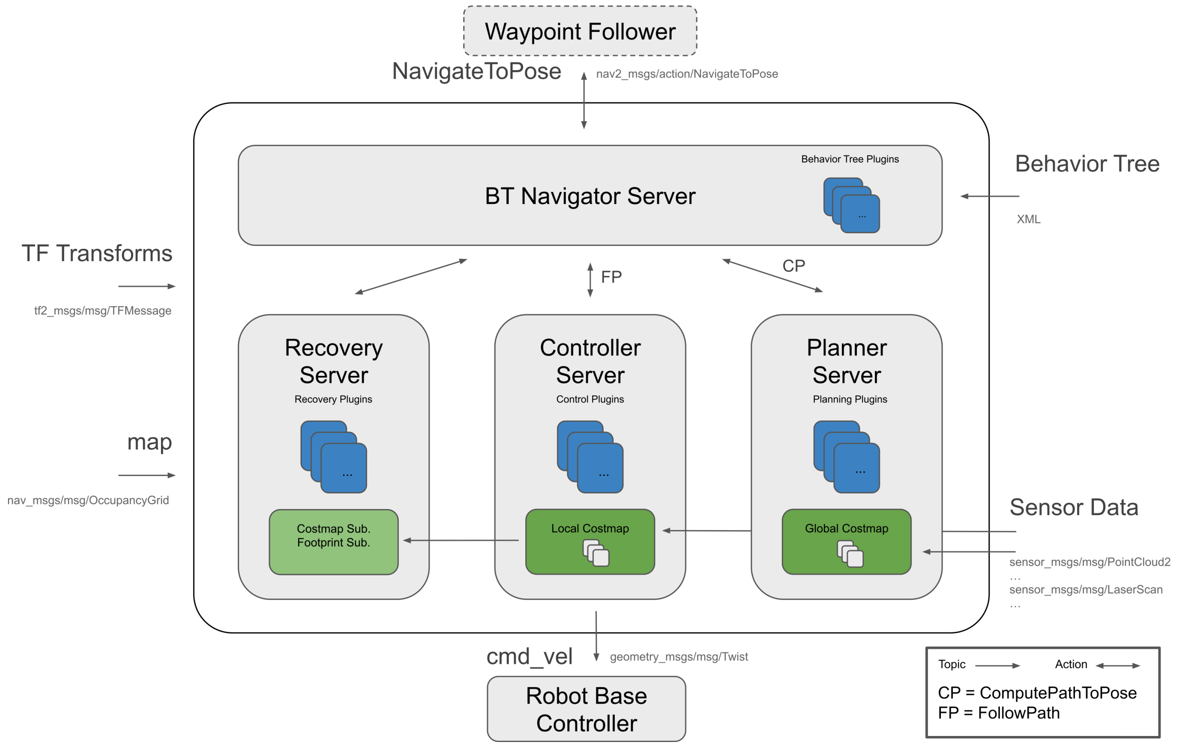
Plugins provided by Navigation2: https://navigation.ros.org/plugins/index.html#plugins

The operating environment and software and hardware reference configurations are as follows:

- REFERENCE MODEL: ROSMASTER R2
- Robot hardware configuration: Arm series main control, Silan A1 lidar, AstraPro Plus depth camera
- Robot system: Ubuntu (version not required) + docker (version 20.10.21 and above)
- PC Virtual Machine: Ubuntu （22.04） + ROS2 （Humble）
- Usage scenario: Use on a relatively clean 2D plane

8.1、Brief introduction

Navigation2 overall architecture diagram



Navigation2 has the following tools:

- Tools for loading, serving, and storing maps (Map Server)
- Tool to locate the robot on the map (AMCL)
- Navigate from point A to point B with Nav2 Planner
- Tool to control the robot during the following path (Nav2 Controller)
- Tools for converting sensor data into cost map representations in the world of robotics (Nav2 Costmap 2D)
- Tools for building complex robot behaviors using behavior trees (Nav2 Behavior Trees and BT Navigator)
- Tool to calculate recovery behavior in the event of a failure (Nav2 Recoveries)
- Nav2 Waypoint Follower
- Tools and watchdog for managing server lifecycles (Nav2 Lifecycle Manager)
- Plugins that enable user-defined algorithms and behaviors (Nav2 Core)

Navigation 2 (Nav 2) is the navigation framework that comes with ROS 2 and aims to move mobile robots from point A to point B in a safe way. As a result, Nav 2 can perform dynamic path planning, calculate motor speed, avoid obstacles, and restore structures.

Nav 2 uses Behavior Trees (BT) to call modular servers to complete an action. Actions can be calculated paths, control efforts, recovery, or other navigation-related actions. These actions are independent nodes that communicate with the Behavior Tree (BT) through the Action Server.

8.2、use

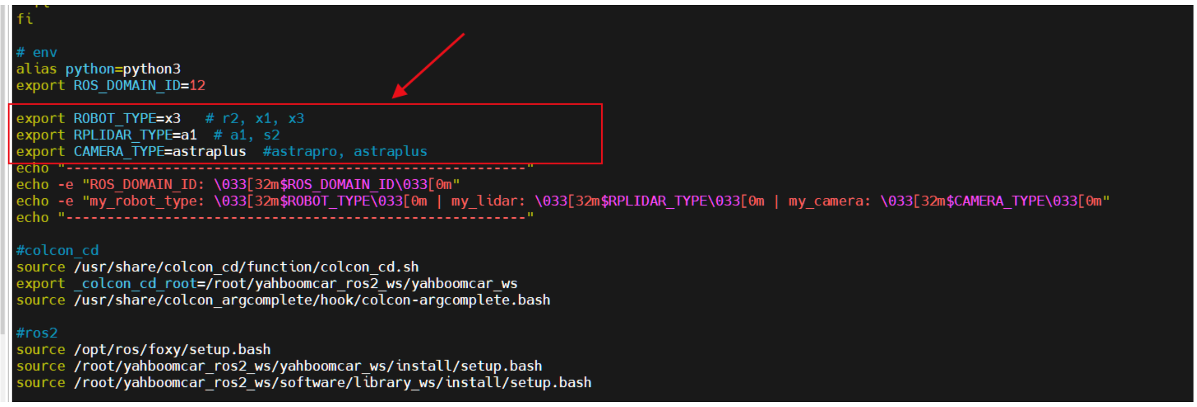
8.2.1、pre-use configuration

Note: Since ROSMASTER series robots are divided into multiple robots and multiple devices, the factory system has been configured with routines for multiple devices, but because the product cannot be automatically identified, it is necessary to manually set the machine type and radar model.

Raspberry Pi PI5 master control needs to enter the docker container first, Orin master control does not need to enter,

according to the model of the car, the type of radar and the type of camera, make the following modifications:

```
root@ubuntu:/# cd
root@ubuntu:~# vim .bashrc
```



```
fi

# env
alias python=python3
export ROS_DOMAIN_ID=12

export ROBOT_TYPE=x3   # r2, x1, x3
export RPLIDAR_TYPE=a1  # a1, s2
export CAMERA_TYPE=astraplus  #astrapro, astraplus
echo "---------------------------------------------------------"
echo -e "ROS_DOMAIN_ID: \033[32m$ROS_DOMAIN_ID\033[0m"
echo -e "my_robot_type: \033[32m$ROBOT_TYPE\033[0m | my_lidar: \033[32m$RPLIDAR_TYPE\033[0m | my_camera: \033[32m$CAMERA_TYPE\033[0m"
echo "---------------------------------------------------------"

#colcon_cd
source /usr/share/colcon_cd/function/colcon_cd.sh
export _colcon_cd_root=/root/yahboomcar_ros2_ws/yahboomcar_ws
source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash

#ros2
source /opt/ros/foxy/setup.bash
source /root/yahboomcar_ros2_ws/yahboomcar_ws/install/setup.bash
source /root/yahboomcar_ros2_ws/software/library_ws/install/setup.bash
```

After the modification is complete, save and exit vim, and then execute:

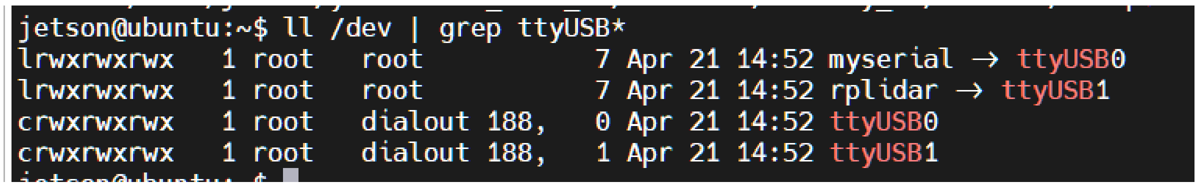
```
root@ubuntu:~# source .bashrc
--------------------------------------------------------
ROS_DOMAIN_ID: 12
my_robot_type: x3 | my_lidar: a1 | my_camera: astraplus
--------------------------------------------------------
root@ubuntu:~#
```

You can see the model of the currently modified car, the type of radar and the type of camera

8.2.2、start chassis and radar-related nodes

First of all, you need to do port binding operations in the host [that is, on the jetson of the car] [see the port binding tutorial chapter], which mainly uses two devices: radar and serial port;

Then check whether the radar and serial device are in the port binding state: on the host [that is, the jetson of the car], refer to the following command to view, and the successful binding is the following state:



```
jetson@ubuntu:~$ ll /dev | grep ttyUSB*
lrwxrwxrwx   1 root   root          7 Apr 21 14:52 myserial -> ttyUSB0
lrwxrwxrwx   1 root   root          7 Apr 21 14:52 rplidar -> ttyUSB1
crwxrwxrwx   1 root   dialout 188,  0 Apr 21 14:52 ttyUSB0
crwxrwxrwx   1 root   dialout 188,  1 Apr 21 14:52 ttyUSB1
```

If the radar or serial device is not bound to the display, you can plug and unplug the USB cable to view it again

Raspberry Pi PI5 master needs to enter the docker container first, Orin master does not need to enter,

Enter the docker container (for steps, please refer to [docker course chapter ----- 5. Enter the robot's docker container]),

and execute it in a terminal:

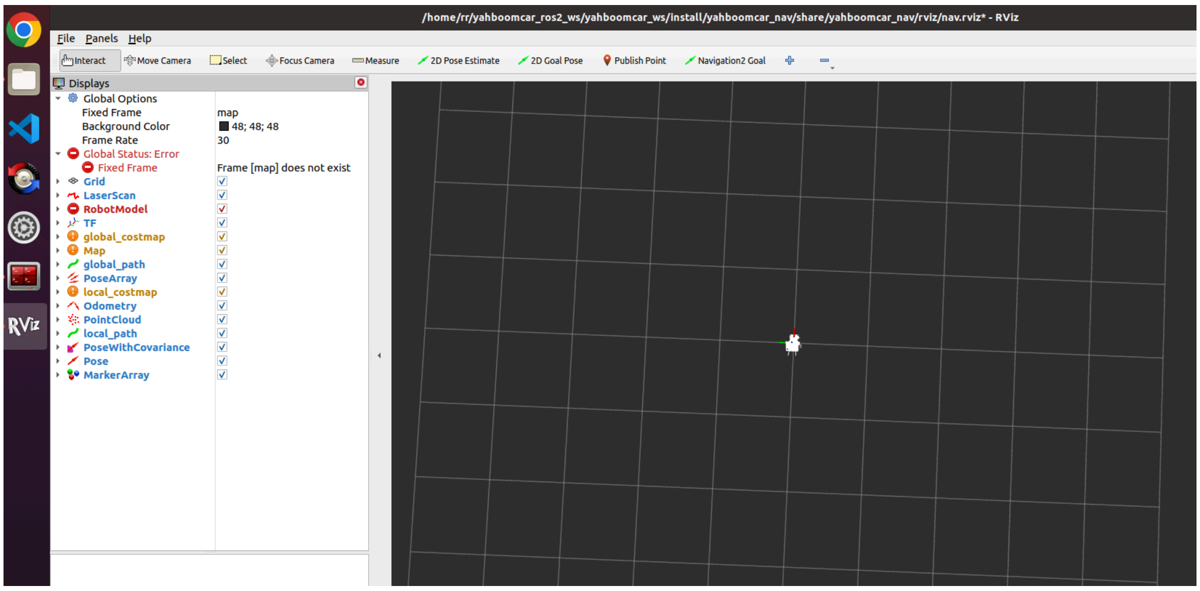
```
ros2 launch yahboomcar_nav laser_bringup_launch.py
```

8.2.3、Start rviz display map

Ubuntu virtual machine and docker container are configured with multi-machine communication, this step is recommended to start in the virtual machine: to keep time synchronized and reduce resource consumption, because if you use VNC, you are very network-dependent, which may lead to navigation failure.

