雷达跟踪

注：虚拟机需要与小车处在同一个局域网下，且ROS_DOMAIN_ID，需要一致，可以查看【使用前必看】来设置板子上的IP和ROS_DOMAIN_ID。

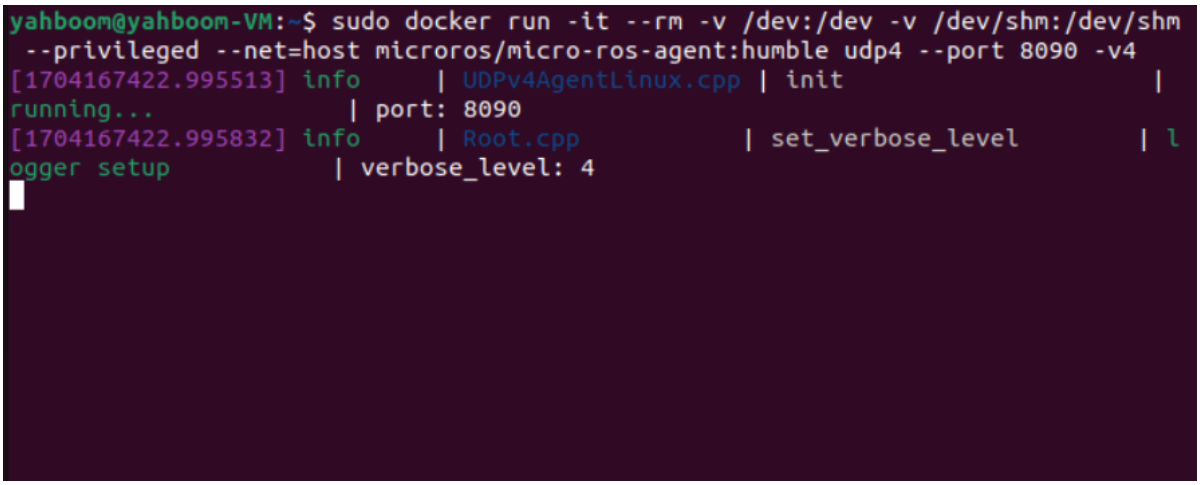
1、程序功能说明

小车连接上代理，运行程序，小车上的雷达扫描设定范围内的最近的一个物体，并且根据设定的跟踪距离，调试自身的速度，与该物体保持一定的距离。通过动态参数调节器可以调整雷达检测的范围和避障检测的距离等参数。

