

# Patrol

---

## 1. Program function description

After the program is started, open the patrol route set by the dynamic parameter setter and click "switch" on the GUI interface. The car will move according to the set patrol route. During the operation, the radar will work at the same time. If obstacles are detected within the detection range, will stop.

## 2. Program code reference path

The source code of this function is located at,

```
#pi4version
/home/pi/cartographer_ws2/src/yahboom_laser/yahboom_laser/laser_Patrol_RS200.py

#pi5version
/root/yahboomcar_ws/src/yahboom_laser/yahboom_laser/laser_Patrol_RS200.py
```

## 3. Program startup

### 3.1. Start command

**PI4 version driver:**

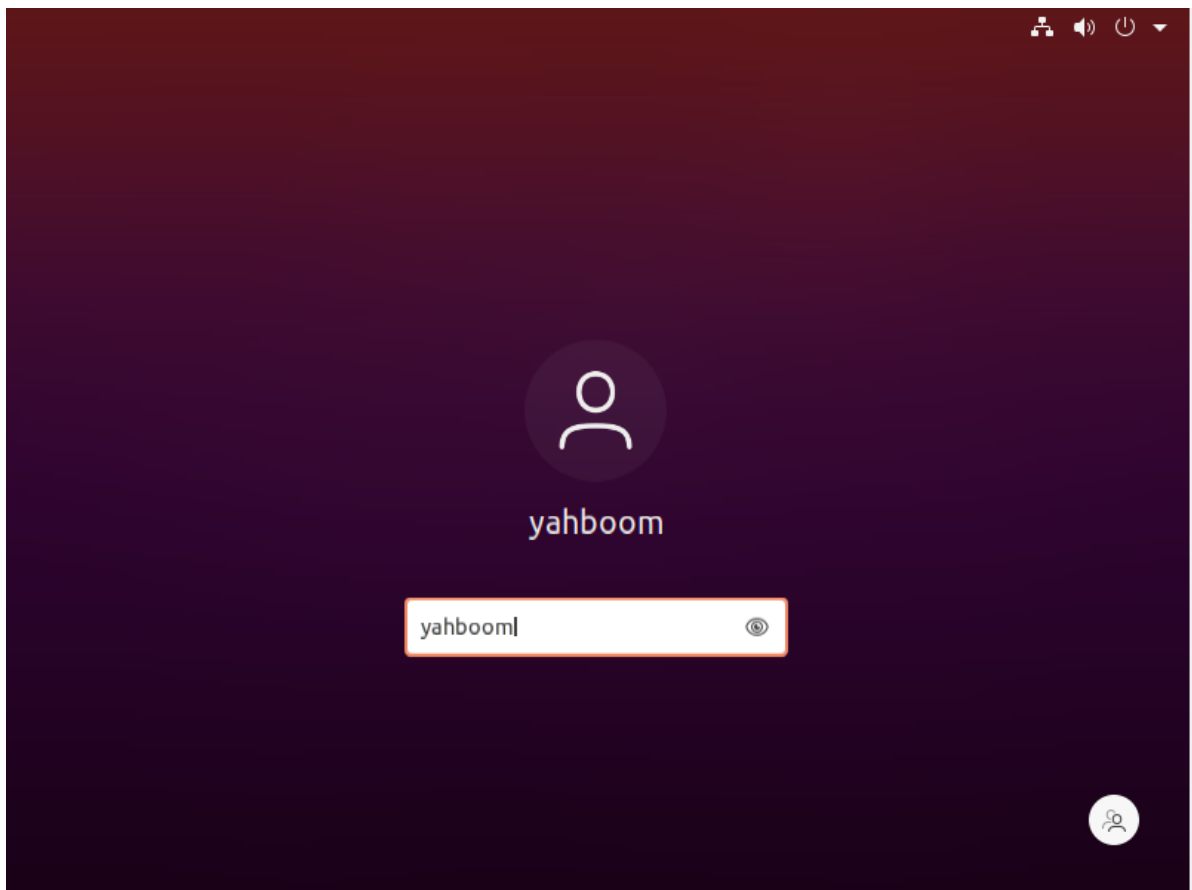
**For multi-machine communication ID modification, please refer to the tutorial: 14. Radar mapping navigation\6. Obtaining the status of the mechanical dog in the ROS2 environment\Acquiring the real joint data of the mechanical dog in the ROS2 environment.pdf**

The mechanical dog chassis and radar have been set to start automatically at boot. If you find that they have not started, please enter in the terminal.