[Note, here must first start the node that displays the map, and then start the navigation node in step 3, this is because the navigation2 terminal map topic is only published once, how to start the navigation node first, and then start the rviz display, it is possible that you may not be able to subscribe to the only map topic published, resulting in the map not being displayed]

```
ros2 launch yahboomcar_nav display_nav_launch.py
```



Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
Global Status: Error	
Fixed Frame	Frame [map] does not exist
Grid	✓
LaserScan	✓
RobotModel	✓
TF	✓
global_costmap	✓
Map	✓
global_path	✓
PoseArray	✓
local_costmap	✓
Odometry	✓
PointCloud	✓
local_path	✓
PoseWithCovariance	✓
Pose	✓
MarkerArray	✓

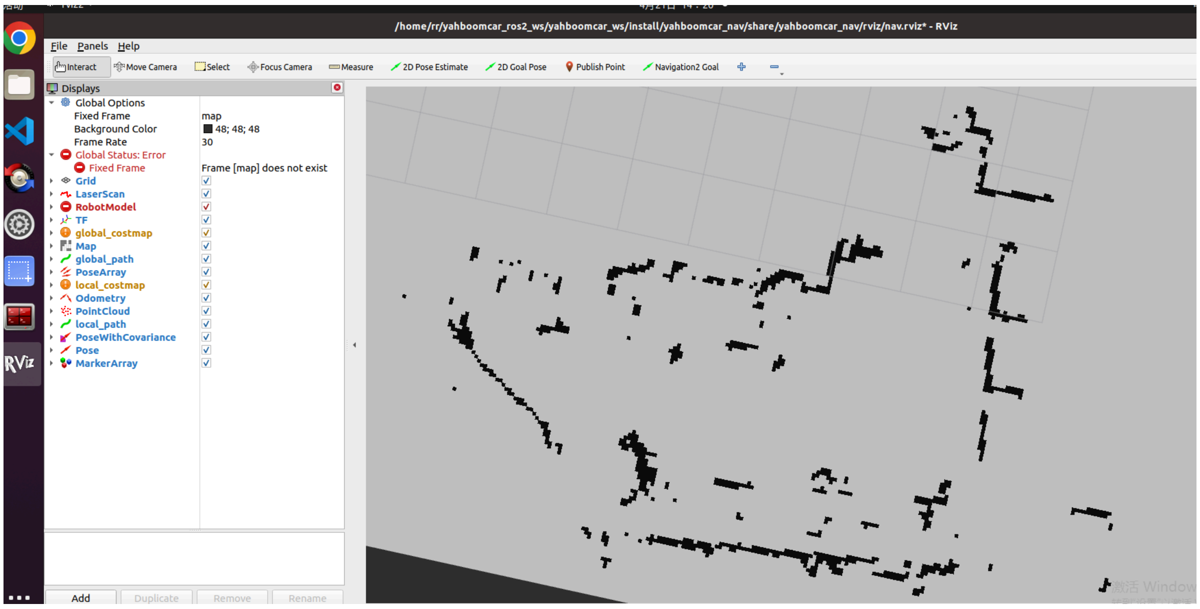
At this time, the map will not be displayed in the screen, and the topics of each node on the left do not need to be red, because the navigation node has not been started.

8.2.4、Start the navigation node

- There are two navigation algorithms: DWA and TEB
- Navigation can be divided into single-point navigation and multi-point navigation, which are described below.

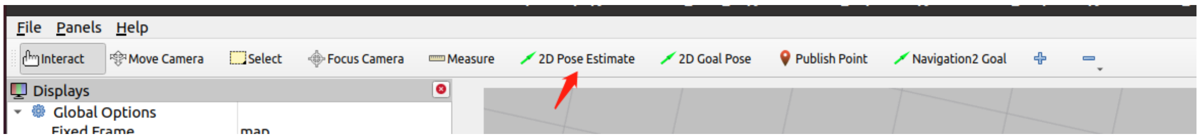
1. Start the navigation node, enter the docker container, and execute it in terminals:

```
ros2 launch yahboomcar_nav navigation_dwa_launch.py   #DWA navigation
or
ros2 launch yahboomcar_nav navigation_teb_launch.py   # TEB navigation
```

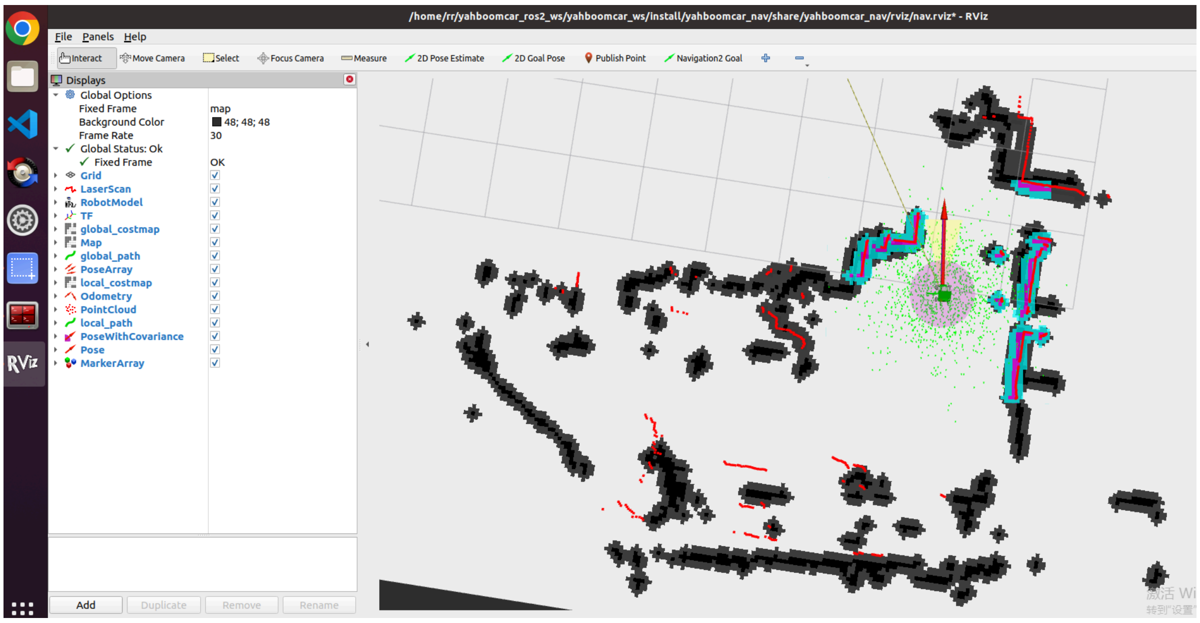


Global Options
Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Global Status: Error
Fixed Frame Frame [map] does not exist
Grid
LaserScan
RobotModel
TF
global_costmap
Map
global_path
PoseArray
local_costmap
Odometry
PointCloud
local_path
PoseWithCovariance
Pose
MarkerArray

2. Click [2D Pose Estimate] on rviz, and then compare the position of the car to mark the initial position of the car on the map;



The display after marking is as follows:

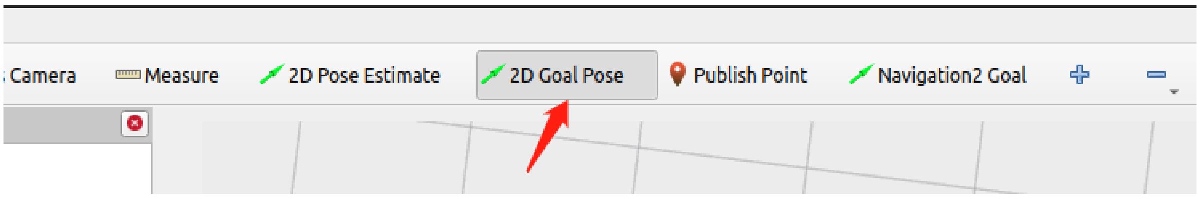


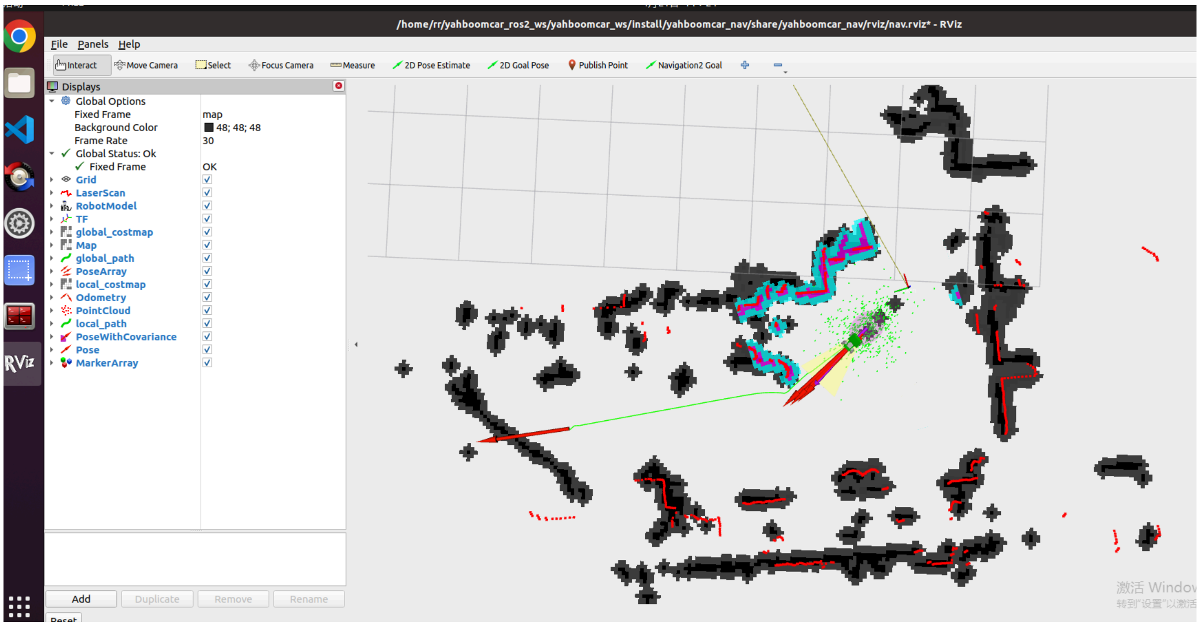
Global Options
Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Global Status: Ok
Fixed Frame OK
Grid
LaserScan
RobotModel
TF
global_costmap
Map
global_path
PoseArray
local_costmap
Odometry
PointCloud
local_path
PoseWithCovariance
Pose
MarkerArray

3. Compare the overlap between the radar scanning point and the obstacle, and you can set the initial posture of the trolley many times until the radar scanning point and the obstacle roughly coincide;

8.2.5、Single point navigation

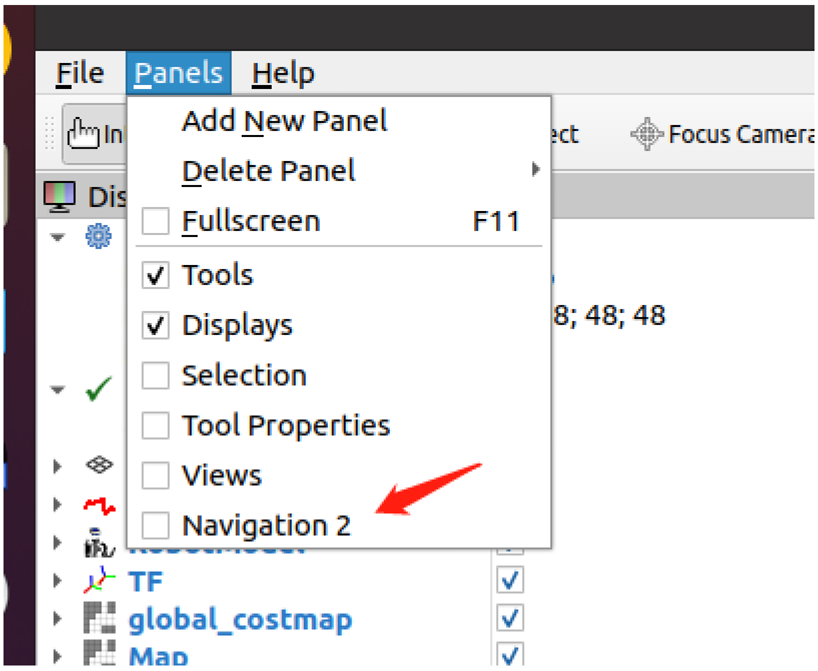
After the initial posture is set, you can click [2D Goal Pose] to set a navigation target point, and the car will start single-point navigation;

