Lidar patrol

This lesson uses the Raspberry Pi 5 as an example.

For Raspberry Pi and Jetson Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this lesson in the terminal. For instructions on entering the Docker container from the host computer, refer to **[01. Robot Configuration and Operation Guide] -- [5.Enter Docker (For JETSON Nano and RPi 5)]**.

For Orin boards, simply open a terminal and enter the commands mentioned in this lesson.

1. Program Functionality

> After running the program, set the patrol route in the dynamic parameter adjuster and click Start. The car will move along the patrol route. Simultaneously, the car's lidar will scan for obstacles within the set lidar angle and obstacle detection distance. If an obstacle is detected, the car will stop and a buzzer will sound. If no obstacle is detected, the car will continue patrolling.

2. Program Code Reference Path

For the Raspberry Pi PI5 controller, you must first enter the Docker container. The Orin controller does not need to enter this.

The source code for this function is located at:

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_bringup/yahboomcar_bringup/patrol_A1.py
```

3. Program Startup

3.1. Startup Command

For the Raspberry Pi PI5 controller, you must first enter the Docker container. The Orin controller does not need to enter this.

Enter the Docker container (for steps, see [Docker Course] --- [4. Docker Startup Script]).

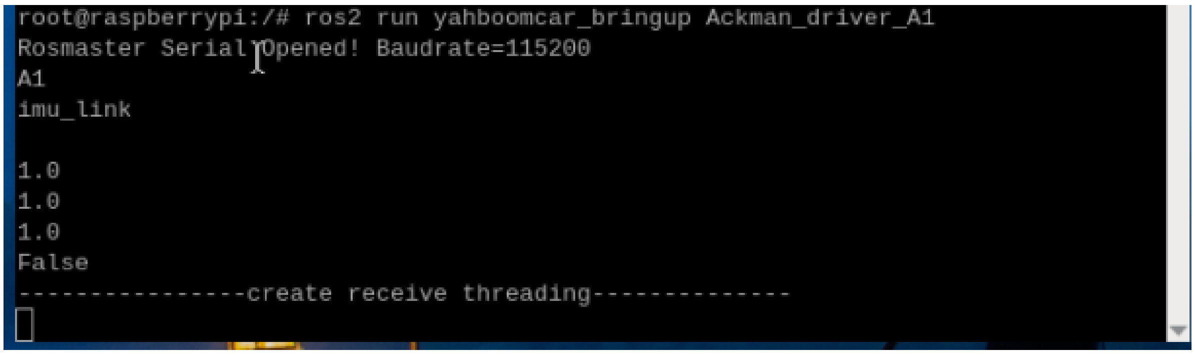
All the following Docker commands must be executed from the same Docker container.**(For steps, see [Docker Course] --- [3. Docker Submission and Multi-Terminal Access]).**

This command must be modified to the corresponding lidar model.

Enter in the terminal:

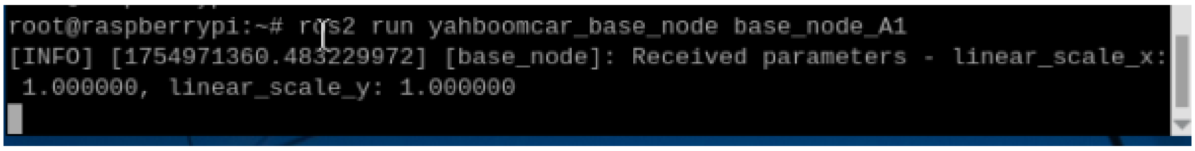
```
# Start the car chassis
ros2 run yahboomcar_bringup Ackman_driver_A1
ros2 run yahboomcar_base_node base_node_A1
# Select one of the two lidars
#timni
ros2 launch ydlidar_ros2_driver ydlidar_launch.py
#c1
ros2 launch sllidar_ros2 sllidar_c1_launch.py
# Start the lidar patrol program
ros2 run yahboomcar_bringup patrol_A1
```

In the chassis terminal,



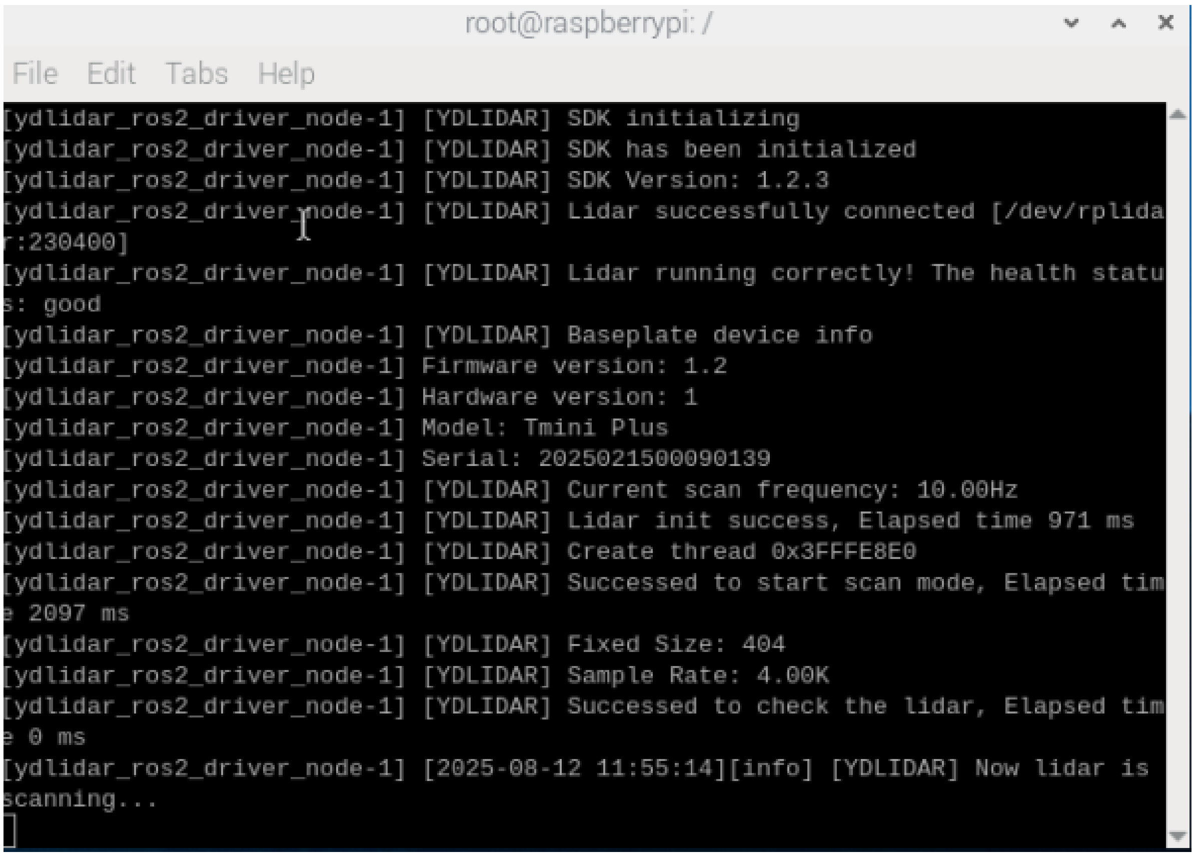
```
root@raspberrypi:/# ros2 run yahboomcar_bringup Ackman_driver_A1
Rosmaster Serial Opened! Baudrate=115200
A1
imu_link

1.0
1.0
1.0
False
----------------create receive threading--------------
```

Starting the odomtf conversion, the terminal displays:



