Patrol

1. Function description of the programme

After the programme starts, open the patrol route set by the dynamic parameter setter, click "switch" in the GUI interface, the car will move according to the set patrol route, during the running process, the lidar will work at the same time, and it will stop if it detects obstacles within the detection range. After the joystick programme is on, you can also pause/continue the movement of the car by pressing R2 button.

2. Programme Code Reference Path

The location of the source code for this function is located at.

```
/home/pi/cartographer_ws2/src/yahboom_laser/yahboom_laser/laser_Patrol_xgo_RS200.py
```

3. Program startup

3.1 Start command

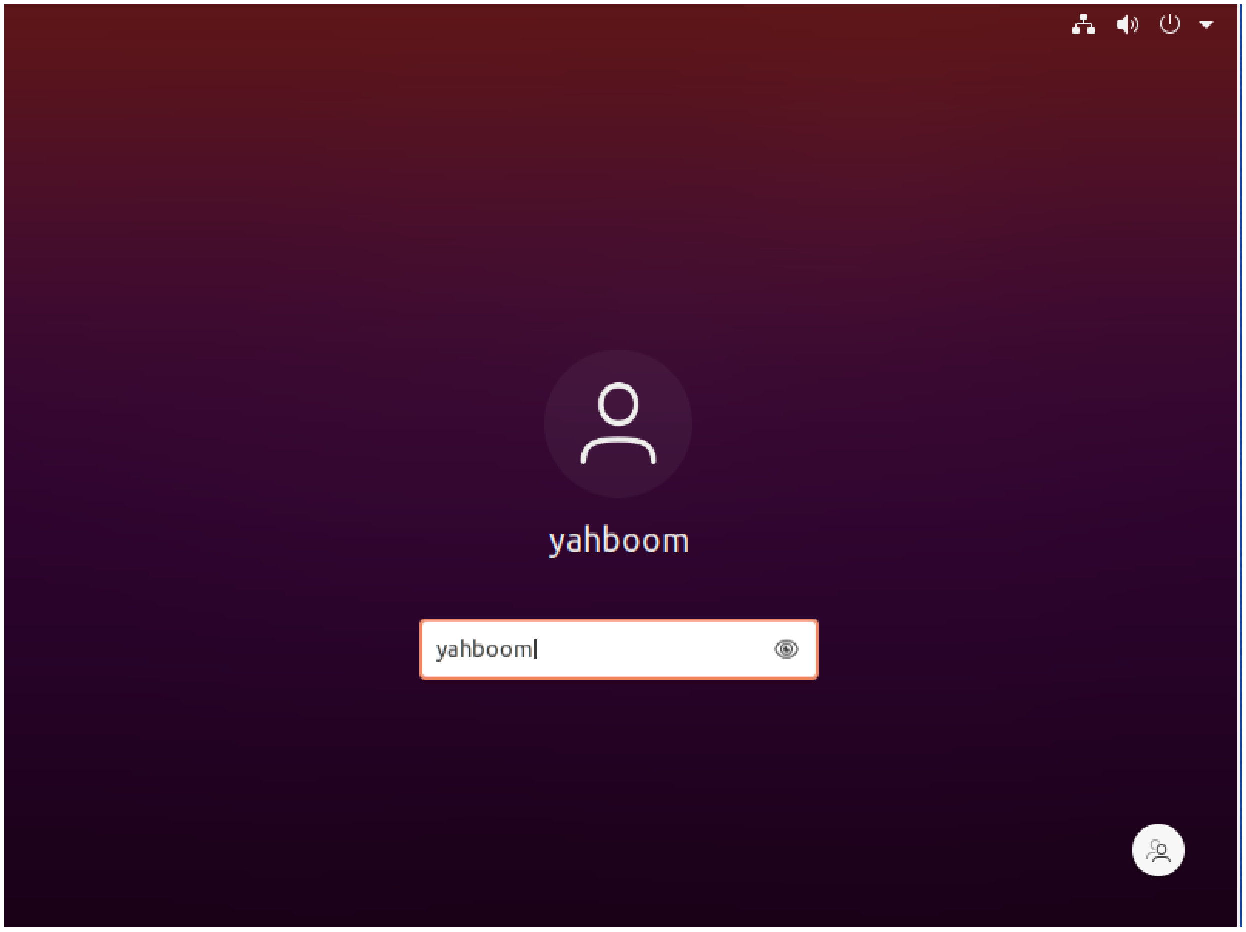
Multi-computer communication id modification can refer to the tutorial: 14. radar build map navigation \6. ROS2 environment entity mechanical dog state acquisition \ ROS2 environment to obtain the real joints of the mechanical dog data.pdf

Mechanical dog chassis and lidar has been set to boot self-start, if you find that it did not start please enter in the terminal.