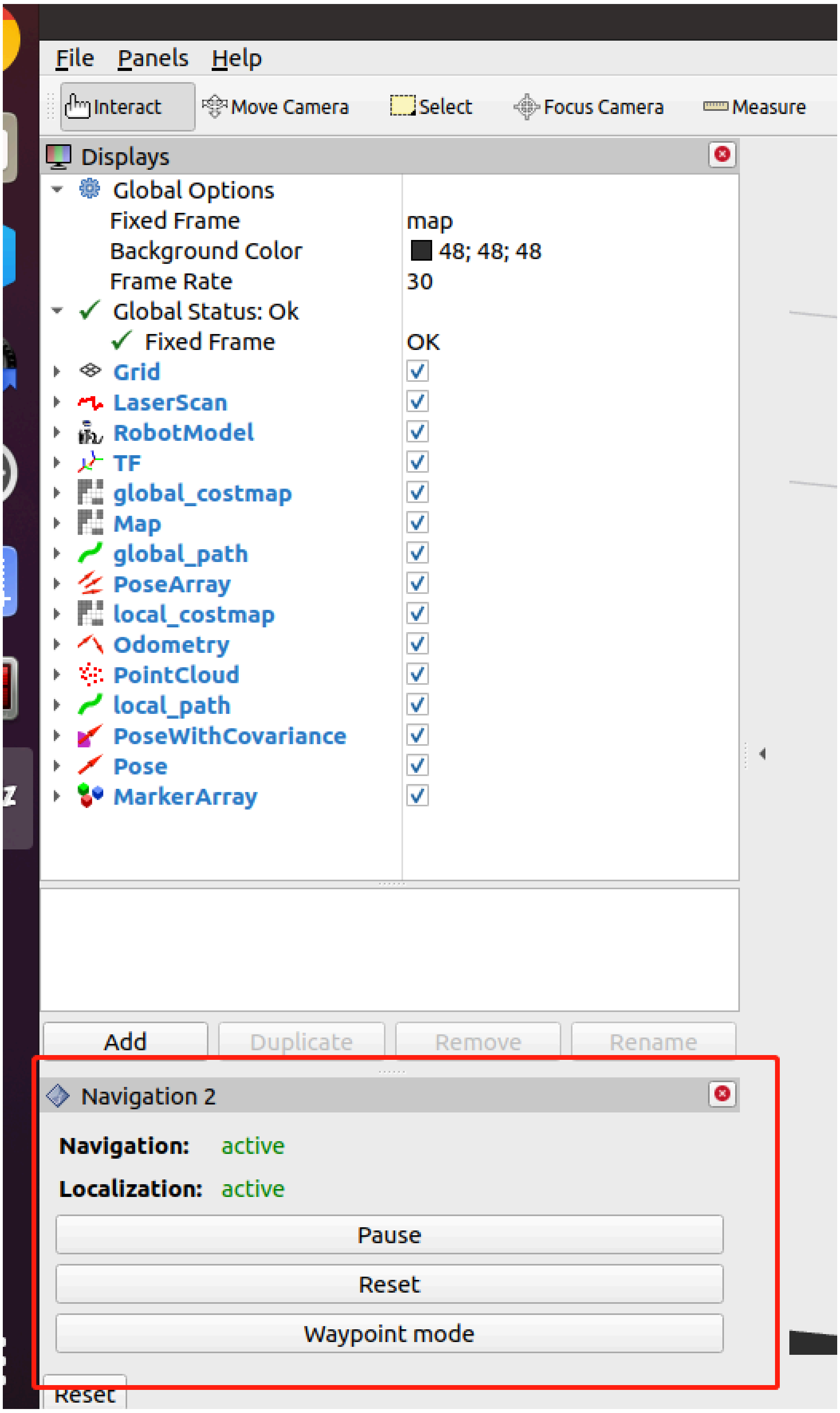




8.2.6、multi-point navigation

1、After the initial posture is set, you can click [Panels] in the upper left corner of rviz --- Select [Navigation 2], and the [Navigation 2] panel will be displayed




File Panels Help

Interact | Move Camera | Select | Focus Camera | Measure

Displays

Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
LaserScan	✓
RobotModel	✓
TF	✓
global_costmap	✓
Map	✓
global_path	✓
PoseArray	✓
local_costmap	✓
Odometry	✓
PointCloud	✓
local_path	✓
PoseWithCovariance	✓
Pose	✓
MarkerArray	✓

Add | Duplicate | Remove | Rename

Navigation 2

Navigation: active

Localization: active

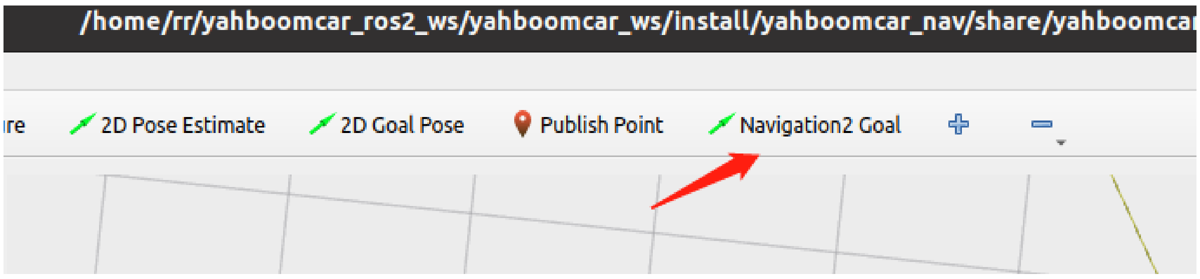
Pause

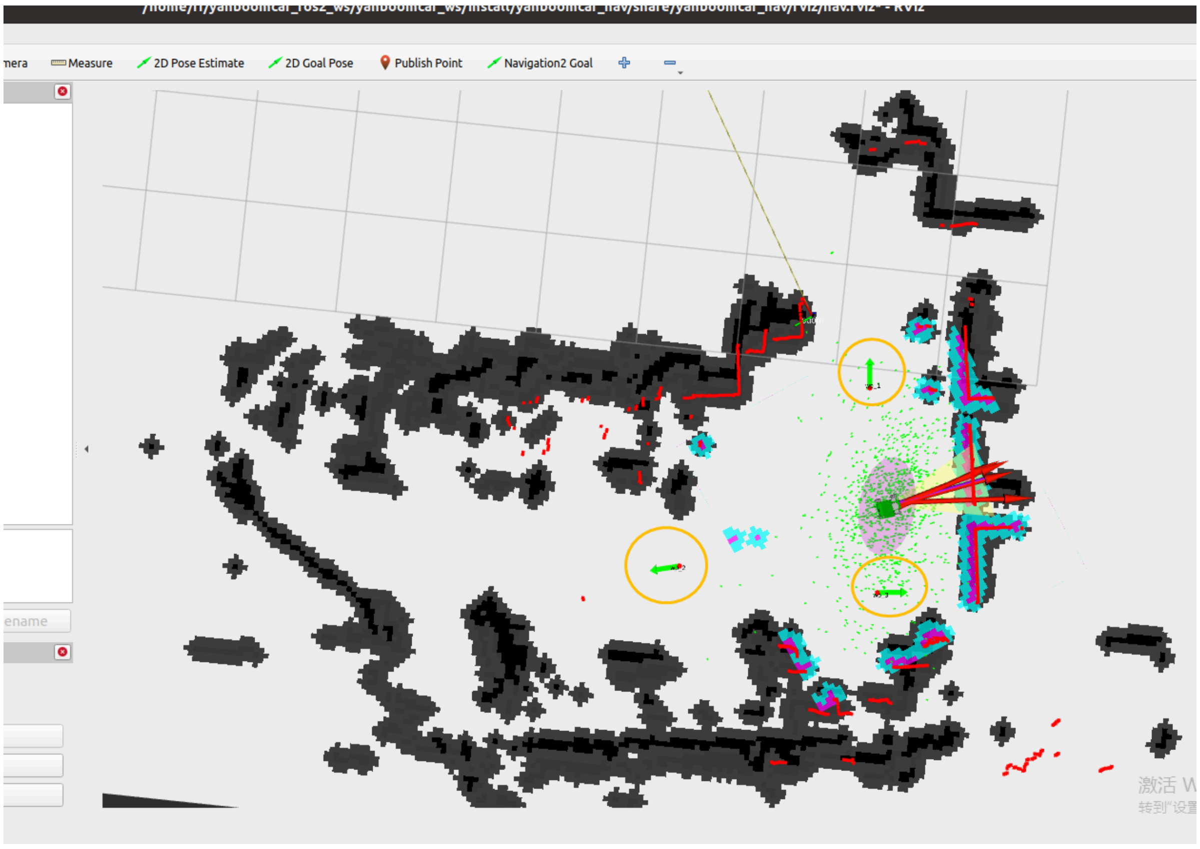
Reset

Waypoint mode

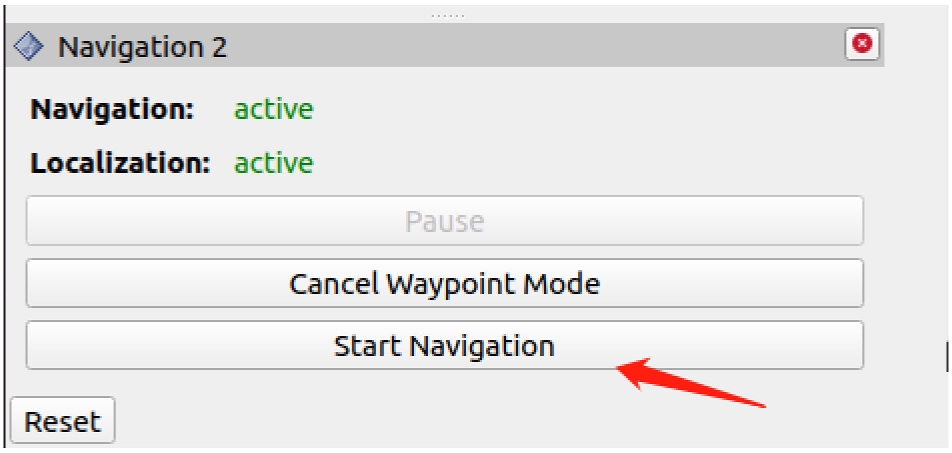
Reset

2、Click [Waypoint mode] in the figure above, and then click [Navigation2 Goal] on rviz to mark a target point in the map, and click [Navigation2 Goal] again to mark the second target point in the map, and if the cycle continues, you can mark multiple target points at once;


/home/rr/yahboomcar_ros2_ws/yahboomcar_ws/install/yahboomcar_nav/share/yahboomca
2D Pose Estimate
2D Goal Pose
Publish Point
Navigation2 Goal



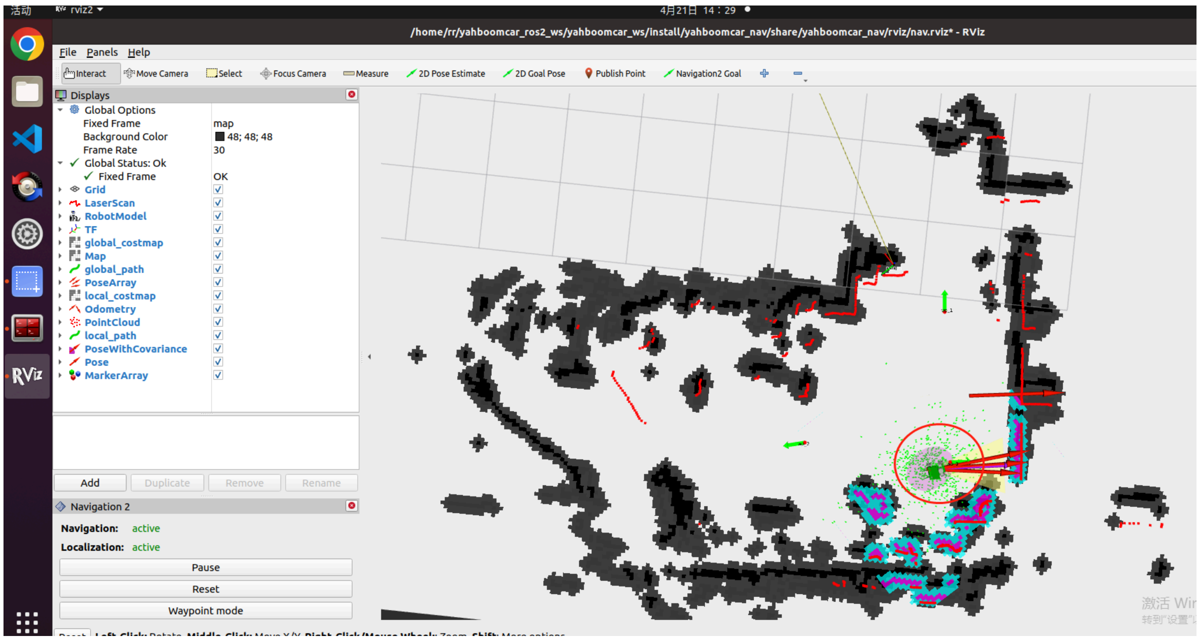
3、After marking multiple target points, click [Start Navigation] to start multi-point navigation;


Navigation 2

Navigation: active

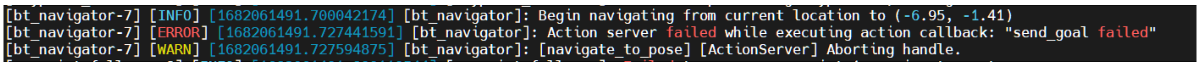
Localization: active

4、After the multi-point navigation is completed, the car will stay in the position of the last target point;



```
Displays
  Global Options
    Fixed Frame          map
    Background Color     48; 48; 48
    Frame Rate           30
  Global Status: Ok
    Fixed Frame          OK
  Grid                   ✓
  LaserScan              ✓
  RobotModel             ✓
  TF                     ✓
  global_costmap         ✓
  Map                    ✓
  global_path            ✓
  PoseArray              ✓
  local_costmap          ✓
  Odometry               ✓
  PointCloud             ✓
  local_path             ✓
  PoseWithCovariance     ✓
  Pose                   ✓
  MarkerArray            ✓

Navigation 2
Navigation:    active
Localization:  active
```

