5. Lidar patrol

5.1. Program function description

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

After starting the dynamic parameter regulator on the virtual machine, click [Switch] to turn on/pause this function, and you can modify the "Command" parameter to set different patrol routes.

In addition, the [L1] button of the handle can lock/open the motion control of the car. When the motion control is turned on, the function will be locked; when the motion control is locked, the function can be turned on.

5.2, Program code reference path

After SSH connects to the car, the source code of this function is located at,

```
#python file
/home/sunrise/yahboomcar_ws/src/yahboomcar_bringup/yahboomcar_bringup/patrol.py
#launch file
/home/sunrise/yahboomcar_ws/src/yahboomcar_bringup/launch/yahboomcar_patrol_laun
ch.py
```

5.3, Program startup

5.3.1, Start command

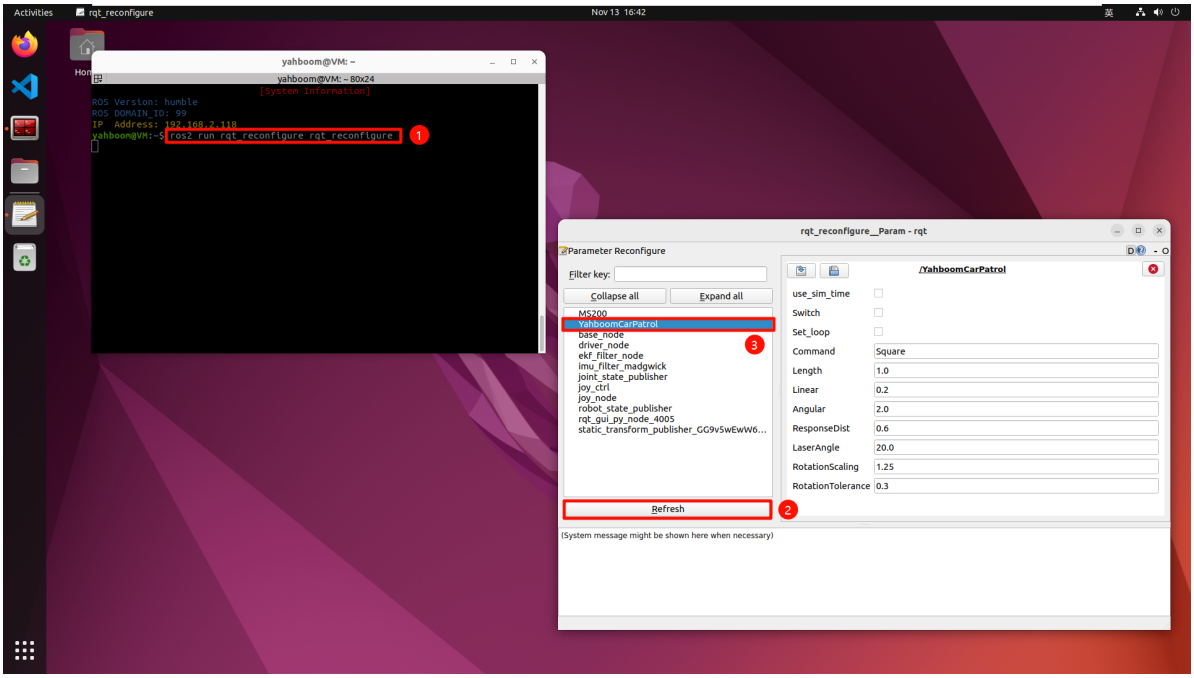
After SSH connects to the car, enter in the terminal,

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

5.3.2, modify parameters on the virtual machine side

Open the dynamic parameter adjuster on the virtual machine side, open the terminal input,

```
ros2 run rqt_reconfigure rqt_reconfigure
```



The meaning of each parameter is as follows,

Parameter name	Parameter meaning
Switch	Gameplay switch
Set_loop	Set loop
Command	Patrol route
Linear	Linear speed
Angular	Angular speed
Length	Linear motion distance
ResponseDist	Radar obstacle avoidance response distance
RotationScaling	Angle ratio coefficient
RotationTolerance	Steering error tolerance

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

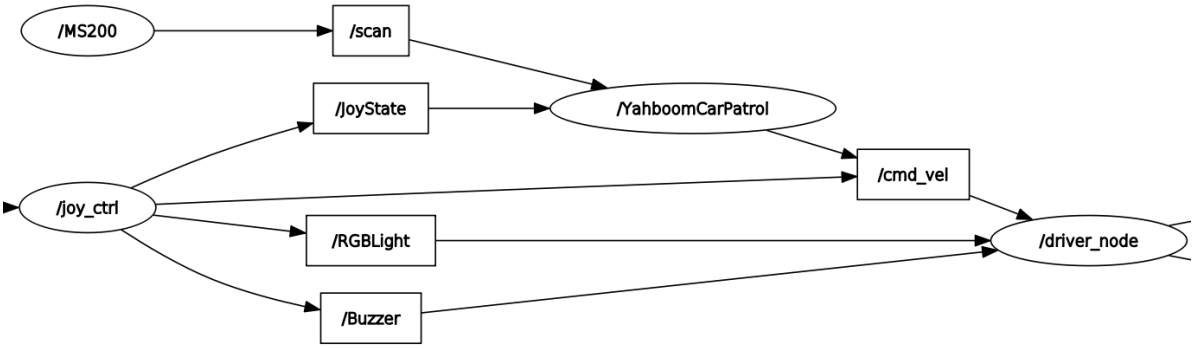
- LengthTest: Straight line test
- Circle: Circular route patrol
- Square: Square route patrol
- Triangle: Triangle route patrol

After selecting the route, click the blank space to write the parameters, and then click the [Switch] button to start the patrol movement. After one movement is completed, if [Set_loop] is checked, the patrol will loop the previous route, otherwise it will stop after completing one 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 analysis

The implementation source code of this code subscribes to the TF transformation of odom and base_footprintf, so that you can know "how long you have walked" at any time, and then issue speed instructions according to the set route. Taking Square as an example, here is the analysis,

```
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)
            if step_length == True:
                self.index += 1
        else:
            step_spin = self.Spin(90)
            if step_spin == True:
                self.index += 1
    else:
        self.index = 0
        return True

def advancing(self,target_distance):
    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)
        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
```