3、雷达警卫

功能包路径：~/rplidar_ws/src/transbot_laser

雷达警卫玩法介绍：

- 设定激光雷达检测角度和响应距离。
- 开启小车后，小车面对距离小车最近的目标。
- 当目标与小车的距离小于响应距离时，蜂鸣器一直响，直到响应距离范围内没有目标。
- 可调节小车角速度PID，使得小车旋转效果最佳。

3.1、使用方法

注意：遥控手柄的【R2】具备所有玩法的【暂停/开启】的功能。

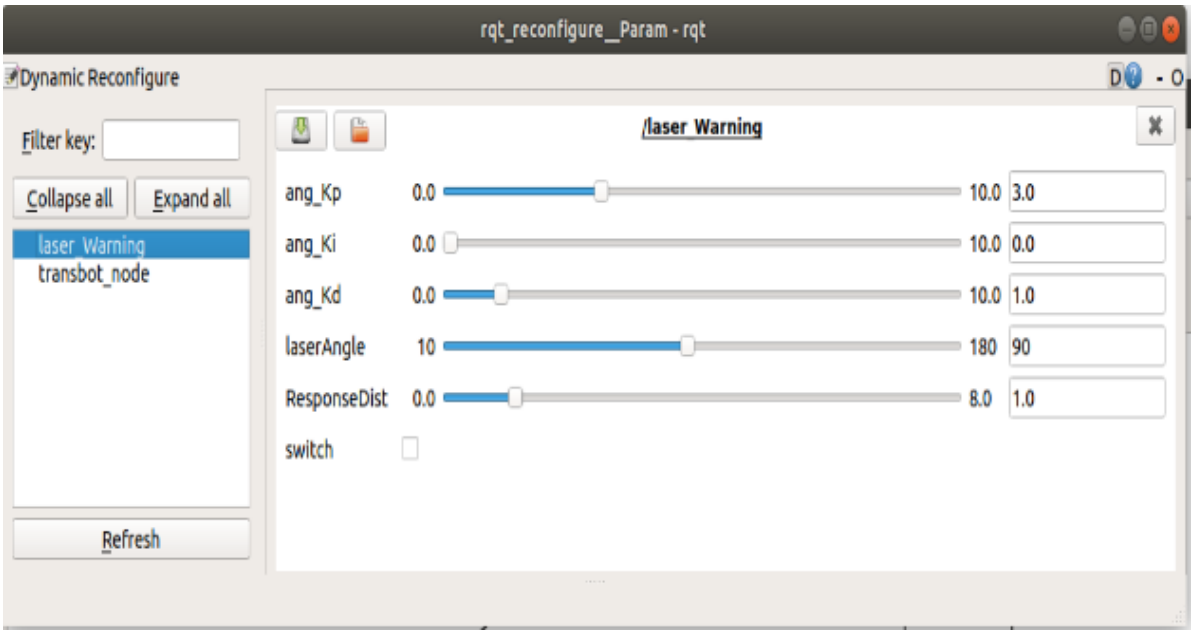
一键启动

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

lidar_type参数：使用激光雷达的型号：[a1,a2,a3,s1,s2]。

动态调试参数

```
rosrun rqt_reconfigure rqt_reconfigure
```



参数解析：

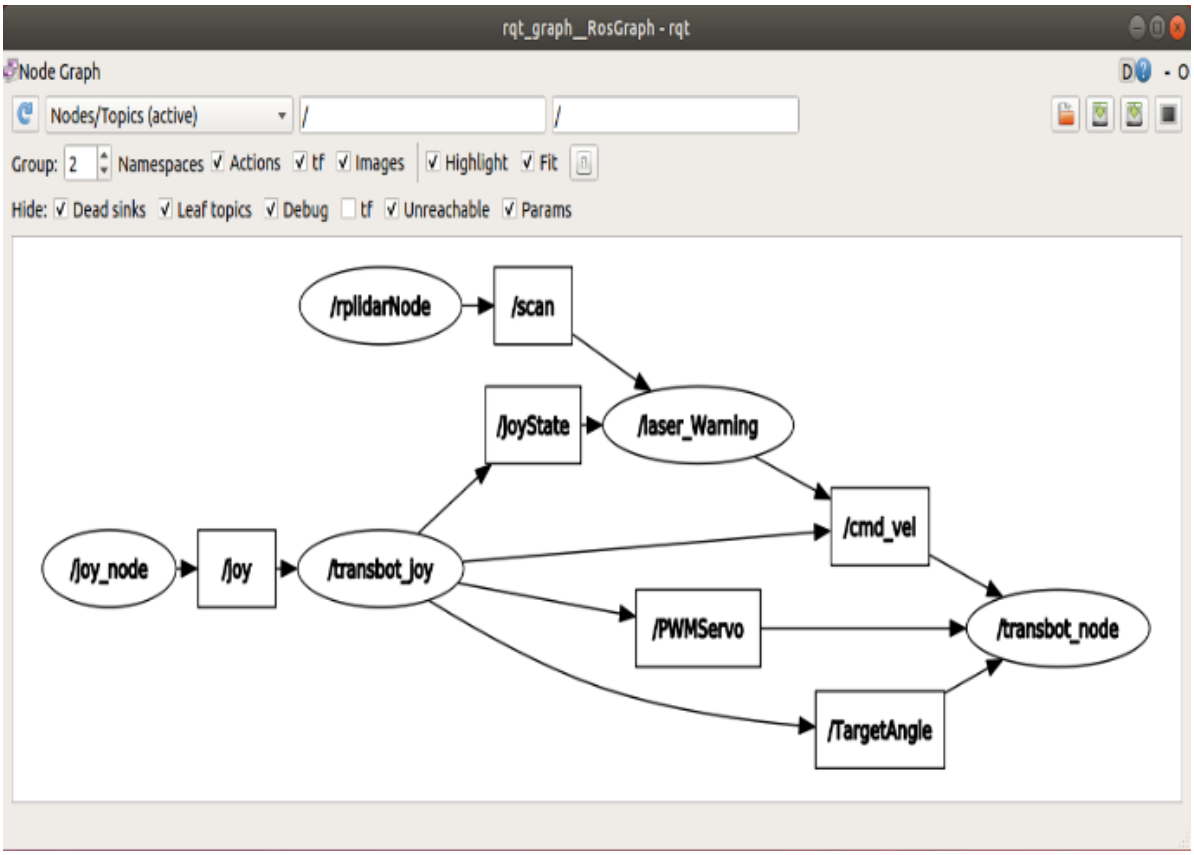
参数	范围	解析
【LaserAngle】	【10，180】	激光雷达检测角度（左右一侧角度）
【ResponseDist】	【0.0，8.0】	小车响应距离
【switch】	【False，True】	小车运动【开始/暂停】

【ang_Kp】、【ang_Ki】、【ang_Kd】：小车角速度PID调试。

【switch】前面的方框，点击【switch】的值为True，小车停止。【switch】默认为False，小车运动。

节点查看

```
rqt_graph
```



3.2、源码解析

launch文件

- base.launch

```
<launch>
    <!-- 启动激光雷达节点-->
    <arg name="lidar_type" default="a1" doc="lidar_type type [a1,a2,a3,s1,s2]"/>
    <include file="$(find rplidar_ros)/launch/rplidar.launch">
        <arg name="lidar_type" value="$(arg lidar_type)"/>
    </include>
    <!-- 动态调试工具节点-->
<!--    <node pkg="rqt_reconfigure" type="rqt_reconfigure" name="rqt_reconfigure"
output="screen"/>-->
    <!-- 启动小车底盘驱动节点-->
```

```
    <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>
