4.Lidar guard

Note: Section 2-10 takes the transbot crawler as an example. Users need to modify it according to their own motion model. These courses are only used as running demos.

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.

4.1、Instructions

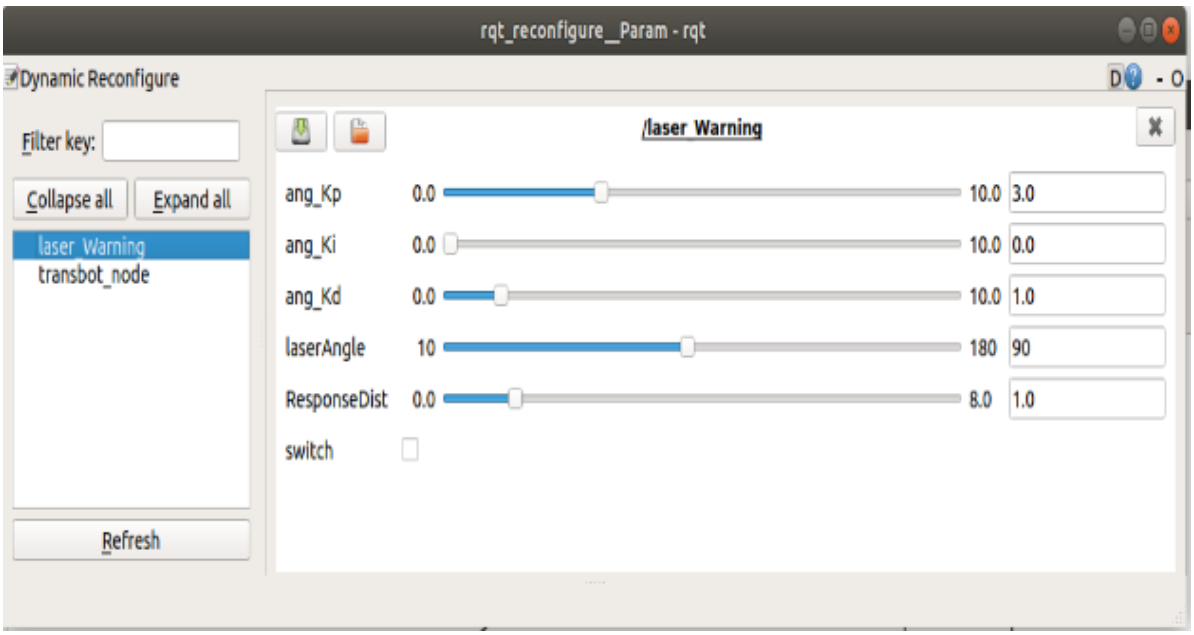
Note: The [R2] of the handle remote controller can [Pause/Open] for all functions of robot car

Start up

```
roslaunch transbot_laser laser_Warning.launch
```

Dynamic debugging parameters

```
rosrun rqt_reconfigure rqt_reconfigure
```



Parameter analysis:

Parameter	Range	Analysis
【LaserAngle】	【10，85】	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.

```
rqt_graph
```

4.2、Source code analysis

launch file

- base.launch

```
<launch>
    <!-- Activate the lidar node -->
    <include file="$(find yahboomcar_laser)/launch/lidar.launch"/>
    <!-- Start the car chassis drive node -->
    <include file="$(find yahboomcar_bringup)/launch/yahboomcar.launch"/>
    <!-- Handle control node -->
    <include file="$(find yahboomcar_ctrl)/launch/yahboom_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：~/ydlidar_ws/src/transbot_laser/scripts/laser_Warning.py

Main code analysis

```
 def registerScan(self, scan_data):
        if not isinstance(scan_data, LaserScan): return
        # Record the laser scan and publish the position of the nearest object
(or point to a point)
        ranges = np.array(scan_data.ranges)
        # create distance list, put the effective distance within the detection
range into the list
        minDistList = []
        # Create a serial number and place the ID corresponding to the valid
distance in the list
        minDistIDList = []
        # if we already have a last scan to compare to:
        for i in range(len(ranges)):
            angle = (scan_data.angle_min + scan_data.angle_increment * i) *
RAD2DEG
            # if angle > 90: print "i: {},angle: {},dist: {}".format(i, angle,
scan_data.ranges[i])
```

```
            #print "i: {},angle: {},dist: {}".format(i, angle,
scan_data.ranges[i])
            #Preserve valid data by clearing data from unneeded sectors
            if abs(angle) < self.laserAngle and ranges[i] != 0.0:
                minDistList.append(ranges[i])
                minDistIDList.append(angle)
        if len(minDistList) == 0: return
        # Find the minimum distance
        minDist = min(minDistList)
        # Find the ID corresponding to the minimum distance
        minDistID = minDistIDList[minDistList.index(minDist)]
        if self.ros_ctrl.Joy_active or self.switch == True:
            if self.Moving == True:
                self.ros_ctrl.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()
        # The PID algorithm is used to make the car move to the corresponding
position steadily
        ang_pid_compute = self.ang_pid.pid_compute((180 - abs(minDistID)) / 128,
0)
        print(str(minDistID)+" "+str(ang_pid_compute))

        if minDistID > 0: velocity.angular.z = ang_pid_compute
        else: velocity.angular.z = -ang_pid_compute
        if ang_pid_compute < 0.2: velocity.angular.z = 0
        if abs(minDistID) < 10: velocity.angular.z = 0
        self.ros_ctrl.pub_vel.publish(velocity)
```

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