```
root@raspberrypi:~# ros2 run yahboomcar_base_node base_node_A1
[INFO] [1754971360.483229972] [base_node]: Received parameters - linear_scale_x:
 1.000000, linear_scale_y: 1.000000
```

Starting the lidar driver terminal,

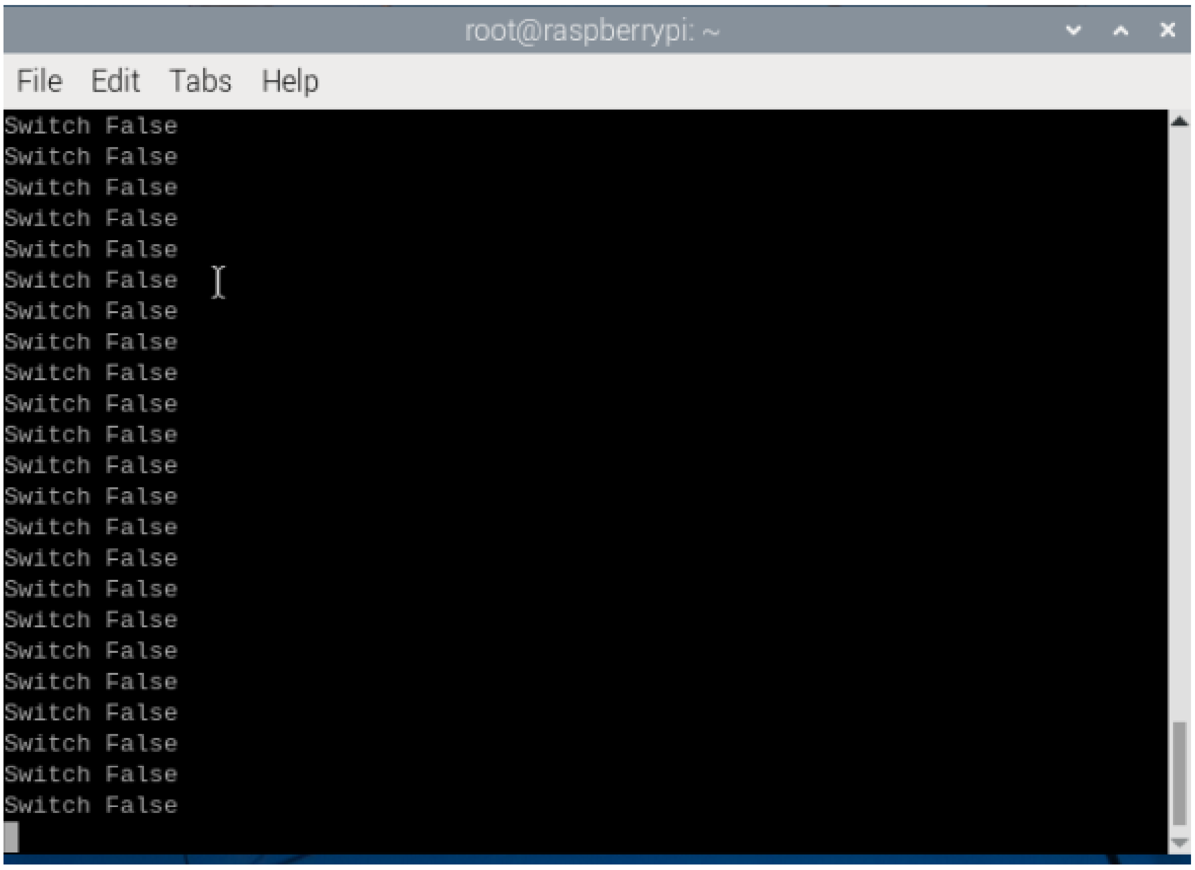


root@raspberrypi: /

File Edit Tabs Help

