Lidar avoiding

The following workspace contains the entire ydlidar_ws function package.

If you need to transplant it to your own main development board, you need to copy all the function packages to the src of the workspace for compilation, and install the corresponding environment.

Note: This course uses Rosmaster-X3Plus as an example. Users need to modify it according to their own motion model. The course only explains the implementation method.

Function package path: ~/ydlidar_ws/src/yahboomcar_laser

Introduction to lidar obstacle avoidance gameplay:

- Set lidar detection angle and response distance
- After turning on the car, the car will drive straight when there are no obstacles.
- Determine the direction in which the obstacle appears on the car (front left, front right, right in front)
- Respond according to the position of the car where obstacles appear (turn left, turn right, turn left in a large circle, turn in a large right circle)

1. Start

Input following command:

```
roslaunch yahboomcar_laser laser_Avoidance.launch
```

Input following command to debug dynamic parameters

```
rosrun rqt_reconfigure rqt_reconfigure
```

View the node graph

```
rqt_graph
```

2. Source code analysis

2.1 launch file

laser_Avoidance.launch

```
<launch>
    <!-- 启动base.launch文件 -->
    <!-- Launch the base.launch file -->
    <include file="$(find yahboomcar_laser)/launch/base.launch"/>
    <!-- 启动激光雷达避障节点 -->
    <!-- Activate lidar obstacle avoidance node -->
    <node name='laser_Avoidance' pkg="yahboomcar_laser"
type="laser_Avoidance.py" required="true" output="screen"/>
</launch>
```

The base.launch file is to start the car chassis and lidar, mainly look at laser_Avoidance.py file.

Code path：~/ydlidar_ws/src/yahboomcar_laser/scripts

```
    def registerScan(self, scan_data):
        if not isinstance(scan_data, LaserScan): return
        # 记录激光扫描并发布最近物体的位置（或指向某点）
        # Record the laser scan and publish the position of the nearest object
(or point to a point)
        ranges = np.array(scan_data.ranges)
        self.Right_warning = 0
        self.Left_warning = 0
        self.front_warning = 0
        # 按距离排序以检查从较近的点到较远的点是否是真实的东西
        # 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])
            # 通过清除不需要的扇区的数据来保留有效的数据
            if -10 > angle > -10-self.LaserAngle:
                if ranges[i] < self.ResponseDist:
                    self.Right_warning += 1
                    #print(angle)
            if 10+self.LaserAngle > angle > 10:
                if ranges[i] < self.ResponseDist:
                    self.Left_warning += 1
                    #print(angle)
            if abs(angle) < 10:
                if ranges[i] <= self.ResponseDist:
                    self.front_warning += 1
                    #print(angle)
```

The judgment range of angle needs to be modified according to the actual direction of the lidar at 0° angle.

self.LaserAngle represents the angle of lidar scanning.

self.ResponseDist represents the distance of the response.

When the distance to an obstacle scanned by the lidar is less than this value, the object is deemed to be an obstacle, and self.Right_warning/self.Left_warning/self.front_warning needs to be accumulated.

The subsequent program will judge based on these three values. Obstacles Where the object is, then the relative processing is done.