3、Lidar avoid

Note: This course takes Rosmaster-X3 as an example. Users need to modify according to their own motion model.

The course only explains the implementation method.

Function packag path: ~/oradar_ws/src/yahboomcar_laser

Introduction to lidar obstacle avoidance:

1. Set lidar detection angle and response distance
2. After the car is started, the car can travel in a straight line without obstacles
3. Determine the position of the obstacle appearing in the car (left front, right front, right front)
4. Respond according to the position of the obstacle in the car (turn left, turn right, spin left, spin right)

3.1、Start

Input following command.

```
roslaunch yahboomcar_laser laser_Avoidance.launch
```

Input following command to dynamic debugging parameters

```
rosrun rqt_reconfigure rqt_reconfigure
```

Input following command to view node picture.

```
rqt_graph
```

3.2、Source code analysis

3.2.1、launch file

laser_Avoidance.launch

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

The base.launch file is used to start the car chassis and radar, mainly to check the laser_ Avoidance.py.

Code path: ~/oradar_ws/src/yahboomcar_laser/scripts
The core code is as follows.

This part is mainly used to enter the callback function after receiving the radar information to determine whether there are real things on the left, front and right.

```
    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 360 > angle > 360 - self.LaserAngle:
                if ranges[i] < self.ResponseDist: self.Right_warning += 1
            if 180 < angle < self.LaserAngle + 180:
                if ranges[i] < self.ResponseDist: self.Left_warning += 1
            if 270-self.LaserAngle<angle<270+self.LaserAngle:
                if ranges[i] <= self.ResponseDist: self.front_warning += 1
```

The judgment range of angle here needs to be modified according to the orientation of the actual lidar 0 ° angle (* * The radar 0 ° angle is the position with a → on the radar * *). Self.LaserAngle represents the lidar scanning angle, and self.ResponseDistrepresents the response distance.

When the distance of the obstacle scanned by the lidar is less than this value, the object is recognized as an obstacle and needs to be accumulated to self.Right_ warning/self.Left_ warning/self.front_ In warning, the latter program will judge where the obstacle is based on these three values, and then make relative processing, such as release speed, etc. For detailed implementation process, please check the code.