Lidar following

Note: The ROS_DOMAIN_ID of the Raspberry Pi and the microROS control board need to be consistent. You can check [MicroROS Control Board Parameter Configuration] to set the microROS control board ROS_DOMAIN_ID. Check the tutorial [Connect MicroROS Agent] to determine whether the IDs are consistent.

1、Program function description

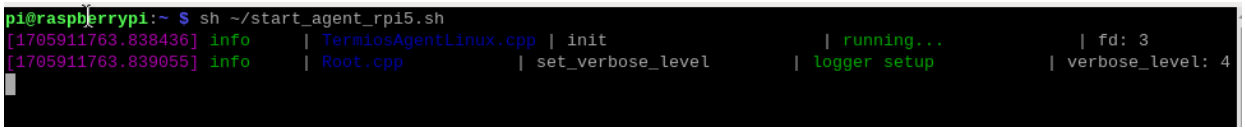
The car connects to the agent and runs the program. The radar on the car scans the nearest object within the set range, and adjusts its own speed based on the set tracking distance to maintain a certain distance from the object. Parameters such as the radar detection range and obstacle avoidance detection distance can be adjusted through the dynamic parameter adjuster.

2、Query car information

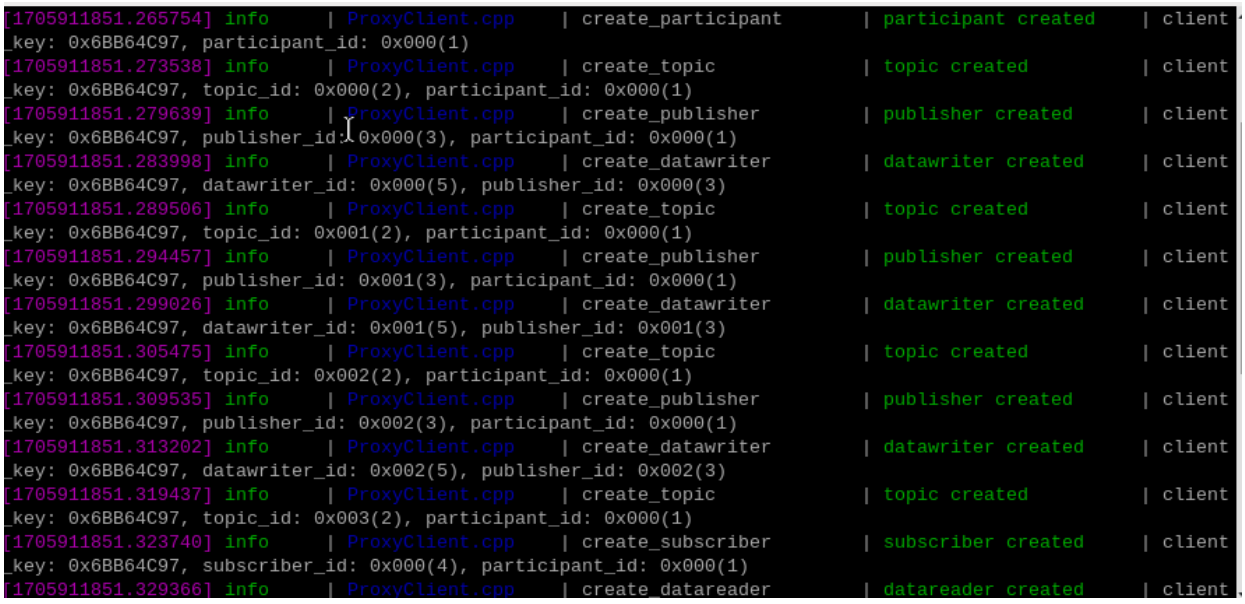
2.1、Start and connect to the agent

After the Raspberry Pi is successfully powered on, open the terminal and enter the following command to open the agent.

```
sh ~/start_agent_rpi5.sh
```



Press the reset button on the microROS control board and wait for the car to connect to the agent. The connection is successful as shown in the figure below.