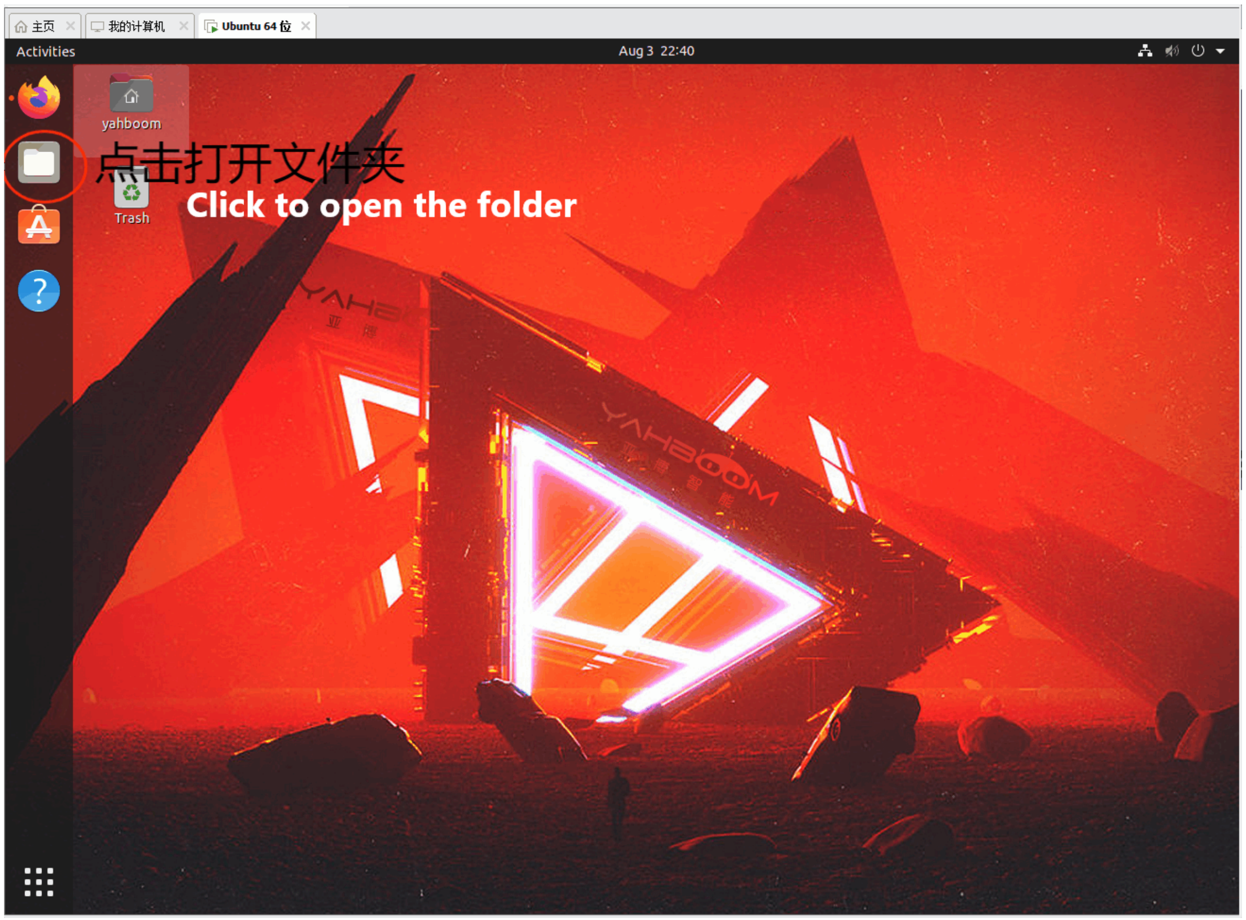
```
sudo systemctl restart YahboomStart.service
```

Since the mechanical dog chassis does not have an odometer, we can use cartographer to post odometer data, so we need the navigation module.

