Lidar following

This lesson uses the Raspberry Pi as an example.

For Raspberry Pi and Jetson-Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this lesson in the terminal. For instructions on entering the Docker container from the host computer, refer to **[01. Robot Configuration and Operation Guide] --[4.Enter Docker (For JETSON Nano and RPi 5)]**.

For RDKX5 and Orin boards, simply open a terminal and enter the commands mentioned in this lesson.

1. Program Functionality

> After running the program, the car's lidar scans for the nearest object within the set range and adjusts its speed based on the set tracking distance to maintain a certain distance from the object. Dynamic parameter adjusters can be used to adjust parameters such as the lidar detection range and obstacle avoidance detection distance.

2. Program Code Reference Path

For Raspberry Pi and Jetson-Nano boards, you need to enter the Docker container first. For RDKX5 and Orin controllers, this is not necessary.

The source code for this function is located at:

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_laser/yahboomcar_laser/laser_Tracker_M1.py
```

3. Program Startup

3.1. Startup Command

For Raspberry Pi and Jetson-Nano boards, you need to enter the Docker container first. For RDKX5 and Orin controllers, this is not necessary.

Enter the Docker container (see [Docker course] --- [4. Docker Startup Script] for steps).

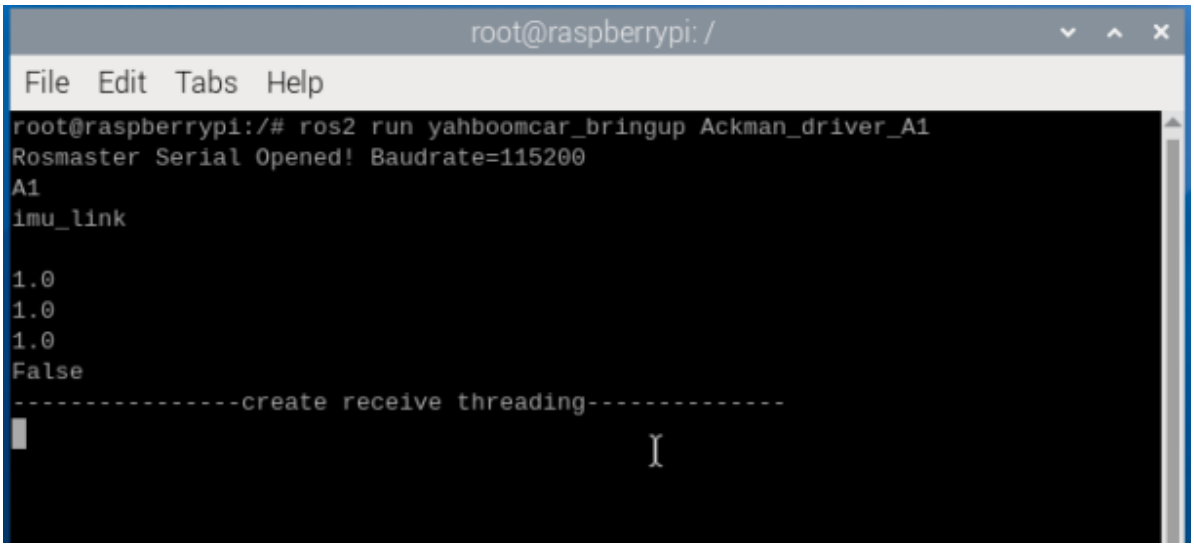
All the following commands must be executed within the same Docker container **(see [Docker course] --- [3. Docker Submission and Multi-Terminal Access] for steps).**

This command must be modified to the corresponding lidar model.

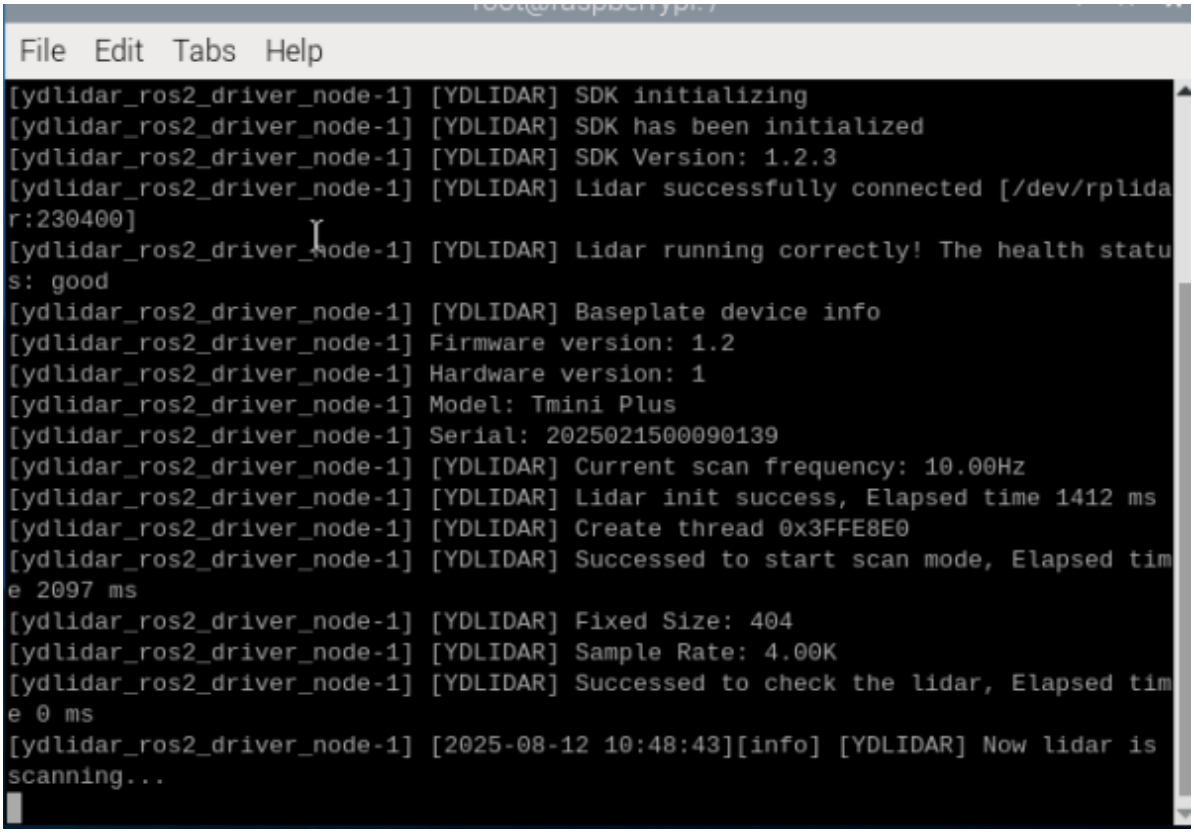
Terminal Input