5、This may occur during navigation due to the ros-foxy version of navigation2 itself, which has been fixed in subsequent ros2 releases

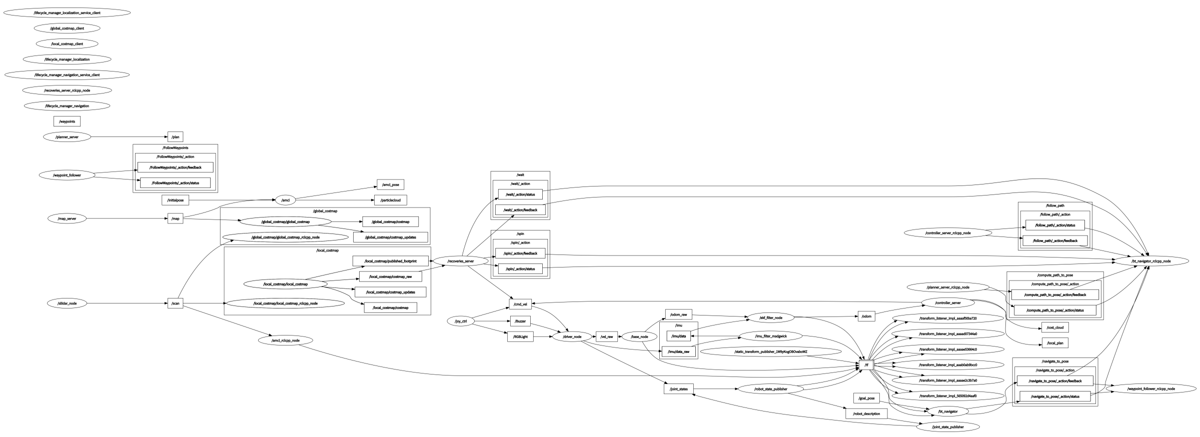


```
[bt_navigator-7] [INFO] [1682061491.700042174] [bt_navigator]: Begin navigating from current location to (-6.95, -1.41)
[bt_navigator-7] [ERROR] [1682061491.727441591] [bt_navigator]: Action server failed while executing action callback: "send_goal failed"
[bt_navigator-7] [WARN] [1682061491.727594875] [bt_navigator]: [navigate_to_pose] [ActionServer] Aborting handle.
```

8.3、node resolution

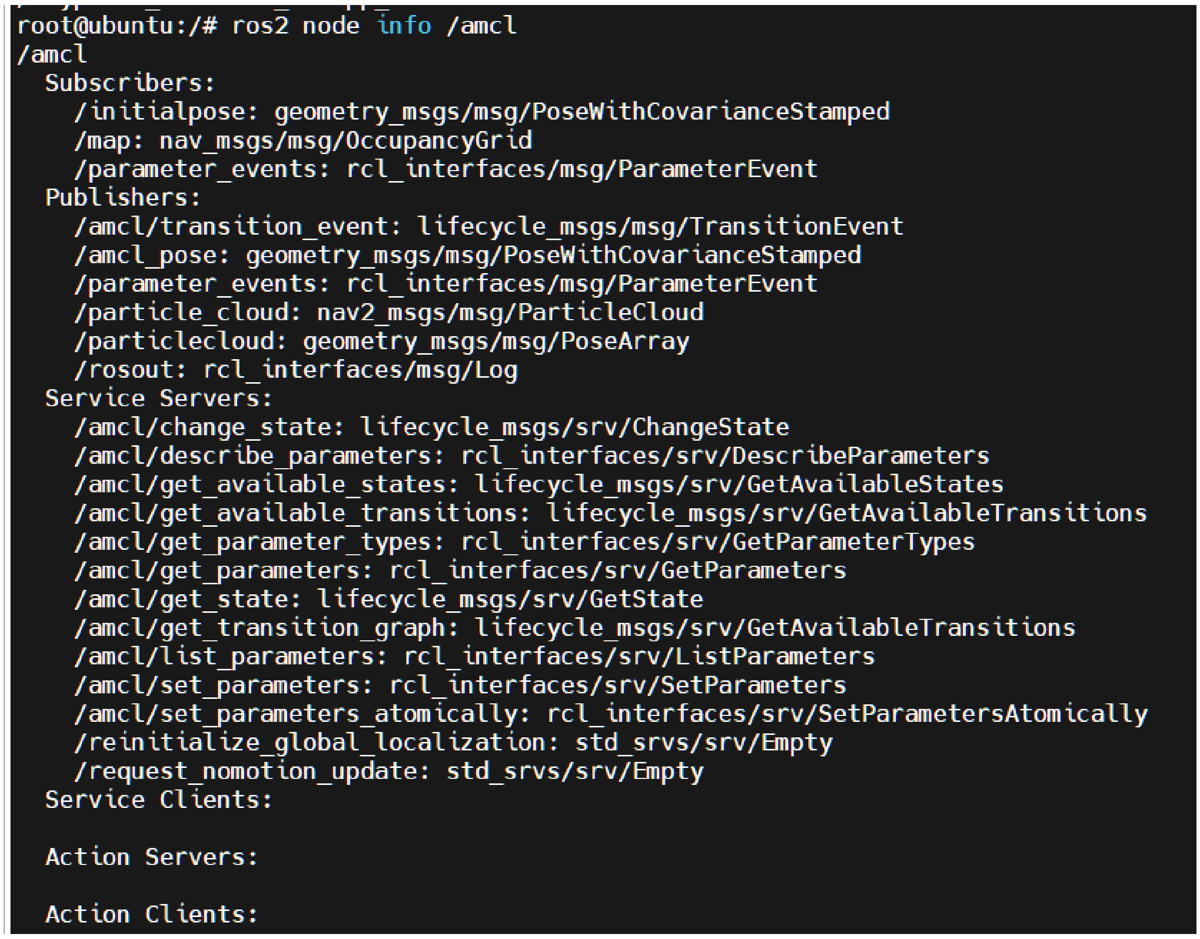
8.3.1、display the computational graph

```
rqt_graph
```



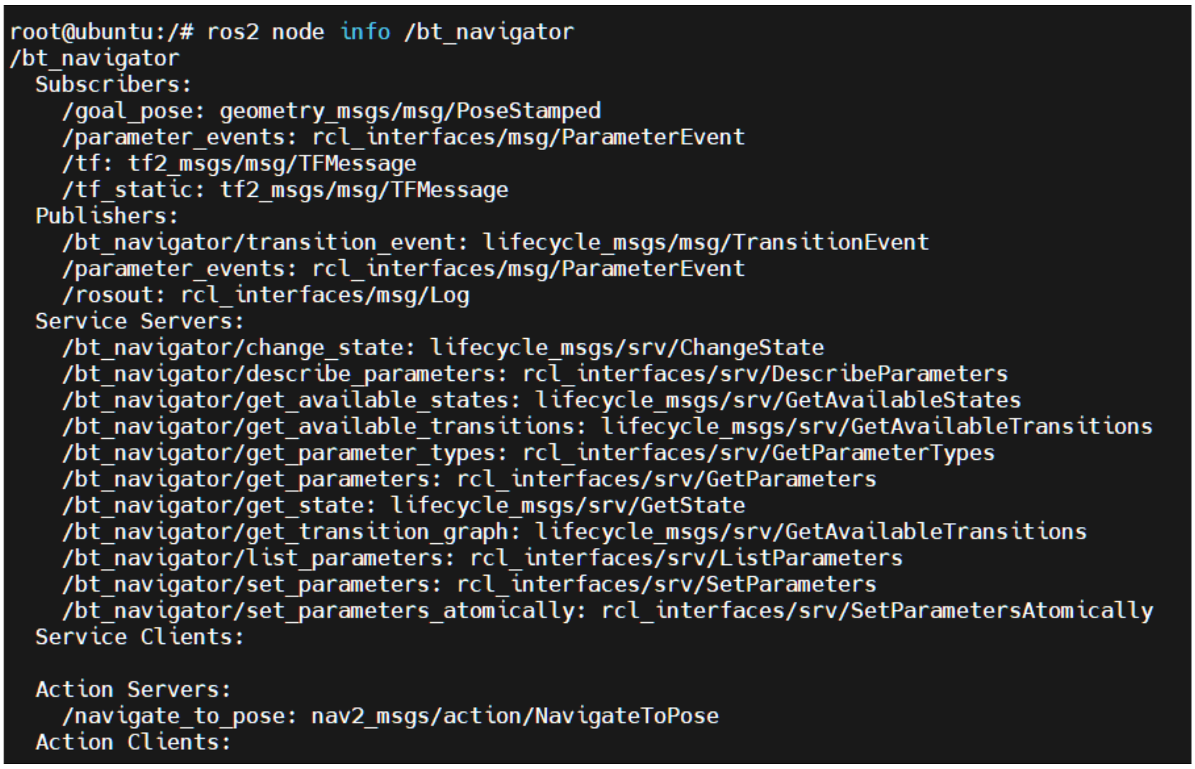
8.3.2、Navigate the details of each node

/amcl



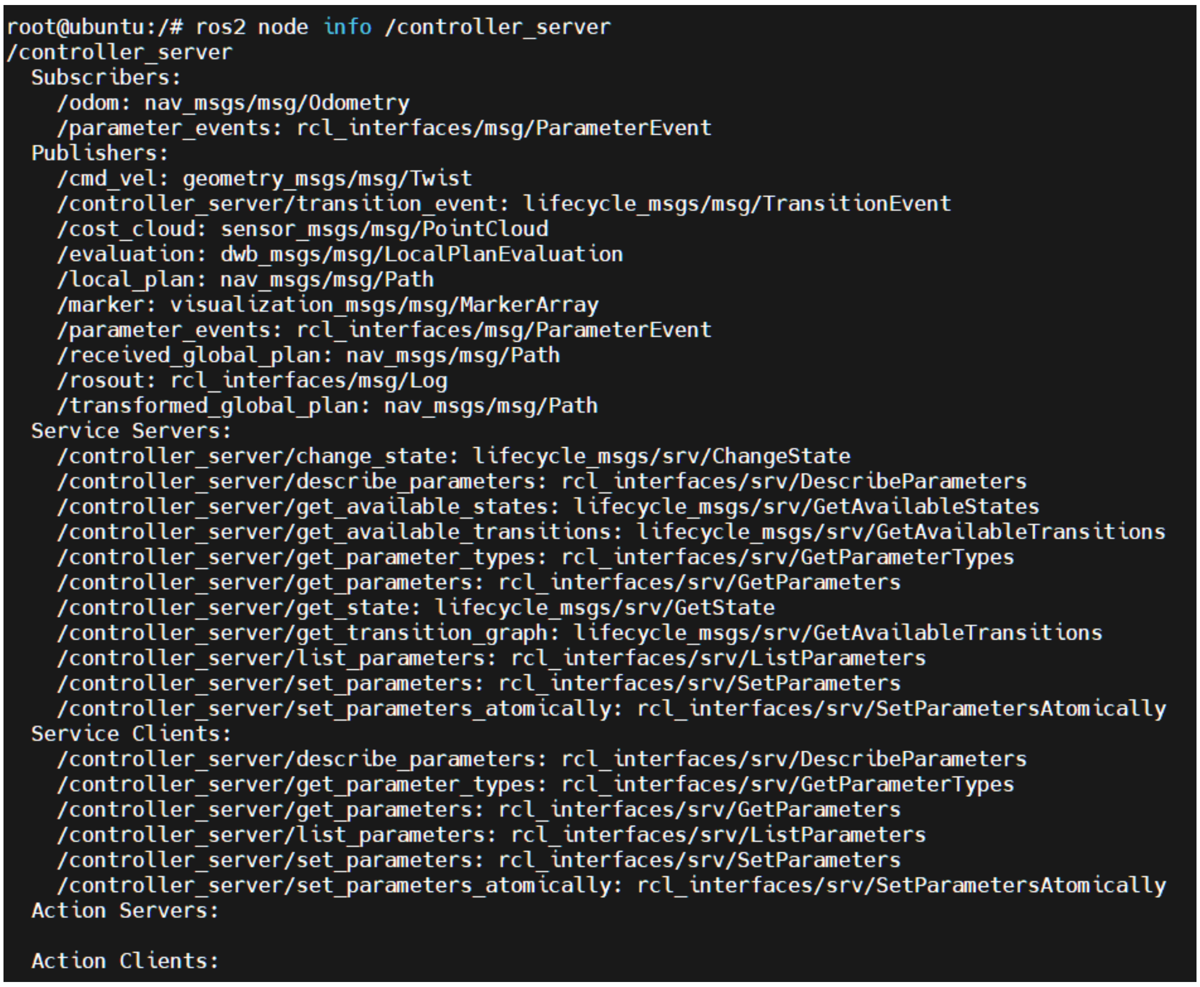
```
root@ubuntu:/# ros2 node info /amcl
/amcl
  Subscribers:
    /initialpose: geometry_msgs/msg/PoseWithCovarianceStamped
    /map: nav_msgs/msg/OccupancyGrid
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /amcl/transition_event: lifecycle_msgs/msg/TransitionEvent
    /amcl_pose: geometry_msgs/msg/PoseWithCovarianceStamped
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /particle_cloud: nav2_msgs/msg/ParticleCloud
    /particlecloud: geometry_msgs/msg/PoseArray
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /amcl/change_state: lifecycle_msgs/srv/ChangeState
    /amcl/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /amcl/get_available_states: lifecycle_msgs/srv/GetAvailableStates
    /amcl/get_available_transitions: lifecycle_msgs/srv/GetAvailableTransitions
    /amcl/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /amcl/get_parameters: rcl_interfaces/srv/GetParameters
    /amcl/get_state: lifecycle_msgs/srv/GetState
    /amcl/get_transition_graph: lifecycle_msgs/srv/GetAvailableTransitions
    /amcl/list_parameters: rcl_interfaces/srv/ListParameters
    /amcl/set_parameters: rcl_interfaces/srv/SetParameters
    /amcl/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
    /reinitialize_global_localization: std_srvs/srv/Empty
    /request_nomotion_update: std_srvs/srv/Empty
  Service Clients:

  Action Servers:

  Action Clients:
```