    <!-- 手柄控制节点 -->
    <include file="$(find transbot_ctrl)/launch/transbot_joy.launch"/>
</launch>
```

- laser_Warning.launch

```
<launch>
    <!-- 启动base.launch文件 -->
    <arg name="lidar_type" default="a1" doc="lidar_type type [a1,a2,a3,s1,s2]"/>
    <include file="$(find transbot_laser)/launch/base.launch">
        <arg name="lidar_type" value="$(arg lidar_type)"/>
    </include>
    <!-- 启动激光雷达警卫节点 -->
    <node name='laser_Warning' pkg="transbot_laser" type="laser_Warning.py"
required="true" output="screen"/>
</launch>
```

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

主要代码解析

```
        # 创建距离列表，将检测范围内的有效距离放入列表中
        minDistList = []
        # 创建序列号，将有效距离对应的ID放入列表中
        minDistIDList = []
        for i in np.argsort(ranges):
            if len(np.array(scan_data.ranges)) == 720:
                # 通过清除不需要的扇区的数据来保留有效的数据
                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:
                # 通过清除不需要的扇区的数据来保留有效的数据
                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
        # 找到最小距离
        minDist = min(minDistList)
        # 找到最小距离对应的ID
        minDistanceAngle = minDistIDList[minDistList.index(minDist)]
```

根据目标出现的位置，小车自主移动到相应的位置。