3、Lidar follow

1、Program function description

After the program starts, the radar scans the nearest object, then locks, the object moves, and the trolley follows. 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 location of the source code of this function is located at,

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_laser/yahboomcar_laser/laser_Tracker_a1_X3.py
```

The A1 radar has the same architecture as the S2 radar and can be shared.

3、The program starts

3.1、start the command

After entering the docker container, according to the actual model and radar model, the terminal input,

```
#Start the trolley chassis
ros2 run yahboomcar_bringup Mcnamu_driver_X3
#Start the A1 radar
ros2 launch sllidar_ros2 sllidar_launch.py
#Start the S2 radar
ros2 launch sllidar_ros2 sllidar_s2_launch.py
#Activate the radar tracker X3 model, A1/S2 radar
ros2 run yahboomcar_laser laser_Tracker_a1_X3
#Start the handle, if needed
ros2 run yahboomcar_ctrl yahboom_joy_X3
ros2 run joy joy_node
```

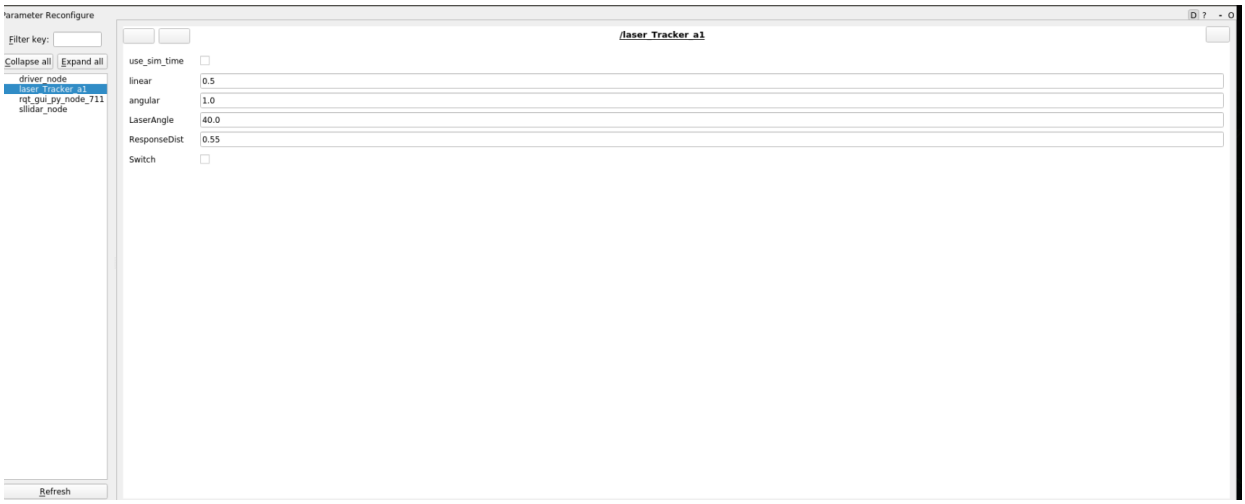
3.2、View the topic communication node diagram

docker terminal input,

```
ros2 run rqt_graph rqt_graph
```

It is also possible to set the size of parameters, terminal input, and terminal input through the dynamic parameter adjuster,

```
ros2 run rqt_reconfigure rqt_reconfigure
```



The meaning of each parameter is as follows,

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

The above parameters are adjustable, except for Switch, the other four need to be set when they need to be decimal, after modification, click on the blank space to write.

4、core code

Taking the X3 model, the source code of the A1 radar as an example, mainly looking at the callback function of the radar, here explains how to obtain the obstacle distance information of each angle, and then find the nearest point, then judge the distance, then calculate the speed data, and finally release it.

```
angle = (scan_data.angle_min + scan_data.angle_increment * i) * RAD2DEG
if abs(angle) > (180 - self.priorityAngle): #priorityAngle is the trolley priority
to follow the 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#Determines the distance from the smallest point
                velocity.linear.x = -self.lin_pid.pid_compute(self.ResponseDist,
minDist)#Calculate the line speed
                ang_pid_compute = self.ang_pid.pid_compute((180 - abs(minDistID)) /
72, 0)#Calculate angular velocity
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)
```