```
# Start the car chassis
ros2 run yahboomcar_bringup Mcnamu_driver_M1
# Select one of the two lidars
#timni
ros2 launch ydlidar_ros2_driver ydlidar_launch.py
#c1
ros2 launch sllidar_ros2 sllidar_c1_launch.py
# Start the lidar tracking program
ros2 run yahboomcar_laser laser_Tracker_M1
```

The chassis terminal,



Start the lidar driver terminal,



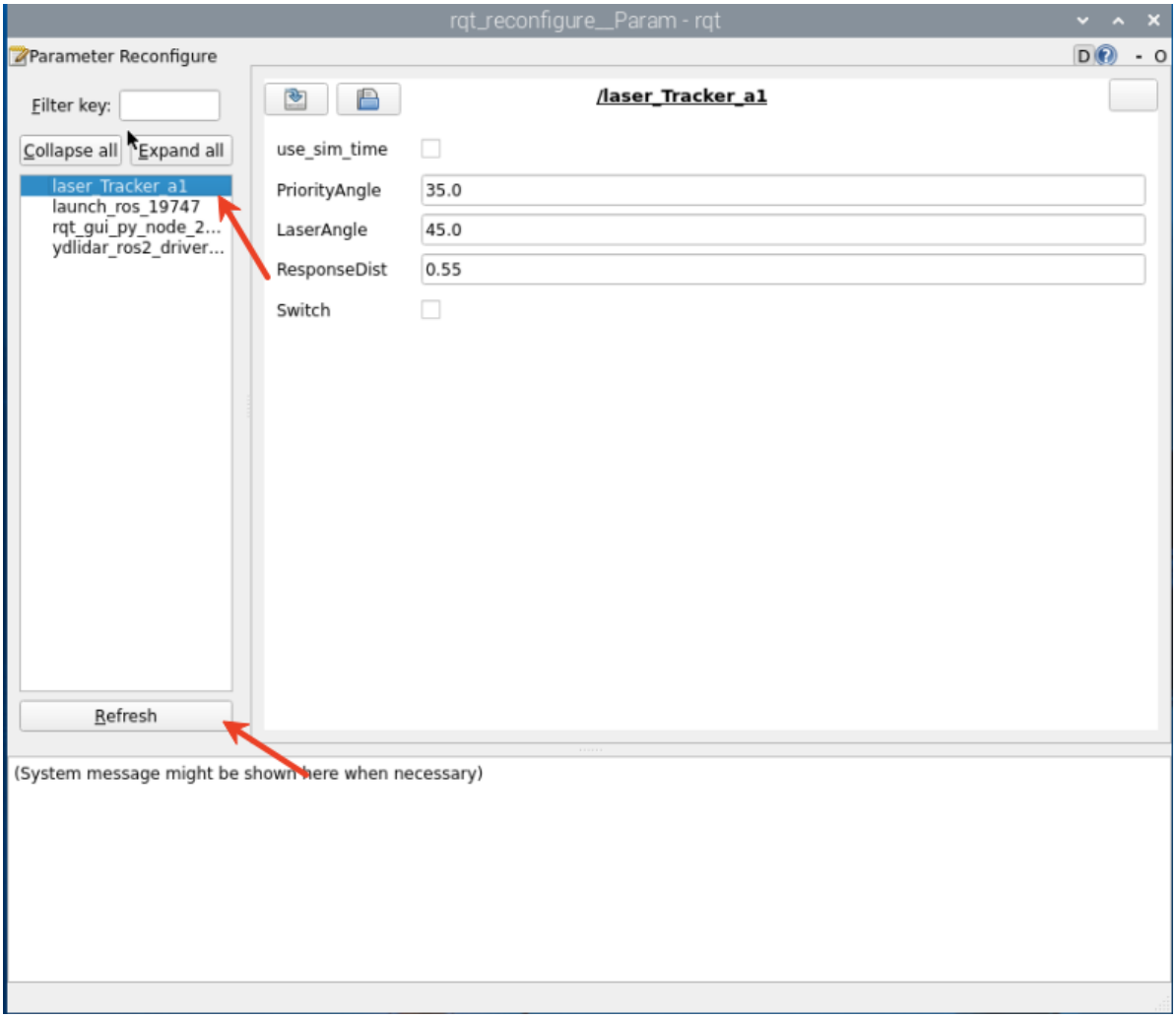
After starting the program, it will search for the nearest object within the lidar's scanning range and maintain a set distance from it. Slowly move the object to be tracked, and the robot will follow its movement.