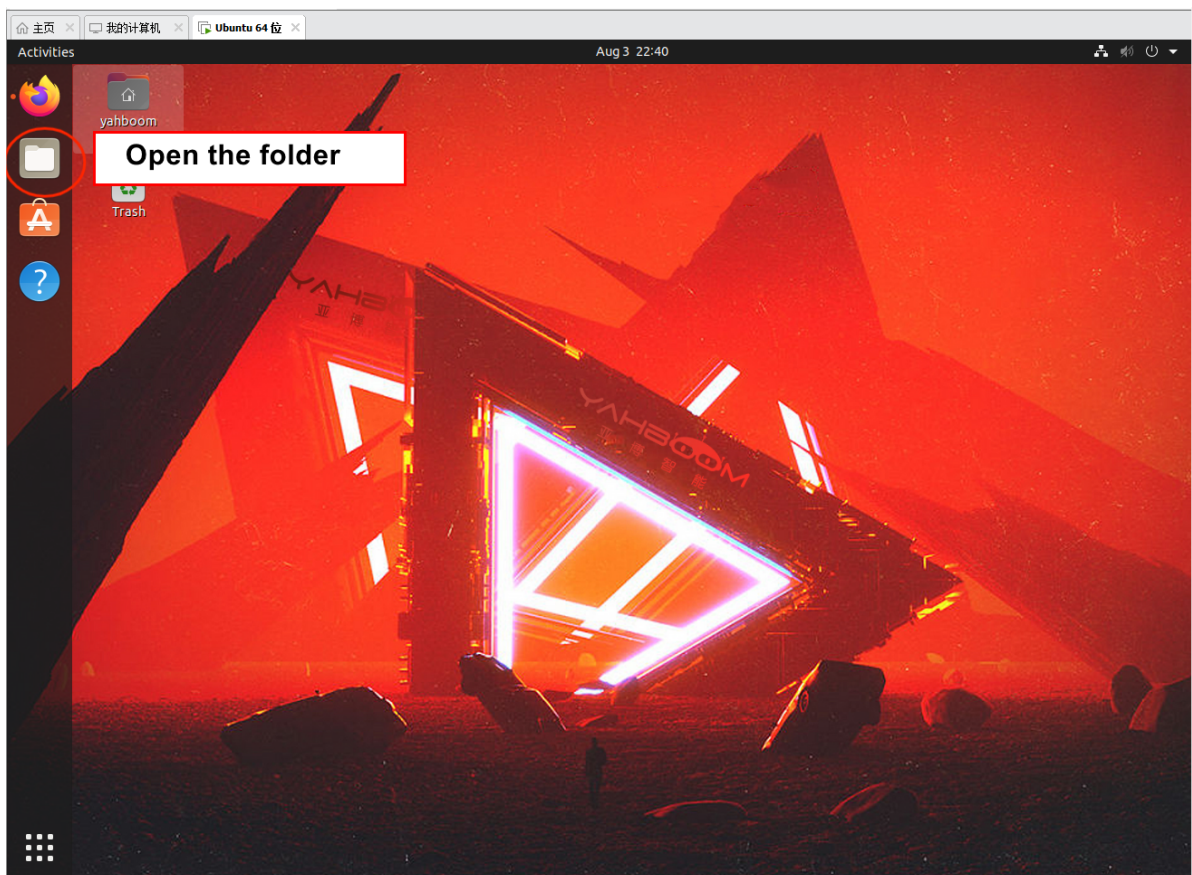
```
sudo systemctl restart YahboomStart.service
```

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

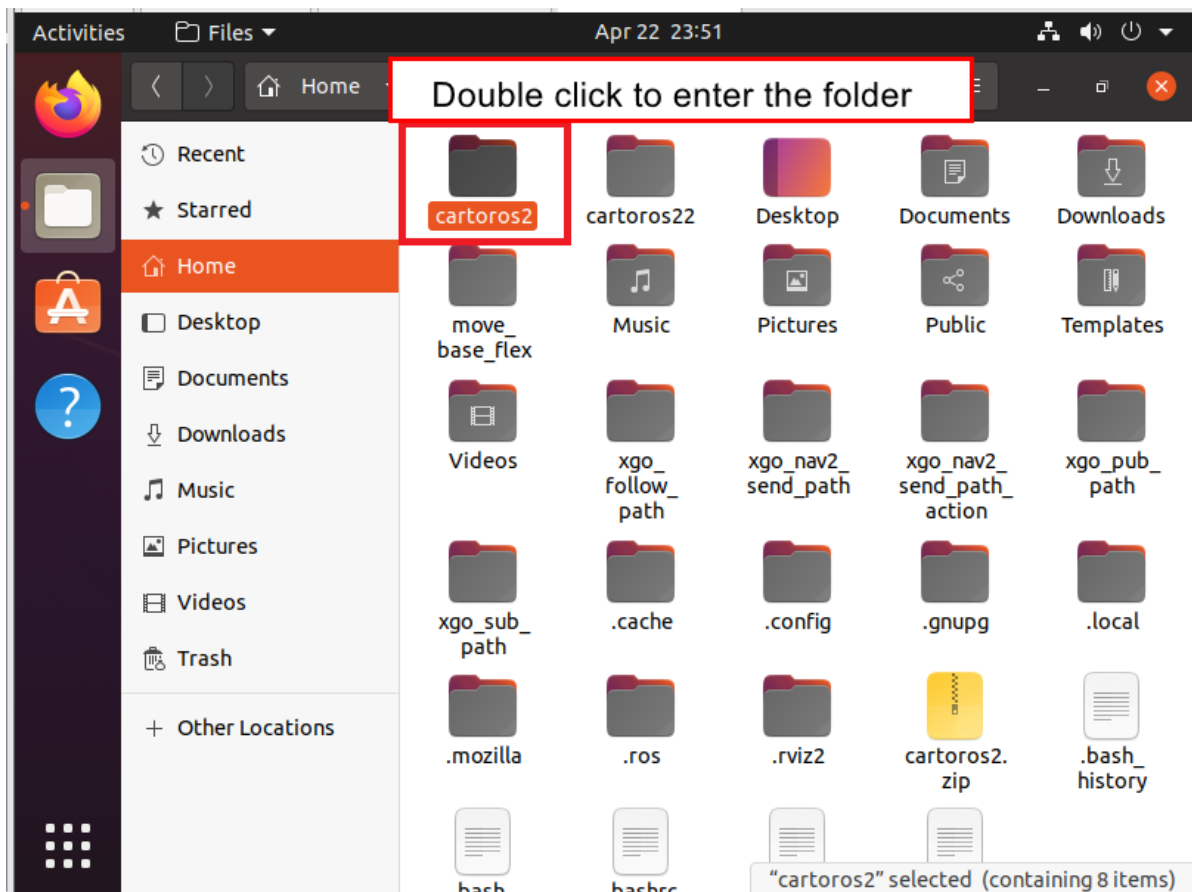
Open the virtual machine, enter the password: yahboom and press the Enter key to enter the system desktop.