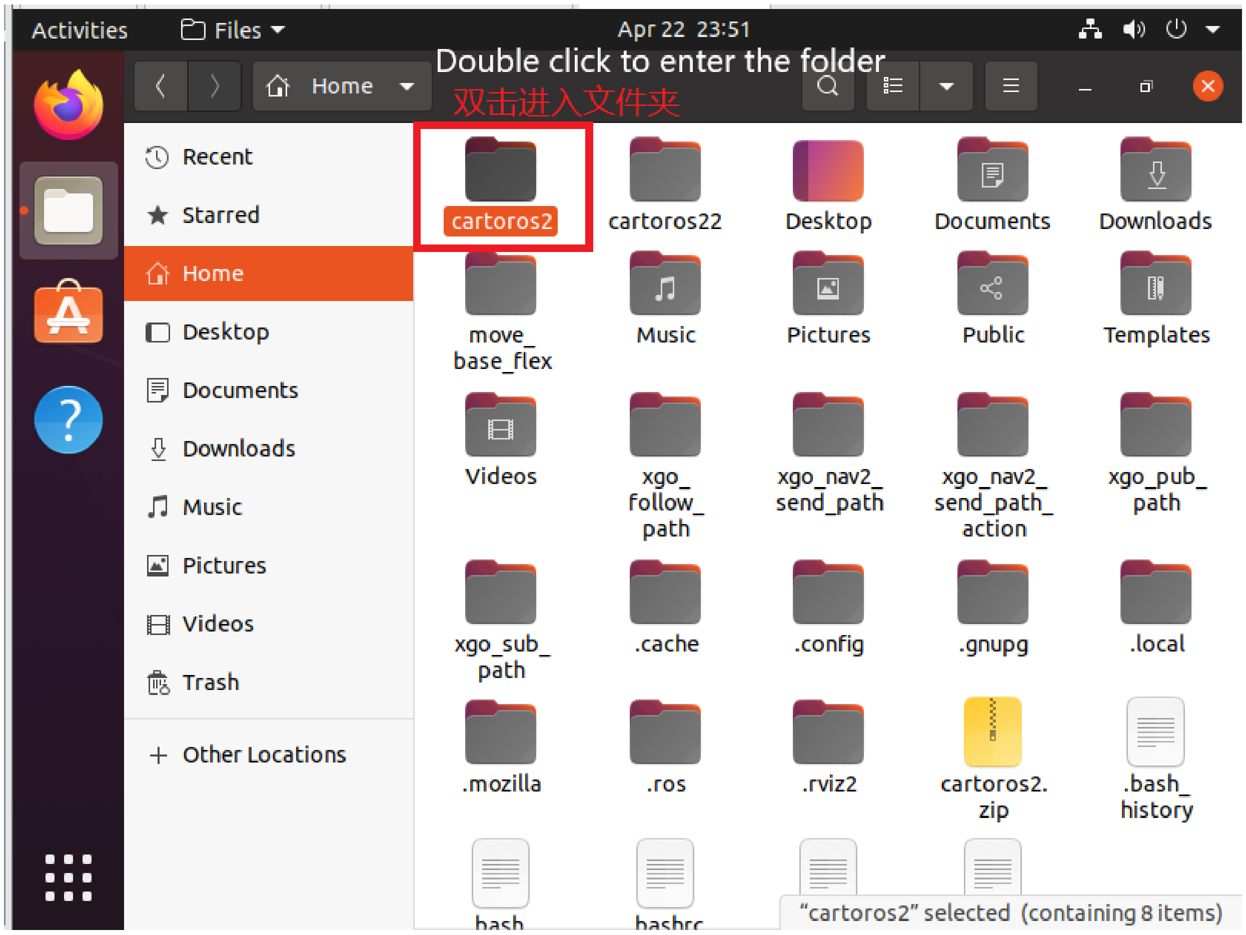
Open the virtual machine, enter the password: yahboom then press enter to enter the system desktop.



