

Guard

1. Programme function description

After the programme starts, the trolley will track the target of the nearest point, when the target point moves laterally, it will move with the target point, when the target point is close to the trolley and less than the set distance, there will be a voice reminder until the target point is far away from the trolley's set distance.

2. Programme Code Reference Path

The location of the source code of this function is located at.

```
/home/pi/cartographer_ws2/src/yahboom_laser/laser_warning_xgo_RS200.py
```

3. Program startup

3.1 Start command

Mechanical dog chassis and lidar has been set to boot self-start, if you find that it did not start please enter in the terminal.

```
sudo systemctl restart YahboomStart.service
```

If the lidar and chassis start-up is complete then you need to enter it in the terminal:

```
# Activate lidar obstacle avoidance procedures lidar MS200  
ros2 run yahboom_laser laser_warning_RS200
```

3.2 Viewing the topic communication node map

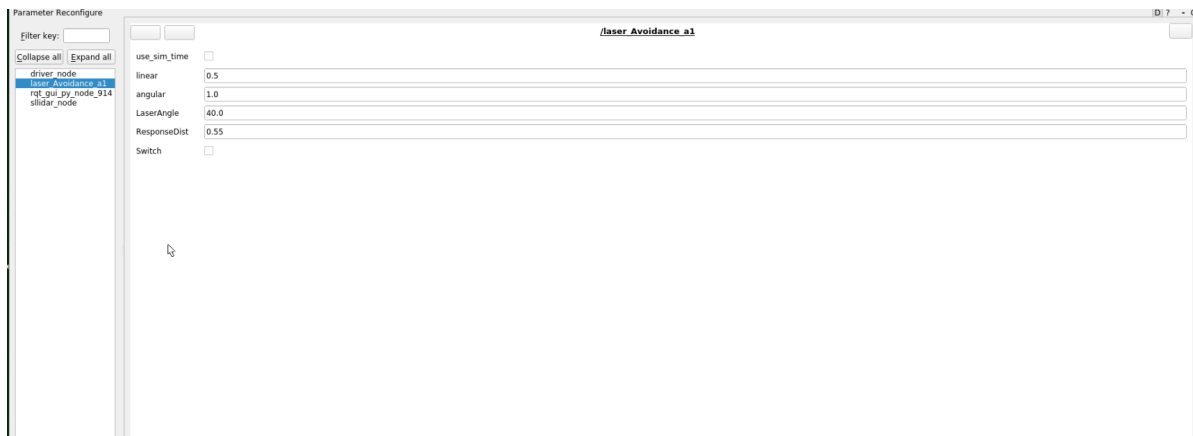
Terminal input.

```
ros2 run rqt_graph rqt_graph
```



It is also possible to set the size of the parameter, the terminal input, by means of a dynamic parameter regulator, the

```
ros2 run rqt_reconfigure rqt_reconfigure
```



The meaning of each parameter is as follows.

Parameter name	Parameter Meaning
linear	linear velocity
angular	angular velocity
<u>LaserAngle</u>	<u>lidar</u> detection angle
<u>ResponseDist</u>	Obstacle detection distance
Switch	Play switch

4. Core source code analysis

Mainly look at the lidar's callback function, here explains how to get to each angle of the obstacle distance information, and then get the ID of the minimum distance, calculate the size of the angular velocity, and then the minimum distance and the set distance to compare, if it is less than the set distance, then let the buzzer ring, and finally release the speed topic data.

```
for i in range(len(ranges)):
    angle = (scan_data.angle_min + scan_data.angle_increment * i) * RAD2DEG
    if abs(angle) > (180 - self.LaserAngle):
        minDistList.append(ranges[i])
        minDistIDList.append(angle)
        if len(minDistList) == 0: return
    minDist = min(minDistList)
    minDistID = minDistIDList[minDistList.index(minDist)]
    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
    if minDist <= self.ResponseDist:
        if self.Buzzer_state == False:
            b = Bool()
            b.data = True
```

```
        self.pub_Buzzer.publish(b)
        self.Buzzer_state = True
    else:
        if self.Buzzer_state == True:
            self.pub_Buzzer.publish(Bool())
            self.Buzzer_state = False
    velocity = Twist()
    ang_pid_compute = self.ang_pid.pid_compute((180 - abs(minDistID)) / 36, 0)
    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)
```