2、Fun gameplay with radar obstacle avoidance

1、Program function description

After the program starts, the car will move forward, and when there is an obstacle within the detection range, it will adjust its attitude, avoid the obstacle, and then continue to walk forward. 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_Avoidance_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
#Start the radar obstacle avoidance program X3 model, A1/S2 radar
ros2 run yahboomcar_laser laser_Avoidance_a1_X3
#Start the handle, if needed
ros2 run yahboomcar_ctrl yahboom_joy_X3
ros2 run joy joy_node
```

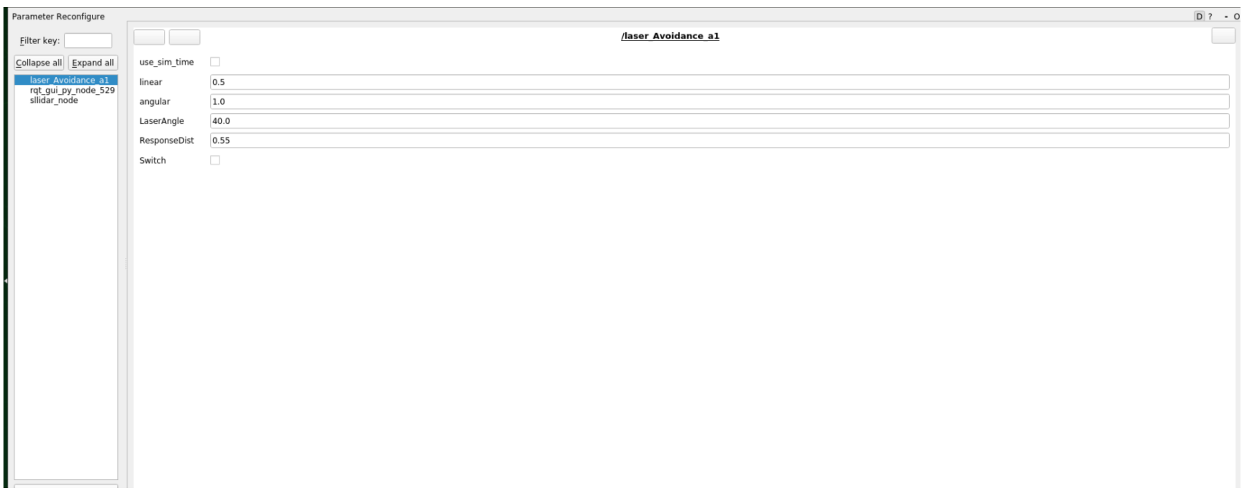
3.2、View the topic communication node diagram

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 Switc, the other four need to be set when they need to be decimal, after modification, click on the blank space to write.

4、Core source code analysis

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.

```
def registerScan(self, scan_data):
    if not isinstance(scan_data, LaserScan): return
    ranges = np.array(scan_data.ranges)
    self.Right_warning = 0
    self.Left_warning = 0
    self.front_warning = 0

    for i in range(len(ranges)):
        angle = (scan_data.angle_min + scan_data.angle_increment * i) * RAD2DEG
        #The angle of radar information is a radian system, and here it is converted
into an angle for calculation
        if 160 > angle > 180 - self.LaserAngle:#The angle is based on the structure
of the radar to set the judgment range
            if ranges[i] < self.ResponseDist*1.5:
        #range[i] is the result of radar scanning, which in this case refers to
distance information
                self.Right_warning += 1
        if - 160 < angle < self.LaserAngle - 180:
            if ranges[i] < self.ResponseDist*1.5:
                self.Left_warning += 1
        if abs(angle) > 160:
            if ranges[i] <= self.ResponseDist*1.5:
                self.front_warning += 1
        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
```