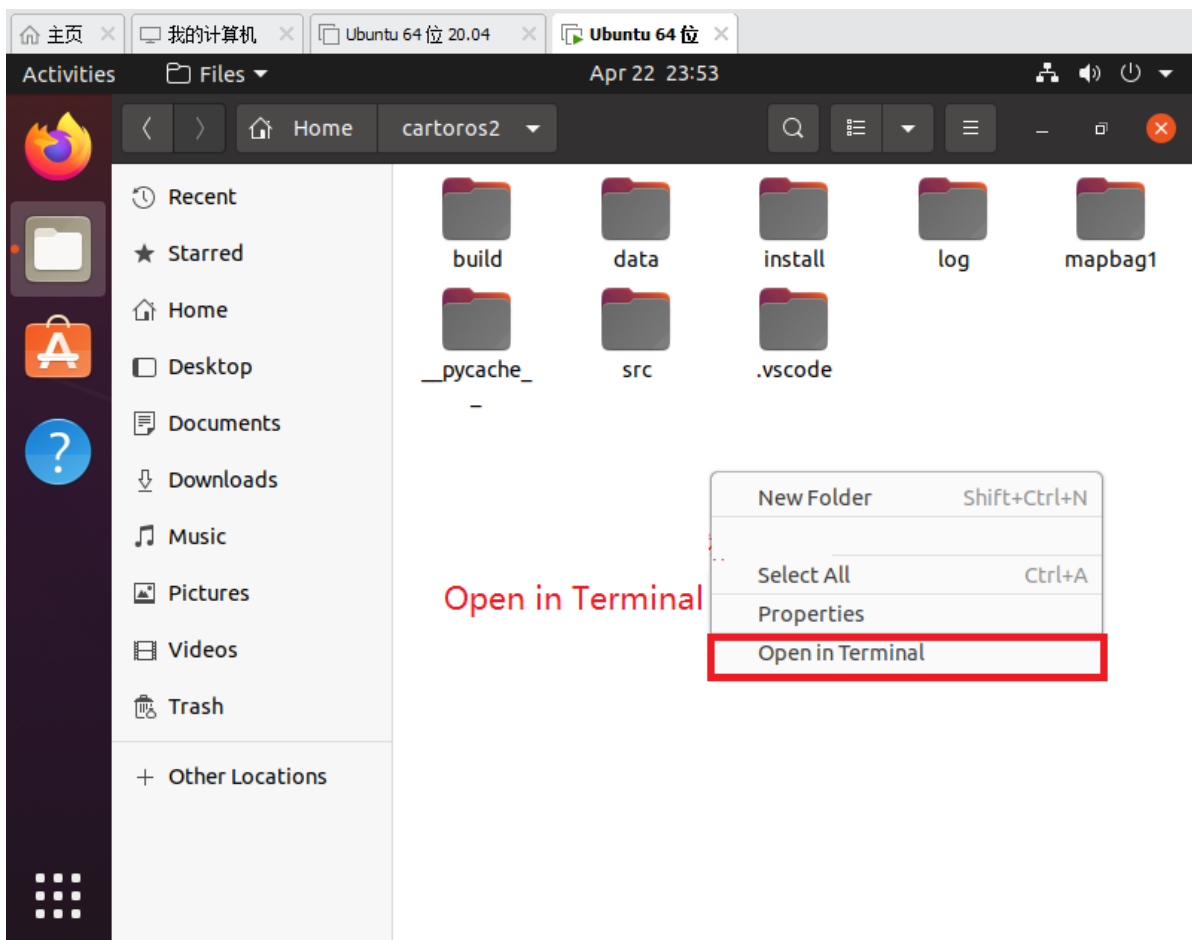
Enter the desktop system and open the folder.