/bt_navigator



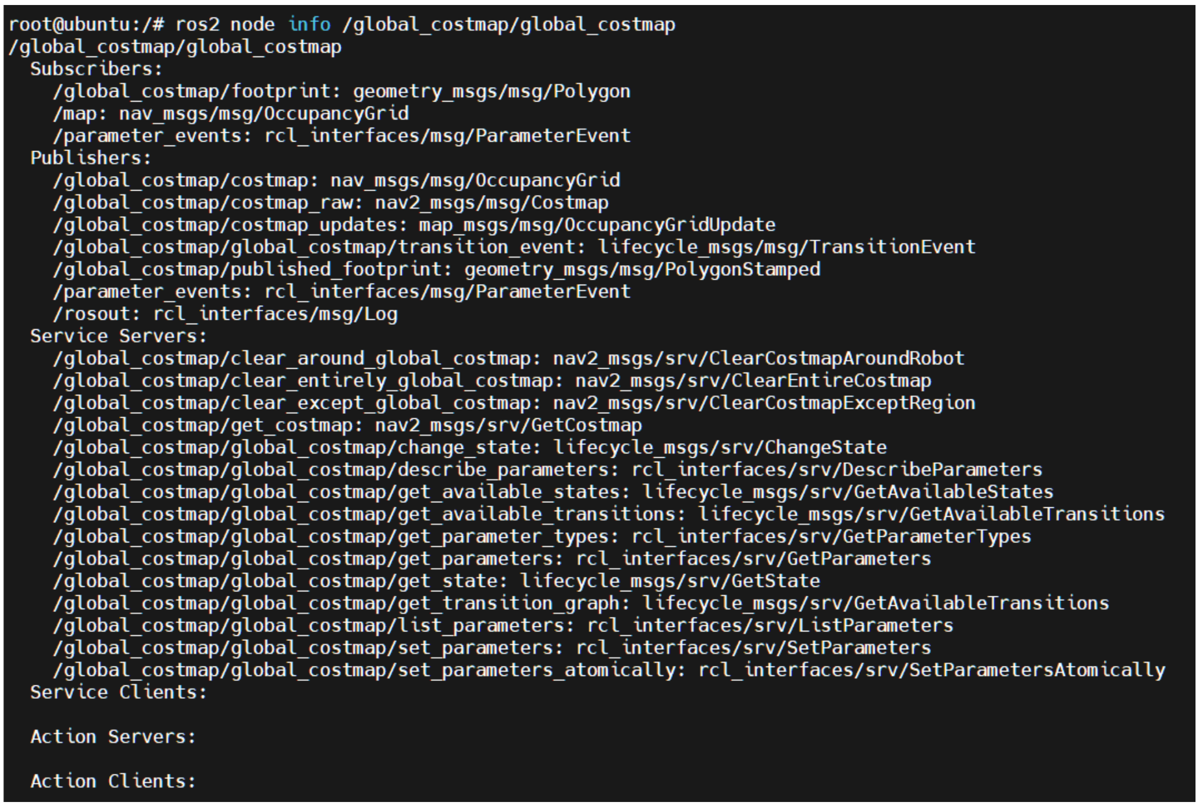
```
root@ubuntu:/# ros2 node info /bt_navigator
/bt_navigator
  Subscribers:
    /goal_pose: geometry_msgs/msg/PoseStamped
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /tf: tf2_msgs/msg/TFMessage
    /tf_static: tf2_msgs/msg/TFMessage
  Publishers:
    /bt_navigator/transition_event: lifecycle_msgs/msg/TransitionEvent
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /bt_navigator/change_state: lifecycle_msgs/srv/ChangeState
    /bt_navigator/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /bt_navigator/get_available_states: lifecycle_msgs/srv/GetAvailableStates
    /bt_navigator/get_available_transitions: lifecycle_msgs/srv/GetAvailableTransitions
    /bt_navigator/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /bt_navigator/get_parameters: rcl_interfaces/srv/GetParameters
    /bt_navigator/get_state: lifecycle_msgs/srv/GetState
    /bt_navigator/get_transition_graph: lifecycle_msgs/srv/GetAvailableTransitions
    /bt_navigator/list_parameters: rcl_interfaces/srv/ListParameters
    /bt_navigator/set_parameters: rcl_interfaces/srv/SetParameters
    /bt_navigator/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:
    /navigate_to_pose: nav2_msgs/action/NavigateToPose
  Action Clients:
```

/controller_server



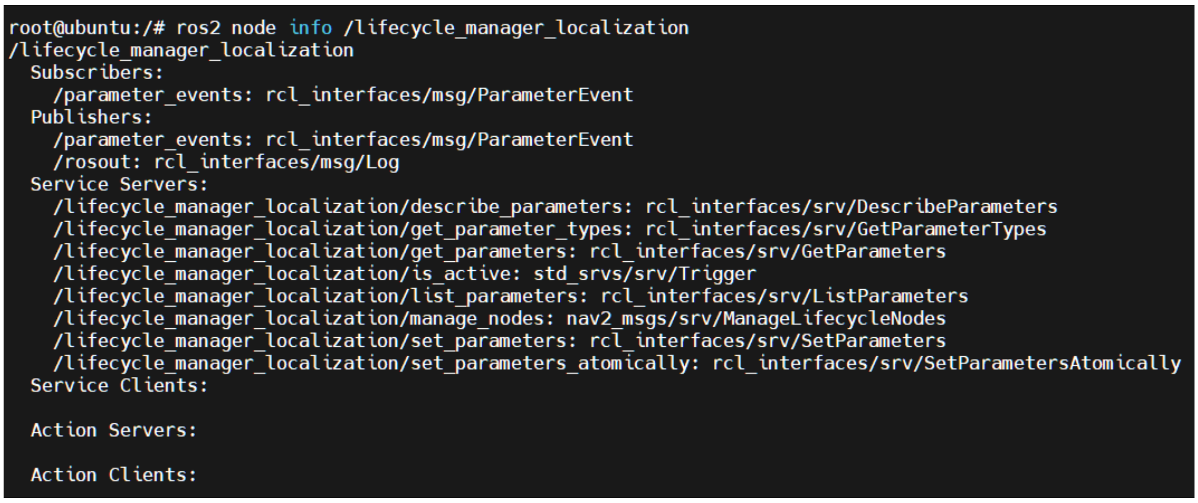
```
root@ubuntu:/# ros2 node info /controller_server
/controller_server
  Subscribers:
    /odom: nav_msgs/msg/Odometry
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /cmd_vel: geometry_msgs/msg/Twist
    /controller_server/transition_event: lifecycle_msgs/msg/TransitionEvent
    /cost_cloud: sensor_msgs/msg/PointCloud
    /evaluation: dwb_msgs/msg/LocalPlanEvaluation
    /local_plan: nav_msgs/msg/Path
    /marker: visualization_msgs/msg/MarkerArray
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /received_global_plan: nav_msgs/msg/Path
    /rosout: rcl_interfaces/msg/Log
    /transformed_global_plan: nav_msgs/msg/Path
  Service Servers:
    /controller_server/change_state: lifecycle_msgs/srv/ChangeState
    /controller_server/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /controller_server/get_available_states: lifecycle_msgs/srv/GetAvailableStates
    /controller_server/get_available_transitions: lifecycle_msgs/srv/GetAvailableTransitions
    /controller_server/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /controller_server/get_parameters: rcl_interfaces/srv/GetParameters
    /controller_server/get_state: lifecycle_msgs/srv/GetState
    /controller_server/get_transition_graph: lifecycle_msgs/srv/GetAvailableTransitions
    /controller_server/list_parameters: rcl_interfaces/srv/ListParameters
    /controller_server/set_parameters: rcl_interfaces/srv/SetParameters
    /controller_server/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:
    /controller_server/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /controller_server/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /controller_server/get_parameters: rcl_interfaces/srv/GetParameters
    /controller_server/list_parameters: rcl_interfaces/srv/ListParameters
    /controller_server/set_parameters: rcl_interfaces/srv/SetParameters
    /controller_server/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Action Servers:

  Action Clients:
```

/global_costmap/global_costmap



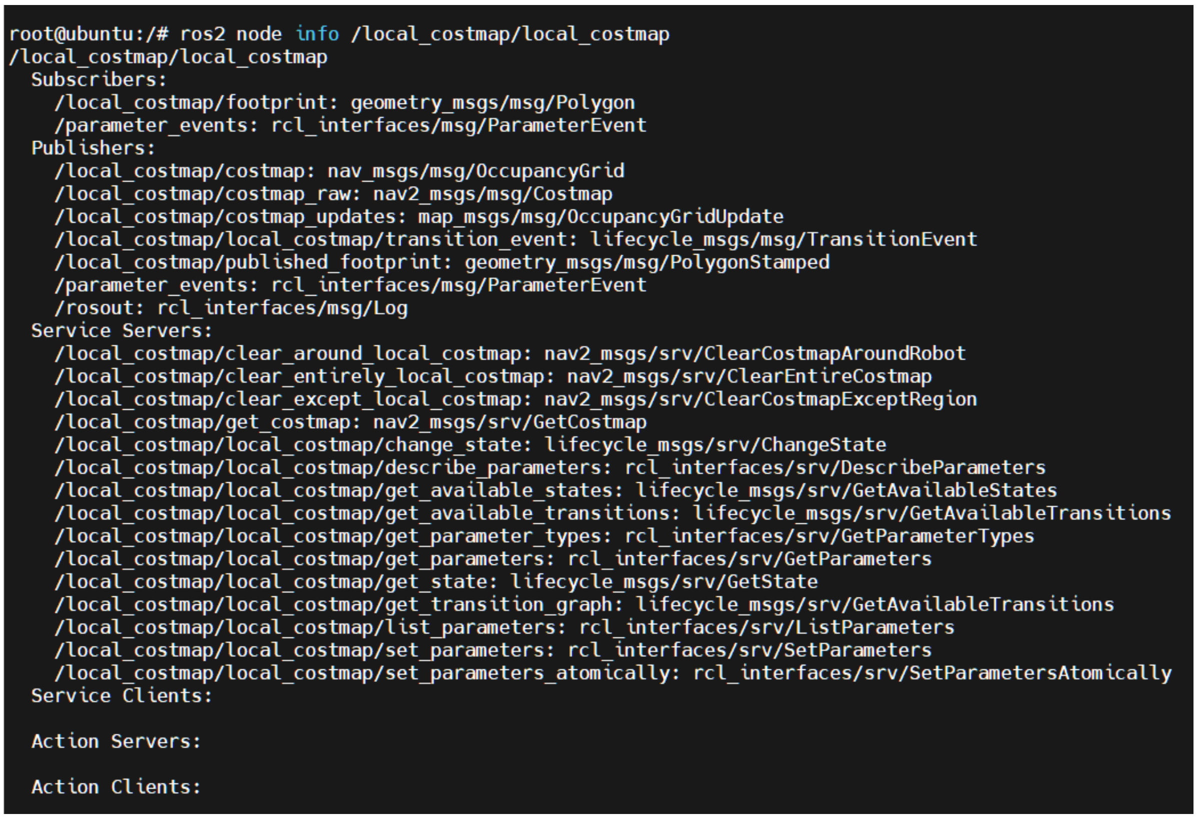
```
root@ubuntu:/# ros2 node info /global_costmap/global_costmap
/global_costmap/global_costmap
  Subscribers:
    /global_costmap/footprint: geometry_msgs/msg/Polygon
    /map: nav_msgs/msg/OccupancyGrid
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /global_costmap/costmap: nav_msgs/msg/OccupancyGrid
    /global_costmap/costmap_raw: nav2_msgs/msg/Costmap
    /global_costmap/costmap_updates: map_msgs/msg/OccupancyGridUpdate
    /global_costmap/global_costmap/transition_event: lifecycle_msgs/msg/TransitionEvent
    /global_costmap/published_footprint: geometry_msgs/msg/PolygonStamped
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /global_costmap/clear_around_global_costmap: nav2_msgs/srv/ClearCostmapAroundRobot
    /global_costmap/clear_entirely_global_costmap: nav2_msgs/srv/ClearEntireCostmap
    /global_costmap/clear_except_global_costmap: nav2_msgs/srv/ClearCostmapExceptRegion
    /global_costmap/get_costmap: nav2_msgs/srv/GetCostmap
    /global_costmap/global_costmap/change_state: lifecycle_msgs/srv/ChangeState
    /global_costmap/global_costmap/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /global_costmap/global_costmap/get_available_states: lifecycle_msgs/srv/GetAvailableStates
    /global_costmap/global_costmap/get_available_transitions: lifecycle_msgs/srv/GetAvailableTransitions
    /global_costmap/global_costmap/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /global_costmap/global_costmap/get_parameters: rcl_interfaces/srv/GetParameters
    /global_costmap/global_costmap/get_state: lifecycle_msgs/srv/GetState
    /global_costmap/global_costmap/get_transition_graph: lifecycle_msgs/srv/GetAvailableTransitions
    /global_costmap/global_costmap/list_parameters: rcl_interfaces/srv/ListParameters
    /global_costmap/global_costmap/set_parameters: rcl_interfaces/srv/SetParameters
    /global_costmap/global_costmap/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:

  Action Clients:
```

