5. Patrol

5.1. Description of program functions

After the program starts, the car moves according to the set patrol route. During operation, the lidar works at the same time, and if an obstacle is detected within the detection range, it will stop.

This function is enabled/suspended by clicking [Switch] after starting the dynamic parameter adjuator on the virtual machine side, and you can modify the "Command" parameter to set different patrol routes.

In addition, the [L1] button on the handle locks/turns on the car's motion controls. When motion control is enabled, the function is locked; This function can be turned on when the motion control is locked.

5.2. Program Code Reference Path

After SSH connection car, the location of the function source code is located at,

```
#python文件#python file
/userdata/yahboomcar_ws/src/yahboomcar_bringup/yahboomcar_bringup/patrol.py
#launch文件#launch file
/userdata/yahboomcar_ws/src/yahboomcar_bringup/launch/yahboomcar_patrol_launch.p
y
```

5.3. Program Startup

5.3.1. Launch commands

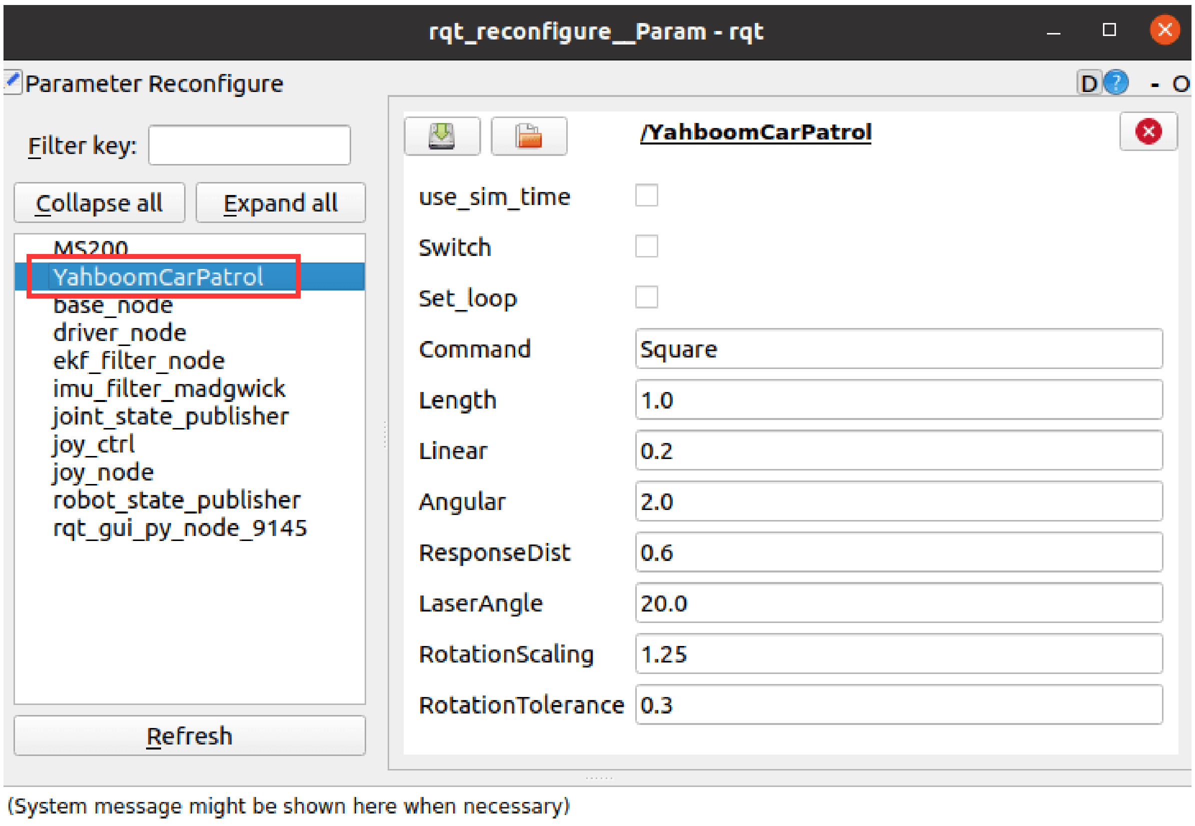
After SSH connects to the car, terminal input,

```
ros2 launch yahboomcar_bringup yahboomcar_patrol_launch.py
```

5.3.2. Modifying parameters on the VM side

On the virtual machine, open the dynamic parameter adjuster, open the terminal input,

```
ros2 run rqt_reconfigure rqt_reconfigure
```



use_sim_time	☐
Switch	☐
Set_loop	☐
Command	Square
Length	1.0
Linear	0.2
Angular	2.0
ResponseDist	0.6
LaserAngle	20.0
RotationScaling	1.25
RotationTolerance	0.3

(System message might be shown here when necessary)

The meanings of the parameters are as follows:,

parameter name	parameter meaning
Switch	Play switch
Set_loop	sets the loop
Command	the patrol route
Linear	The linear velocity
Angular	Angular velocity
Length	The linear distance
ResponseDist	lidar obstacle avoidance response range
RotationScaling	Angle proportion coefficient
RotationTolerance	steering error bear degrees

After the program starts, in the GUI interface of the dynamic parameter regulator interface, enter one of the following routes in the [Command] field:

- LengthTest: indicates a straight line test

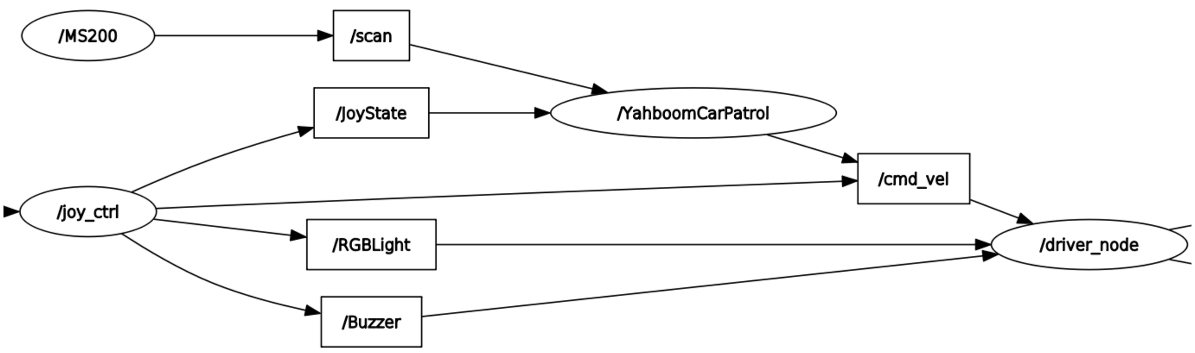
- Circle: patrol on a circular route
- Square: Square route patrol
- Triangle: Patrol the triangle route

After selecting the route, click the blank to write parameters, and then click the [Switch] button to start the patrol movement. After the end of a movement, if [Set_loop] is checked, the last route will be cycled for patrol, otherwise it will stop after completing a patrol.

5.3.3. View the topic communication node graph

Virtual machine terminal input,

```
ros2 run rqt_graph rqt_graph
```



5.4. Core Source Code Parsing

The source code of this code is to subscribe to the TF transformation of odom and base_footprintf, so that you can always know "how long you have gone", and then release speed instructions according to the set route, taking Square as an example, which is analyzed here.

```
#设定巡逻的路线，进入self.Square函数
# To set a patrol route, enter the self.Square function
self.command_src = "Square"
square = self.Square()

def Square(self):
    if self.index < 8:
        if self.index % 2 == 0:
            step_length = self.advancing(self.Length) #走直线# Go straight
            if step_length == True:
                self.index += 1
        else:
            step_spin = self.Spin(90) #旋转90度# Rotate 90 degrees
            if step_spin == True:
                self.index += 1
    else:
        self.index = 0
        return True # 完成4次直线和4次转向后，返回True
# Returns True after completing 4 straight lines and 4 turns
def advancing(self,target_distance):
    #以下是获取xy坐标，与上一时刻的坐标进行计算，计算出自己走了多远
    #获取xy坐标的方式监听odom与base_footprint的tf变换，这部分可参考self.get_position()函
数
```

```
    # The following is to get the xy coordinates and calculate with the
coordinates of the previous moment to calculate how far you have gone
    # The method of obtaining xy coordinates listens to the tf transformation of
odom and base_footprint, which can be referred to the self.get_position()
function
    self.position.x = self.get_position().transform.translation.x
    self.position.y = self.get_position().transform.translation.y
    move_cmd = Twist()
    self.distance = sqrt(pow((self.position.x - self.x_start), 2) +
                        pow((self.position.y - self.y_start), 2))
    self.distance *= self.LineScaling
    print("distance: ",self.distance)
    self.error = self.distance - target_distance
    move_cmd.linear.x = self.Linear
    if abs(self.error) < self.LineTolerance :
        print("stop")
        self.distance = 0.0
        self.pub_cmdVel.publish(Twist())
        self.x_start = self.position.x
        self.y_start = self.position.y
        return True
    else:
        if self.Joy_active == True or self.warning > 10:
            if self.moving == True:
                self.pub_cmdVel.publish(Twist())
                self.moving = False
                print("obstacles")
        else:
            self.pub_cmdVel.publish(move_cmd)
            self.moving = True
        return False

def Spin(self,angle):
    self.target_angle = radians(angle)
    #以下是获取位姿，计算出自己转了多少度，获取位姿可以参考self.get_odom_angle函数，同样也是
监听了odom与base_footprint的TF变换得来的。
    # The following is to obtain the pose and calculate how much you have turned.
To obtain the pose, refer to the self.get_odom_angle function, which is also
obtained by monitoring the TF transformation of odom and base_footprint.
    self.odom_angle = self.get_odom_angle()
    self.delta_angle = self.RotationScaling *
self.normalize_angle(self.odom_angle - self.last_angle)
    self.turn_angle += self.delta_angle
    print("turn_angle: ",self.turn_angle)
    self.error = self.target_angle - self.turn_angle
    print("error: ",self.error)
    self.last_angle = self.odom_angle
    move_cmd = Twist()
    if abs(self.error) < self.RotationTolerance:
        self.pub_cmdVel.publish(Twist())
        self.turn_angle = 0.0
        return True
    if self.Joy_active == True or self.warning > 10:
        if self.moving == True:
            self.pub_cmdVel.publish(Twist())
```

```
                self.moving = False
                print("obstacles")
        else:
            move_cmd.linear.x = self.Linear
            move_cmd.angular.z = copysign(self.Angular, self.error)
            self.pub_cmdVel.publish(move_cmd)
            self.moving = True
```