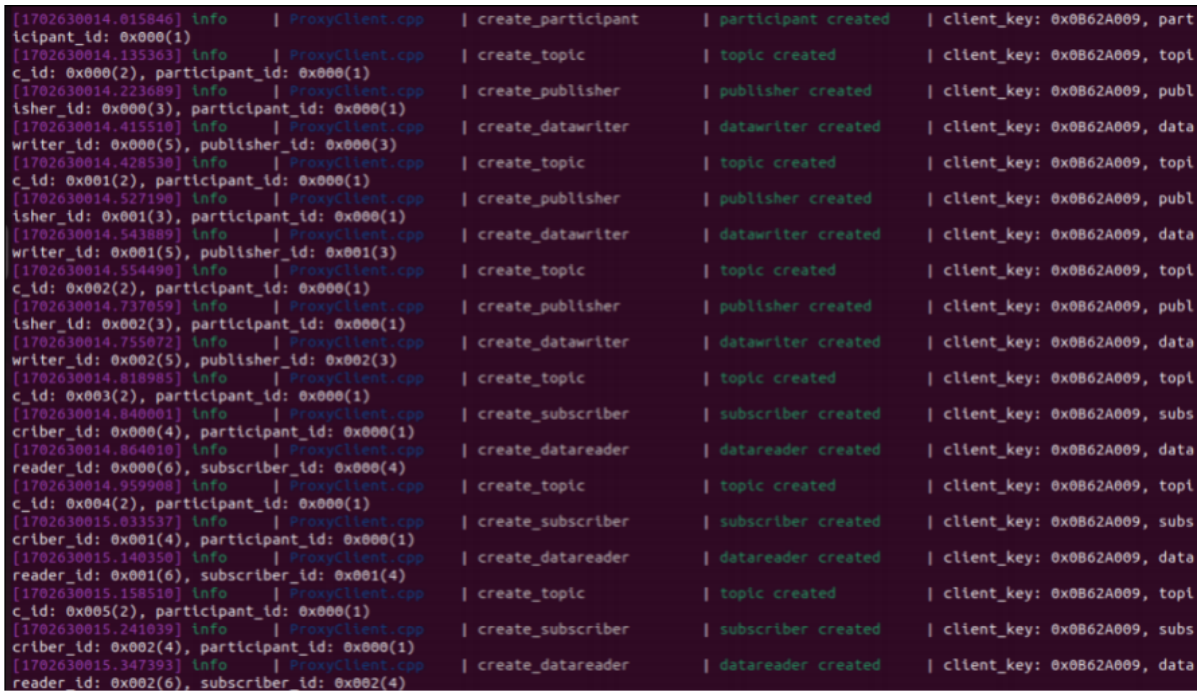
2、启动并连接代理

以配套虚拟机为例，输入以下指令启动代理

```
sudo docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privileged --
net=host microros/micro-ros-agent:humble udp4 --port 8090 -v4
```



然后，打开小车开关，等待小车连接上代理，连接成功如下图所示，



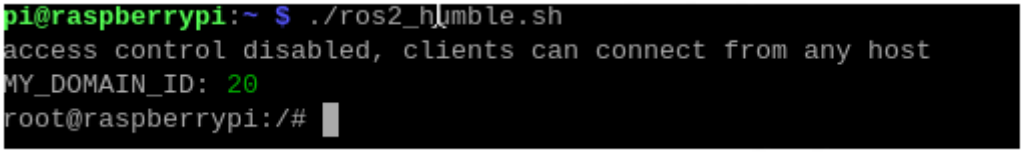
3、启动程序

3.1运行指令

如果是树莓派桌面版本和jetson nano桌面版本，需要提前进入docker，终端输入，

```
sh ros2_humble.sh
```

出现以下界面就是进入docker成功，

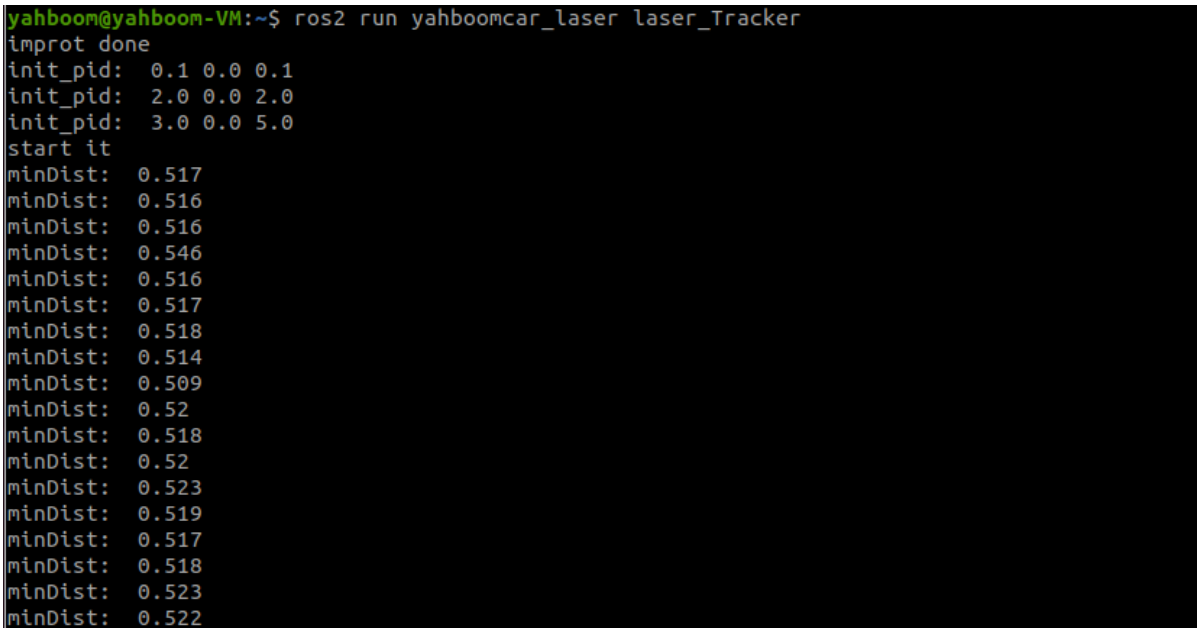


之后在docker里输入，**（查看【docker环境】章节，如何进入同一个docker终端）**

```
ros2 run yahboomcar_laser laser_Tracker          #雷达跟踪
ros2 run rqt_reconfigure rqt_reconfigure          #参数动态调节
```

