3. Lidar tracking

3.1, Program function description

After the program is started, the radar scans the nearest object and then locks it. When the object moves, the car moves with it.

Start the dynamic parameter regulator on the virtual machine and click [Switch] to turn on/pause this function.

In addition, the [L1] button of the handle can lock/open the motion control of the car. When the motion control is turned on, the function will be locked; when the motion control is locked, the function can be turned on.

3.2, Program code reference path

After SSH connects to the car, the location of the function source code is,

```
#python file
/home/sunrise/yahboomcar_ws/src/yahboomcar_laser/yahboomcar_laser/laser_tracker.
py
#launch file
/home/sunrise/yahboomcar_ws/src/yahboomcar_laser/launch/laser_tracker_launch.py
```

3.3, Program startup

3.3.1, Car startup command

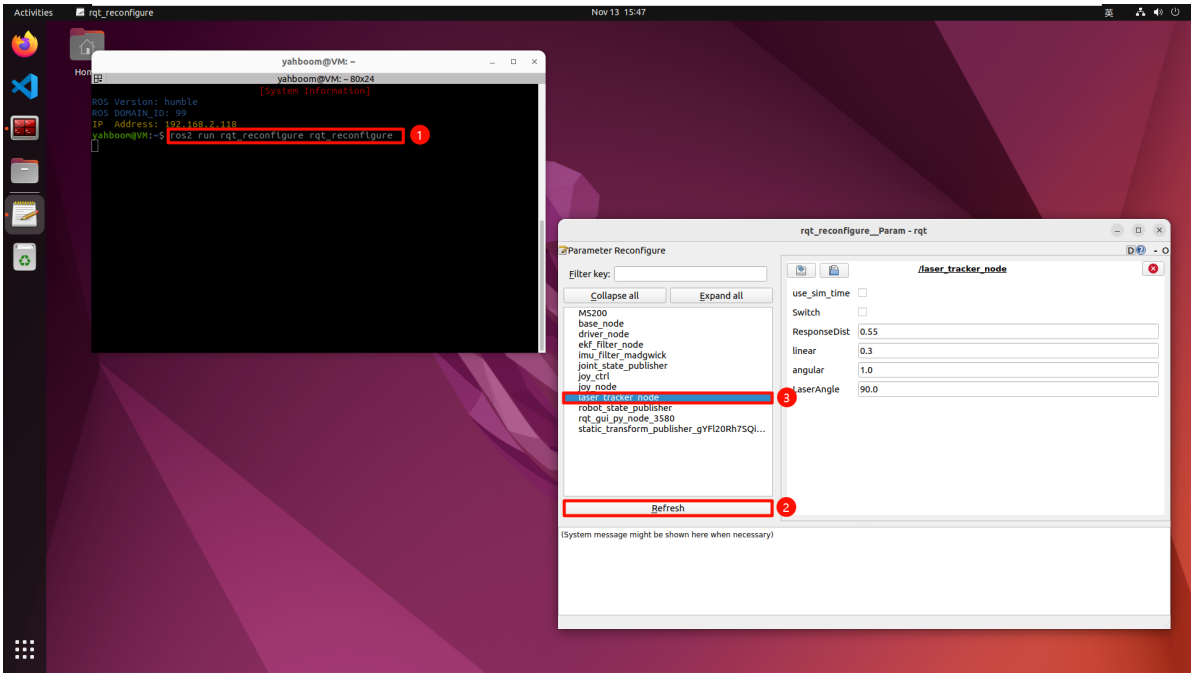
After SSH connects to the car, enter in the terminal,

```
ros2 launch yahboomcar_laser laser_tracker_launch.py
```

3.3.2, Modify parameters on the virtual machine side

Open the dynamic parameter adjuster on the virtual machine side, open the terminal and enter,

```
ros2 run rqt_reconfigure rqt_reconfigure
```



The meaning of each parameter is as follows,

Parameter name	Parameter meaning
Switch	Gameplay switch
ResponseDist	Obstacle detection distance
linear	Linear speed
angular	Angular speed
LaserAngle	Radar detection angle

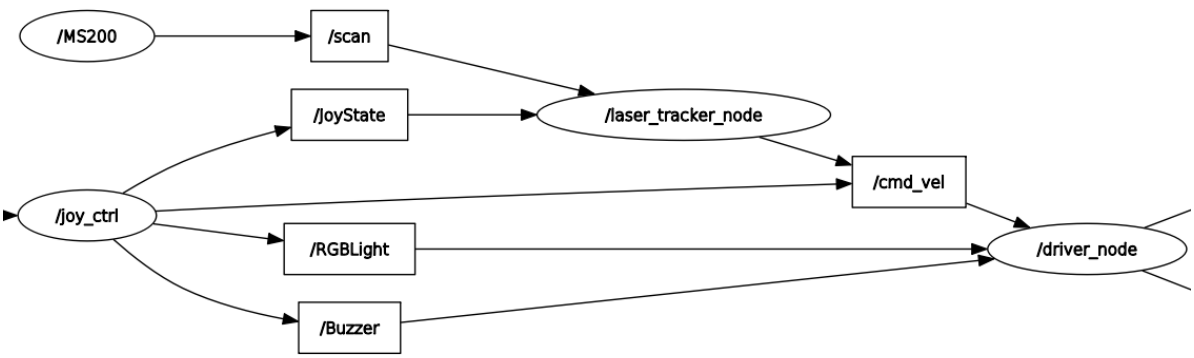
The above parameters can be adjusted. Check or uncheck [Switch] to turn on or pause the radar tracking function.

The other four need to be set in decimals. After modification, press the Enter key or click on the blank space to write.

3.3.3, view the topic communication node diagram

Virtual machine terminal input,

```
ros2 run rqt_graph rqt_graph
```



3.4, core source code analysis

Mainly look at the callback function of the radar, which explains how to obtain the obstacle distance information at each angle, then find the nearest point, then judge the distance, then calculate the speed data, and finally publish it,

```
ranges = np.array(scan_data.ranges)
for i in range(len(ranges)):
angle = (scan_data.angle_min + scan_data.angle_increment * i) * 180 / pi
if angle > 180: angle = angle - 360
if abs(angle) < self.priorityAngle: #priorityAngle is the range that the car
prioritizes to follow
if 0 < ranges[i] < self.ResponseDist + self.offset:
frontDistList.append(ranges[i])
frontDistIDList.append(angle)
elif abs(angle) < self.LaserAngle and ranges[i] > 0:
minDistList.append(ranges[i])
minDistIDList.append(angle)

#Calculate the minimum distance point and ID
if len(frontDistIDList) != 0:
minDist = min(frontDistList)
minDistID = frontDistIDList[frontDistList.index(minDist)]
else:
minDist = min(minDistList)
minDistID = minDistIDList[minDistList.index(minDist)]

velocity = Twist()
#Calculate linear velocity
if abs(minDist - self.ResponseDist) < 0.1: velocity.linear.x = 0.0 else:
velocity.linear.x = -self.lin_pid.pid_compute(self.ResponseDist, minDist)
#Calculate angular velocity angle_pid_compute =
self.ang_pid.pid_compute(minDistID / 72, 0) if abs(angle_pid_compute) < 0.02:
velocity.angular.z = 0.0 else: velocity.angular.z = angle_pid_compute
self.pub_vel.publish(velocity) ```
```