Tracking

1. Description of programme functions

When the programme starts, the lidar scans for the nearest object, then locks on, the object moves, and the mechanical dog follows. If the joystick node is activated, the R2 button on the joystick can pause/enable this function.

2. Programme Code Reference Path

The source code for this function is located at.

```
/home/pi/cartographer_ws2/src/yahboom_laser/yahboom_laser/laser_Tracker_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:

```
cd /home/pi/cartographer_ws2
source install/setup.bash
# Initiate lidar tracking procedure lidar MS200
ros2 run yahboom_laser laser_Tracke_RS200
```

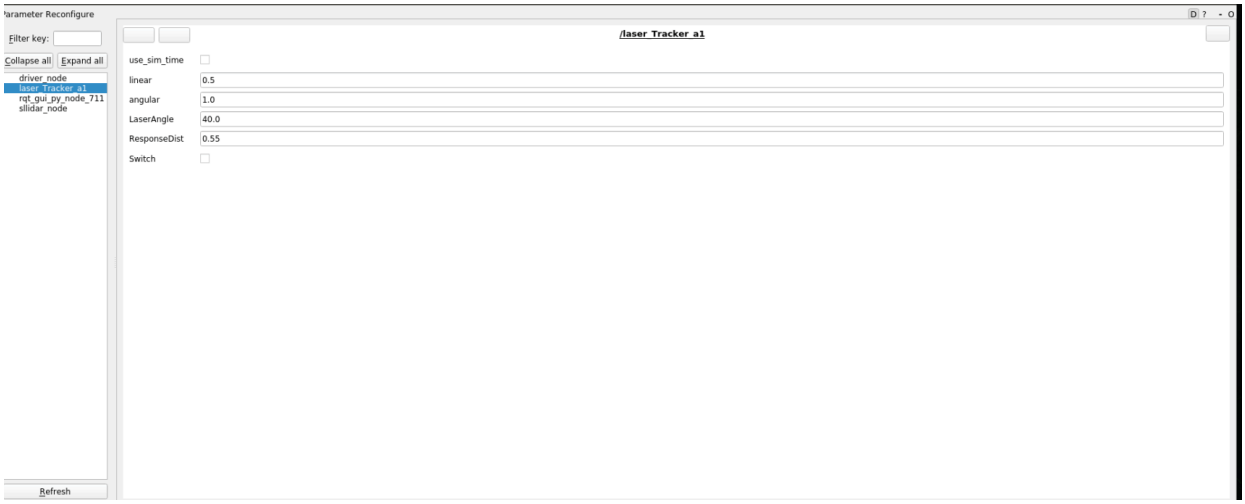
3.2 Viewing the topic communication node map

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
Switch	Play switch
ResponseDist	Obstacle detection distance
linear	The linear velocity
angular	Angular velocity
LaserAngle	Lidar detection Angle

The above parameters are adjustable, except Switc, the other four need to be set when you need to be a decimal, modified, click on the blank before you can write.

4. Core Code

Mainly look at the lidar's callback function, here explains how to get to each angle of the obstacle distance information, and then find out the nearest point, and then judge the distance, and then it is to calculate the speed data, and finally released, the

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