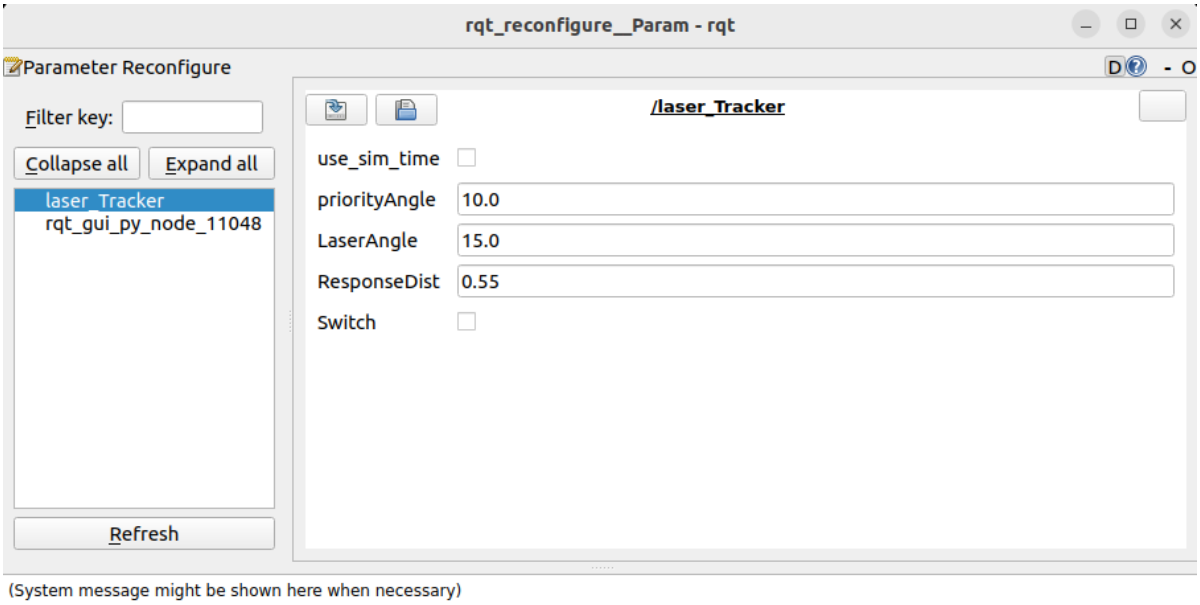
以配套的虚拟机为例，终端输入，

```
ros2 run yahboomcar_laser laser_Tracker
```



程序启动后，会寻找雷达扫描范围内最近的物体，并且与它保持设定的距离。缓慢移动被跟踪的物体，小车会跟踪物体移动。可以通过动态参数调节器去设置一些参数，终端输入，

```
ros2 run rqt_reconfigure rqt_reconfigure
```



注：刚开始打开的时候可能没有以上节点，点击Refresh刷新后可以看到全部节点。显示的laser_Tracker就是雷达跟踪的节点。

以上参数说明：

- priorityAngle：雷达优先检测的角度
- LaserAngle：雷达检测角度
- ResponseDist：跟踪的距离
- Switch：玩法开关

修改完以上的参数，需要点击空白处，才能把参数传入程序中。

4、代码解析

源码参考路径（以配套虚拟机为例）：

```
/home/yahboom/yahboomcar_ws/src/yahboomcar_laser/yahboomcar_laser
```

jetson nano代码路径：

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

树莓派代码路径：

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

laser_Tracker，核心代码如下，

```
#创建雷达订阅者订阅雷达数据和遥控控制数据以及速度发布者发布速度数据
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)
```

```
#雷达回调函数：处理订阅到的雷达数据，这里有个priorityAngle，表示优先检测雷达的范围，该范围内有物体上则是优先选择跟踪，设定的角度是10度
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)
#找到最近的一个物体minDistID
if len(frontDistIDList) != 0:
    minDist = min(frontDistList)
    minDistID = frontDistIDList[frontDistList.index(minDist)]
else:
    minDist = min(minDistList)
    minDistID = minDistIDList[minDistList.index(minDist)]
#根据需要跟踪的物体，计算角速度和线速度，然后发布速度数据
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)
```