/lifecycle_manager_localization



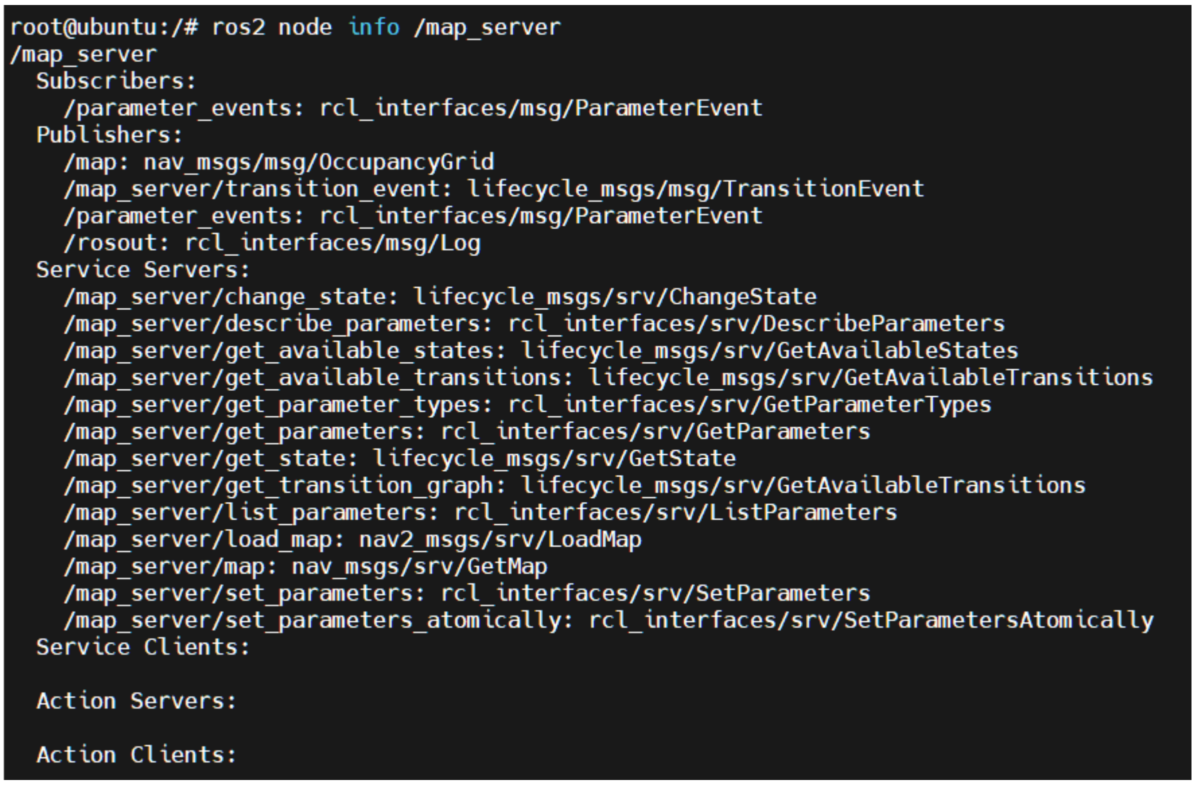
```
root@ubuntu:/# ros2 node info /lifecycle_manager_localization
/lifecycle_manager_localization
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /lifecycle_manager_localization/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /lifecycle_manager_localization/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /lifecycle_manager_localization/get_parameters: rcl_interfaces/srv/GetParameters
    /lifecycle_manager_localization/is_active: std_srvs/srv/Trigger
    /lifecycle_manager_localization/list_parameters: rcl_interfaces/srv/ListParameters
    /lifecycle_manager_localization/manage_nodes: nav2_msgs/srv/ManageLifecycleNodes
    /lifecycle_manager_localization/set_parameters: rcl_interfaces/srv/SetParameters
    /lifecycle_manager_localization/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:

  Action Clients:
```

/local_costmap/local_costmap

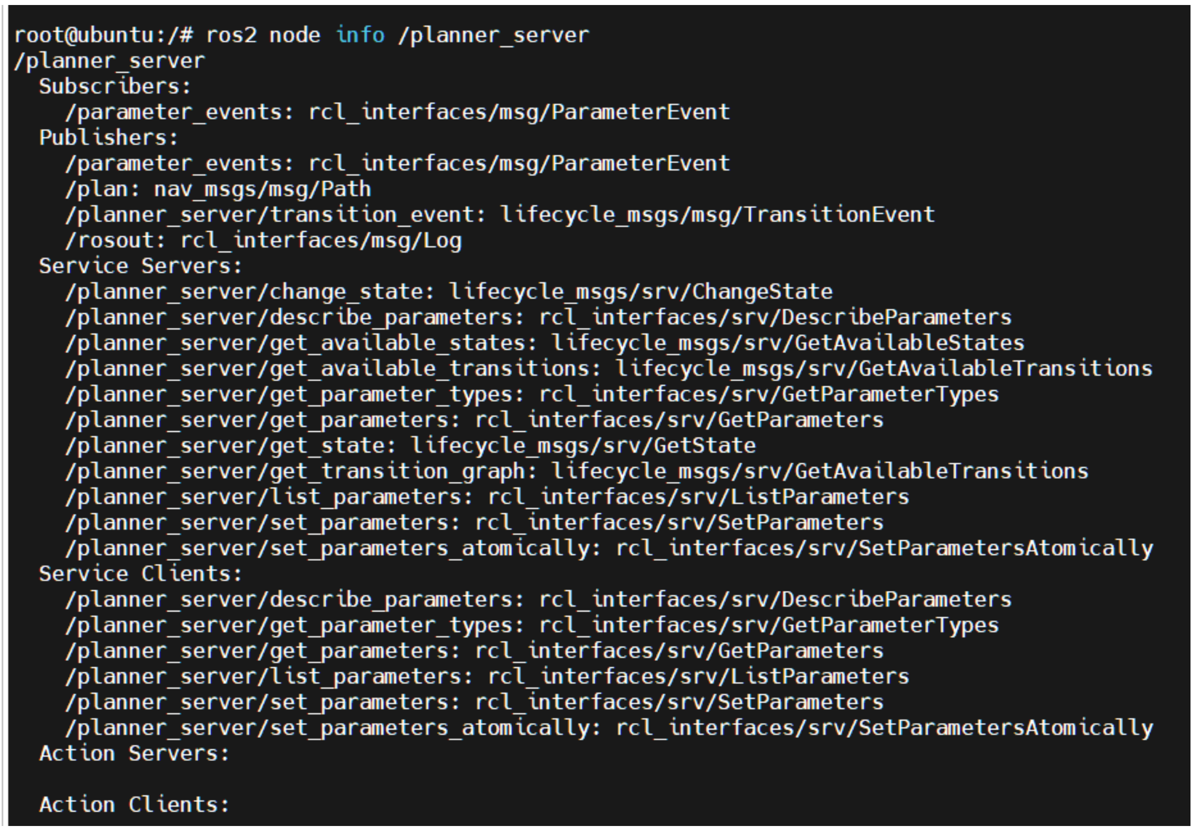


```
root@ubuntu:/# ros2 node info /local_costmap/local_costmap
/local_costmap/local_costmap
  Subscribers:
    /local_costmap/footprint: geometry_msgs/msg/Polygon
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /local_costmap/costmap: nav_msgs/msg/OccupancyGrid
    /local_costmap/costmap_raw: nav2_msgs/msg/Costmap
    /local_costmap/costmap_updates: map_msgs/msg/OccupancyGridUpdate
    /local_costmap/local_costmap/transition_event: lifecycle_msgs/msg/TransitionEvent
    /local_costmap/published_footprint: geometry_msgs/msg/PolygonStamped
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /local_costmap/clear_around_local_costmap: nav2_msgs/srv/ClearCostmapAroundRobot
    /local_costmap/clear_entirely_local_costmap: nav2_msgs/srv/ClearEntireCostmap
    /local_costmap/clear_except_local_costmap: nav2_msgs/srv/ClearCostmapExceptRegion
    /local_costmap/get_costmap: nav2_msgs/srv/GetCostmap
    /local_costmap/local_costmap/change_state: lifecycle_msgs/srv/ChangeState
    /local_costmap/local_costmap/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /local_costmap/local_costmap/get_available_states: lifecycle_msgs/srv/GetAvailableStates
    /local_costmap/local_costmap/get_available_transitions: lifecycle_msgs/srv/GetAvailableTransitions
    /local_costmap/local_costmap/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /local_costmap/local_costmap/get_parameters: rcl_interfaces/srv/GetParameters
    /local_costmap/local_costmap/get_state: lifecycle_msgs/srv/GetState
    /local_costmap/local_costmap/get_transition_graph: lifecycle_msgs/srv/GetAvailableTransitions
    /local_costmap/local_costmap/list_parameters: rcl_interfaces/srv/ListParameters
    /local_costmap/local_costmap/set_parameters: rcl_interfaces/srv/SetParameters
    /local_costmap/local_costmap/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:

  Action Clients:
```

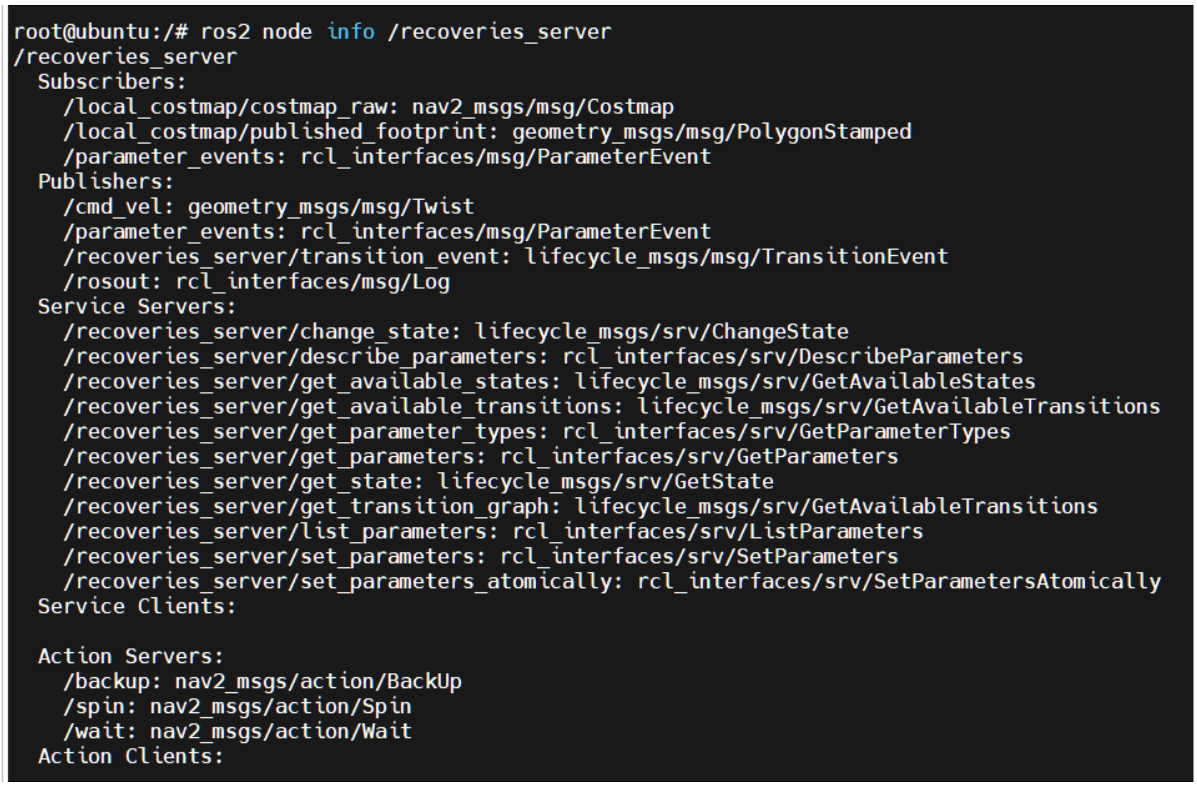
/map_server


```
root@ubuntu:/# ros2 node info /map_server
/map_server
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /map: nav_msgs/msg/OccupancyGrid
    /map_server/transition_event: lifecycle_msgs/msg/TransitionEvent
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /map_server/change_state: lifecycle_msgs/srv/ChangeState
    /map_server/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /map_server/get_available_states: lifecycle_msgs/srv/GetAvailableStates
    /map_server/get_available_transitions: lifecycle_msgs/srv/GetAvailableTransitions
    /map_server/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /map_server/get_parameters: rcl_interfaces/srv/GetParameters
    /map_server/get_state: lifecycle_msgs/srv/GetState
    /map_server/get_transition_graph: lifecycle_msgs/srv/GetAvailableTransitions
    /map_server/list_parameters: rcl_interfaces/srv/ListParameters
    /map_server/load_map: nav2_msgs/srv/LoadMap
    /map_server/map: nav_msgs/srv/GetMap
    /map_server/set_parameters: rcl_interfaces/srv/SetParameters
    /map_server/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:

  Action Clients:
```