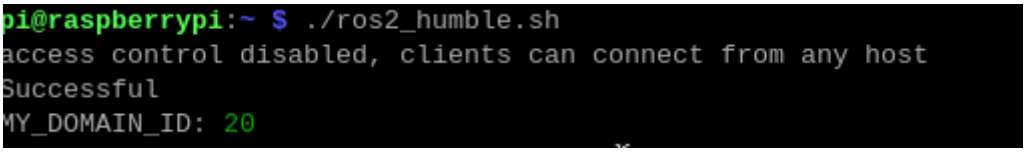


2.2、Enter the car docker

Open another terminal and enter the following command to enter docker.

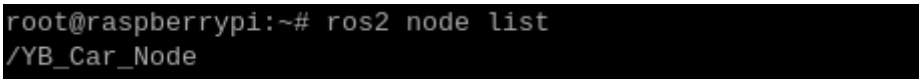
```
sh ros2_humble.sh
```

When the following interface appears, you have successfully entered docker. Now you can control the car through commands.



Enter the following command in the terminal to query the agent node.

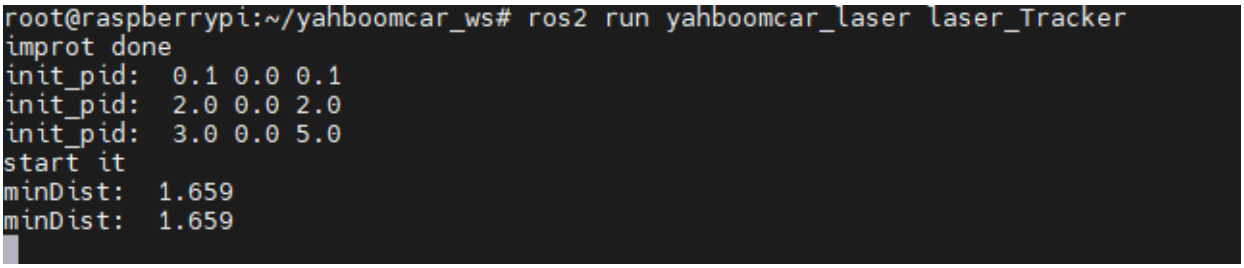
```
ros2 node list
```



3、starting program

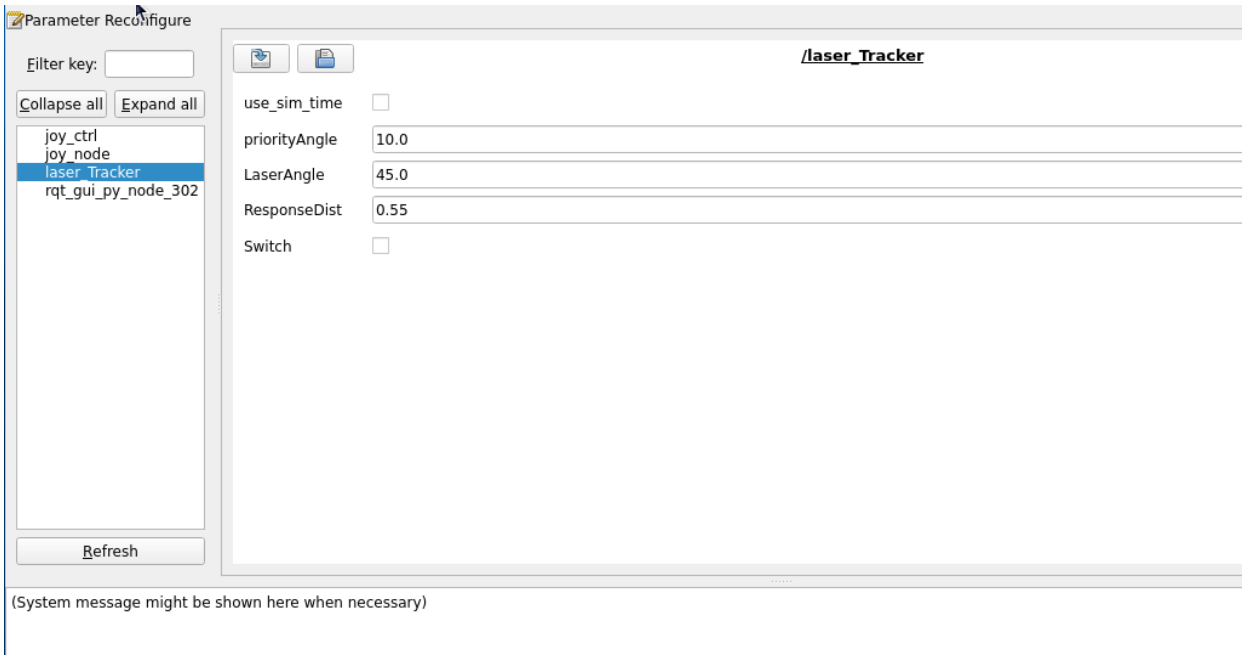
Enter the following command in the terminal to start.

```
ros2 run yahboomcar_laser laser_Tracker
```