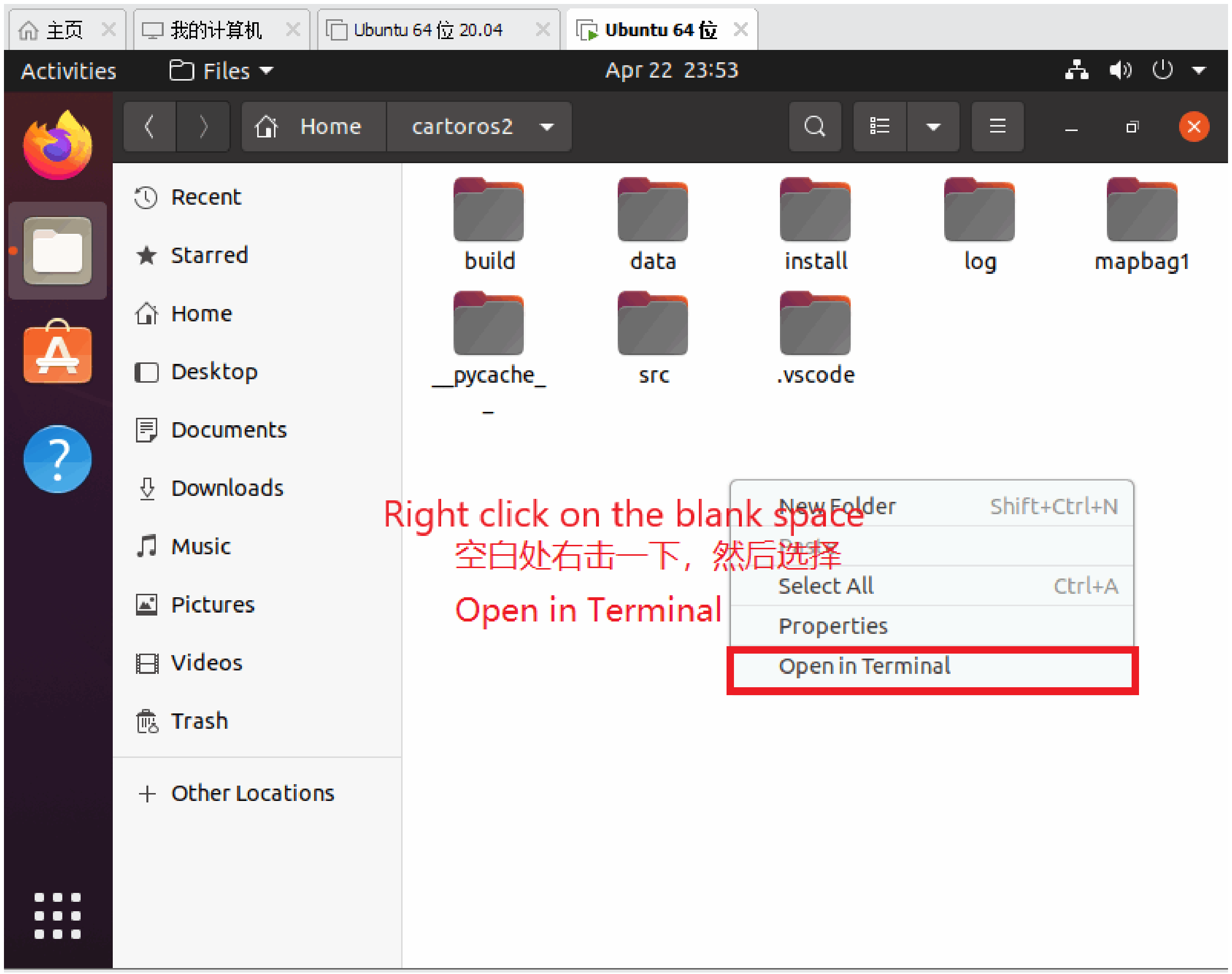
Go to the desktop system and open the folder.