Then double click on the cartoros2 folder



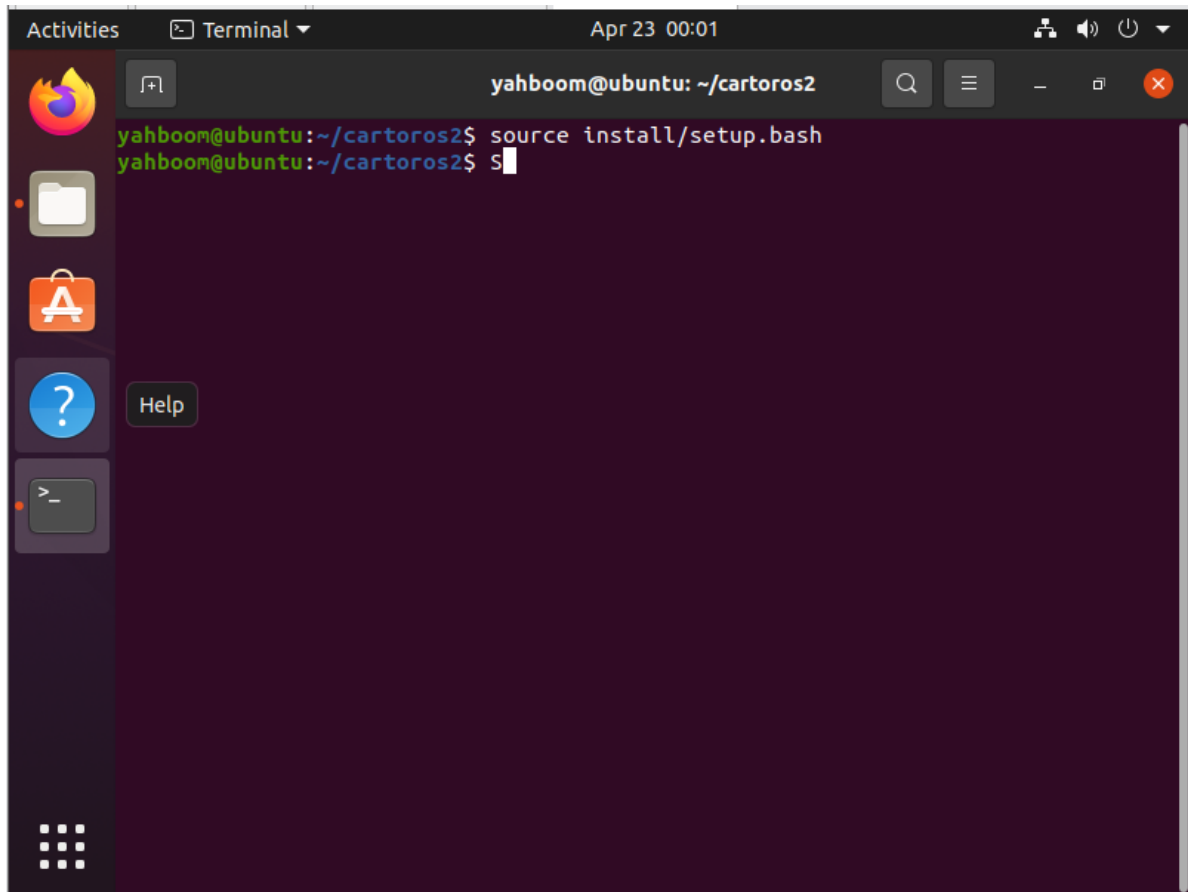
Then right-click in an empty space of the folder and select Open in Terminal



Then enter the following command in the terminal to activate the environment

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

After completing the input, press the Enter key.



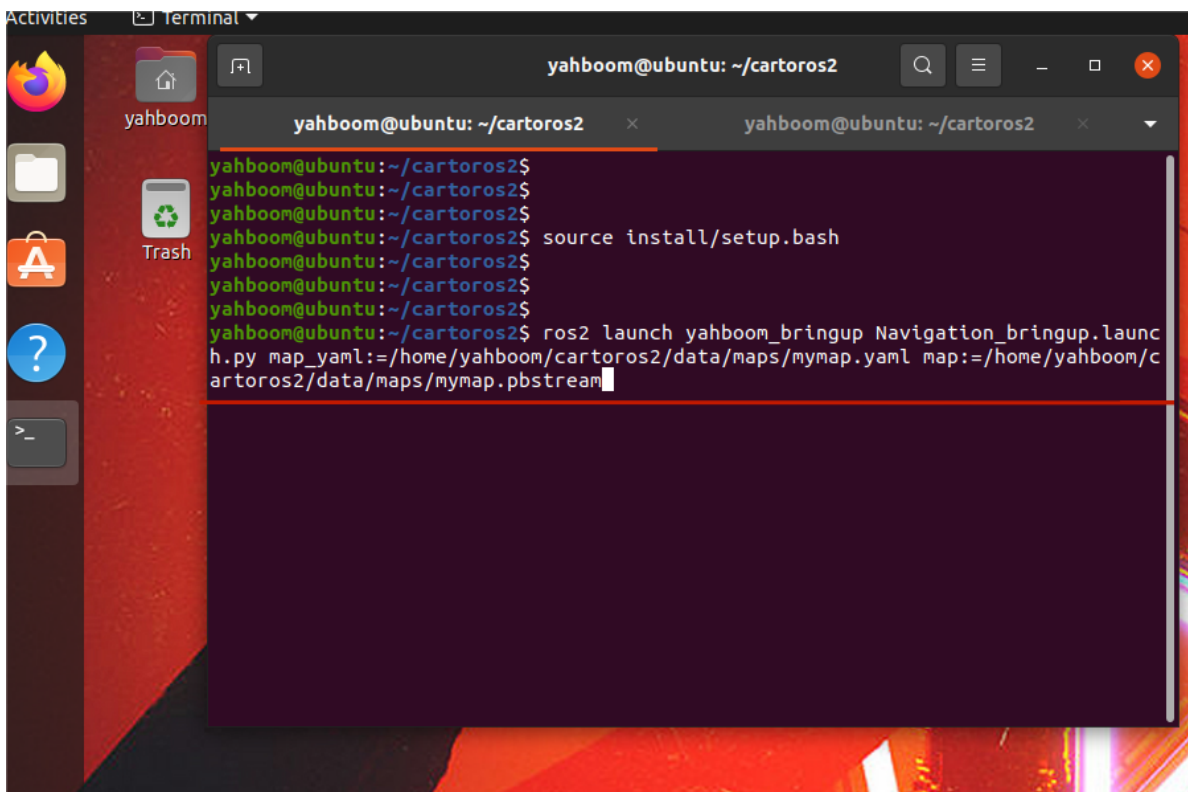
A terminal window titled 'Terminal' with the date 'Apr 23 00:01'. The prompt is 'yahboom@ubuntu: ~/cartoros2'. The user has entered the command 'source install/setup.bash' and the prompt has changed to 'yahboom@ubuntu: ~/cartoros2\$'. The terminal window is part of a desktop environment with a sidebar on the left containing icons for Firefox, Files, Applications, Help, and a terminal icon. The top bar shows 'Activities' and 'Terminal'.