```
root@raspberrypi:/# ^C
root@raspberrypi:/# ros2 run yahboomcar_laser laser_Tracker_A1
improt done
init_pid:  2.0 0.0 2.0
init_pid:  3.5 0.0 5.0
start it
```

3.2 Dynamic Parameter Adjuster

You can also use the dynamic parameter adjuster to set parameter values. In the terminal, enter:

```
ros2 run rqt_reconfigure rqt_reconfigure
```



☑ After modifying the parameters, click a blank area in the GUI to enter the parameter value. Note that this will only take effect for the current boot. To permanently change the value, you need to modify the parameter in the source code.

Parameter Analysis:

[priorityAngle]: lidar priority detection angle

[LaserAngle]: lidar detection angle

[ResponseDist]: Tracking distance

[Switch]: Gameplay switch

All of the above parameters are adjustable. Except for Switch, the other four must be set as decimals. After changing, click the blank space to write.

4. Core Code

4.1. laser_Tracker_M1.py

This program has the following main functions:

- Subscribes to the lidar topic and obtains surrounding lidar data;
- Obtains and pre-processes the nearest lidar data;
- Publishes the vehicle's forward speed, distance, and turning angle based on the following strategy.

Some of the core code is as follows.

```
#数据预处理 #Data preprocessing
if not isinstance(scan_data, LaserScan): return
ranges = np.array(scan_data.ranges)  # 转换为数组 #Convert to array
offset = 0.5  # 安全距离偏移量 #Safe distance offset
frontDistList = []  # 优先区域距离列表 #Priority Area Distance List
frontDistIDList = []  # 优先区域角度列表 Priority Area Angle List
minDistList = []  # 次要区域距离列表 Secondary Area Distance List
minDistIDList = []  # 次要区域角度列表 List of secondary area angles

#优先跟随（前方区域） #Priority Follow (Forward Area)
if abs(angle) > (180 - self.priorityAngle):
    if ranges[i] < (self.ResponseDist + offset) and ranges[i]!=0:
        frontDistList.append(ranges[i])
        frontDistIDList.append(angle)

#跟随决策逻辑 #Follow decision logic
if len(frontDistIDList) != 0:
    minDist = min(frontDistList)  # 取最小距离 #Take the minimum distance
    minDistID = frontDistIDList[frontDistList.index(minDist)]  # 对应角度
#Corresponding angle
else:
    minDist = min(minDistList)  # 取次要区域最小距离 #Take the minimum distance from
the secondary area
    minDistID = minDistIDList[minDistList.index(minDist)]

# 距离微调（防抖动） #Distance adjustment (anti shake)
if abs(minDist - self.ResponseDist) < 0.1:
    minDist = self.ResponseDist
# 线速度PID控制器 #Linear velocity PID controller
velocity.linear.x = -self.lin_pid.pid_compute(self.ResponseDist, minDist)
# 角速度误差计算 #Calculation of angular velocity error
error = (180 - abs(minDistID)) / 72  # 角度归一化 #Angle normalization
ang_pid_compute = self.ang_pid.pid_compute(error, 0)  # 目标角度=0（正前方） Target
angle=0 (straight ahead)

# 方向决策 #Direction decision
if minDistID > 0:  # 障碍物在右侧 #The obstacle is on the right side
    velocity.angular.z = -ang_pid_compute  # 左转 #Turn left
else:
    velocity.angular.z = ang_pid_compute  # 右转 #Turn right
```