Then double click on the cartoros2 folder



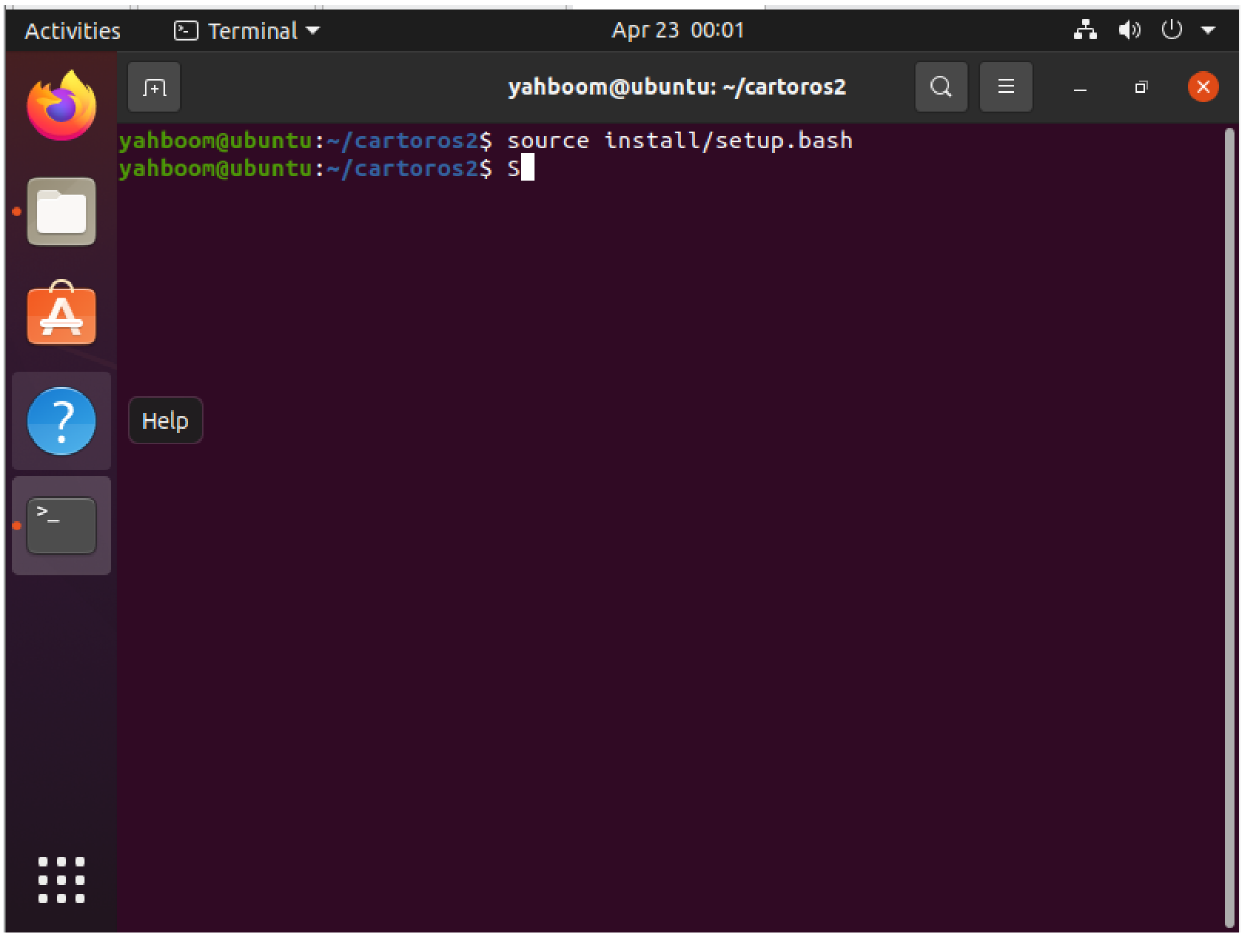
Then right-click in a blank space in the folder and select Open in Terminal



Then activate the environment by typing the following command in the terminal

```
source install/setup.bash
```

Press the Enter key when you have finished typing.

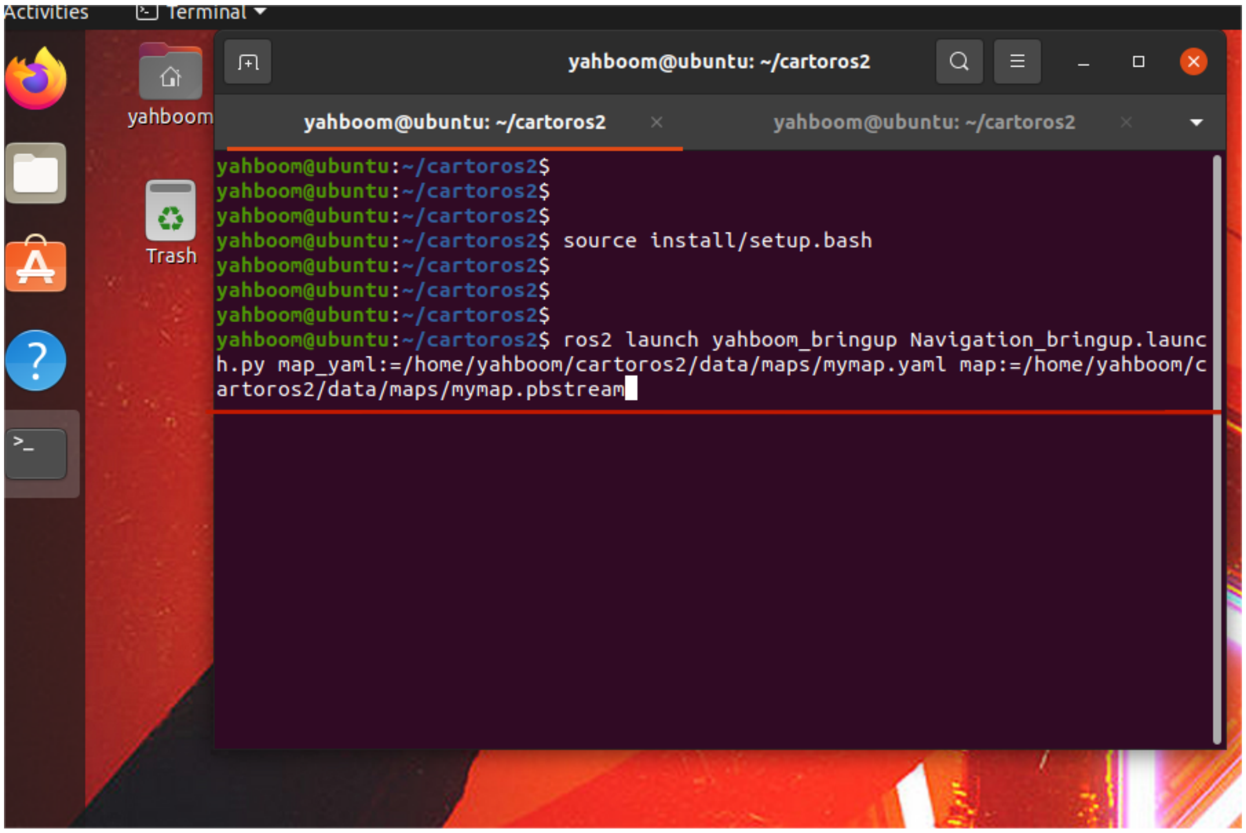


```
yahboom@ubuntu:~/cartoros2$ source install/setup.bash
yahboom@ubuntu:~/cartoros2$ S
```

