3.Lidar tracking

1. Program function description

After the program is started, the lidar scans the nearest object, then locks it, the object moves, and Muto moves with it. If the controller node is activated, the R2 key of the controller can pause/enable this function.

2. Program code reference path

After entering the docker container, the source code of this function is located at

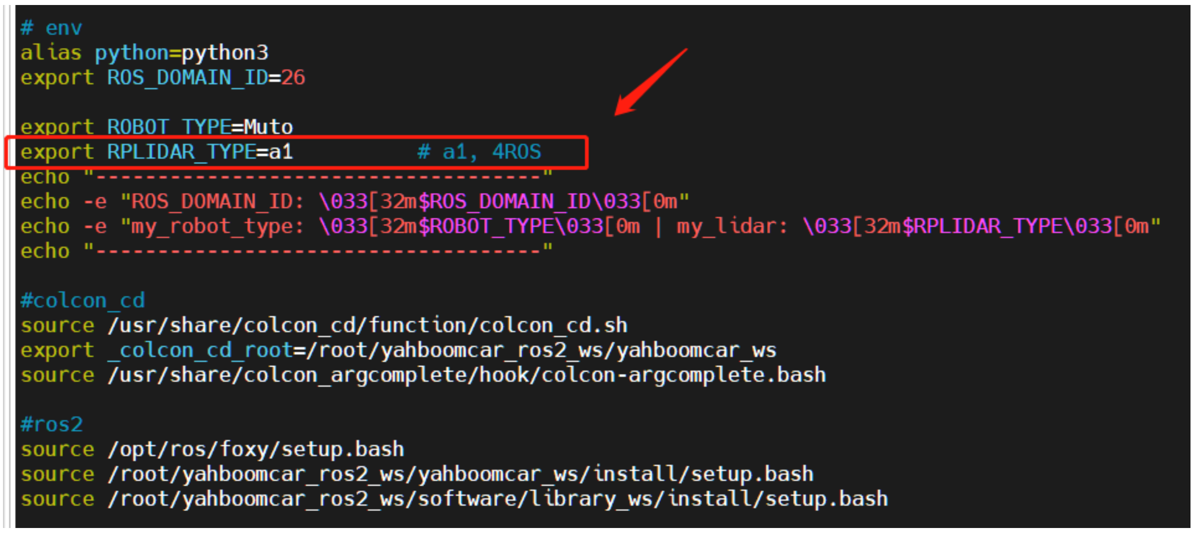
```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_laser/yahboomcar_laser/laser_Tracker_a1.py      # a1雷达
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_laser/yahboomcar_laser/laser_Tracker_4ROS.py    # 4ROS雷达
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_laser/launch/laser_tracker_launch.py
```

3. Configuration before use

Note: Since the Muto series robots are equipped with multiple lidar devices, the factory system has been configured with routines for multiple devices. However, since the product cannot be automatically recognized, the lidar model needs to be manually set.

After entering the container: Make the following modifications according to the lidar type:

```
root@ubuntu:/# cd
root@ubuntu:~# vim .bashrc
```



```
# env
alias python=python3
export ROS_DOMAIN_ID=26

export ROBOT_TYPE=Muto
export RPLIDAR_TYPE=a1          # a1, 4ROS
echo "------------------------------------"
echo -e "ROS_DOMAIN_ID: \033[32m$ROS_DOMAIN_ID\033[0m"
echo -e "my_robot_type: \033[32m$ROBOT_TYPE\033[0m | my_lidar: \033[32m$RPLIDAR_TYPE\033[0m"
echo "------------------------------------"

#colcon_cd
source /usr/share/colcon_cd/function/colcon_cd.sh
export _colcon_cd_root=/root/yahboomcar_ros2_ws/yahboomcar_ws
source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash

#ros2
source /opt/ros/foxy/setup.bash
source /root/yahboomcar_ros2_ws/yahboomcar_ws/install/setup.bash
source /root/yahboomcar_ros2_ws/software/library_ws/install/setup.bash
```

After the modification is completed, save and exit vim, and then execute:

```
root@jetson-desktop:~# source .bashrc
------------------------------------
ROS_DOMAIN_ID: 26
my_robot_type: Muto | my_lidar: a1
------------------------------------
root@jetson-desktop:~#
```

You can see the current modified lidar type.

