5.Lidar follow

The following workspace contains the entire rplidar_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-X3 as an example. Users need to modify it according to their own motion model. The course only explains the implementation method.

Function package path: ~/rplidar_ws/src/yahboomcar_laser

Introduction to lidar following funtion:

- Set lidar detection angle and distance.
- After turning on the car, the car will follow the target closest to the car and keep a certain distance.
- The PID can adjust the linear speed and angular speed of the car to make the car follow the best effect.

5.1 Start

Input following command:

```
roslaunch yahboomcar_laser laser_Tracker.launch
```

Input following command to debug dynamic parameters

```
rosrun rqt_reconfigure rqt_reconfigure
```

View the node graph

```
rqt_graph
```

5.2 Source code analysis

5.2.1 launch file

laser_Tracker.launch

```
<launch>
    <!-- 启动base.launch文件 -->
    <!-- Launch the base.launch file -->
    <include file="$(find yahboomcar_laser)/launch/base.launch"/>
    <!-- 启动激光雷达跟随节点 -->
    <!-- Activate lidar follow node -->
    <node name='laser_Tracker' pkg="yahboomcar_laser" type="laser_Tracker.py"
required="true" output="screen"/>
</launch>
```

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

Source code path: ~/rplidar_ws/src/yahboomcar_laser/scripts

The core code part is as follows. This part mainly enters the callback function after receiving the radar information.

```
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)
        offset = 0.5
        frontDistList = []
        frontDistIDList = []
        minDistList = []
        minDistIDList = []
        # 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 270-self.priorityAngle<angle<270self.priorityAngle:
                if ranges[i] < (self.ResponseDist + offset):
                    frontDistList.append(ranges[i])
                    frontDistIDList.append(angle)
            elif 270-self.laserAngle < angle < 270-self.priorityAngle:
                minDistList.append(ranges[i])
                minDistIDList.append(angle)
            elif 270+self.priorityAngle<angle<270+self.laserAngle:
                minDistList.append(ranges[i])
                minDistIDList.append(angle)
        # 找到最小距离和最小距离对应的ID
        # Find the minimum distance and the ID corresponding to the minimum
distance
        if len(frontDistIDList) != 0:
            minDist = min(frontDistList)
            minDistID = frontDistIDList[frontDistList.index(minDist)]
        else:
            minDist = min(minDistList)
            minDistID = minDistIDList[minDistList.index(minDist)]
```

```
        # rospy.loginfo('minDist: {}, minDistID: {}'.format(minDist, minDistID))
        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
        velocity = Twist()
        if abs(minDist - self.ResponseDist) < 0.1: minDist = self.ResponseDist
        velocity.linear.x = -self.lin_pid.pid_compute(self.ResponseDist,
minDist)
        ang_pid_compute = self.ang_pid.pid_compute((180 - abs(minDistID)) / 72,
0)
        if minDistID > 0: velocity.angular.z = -ang_pid_compute
        else: velocity.angular.z = ang_pid_compute
        if ang_pid_compute < 0.02: velocity.angular.z = 0
        self.ros_ctrl.pub_vel.publish(velocity)
```

Among them, self.priorityAngle is the angle to be followed first. This value cannot be larger than self.laserAngle, otherwise it is meaningless.

After calculating the minimum ID, the speed data is released to the bottom layer through PID calculation, so that the car follows the movement of the nearest object.

The angle judgment here also needs to be modified based on the position of the radar at 0° angle.