Then enter the command

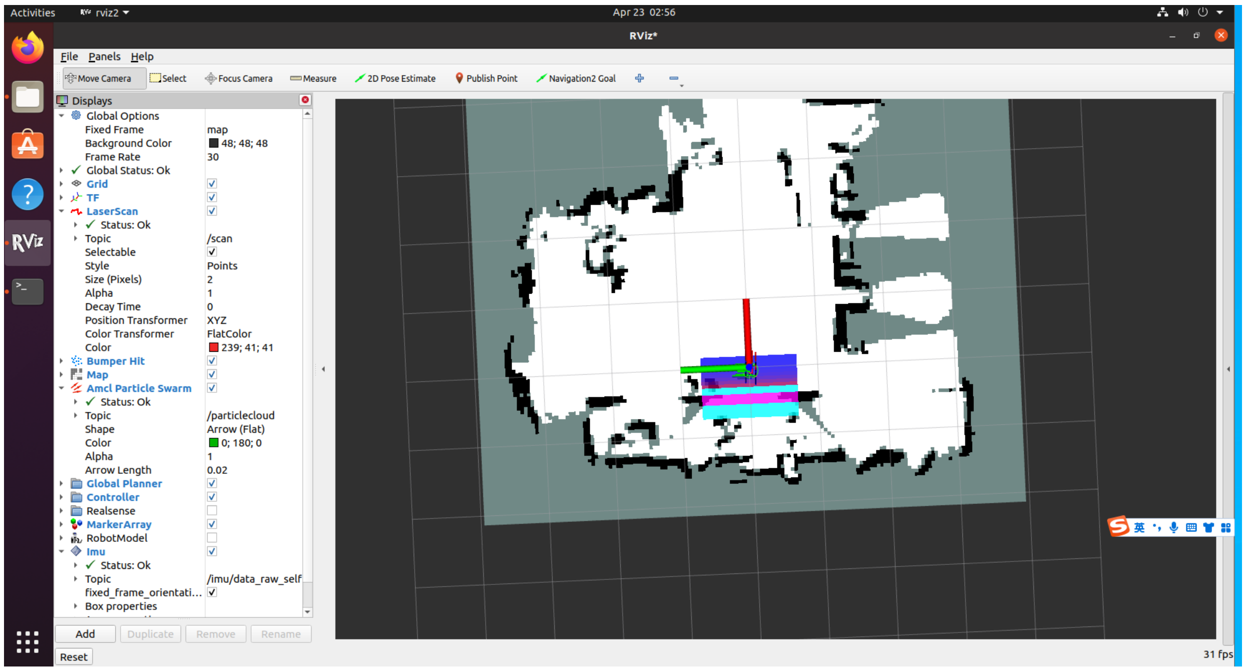
```
ros2 launch yahboom_bringup Navigation_bringup.launch.py
map_yaml:=/home/yahboom/cartoros2/data/maps/mymap.yaml
map:=/home/yahboom/cartoros2/data/maps/mymap.pbstream
```

Where map_yaml is the map's yaml file and map is the map's pbstream file.