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

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
```

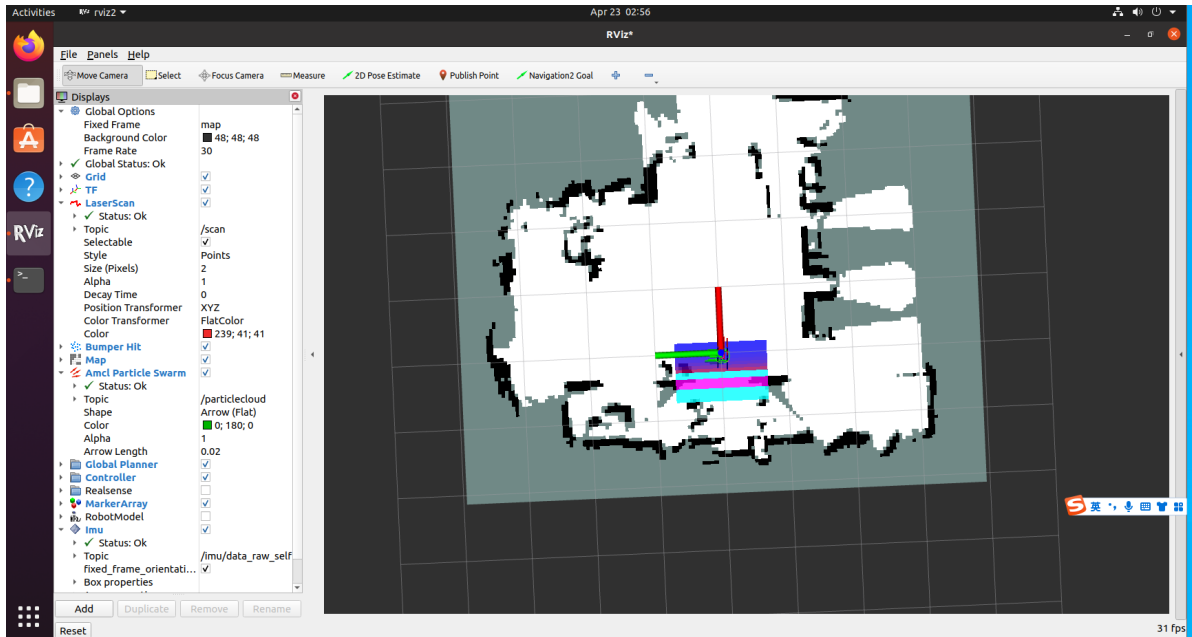
Where map\_yaml is the yaml file of the map, and map is the pbstream file of the map.



A terminal window titled 'Terminal' with the date 'Apr 23 00:01'. The prompt is 'yahboom@ubuntu: ~/cartoros2'. The user has entered 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' and the prompt has changed to 'yahboom@ubuntu: ~/cartoros2\$'. The terminal window is part of a desktop environment with a sidebar on the left containing icons for Firefox, Files, Applications, Help, and a terminal icon. The top bar shows 'Activities' and 'Terminal'.

```
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$ 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
yahboom@ubuntu:~/cartoros2$
```

Then press the Enter key to navigate.



If the radar and chassis are started, you need to enter in the terminal of the mechanical dog:

```
cd /home/pi/cartographer_ws2
source install/setup.bash
#Start patrol program Radar MS200
ros2 run yahboom_laser laser_Patrol_RS200
```

**PI5 version driver:**

**For multi-machine communication ID modification, please refer to the tutorial: 14. Radar mapping navigation\6. Obtaining the status of the mechanical dog in the ROS2 environment\Acquiring the real joint data of the mechanical dog in the ROS2 environment.pdf**

Open a terminal in the root directory of the Raspberry Pi and enter the following command to enter docker:

```
./run_humble.sh
```

```
pi@raspberrypi:~$ ./run_humble.sh
access control disabled, clients can connect from any host
root@raspberrypi:/# cd
root@raspberrypi:~# ls
core DOGZILLALib py_install_V0.0.1 yahboomcar_ws YDLidar-SDK
root@raspberrypi:~#
```

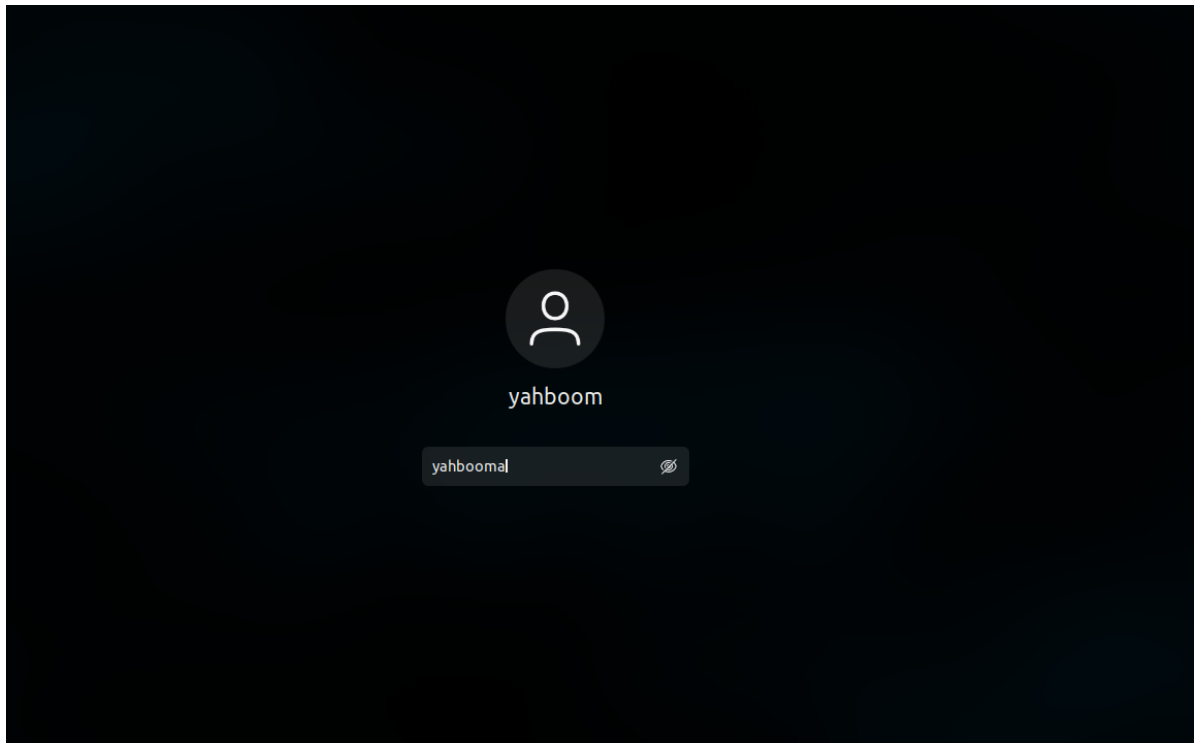
Then enter the following command in the docker terminal to start the radar data:

```
ros2 launch bringup Navigation_bringup.launch.py
```

Then enter the command to run the patrol program, \*\* (PI5 version needs to enter the same docker to run, please refer to "Introduction and Use of Radar" for steps)\*\*