After the program is started, it will search for the nearest object within the radar scanning range and maintain a set distance from it. Move the tracked object slowly, and the car will track the movement of the object. You can set some parameters through the dynamic parameter adjuster. Enter the following command at the terminal.

```
ros2 run rqt_reconfigure rqt_reconfigure
```



Note: There may not be the above nodes when you first open it. You can see all nodes after clicking Refresh. The displayed laser_Tracker is the node tracked by the radar.

Description of the above parameters:

- priorityAngle：Radar priority detection angle
- LaserAngle：Radar detection angle value
- ResponseDist：tracking distance
- Switch：function enable switch

After modifying the above parameters, you need to click on the blank space to transfer the parameters into the program.

4、Code analysis

Source code reference path.

```
/root/yahboomcar_ws/src/yahboomcar_laser/yahboomcar_laser
```

laser_Tracker，The core code is as follows.

```
#Create a radar subscriber to subscribe to radar data and remote control data and a
speed publisher to publish speed data
self.sub_laser = self.create_subscription(LaserScan,"/scan",self.registerScan,1)
self.sub_JoyState = self.create_subscription(Bool,'/JoyState',
self.JoyStateCallback,1)
self.pub_vel = self.create_publisher(Twist,'/cmd_vel',1)
#Radar callback function: Process the subscribed radar data. There is a
priorityAngle here, which indicates the range of priority radar detection. If there
are objects within this range, priority will be selected for tracking. The set angle
is 10 degrees.
ranges = np.array(scan_data.ranges)
for i in range(len(ranges)):
```

```
    angle = (scan_data.angle_min + scan_data.angle_increment * i) * RAD2DEG
if abs(angle) < self.priorityAngle:
    if 0 < ranges[i] < (self.ResponseDist + offset):
        frontDistList.append(ranges[i])
        frontDistIDList.append(angle)
    elif abs(angle) < self.LaserAngle and ranges[i] > 0:
        minDistList.append(ranges[i])
        minDistIDList.append(angle)
#Find the nearest object minDistID
if len(frontDistIDList) != 0:
    minDist = min(frontDistList)
    minDistID = frontDistIDList[frontDistList.index(minDist)]
else:
    minDist = min(minDistList)
    minDistID = minDistIDList[minDistList.index(minDist)]
#According to the object that needs to be tracked, calculate the angular velocity
and linear velocity, and then publish the velocity data
if abs(minDist - self.ResponseDist) < 0.1: minDist = self.ResponseDist
velocity.linear.x = -self.lin_pid.pid_compute(self.ResponseDist, minDist)
ang_pid_compute = self.ang_pid.pid_compute(minDistID/48, 0)
if minDistID > 0: velocity.angular.z = ang_pid_compute
else: velocity.angular.z = ang_pid_compute
velocity.angular.z = ang_pid_compute
if abs(ang_pid_compute) < 0.1: velocity.angular.z = 0.0
self.pub_vel.publish(velocity)
```