Activities
Terminal
yahboom
Trash
yahboom@ubuntu: ~/cartoros2
yahboom@ubuntu: ~/cartoros2
yahboom@ubuntu: ~/cartoros2

```
yahboom@ubuntu:~/cartoros2$
yahboom@ubuntu:~/cartoros2$
yahboom@ubuntu:~/cartoros2$
yahboom@ubuntu:~/cartoros2$ source install/setup.bash
yahboom@ubuntu:~/cartoros2$
yahboom@ubuntu:~/cartoros2$
yahboom@ubuntu:~/cartoros2$
yahboom@ubuntu:~/cartoros2$ ros2 launch yahboom_bringup Navigation_bringup.launc
h.py map_yaml:=/home/yahboom/cartoros2/data/maps/mymap.yaml map:=/home/yahboom/c
artoros2/data/maps/mymap.pbstream
```

Then pressing the enter key can will allow you to navigate.



```
Displays
Global Options
    Fixed Frame                  map
    Background Color             48; 48; 48
    Frame Rate                   30
Global Status: Ok
Grid                             ✓
TF                               ✓
LaserScan                        ✓
    Status: Ok
    Topic                        /scan
    Selectable                   ✓
    Style                        Points
    Size (Pixels)                2
    Alpha                        1
    Decay Time                   0
    Position Transformer         XYZ
    Color Transformer            FlatColor
    Color                        239; 41; 41
Bumper Hit                       ✓
Map                              ✓
Amcl Particle Swarm              ✓
    Status: Ok
    Topic                        /particlecloud
    Shape                        Arrow (Flat)
    Color                        0; 180; 0
    Alpha                        1
    Arrow Length                 0.02
Global Planner                   ✓
Controller                       ✓
Realsense                        ☐
MarkerArray                      ✓
RobotModel                       ☐
Imu                              ✓
    Status: Ok
    Topic                        /imu/data_raw_self
    fixed_frame_orientati...     ✓
    Box properties
```

If the lidar and chassis start-up is complete then you need to enter it in the terminal:

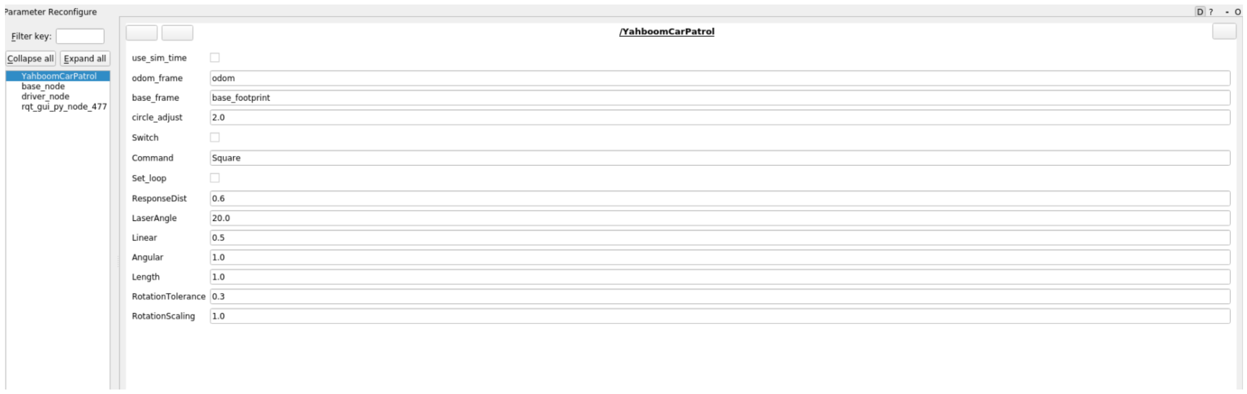
```
cd /home/pi/cartographer_ws2
source install/setup.bash
# Initiate patrol procedures lidar MS200
ros2 run yahboom_laser laser_Patrol_xgo_RS200
```

3.2 Viewing the topic communication node map

Setting the size of a parameter by means of a dynamic parameter regulator, the terminal input, the

```
ros2 run rqt_reconfigure rqt_reconfigure
```

The individual parameters of the dynamic parameter regulator are described as follows.



use_sim_time	☐
odom_frame	odom
base_frame	base_footprint
circle_adjust	2.0
Switch	☐
Command	Square
Set_loop	☐
ResponseDist	0.6
LaserAngle	20.0
Linear	0.5
Angular	1.0
Length	1.0
RotationTolerance	0.3
RotationScaling	1.0

