3、Lidar guard

Function package: ~/rplidar_ws/src/transbot_laser

Introduction to lidar guard gameplay:

- Set the detection angle and response distance of the lidar.
- After turning on the car, the car faces the target closest to the car.
- When the distance between the target and the car is less than the response distance, the buzzer keeps beeping until there is no target within the response distance.
- Adjustable trolley angular velocity PID to make the robot to rotate best status.

3.1、Instructions

Note: The [R2] of the handle remote controller can [Pause/Open] for all functions of robot car

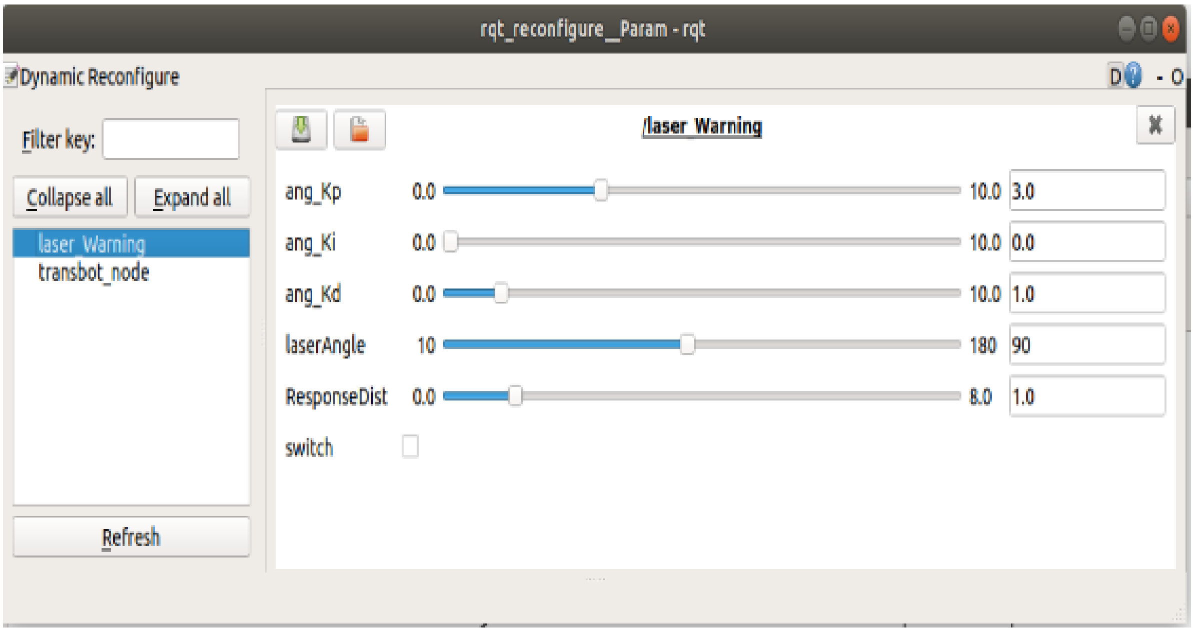
Start up

```
roslaunch transbot_laser laser_Warning.launch lidar_type:=a1
```

lidar_type parameter: the type of lidar used: [a1, a2, a3, s1, s2].

Dynamic debugging parameters

```
rosrun rqt_reconfigure rqt_reconfigure
```



ang_Kp 0.0 10.0 3.0

ang_Ki 0.0 10.0 0.0

ang_Kd 0.0 10.0 1.0

laserAngle 10 180 90

ResponseDist 0.0 8.0 1.0

switch □

Parameter analysis:

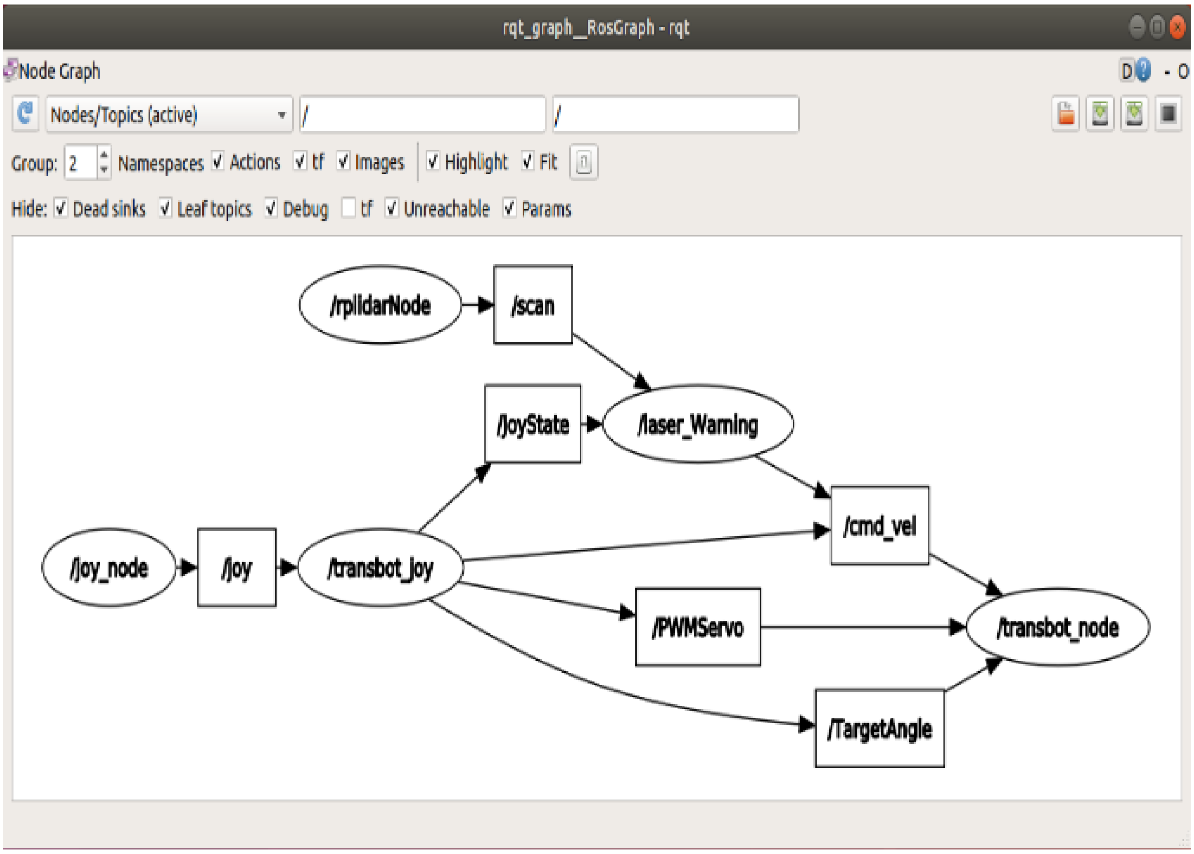
Parameter	Range	Analysis
【LaserAngle】	【10，180】	Lidar detection angle (angle of left and right side)
【ResponseDist】	【0.0，8.0】	Robot response distance
【switch】	【False，True】	Robot movement【start/pause】

【ang_Kp】、【ang_Ki】、【ang_Kd】：PID debugging of car angular speed.

【switch】Click the box in front of [switch], the value of [switch] is True, and the car will stop. [Switch] The default is False, and the car moves.

View node

```
rqt_graph
```



3.2、Source code analysis

launch file

- base.launch

```
<launch>
    <!-- Start the lidar node -->
    <include file="$(find rplidar_ros)/launch/rplidar.launch"/>
    <!-- Dynamic debugging tool node -->
<!--    <node pkg="rqt_reconfigure" type="rqt_reconfigure" name="rqt_reconfigure"
output="screen"/>-->
    <!-- Start the car chassis drive node -->
    <node pkg="transbot_bringup" type="transbot_driver.py" name="transbot_node"
required="true" output="screen">
        <param name="imu" value="/transbot/imu"/>
        <param name="vel" value="/transbot/get_vel"/>
    </node>
    <!-- Handle control node  -->
    <include file="$(find transbot_ctrl)/launch/transbot_joy.launch"/>
</launch>
```

- laser_Warning.launch

```
<launch>
    <!-- Start base.launch file -->
    <include file="$(find transbot_laser)/launch/base.launch"/>
    <!-- Start the lidar guard node  -->
    <node name='laser_Warning' pkg="transbot_laser" type="laser_Warning.py"
required="true" output="screen"/>
</launch>
```

py code: ~/rplidar_ws/src/transbot_laser/scripts/laser_Warning.py

Main code analysis

```
 # Create a distance list, put the effective distance in the detection range
into the list
        minDistList = []
        # Create a serial number and put the ID corresponding to the effective
distance into the list
        minDistIDList = []
        for i in np.argsort(ranges):
            if len(np.array(scan_data.ranges)) == 720:
                # Retain valid data by clearing the data of unnecessary sectors
                if i < self.laserAngle * 2:
                    minDistList.append(ranges[i])
                    minDistIDList.append(i / 2)
                elif (720 - self.laserAngle * 2) <= i:
                    minDistList.append(ranges[i])
                    minDistIDList.append(i / 2 - 360)
            if len(np.array(scan_data.ranges)) == 360:
                # Retain valid data by clearing the data of unnecessary sectors
                if i < self.laserAngle:
                    minDistList.append(ranges[i])
                    minDistIDList.append(i)
                elif (360 - self.laserAngle) <= i :
                    minDistList.append(ranges[i])
                    minDistIDList.append(i - 360)
        if len(minDistList) == 0: return
        # Find the minimum distance
        minDist = min(minDistList)
        # Find the ID corresponding to the minimum distance
        minDistanceAngle = minDistIDList[minDistList.index(minDist)]
```

According to the position of the target, the car will move to the corresponding position autonomously.