5.Lidar follow

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.

ROS package path：~/ydlidar_ws/src/transbot_laser

Introduction of lidar follow:

- Set the detection angle and distance of the lidar.
- After turning on the car, the car will follow the target closest to the car and keep a certain distance.
- When there are obstacles behind the trolley, the buzzer keeps beeping and stops moving backwards until there are no obstacles.
- The PID of the linear speed and angular velocity of the trolley can be adjusted to make the car follow the best effect.

5.1、Instructions

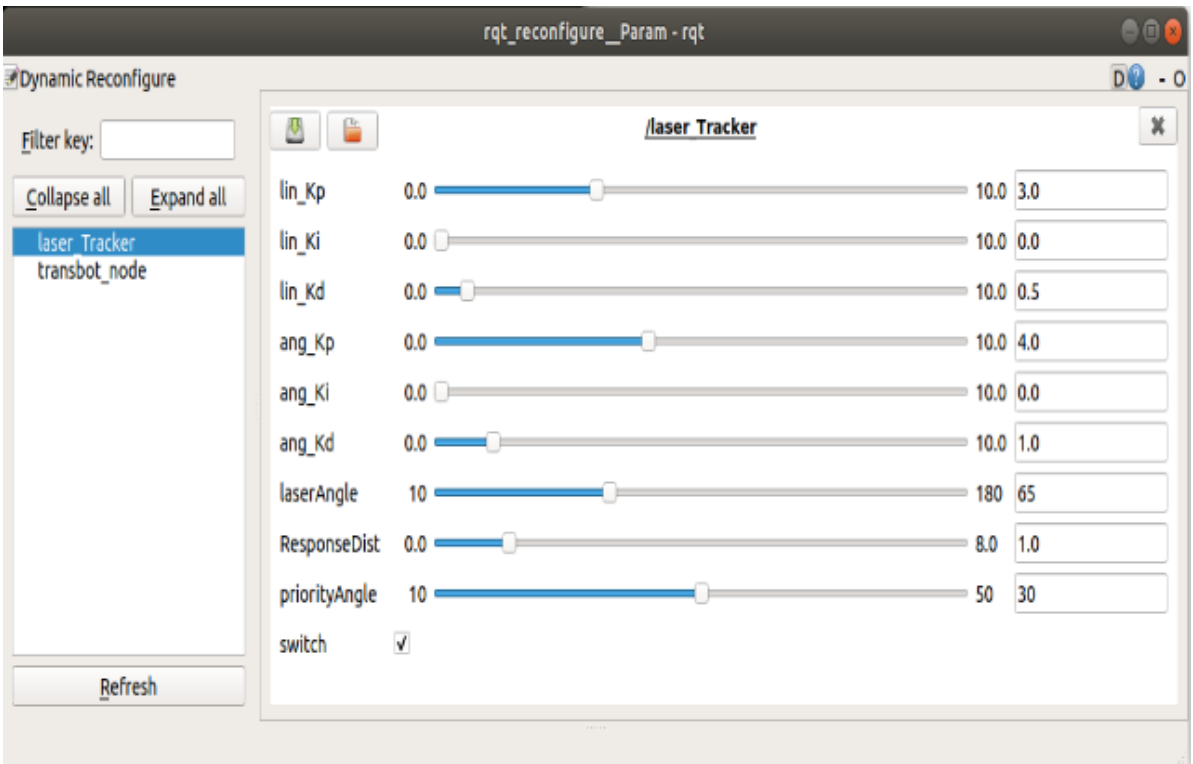
Note: The [R2] of the handle remote controller can [Pause/Open] for all functions of robot car

Start up

```
roslaunch transbot_laser laser_Tracker.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
【priorityAngle】	【10，50】	The car prioritizes the following range (angle of left and right side)
【switch】	【False，True】	Robot movement【start/pause】

【lin_Kp】、【lin_Ki】、【lin_Kd】：PID debugging of car linear speed.

【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.

【priorityAngle】cannot be smaller than【laserAngle】.

View node

```
rqt_graph
```

5.2、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_Avoidance.launch

```
<launch>
    <!-- Launch the base.launch file -->
    <include file="$(find yahboomcar_laser)/launch/base.launch"/>
    <!-- Activate lidar follow node -->
    <node name='laser_Tracker' pkg="yahboomcar_laser" type="laser_Tracker.py"
required="true" output="screen"/>
</launch>
```

py code path：~/ydlidar_ws/src/transbot_laser/scripts/laser_Tracker.py

```
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)
        offset = 0.4
        frontDistList = []
        frontDistIDList = []
        minDistList = []
        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])
            # Preserve valid data by clearing data from unneeded sectors
            if abs(angle) < self.priorityAngle:
                if ranges[i] < (self.ResponseDist + offset) and ranges[i] !=
0.0:
                    frontDistList.append(ranges[i])
                    frontDistIDList.append(angle)
            elif abs(angle) > self.priorityAngle and abs(angle) <
self.laserAngle and ranges[i] != 0.0:
                minDistList.append(ranges[i])
                minDistIDList.append(angle)
        # 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
        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()
        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 < 0.2: velocity.angular.z = 0
        if abs(minDistID) < 10: velocity.angular.z = 0
        self.ros_ctrl.pub_vel.publish(velocity)
```