Parameter name	Parameter Meaning
odom_frame	odometer coordinate system
base_frame	base elevation system
circle_adjust	Turning circle angular speed adjustment factor
Switch	Play switch
Command	Patrol routes
Set_loop	Setting up the loop
ResponseDist	lidar Obstacle Avoidance Response Distance
LaserAngle	lidar scanning angle
Linear	linear velocity
Angular	angular velocity
Length	Straight line test distance
RotationTolerance	Steering Error Tolerance
RotationScaling	Corner scaling factor

When the programme starts, enter any of the following routes in the Comand field in the GUI screen of the Dynamic Parameter Regulator's interface:

- 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. If you set loop, you can loop the last route to patrol, if loop is false, then the patrol will stop after finishing.

4. Core source code analysis

The source code of this code is subscribed to odom and base_footprintf's TF transformation, so that we can always know "how long we have travelled", and then according to the set route, issue the speed instruction, take Triangle as an example, here is the analysis.

```
# Set the route of the patrol, enter self.Triangle function
self.command_src = "Triangle"
triangle = self.Triangle()
# Parsed as part of the self.Triangle code.
def Triangle(self):
    if self.index == 0:
        print("Length")
        step1 = self.advancing(self.Length)  # Straight line first, one side of the
triangle.
        #sleep(0.5)
        if step1 == True:
            #self.distance = 0.0
            self.index = self.index + 1;
            self.Switch  =
rclpy.parameter.Parameter('Switch',rclpy.Parameter.Type.BOOL,True)
            all_new_parameters = [self.Switch]
            self.set_parameters(all_new_parameters)
    elif self.index == 1:
            print("Spin")
            step2 = self.Spin(120)# Then change direction and turn to 120, triangle
3*120=360
            #sleep(0.5)
            if step2 == True:
                self.index = self.index + 1;
                self.Switch  =
rclpy.parameter.Parameter('Switch',rclpy.Parameter.Type.BOOL,True)
                all_new_parameters = [self.Switch]
                self.set_parameters(all_new_parameters)
# The following walk through 3 loops, that is, the completion of the triangle
patrol, mainly to see self.advancing and self.Spin function, these two functions
will return True when the execution is complete.
 def advancing(self,target_distance):
     # The following is to get the xy coordinates, and calculate with the
coordinates of the previous moment, to calculate how far you have travelled.
    #Get the xy coordinate by listening to odom and base_footprint's tf
transformation, this part can refer to 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;
        self.Switch  =
rclpy.parameter.Parameter('Switch',rclpy.Parameter.Type.BOOL,False)
        all_new_parameters = [self.Switch]
        self.set_parameters(all_new_parameters)
        return True
    else:
        if self.Joy_active or self.warning > 10:
            if self.moving == True:
                self.pub_cmdVel.publish(Twist())
                self.moving = False
                print("obstacles")
         else:
                #print("Go")
                self.pub_cmdVel.publish(move_cmd)
            self.moving = True
            return False

def Spin(self,angle):
        self.target_angle = radians(angle)
         #The following is to get the position, calculate how many degrees you have
turned, to get the position you can refer to self.get_odom_angle function, also
listen to the TF transformation of odom and base_footprint to get the position.
        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 or self.Switch==False :
            self.pub_cmdVel.publish(Twist())
            self.turn_angle = 0.0
            '''self.Switch  =
rclpy.parameter.Parameter('Switch',rclpy.Parameter.Type.BOOL,False)
            all_new_parameters = [self.Switch]
            self.set_parameters(all_new_parameters)'''
            return True
        if self.Joy_active or self.warning > 10:
            if self.moving == True:
                self.pub_cmdVel.publish(Twist())
                self.moving = False
                print("obstacles")
        else:
            if self.Command == "Square" or self.Command == "Triangle":
                #move_cmd.linear.x = 0.2
                move_cmd.angular.z = copysign(self.Angular, self.error)
            elif self.Command == "Circle":
                length = self.Linear * self.circle_adjust / self.Length
                #The circle_adjust here is the coefficient of the angle of rotation,
calculated length can be understood that the larger, the larger the radius of the
circle will be.
                #print("length: ",length)
                move_cmd.linear.x = self.Linear
                move_cmd.angular.z = copysign(length, self.error)
                #print("angular: ",move_cmd.angular.z)
                '''move_cmd.linear.x = 0.2
                move_cmd.angular.z = copysign(2, self.error)'''
            self.pub_cmdVel.publish(move_cmd)
        self.moving = True
```