/planner_server

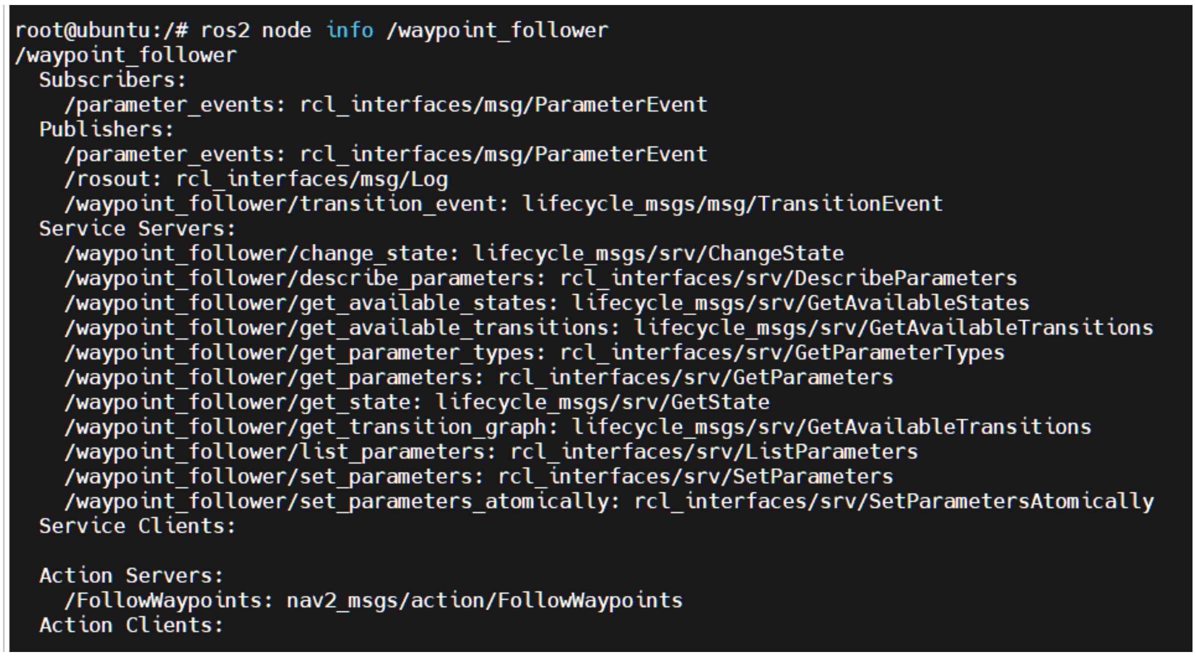

```
root@ubuntu:/# ros2 node info /planner_server
/planner_server
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /plan: nav_msgs/msg/Path
    /planner_server/transition_event: lifecycle_msgs/msg/TransitionEvent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /planner_server/change_state: lifecycle_msgs/srv/ChangeState
    /planner_server/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /planner_server/get_available_states: lifecycle_msgs/srv/GetAvailableStates
    /planner_server/get_available_transitions: lifecycle_msgs/srv/GetAvailableTransitions
    /planner_server/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /planner_server/get_parameters: rcl_interfaces/srv/GetParameters
    /planner_server/get_state: lifecycle_msgs/srv/GetState
    /planner_server/get_transition_graph: lifecycle_msgs/srv/GetAvailableTransitions
    /planner_server/list_parameters: rcl_interfaces/srv/ListParameters
    /planner_server/set_parameters: rcl_interfaces/srv/SetParameters
    /planner_server/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:
    /planner_server/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /planner_server/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /planner_server/get_parameters: rcl_interfaces/srv/GetParameters
    /planner_server/list_parameters: rcl_interfaces/srv/ListParameters
    /planner_server/set_parameters: rcl_interfaces/srv/SetParameters
    /planner_server/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Action Servers:

  Action Clients:
```

/recoveries_server



```
root@ubuntu:/# ros2 node info /recoveries_server
/recoveries_server
  Subscribers:
    /local_costmap/costmap_raw: nav2_msgs/msg/Costmap
    /local_costmap/published_footprint: geometry_msgs/msg/PolygonStamped
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /cmd_vel: geometry_msgs/msg/Twist
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /recoveries_server/transition_event: lifecycle_msgs/msg/TransitionEvent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /recoveries_server/change_state: lifecycle_msgs/srv/ChangeState
    /recoveries_server/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /recoveries_server/get_available_states: lifecycle_msgs/srv/GetAvailableStates
    /recoveries_server/get_available_transitions: lifecycle_msgs/srv/GetAvailableTransitions
    /recoveries_server/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /recoveries_server/get_parameters: rcl_interfaces/srv/GetParameters
    /recoveries_server/get_state: lifecycle_msgs/srv/GetState
    /recoveries_server/get_transition_graph: lifecycle_msgs/srv/GetAvailableTransitions
    /recoveries_server/list_parameters: rcl_interfaces/srv/ListParameters
    /recoveries_server/set_parameters: rcl_interfaces/srv/SetParameters
    /recoveries_server/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:
    /backup: nav2_msgs/action/BackUp
    /spin: nav2_msgs/action/Spin
    /wait: nav2_msgs/action/Wait
  Action Clients:
```

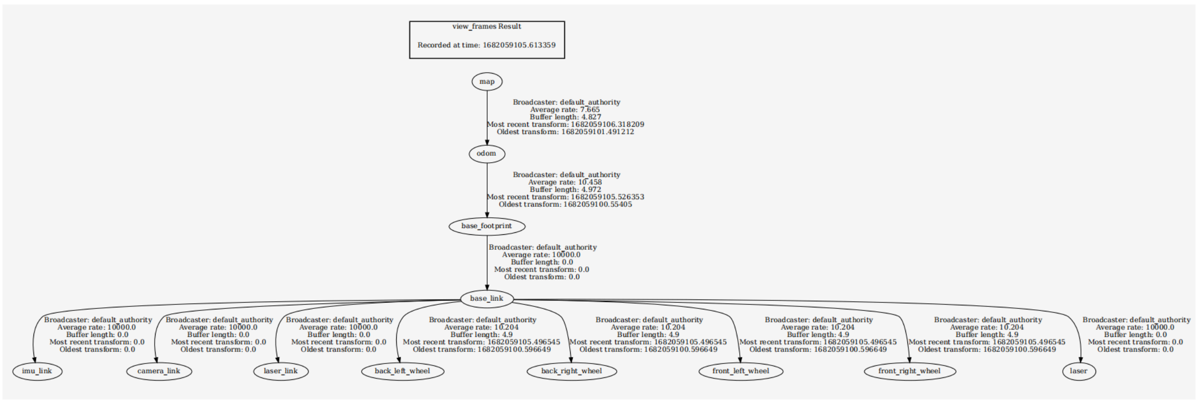
/waypoint_follower



```
root@ubuntu:/# ros2 node info /waypoint_follower
/waypoint_follower
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
    /waypoint_follower/transition_event: lifecycle_msgs/msg/TransitionEvent
  Service Servers:
    /waypoint_follower/change_state: lifecycle_msgs/srv/ChangeState
    /waypoint_follower/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /waypoint_follower/get_available_states: lifecycle_msgs/srv/GetAvailableStates
    /waypoint_follower/get_available_transitions: lifecycle_msgs/srv/GetAvailableTransitions
    /waypoint_follower/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /waypoint_follower/get_parameters: rcl_interfaces/srv/GetParameters
    /waypoint_follower/get_state: lifecycle_msgs/srv/GetState
    /waypoint_follower/get_transition_graph: lifecycle_msgs/srv/GetAvailableTransitions
    /waypoint_follower/list_parameters: rcl_interfaces/srv/ListParameters
    /waypoint_follower/set_parameters: rcl_interfaces/srv/SetParameters
    /waypoint_follower/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:
    /FollowWaypoints: nav2_msgs/action/FollowWaypoints
  Action Clients:
```

8.3.3、TF transform

```
ros2 run tf2_tools view_frames.py
```


view_frames Result
Recorded at time: 1682059105.613359
map
Broadcaster: default_authority
Average rate: 7.665
Buffer length: 4.827
Most recent transform: 1682059106.318209
Oldest transform: 1682059101.491212
odom
Broadcaster: default_authority
Average rate: 10.458
Buffer length: 4.972
Most recent transform: 1682059105.526353
Oldest transform: 1682059100.55405
base_footprint
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
base_link
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
imu_link
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
camera_link
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
laser_link
Broadcaster: default_authority
Average rate: 10.204
Buffer length: 4.9
Most recent transform: 1682059105.496545
Oldest transform: 1682059100.596649
back_left_wheel
Broadcaster: default_authority
Average rate: 10.204
Buffer length: 4.9
Most recent transform: 1682059105.496545
Oldest transform: 1682059100.596649
back_right_wheel
Broadcaster: default_authority
Average rate: 10.204
Buffer length: 4.9
Most recent transform: 1682059105.496545
Oldest transform: 1682059100.596649
front_left_wheel
Broadcaster: default_authority
Average rate: 10.204
Buffer length: 4.9
Most recent transform: 1682059105.496545
Oldest transform: 1682059100.596649
front_right_wheel
Broadcaster: default_authority
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
laser

8.4、Navigation2 details

- **Comparison of navigation frames in ROS 1 and ROS 2**

In Nav 2, move_base is split into multiple components. Unlike a single state machine (ROS1), Nav 2 separates ideas by leveraging the low-latency, reliable communication of the Action Server and ROS 2. The Behavior Tree is used to orchestrate and organize these tasks, which allows Nav 2 to have highly configurable navigation behavior through the Behavior Tree XML file without the need to program the task.

nav2_bt_navigator replaces the move_base at the top level, which uses an Action interface to complete the navigation tasks of a tree-based action model. Implement more complex state machines via BT and add recovery behavior as additional Action Servers. These behavior trees are configurable XML.

The planning, recovery, and controller servers are also action servers that the BT Navigator can call for calculations. All 3 servers can host many plugins for many algorithms, each of which can individually invoke specific behaviors from the navigation behavior tree. The default plugins provided are ported from ROS 1, namely: DWB, NavFn and similar recovery behaviors such as rotating and clearing costmaps, plus a new recovery mechanism that waits for a fixed time. These servers are called from the BT Navigator through their action servers to calculate results or complete tasks. The state is maintained by the BT Navigator behavior tree. All these changes make it possible to replace these nodes at startup/runtime by implementing any algorithm of the same interface.

