Radar tracking

Note: The virtual machine needs to be in the same LAN as the car, and the ROS_DOMAIN_ID needs to be the same. You can check [Must-read before use] to set the IP and ROS_DOMAIN_ID on the board.

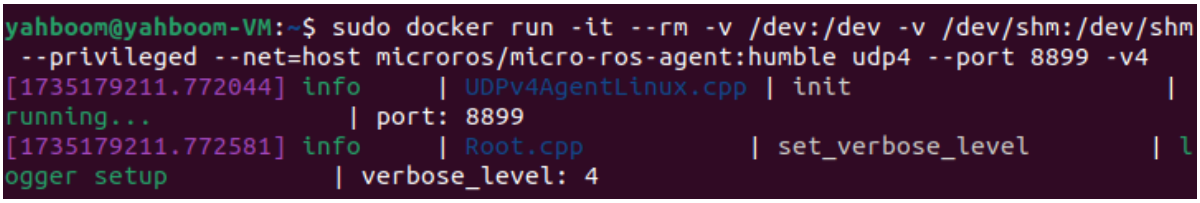
1. Program function description

The car connects to the proxy and runs the program. The radar on the car scans the nearest object within the set range, and adjusts its own speed according to the set tracking distance to keep a certain distance from the object. The dynamic parameter regulator can adjust the parameters such as the radar detection range and obstacle avoidance detection distance.

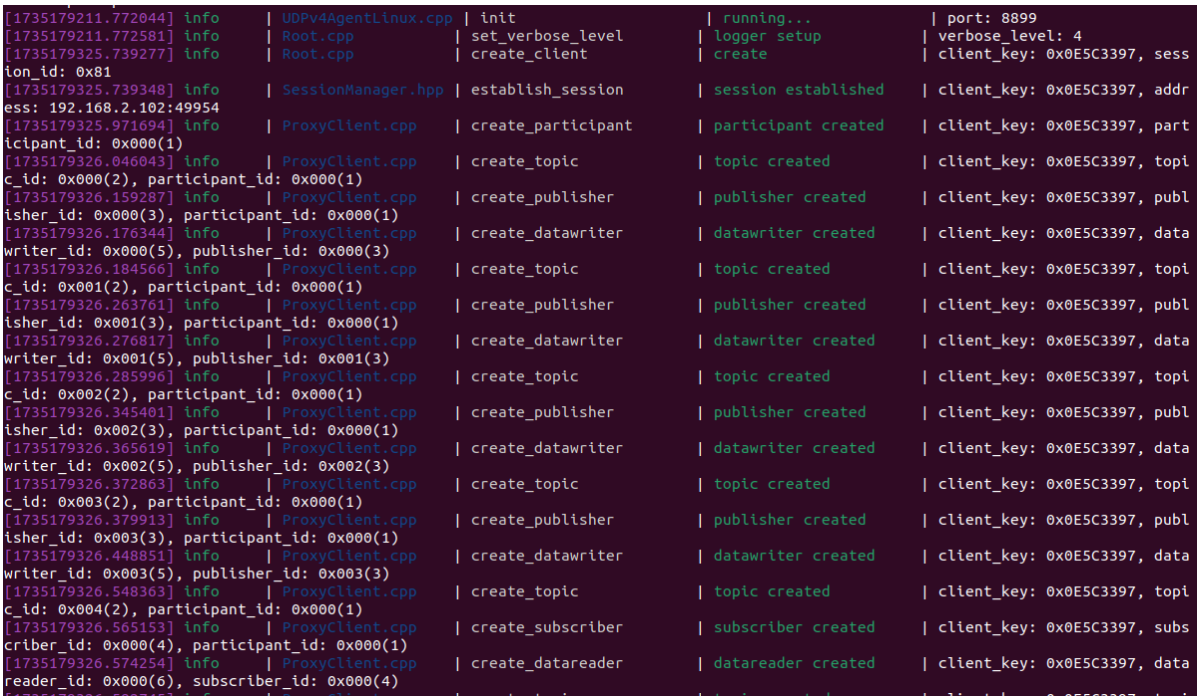
2. Start and connect the agent

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 agent. The connection is successful as shown in the figure below,