```
#Start radar obstacle avoidance program Radar MS200
ros2 run yahboom_laser laser_Patrol_RS200
```

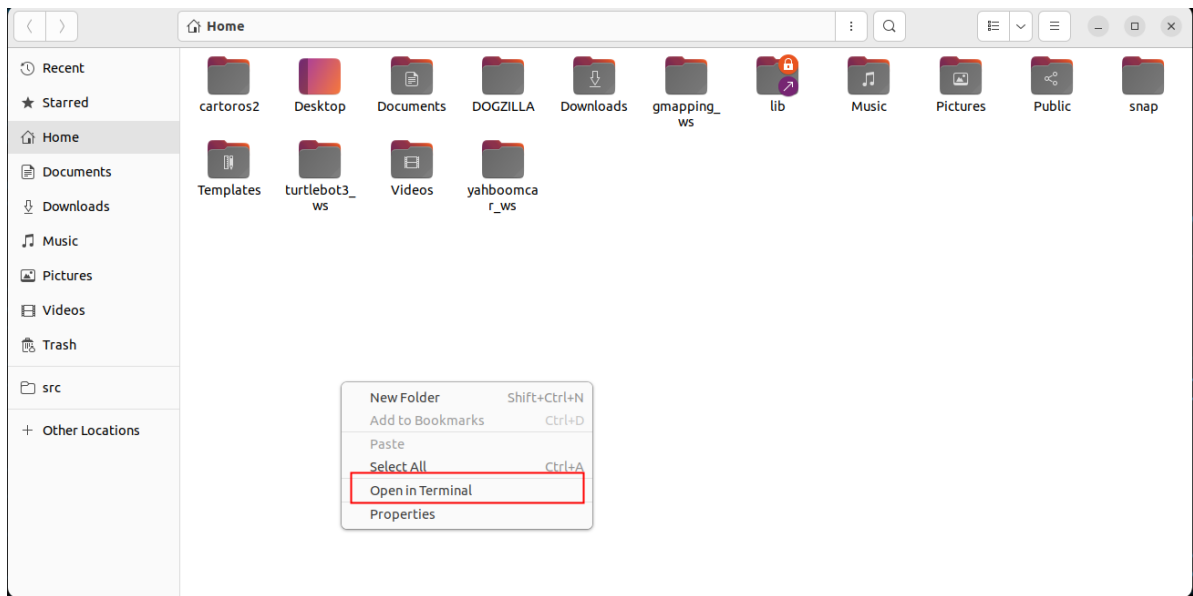
You need to open the virtual machine, enter the password: yahboom and then press the Enter key to enter the system desktop.



Enter the desktop system and open the folder.



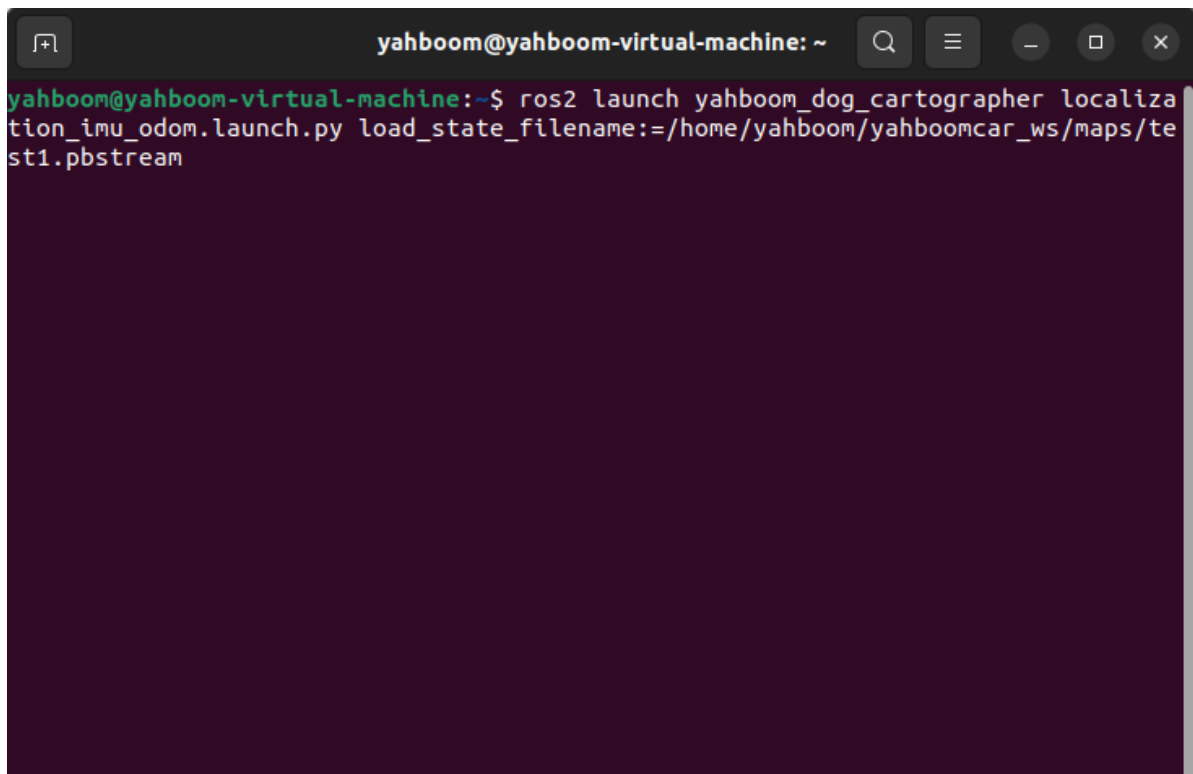
Then right-click in an empty space of the folder and select Open in Terminal



Then enter the command,