Porting kit:

- AMCL: Ported to nav2_amcl
- map_server: Transplant to nav2_map_server
- nav2_planner: Replace global_planner and manage 'N' planner plugins
- nav2_controlle: Replace local_planner and manage 'N' controller plugins
- Navfn: Ported to nav2_navfn_planner
- DWB: Replaces DWA and ported to the nav2_dwb_controller metapackage in ROS 2
- nav_core: Ported to nav2_core and updated interfaces
- costmap_2d: Transplant to nav2_costmap_2d

New Package:

- Nav2_bt_navigator: Replace move_base state machine
- nav2_lifecycle_manager: Handles the life cycle of the server program

- nav2_waypoint_follower: Complex tasks can be performed through many waypoints
- nav2_system_tests: A set of basic tutorials in integration testing and simulation for CI
- nav2_rviz_plugins: An rviz plugin that controls the Navigation2 server, commands, cancellation, and navigation
- nav2_experimental: Experimental (and incomplete) work of deep reinforcement learning controllers
- navigation2_behavior_trees: A wrapper for calling the behavior tree library of the ROS action server

8.4.1、amcl

The full name of amcl in English is adaptive Monte Carlo localization, which is a probabilistic positioning system for two-dimensional mobile robots. It's an upgraded version of the Monte Carlo positioning method, using an adaptive KLD method to update particles and a particle filter to track the robot's pose based on a known map. As currently implemented, this node is only suitable for laser scanning and laser maps. It can be extended to handle other sensor data. AMCL receives laser-based maps, laser scans, and transformation information, and outputs pose estimates. On startup, AMCL initializes its particle filter based on the provided parameters. Note that due to the default settings, if no parameters are set, the initial filter state will be a medium-sized particle cloud centered on (0,0,0).

Monte Carlo

- Monte Carlo method, also known as statistical simulation method, statistical test method, is an idea or method. It is a numerical simulation method that uses probability phenomena as research objects. It is a calculation method that estimates unknown characteristic quantities by obtaining statistical values according to the sampling survey method.
- For example, if there is an irregular shape in a rectangle, how to calculate the area of the irregular shape? It's not easy to count. But we can approximate. Take a bunch of beans, sprinkle them evenly on the rectangle, and then count the number of beans in the irregular shape and the number of beans in the remaining places. The rectangular area is known, so the irregularly shaped area is obtained by estimating. In terms of robot positioning, it is possible that it is located in any location on the map, how can we express the confidence of a location in this case? We also use particles, where there are more particles, it means that the probability of where the robot is is high.

The greatest advantage of Monte Carlo

- The error of the method is independent of the dimensionality of the problem.
- Problems of a statistical nature can be solved directly.
- Discretization is not necessary for continuity problems

Disadvantages of Monte Carlo Law

- For deterministic problems, it needs to be converted into random problems.
- Error is probability error.
- Usually requires more counting steps N.

Particle filtering

- The number of particles represents the likelihood of something. Through some kind of evaluation method (evaluating the likelihood of this thing), the distribution of particles is changed. For example, in robot positioning, a certain particle A, I think the probability of this particle in this coordinate (for example, this coordinate belongs to the "this thing" mentioned earlier), then I give him a high score. The next time you reposition all the particles, arrange more near this position. After a few more rounds in this way, the particles will all concentrate in the position with high probability.

Adaptive Monte Carlo

- Solved the robot kidnapping problem that will re-scatter some particles globally when it finds that the average score of the particles has suddenly decreased (meaning that the correct particle was discarded in an iteration).
- Solved the problem of fixed particle number, because sometimes when the robot positioning is almost obtained, such as these particles are concentrated together, there is no need to maintain so many particles, this time the number of particles can be less.

8.4.2、costmaps and layers

The current environmental representation is a cost map. A cost map is a grid of regular 2D cells containing cells from unknown, idle, occupied, or bloated costs. This cost map is then looked for to calculate global plans or sampling to calculate local control efforts.
Various cost map layers are implemented as pluginlib plugins to buffer information into cost maps. This includes information from LIDAR, RADAR, sonar, depth sensors, image sensors, and more. It is best to process sensor data before it is entered into the layer map, but it is up to the developer.
You can use a camera or depth sensor to create a cost map layer to detect and track obstacles in your scene to avoid collisions. In addition, layers can be created to vary the underlying cost plot based on some rules or heuristics. Finally, they can be used to buffer real-time data into the 2D or 3D world for binarized labeling of obstacles.

8.4.3、planner_server

The task of the planner is to calculate the path based on a specific objective function. Depending on the nomenclature and algorithm chosen, a path is also called an itinerary route. There are two typical examples, one is to calculate a path plan to the target location (e.g. from the current location to a target pose) or completely override the complete coverage (e.g. a path plan that covers all free space). The planner has access to cached data for global environment representations and sensors. The planner can have the following features:

- Calculates the shortest path
- Calculate the full coverage path
- Calculates routes along sparse or predefined routes

The general task of the planner in Nav2 is to calculate an efficient and possibly optimal path from the current position to the target pose.

The algorithm plug-in in Planning Server uses the environment information captured by different sensors to search for the path of the bot. Some of these algorithms obtain paths by searching for environment rasters, while others extend the possible states of the robot by considering the feasibility of the paths.

As mentioned earlier, Planning Server uses plugins that work on grid spaces, such as NavFn Planner, Smac Planner 2D, and Theta Star Planner. NavFn planner is a navigation function planner that uses the Dijkstra or A algorithm. Smac 2D planner implements a 2D A algorithm with 4 or 8 neighbors connected and with smoother and multi-resolution characteristics. Theta Star planner (smooth arbitrari-angle path planning in a continuous environment, Theta is a variant of A, it propagates information along the edges of the graph, but does not constrain the path to the edges of the graph (looking for paths at "arbitrary angles))) is achieved using a line-of-sight algorithm by creating non-discrete directional path segments. Commonly used planners are as follows:

Plugin Name	**Supported bot types**
NavFn Planner	Circular differential, circular omnidirectional
Smac2D Planner	Circular differential, circular omnidirectional
Theta Star Planner	Circular differential, circular omnidirectional
Smac mix A* Planner	Non-round or round Ackerman, non-round or round leg type
Smac Lattice Planner	Non-circular microvelocity, non-circular omnidirectional

8.4.4、controller_server

The controller, also known as the local planner in ROS 1, completes the following of the global calculation path or completes a local task. The controller has access to the local environment representation and attempts to calculate a possible path for the reference path to follow. Many controllers project the robot forward into space and calculate locally feasible paths with each update iteration. The controller has the following functions:

- Follow a path
- Use the detector to complete the docking with the charging pile in the mileage coordinate system
- Board an elevator
- Interface with a tool

In Nav2, the general task of the controller is to calculate an effective control effort to follow the global planning path. However, there are many controller classes and local planner classes. The goal of the Nav2 project is that all controller algorithms can be used as plug-ins in this server for general research and industrial tasks.

Common controller plug-ins are DWA Controller and TEB Controller

1、DWA Controller
The idea of DWA algorithm is to sample different possible speeds, and then simulate different trajectories, use the distance to the target point, obstacle distance and time to evaluate, select the trajectory with the best score, and issue speed commands to control the trolley forward.
The speed sampling simulated by the DWA algorithm is not randomly selected, but there is a range, the size of the speed is limited by the maximum speed and minimum speed of the trolley, the size of the acceleration is affected by the performance of the trolley motor, in addition, in order to ensure safety, the distance of the obstacle will also affect the value of the speed.

Differential trolley, wheat wheel car in the adjacent time, the movement distance is short, can see the motion trajectory between the adjacent two points, as a straight line, the motion trajectory of other models as an arc.

Three evaluation functions:

- heading(v, w): azimuth evaluation function, the angle difference between the car and the target, the smaller the angle difference, the higher the score
- dist(v, w): The distance between the trolley and the nearest obstacle, the farther the distance, the higher the score
- velocity (v, w): The speed corresponding to the trajectory, the higher the speed, the higher the score

Physical meaning: make the car towards the target point, avoid obstacles, and drive quickly

2、TEB Controller
The idea of TEB algorithm is to regard the path connecting the starting point and the end point as similar to a rubber band that can be deformed, and then treat the external constraint as an external force to deform the path.

- Follow Path + Obstacle Avoidance: The constraint has two main goals, following a consistent global path planning and obstacle avoidance. Both objective functions are very similar. Follow the path to apply external force to pull the local path to the global path, and the obstacle avoidance constraint force makes the local path away from the obstacle
- Velocity/acceleration constraints: Speed and acceleration should be within a certain range
- Kinematic constraints: a smooth trajectory composed of several arcs, the control quantity is only linear and angular velocity, the Ackermann structure has a minimum turning radius, and the wheat wheel/omnidirectional/differential speed is 0
- Fastest Path Constraint: The objective function allows the robot to obtain the fastest path, and the poses on the path are evenly separated in time, instead of the shortest path in traditional space

TEB can be formulated as a multi-objective optimization problem, most of which are local and related to only a small set of parameters, since they depend on only a few consecutive robot states. After determining the location of N control points, the path is optimized using the open source framework g2o (General Graph Optimization) general graph optimization method. In the case of the trolley, all pose points, time intervals, obstacles, etc. are described as points, and constraints are described as edges. The minimum distance from an obstacle connects the obstacle to the pose point, and the velocity constraint connects two adjacent pose points with their time difference. In general, the local trajectory generated by TEB is composed of a series of discrete poses with time information, and the goal optimized by the g2o algorithm is that the trajectory composed of these discrete poses can finally reach the shortest time, shortest distance, away from obstacles and other goals, while limiting the speed and acceleration to make the trajectory meet the robot dynamics.

8.4.5、recoveries_server

Recovery Behaviors are the core module of the fault-tolerant system in robot navigation. Their goal is to autonomously handle abnormal states or environmental changes during navigation through predefined behavior logic. The Humble version of `behavior_server` is based on the **Behavior Tree** architecture, encapsulating recovery behaviors as pluggable nodes, supporting dynamic combination and priority scheduling, thereby achieving a more flexible fault recovery strategy. **Examples include sensor failures in the perception system that cause the environmental representation (such as the cost map) to be filled with false obstacles**. Such failures trigger the `ClearCostmap` node in the behavior tree, calling the service to clear the obstacle layer of the local or global cost map, and restore the accuracy of the environmental representation to allow the robot to continue moving. **Another example is that the robot is stuck due to dynamic obstacle blocking, path planner failure, or control error accumulation**. At this point, the behavior tree will execute a recovery sequence: first, the `BackUp` node is used to control the robot to move a specified distance (such as 0.5 meters) in the opposite direction along the trajectory. If it still cannot escape, the `Spin` node is triggered to rotate the robot in place by a specific angle (such as 180°) to reset the local planning state and try to enter the free space. **Finally, when multiple recovery failures or hardware-level failures (such as motor communication interruption and navigation stack crash) are detected**, the behavior tree will execute a safe recovery strategy: send an alarm notification to the operator through the `NotifyOperator` node (integrated with Slack, Telegram, or email services), and call the `EmergencyStop` node to forcefully stop all motion controllers and switch to safe standby mode to wait for human intervention.

8.4.6、waypoint following

Waypoint following is one of the basic functions of a navigation system. It tells the system how to use the navigator to reach multiple destinations. The nav2_waypoint_follower package contains a waypoint tracker that has a plugin interface for specific executors. This is useful if you need to have the robot travel to position and complete specific tasks like taking a picture, picking up a box, or waiting for user input. This is a nice demo application to show how to use Nav2 in a sample application.

However, the package can be used not only for sample applications. There are two schools of thought about robot fleet manager/scheduler: dumb robot + intelligent centralized scheduler; Smart robot + dumb centralized scheduler.

In the first idea, nav2_waypoint_follower software packages are sufficient to create a production-level robotic solution. Since autonomous systems/schedulers take into account factors such as the robot's posture, battery level, current task, etc. when assigning tasks, applications on robots only need to care about the task at hand and not with other complexities of completing the task that the system requires. In this case, a request sent to a waypoint follower should be treated as 1 unit of work (e.g., 1 pick in the warehouse, 1 safety patrol loop, 1 aisle, etc.) to perform the task, and then return to the scheduler for the next task or request charging. In this school of thought, waypoint following applications are just a step above the navigation software stack and below the system's autonomous applications.

In the second thought, the nav2_waypoint_follower package is a nice example application/proof of concept, but it does require a waypoint tracking/autonomous system on a robot to take on more tasks to develop a robust solution. In this case, you should use the nav2_behavior_tree package to create a custom application-level behavior tree to use navigation to complete the task. This can contain subtrees, such as checking the charging status in a task to return to the dock, or handling more than 1 unit of work in a more complex task. Soon, there will be an nav2_bt_waypoint_follower (name to be adjusted) that will make it easier for users to create this application. In this school of thought, waypoint following applications are more closely tied to autonomous systems, or in many cases, waypoint following applications are the autonomous system itself.

These two schools of thought can't simply say who is better than whom, and who is better depends largely on what task the bot is accomplishing, what type of environment it is in, and what cloud resources are available. Often, for a given business case, this distinction is very obvious.

8.4.7、bt_navigator

The BT Navigator (Behavior Tree Navigator) module implements the NavigateToPose task interface. It is a behavior tree-based navigation implementation designed to allow flexibility in navigation tasks and provide a way to easily specify complex bot behaviors, including recovery.

Planer Server, Controller Server, and Recovery Server each implement their respective functions, namely global path planning, local path planning, and recovery operations. But their functions are independent and do not interfere with each other. To achieve a complete navigation function, the cooperation of various modules is required. BT Navigator Server was the one who assembled the Lego bricks.

For complete parameter configuration, see:

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/params/dwa_nav_params.yaml
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/params/teb_nav_params.yaml
```