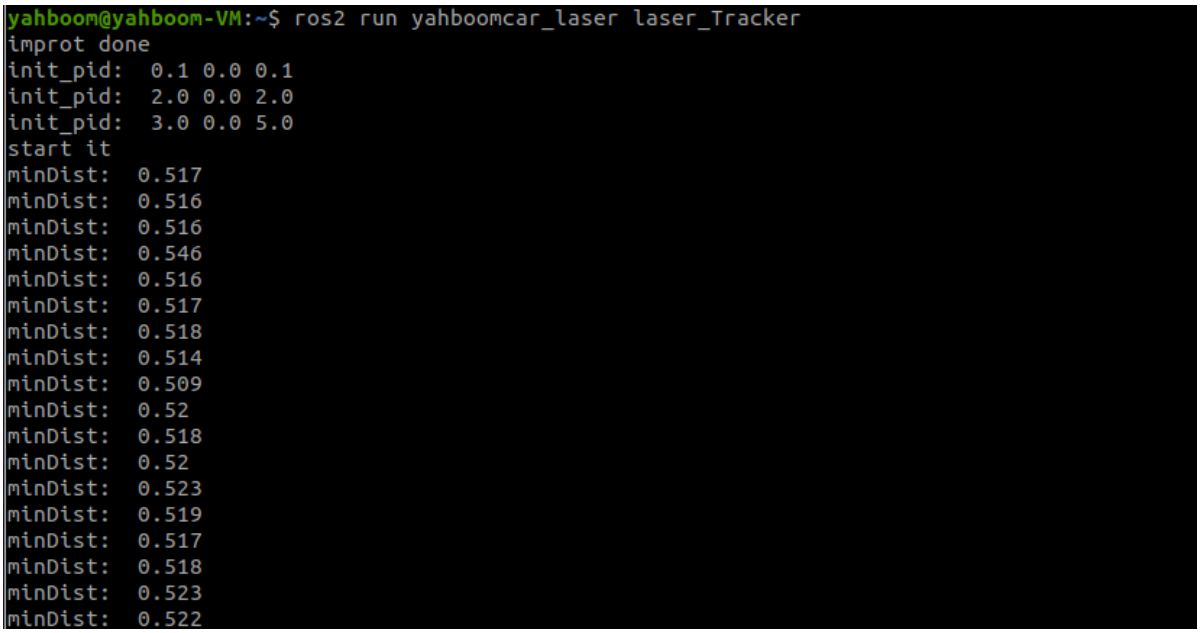


3. Start the program

3.1 Run command

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

```
ros2 run yahboomcar_laser laser_Tracker
```



After the program is started, it will search for the nearest object within the radar scanning range and keep a set distance from it. Slowly move the tracked object, and the car will track the movement of the object. The car has a backward obstacle avoidance function. If there is an obstacle within a certain distance of 70 degrees from the back, the buzzer will sound. You can set some parameters through the dynamic parameter regulator, terminal input,

```
ros2 run rqt_reconfigure rqt_reconfigure
```

Note: The above nodes may not be available when you first open it. Click Refresh to see all nodes. The displayed laser_Tracker is the node tracked by the radar.

Explanation of the above parameters:

- priorityAngle: Radar priority detection angle

- LaserAngle: Radar detection angle
- ResponseDist: Tracking distance
- Switch: Gameplay switch

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

4. Code analysis

Source code reference path (taking the supporting virtual machine as an example):

```
/home/yahboom/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
#create a sub
self.sub_laser = self.create_subscription(LaserScan,"/scan",self.registerScan,1)
self.sub_JoyState = self.create_subscription(Bool,'/JoyState',
self.JoyStateCallback,1)
#create a pub
self.pub_vel = self.create_publisher(Twist,'/cmd_vel_bl',1)
self.pub_Buzzer = self.create_publisher(UInt16,'/beep',1)

#Radar callback function: Process the subscribed radar data. There is a
priorityAngle here, which indicates the priority detection range of the radar. If
there is an object in this range, it will be tracked first. The set angle is 25
degrees
def registerScan(self, scan_data):
    if not isinstance(scan_data, LaserScan): return
    ranges = np.array(scan_data.ranges)
    offset = 0.5
    frontDistList = []
    frontDistIDList = []
    minDistList = []
    minDistIDList = []
    rearDistList = []

    for i in range(len(ranges)):
        angle = (scan_data.angle_min + scan_data.angle_increment * i) * RAD2DEG
        if angle > 180: angle = angle - 360
        if abs(angle) > 145 and ranges[i] != 0.0:
            rearDistList.append(ranges[i])

        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)
    #rearDist is used as the distance for the rear of the car to retreat 70
degrees to avoid obstacles
```

```
    rearDist = min(rearDistList)

#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)]

#Calculate the angular velocity and linear velocity based on the object to be
tracked, 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)
velocity.linear.x = velocity.linear.x * 45
ang_pid_compute = self.ang_pid.pid_compute(minDistID/48, 0)
ang_pid_compute = ang_pid_compute * 200
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

if ((rearDist <= 0.3  and velocity.linear.x < 0) ):
    print("rear obstacles")
    b = UInt16()
    b.data = 1
    self.pub_Buzzer.publish(b)
    velocity.linear.x = 0.0
else:
    b = UInt16()
    b.data = 0
    self.pub_Buzzer.publish(UInt16())

if b.data == 1:
    velocity.linear.x = 0.0
if minDist< self.ResponseDist * 2:
    self.pub_vel.publish(velocity)
else:
    self.pub_Buzzer.publish(UInt16())
    self.pub_vel.publish(Twist())
```