```
ros2 launch yahboom_dog_cartographer localization_imu_odom.launch.py  
load_state_filename:=/home/yahboom/yahboomcar_ws/maps/test1.pbstream
```

The name is the map name is test1, and the map data is test1.pbstream\*\* (note that the map needs to be built in advance, because patrols need to use odom, and the map needs to be built before it can be provided)\*\*



Then press the Enter key to start the patrol function.



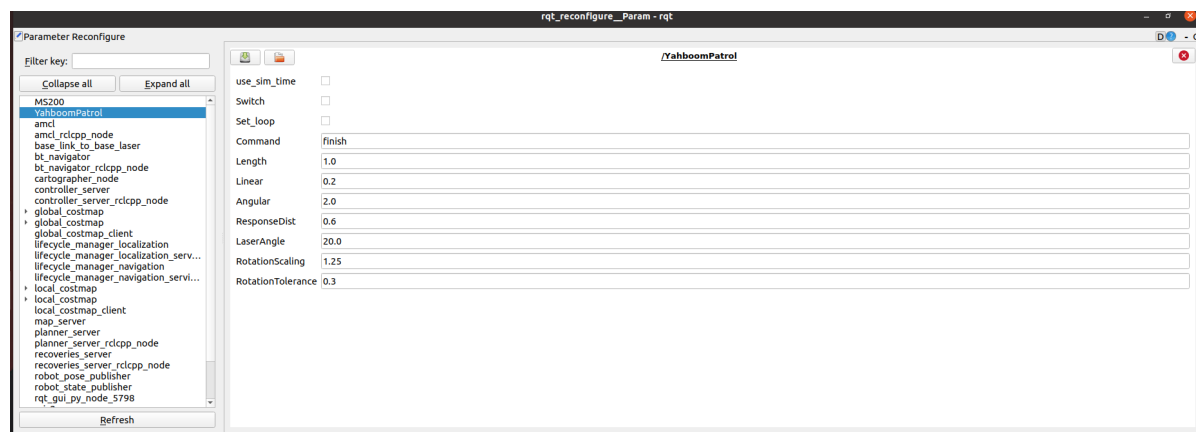
```
yahboom@yahboom-virtual-machine: ~  
yahboom@yahboom-virtual-machine:~$ ros2 launch yahboom_dog_cartographer localization_imu_odom.launch.py load_state_filename:=/home/yahboom/yahboomcar_ws/maps/test1.pbstream  
[INFO] [launch]: All log files can be found below /home/yahboom/.ros/log/2024-04-28-15-51-48-410819-yahboom-virtual-machine-5272  
[INFO] [launch]: Default logging verbosity is set to INFO  
[INFO] [cartographer_node-1]: process started with pid [5273]  
[cartographer_node-1] [INFO] [1714290708.635245656] [cartographer logger]: I0428 15:51:48.000000 5273 configuration_file_resolver.cc:41] Found '/home/yahboom/yahboomcar_ws/install/yahboom_dog_cartographer/share/yahboom_dog_cartographer/config/dog_2d.lua' for 'dog_2d.lua'.  
[cartographer_node-1] [INFO] [1714290708.635670480] [cartographer logger]: I0428 15:51:48.000000 5273 configuration_file_resolver.cc:41] Found '/opt/ros/humble/share/cartographer/configuration_files/map_builder.lua' for 'map_builder.lua'.  
[cartographer_node-1] [INFO] [1714290708.635701732] [cartographer logger]: I0428 15:51:48.000000 5273 configuration_file_resolver.cc:41] Found '/opt/ros/humble/share/cartographer/configuration_files/map_builder.lua' for 'map_builder.lua'.  
[cartographer_node-1] [INFO] [1714290708.635796758] [cartographer logger]: I0428 15:51:48.000000 5273 configuration_file_resolver.cc:41] Found '/opt/ros/humble/share/cartographer/configuration_files/pose_graph.lua' for 'pose_graph.lua'.  
[cartographer_node-1] [INFO] [1714290708.635833456] [cartographer logger]: I0428 15:51:48.000000 5273 configuration_file_resolver.cc:41] Found '/opt/ros/humble/share/cartographer/configuration_files/pose_graph.lua' for 'pose_graph.lua'.  
[cartographer_node-1] [INFO] [1714290708.636119230] [cartographer logger]: I0428
```

### 3.2. View the topic communication node diagram

Set the parameter size through the dynamic parameter adjuster, terminal input, \*\* (PI5 version needs to run in the same docker, please refer to "Introduction and Use of Radar" for steps)\*\*

```
ros2 run rqt_reconfigure rqt_reconfigure
```

Each parameter of the dynamic parameter regulator is explained as follows:





Parameter name	Parameter meaning
odom_frame	Odometer coordinate system
base_frame	Base coordinate system
circle_adjust	Circle angular speed adjustment coefficient
Switch	Gameplay switch
Command	Patrol Route
Set_loop	Set loop
ResponseDist	Radar obstacle avoidance response distance
LaserAngle	Radar scanning angle
Linear	Linear speed
Angular	Angular velocity
Length	Straight line test distance
RotationTolerance	Steering error tolerance
RotationScaling	Rotation scaling coefficient

After the program starts, in the GUI interface of the dynamic parameter adjuster interface, enter any of the following routes in the Comand column:

- LengthTest: straight line test
- Circle: circular route patrol
- Square: Square route patrol
- Triangle: Triangular route patrol

After selecting the route, click on the blank space to write the parameters, and then click the Switch button to start patrolling. If loop is set, you can loop the last route for patrol. If loop is false, it will stop after completing the patrol.

## 4. Core source code analysis

The implementation source code of this code is to subscribe 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 Triangle as an example, we will analyze it here.

```
#Set the patrol route and enter the self.Triangle function
self.command_src = "Triangle"
triangle = self.Triangle()
#Parse with part of self.Triangle code,
def Triangle(self):
    if self.index == 0:
        print("Length")
        step1 = self.advancing(self.Length) #Start with a straight line and
        walk through 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 the direction, 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)
#After completing the following 3 loops, the triangle patrol is completed, mainly
looking at the self.advancing and self.Spin functions. After these two functions
are executed, True will be returned.
        def advancing(self,target_distance):
            #The following is to obtain the xy coordinates, calculate them with the
            coordinates of the previous moment, and calculate how far you have gone
            #The way to obtain xy coordinates is to monitor the tf transformation of
            odom and base_footprint. For this part, please refer 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;
                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 obtain the pose and calculate how many degrees you
    have turned. To obtain the pose, you can refer to the self.get_odom_angle
    function. It 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 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#circle_adjust here is the coefficient of the rotation angle. It can
                be understood that the larger the length, the larger the radius of the circle.
                #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

```