4. Program startup

4.1. Start command

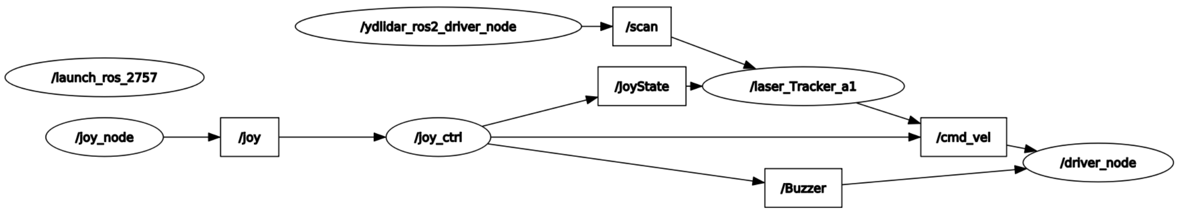
After entering the docker container, enter in the terminal,

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

4.2. View the topic communication node diagram

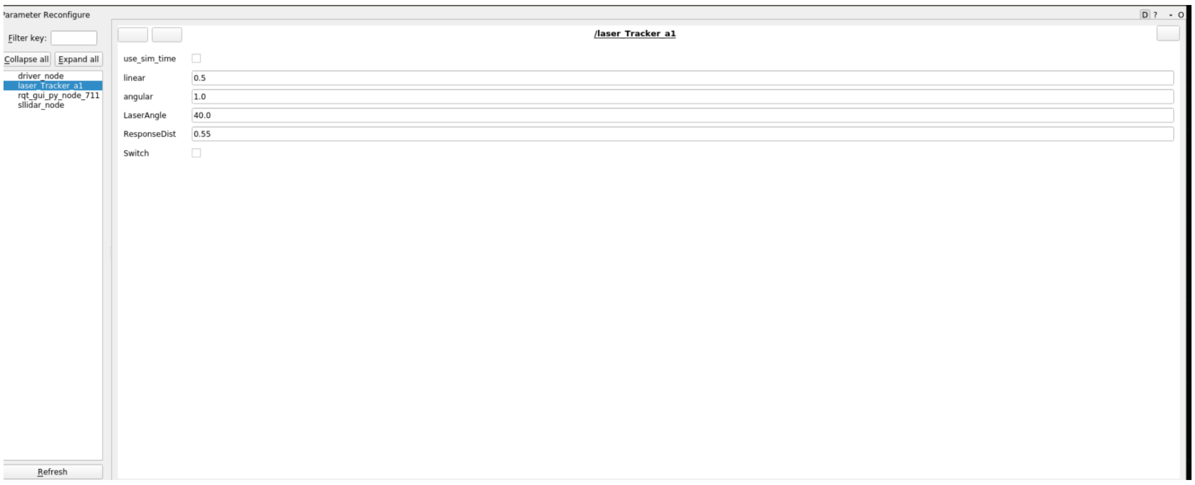
docker terminal input

```
ros2 run rqt_graph rqt_graph
```



You can also set the parameter size through the dynamic parameter adjuster, terminal input,

```
ros2 run rqt_reconfigure rqt_reconfigure
```



/laser_Tracker_a1

use_sim_time
linear 0.5
angular 1.0
LaserAngle 40.0
ResponseDist 0.55
Switch

5. Core code

Taking the source code of a1 lidar as an example, we mainly look at the callback function of lidar. Here we explain how to obtain the obstacle distance information at each angle, then find the nearest point, then determine the distance, and then calculate the speed data. Finally publish it,

```
angle = (scan_data.angle_min + scan_data.angle_increment * i) * RAD2DEG
if abs(angle) > (180 - self.priorityAngle): #priorityAngle is the car’s priority
following range
    if ranges[i] < (self.ResponseDist + offset):
        frontDistList.append(ranges[i])
        frontDistIDList.append(angle)
    elif (180 - self.LaserAngle) < angle < (180 - self.priorityAngle):
        minDistList.append(ranges[i])
        minDistIDList.append(angle)
    elif (self.priorityAngle - 180) < angle < (self.LaserAngle - 180):
        minDistList.append(ranges[i])
        minDistIDList.append(angle)
        if len(frontDistIDList) != 0:
            minDist = min(frontDistList)
            minDistID = frontDistIDList[frontDistList.index(minDist)]
        else:
            minDist = min(minDistList)
            minDistID = minDistIDList[minDistList.index(minDist)]#Calculate the
ID of the minimum distance point
            if self.Joy_active or self.Switch == True:
                if self.Moving == True:
                    self.pub_vel.publish(Twist())
                    self.Moving = not self.Moving
                    return
                self.Moving = True
                velocity = Twist()
                if abs(minDist - self.ResponseDist) < 0.1: minDist =
self.ResponseDist#Determine the distance from the smallest point
                velocity.linear.x = -self.lin_pid.pid_compute(self.ResponseDist,
minDist)#Calculate linear speed
                ang_pid_compute = self.ang_pid.pid_compute((180 -
abs(minDistID)) / 72, 0)#Calculate angular speed
if minDistID > 0: velocity.angular.z = -ang_pid_compute
else: velocity.angular.z = ang_pid_compute
if ang_pid_compute < 0.02: velocity.angular.z = 0.0
self.pub_vel.publish(velocity)
```