```
[ydlidar_ros2_driver_node-1] [YDLIDAR] SDK initializing
[ydlidar_ros2_driver_node-1] [YDLIDAR] SDK has been initialized
[ydlidar_ros2_driver_node-1] [YDLIDAR] SDK Version: 1.2.3
[ydlidar_ros2_driver_node-1] [YDLIDAR] Lidar successfully connected [/dev/rplida
r:230400]
[ydlidar_ros2_driver_node-1] [YDLIDAR] Lidar running correctly! The health statu
s: good
[ydlidar_ros2_driver_node-1] [YDLIDAR] Baseplate device info
[ydlidar_ros2_driver_node-1] Firmware version: 1.2
[ydlidar_ros2_driver_node-1] Hardware version: 1
[ydlidar_ros2_driver_node-1] Model: Tmini Plus
[ydlidar_ros2_driver_node-1] Serial: 2025021500090139
[ydlidar_ros2_driver_node-1] [YDLIDAR] Current scan frequency: 10.00Hz
[ydlidar_ros2_driver_node-1] [YDLIDAR] Lidar init success, Elapsed time 971 ms
[ydlidar_ros2_driver_node-1] [YDLIDAR] Create thread 0x3FFFE8E0
[ydlidar_ros2_driver_node-1] [YDLIDAR] Successed to start scan mode, Elapsed tim
e 2097 ms
[ydlidar_ros2_driver_node-1] [YDLIDAR] Fixed Size: 404
[ydlidar_ros2_driver_node-1] [YDLIDAR] Sample Rate: 4.00K
[ydlidar_ros2_driver_node-1] [YDLIDAR] Successed to check the lidar, Elapsed tim
e 0 ms
[ydlidar_ros2_driver_node-1] [2025-08-12 11:55:14][info] [YDLIDAR] Now lidar is
scanning...
```

After the program starts, it will continuously print "The switch is now closed, waiting for the parameter adjuster to be used to specify the patrol route."



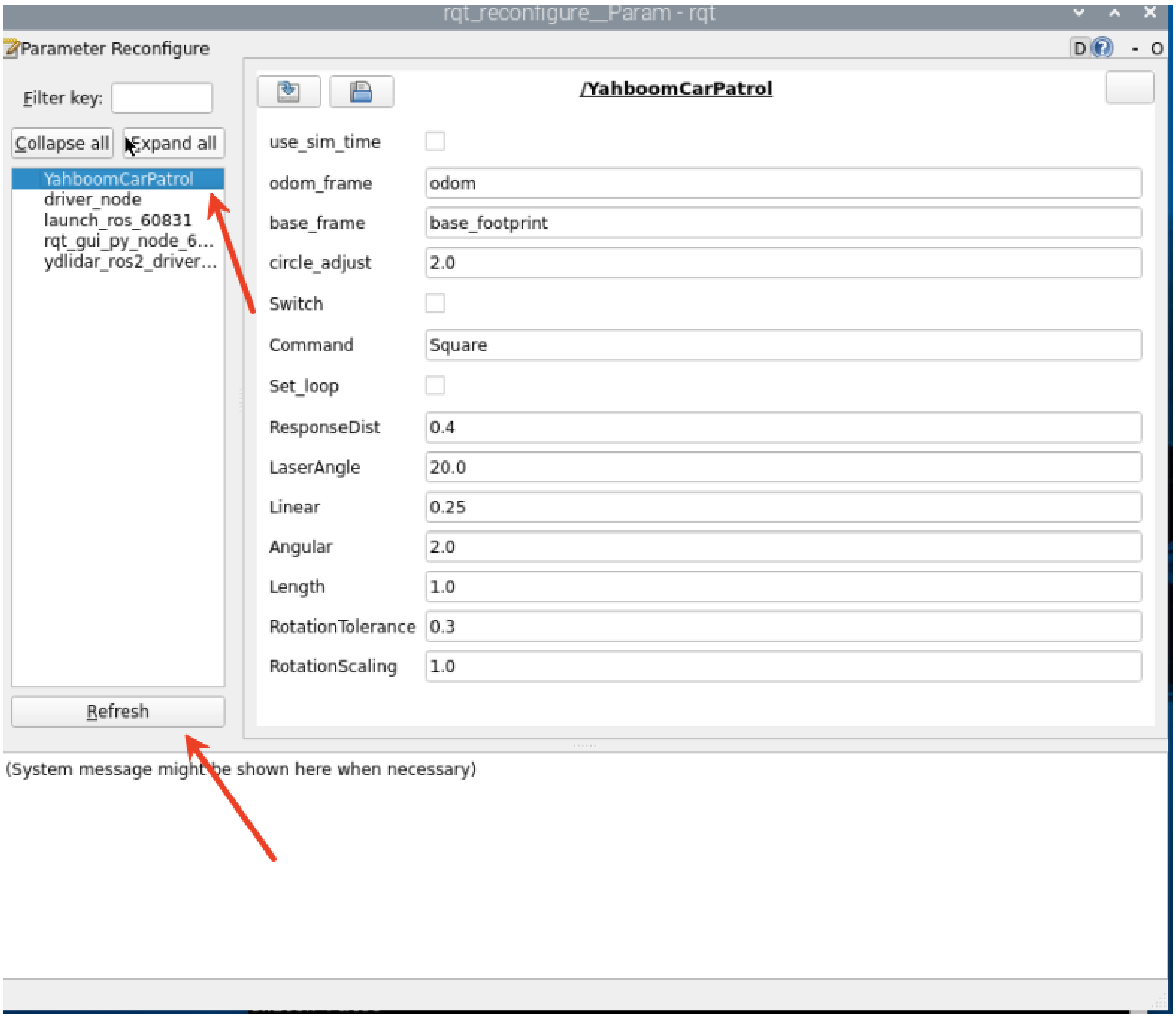
```
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
Switch False
```

3.2 Dynamic Parameter Adjuster

You can also use the dynamic parameter adjuster to set parameter values. In the terminal, enter:

```
ros2 run rqt_reconfigure rqt_reconfigure
```

Note: The above nodes may not appear when you first open the system. Click Refresh to see all nodes. The YahboomCarPatrol node displayed is the patrol node.


rqt_reconfigure__Param - rqt

Parameter Reconfigure

Filter key:

Collapse all | Expand all

YahboomCarPatrol
driver_node
launch_ros_60831
rqt_gui_py_node_6...
ydlidar_ros2_driver...

Refresh

/YahboomCarPatrol

use_sim_time	☐
odom_frame	odom
base_frame	base_footprint
circle_adjust	2.0
Switch	☐
Command	Square
Set_loop	☐
ResponseDist	0.4
LaserAngle	20.0
Linear	0.25
Angular	2.0
Length	1.0
RotationTolerance	0.3
RotationScaling	1.0

(System message might be shown here when necessary)

☑ After modifying the parameters, click a blank area in the GUI to enter the parameter value. Note that this will only take effect for the current startup. To permanently change the value, you need to modify the parameter in the source code.

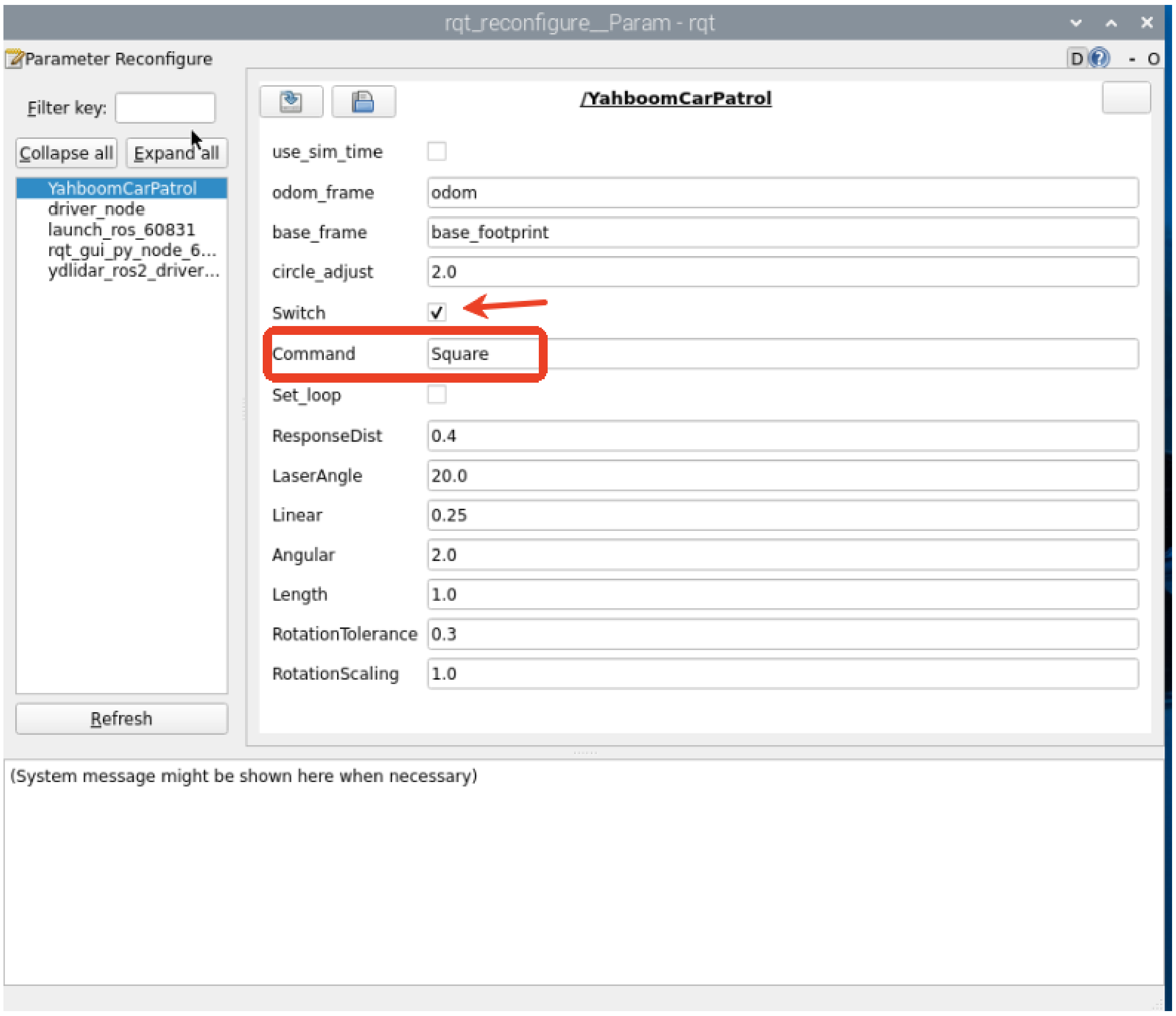
Parameter Explanation:

- odom_frame: Odometry coordinate system name
- base_frame: Base coordinate system name
- circle_adjust: When the patrol route is circular, this value can be used as a coefficient to adjust the circle's size. See the code for details.
- Switch: Gameplay switch
- Command: Patrol route. There are the following routes: [LengthTest] - Linear patrol, [Circle] - Circle patrol, [Square] - Square patrol.
- Set_loop: Restart patrol. Once set, the patrol will continue in a loop along the specified route.
- ResponseDist: Obstacle detection distance
- LaserAngle: lidar detection angle
- Linear: Linear velocity
- Angular: Angular velocity
- Length: Linear distance
- RotationTolerance: Rotation error tolerance
- RotationScaling: Rotation scaling factor

After modifying the above parameters, click a blank space to transfer the parameters to the program.

3.3. Starting a Patrol

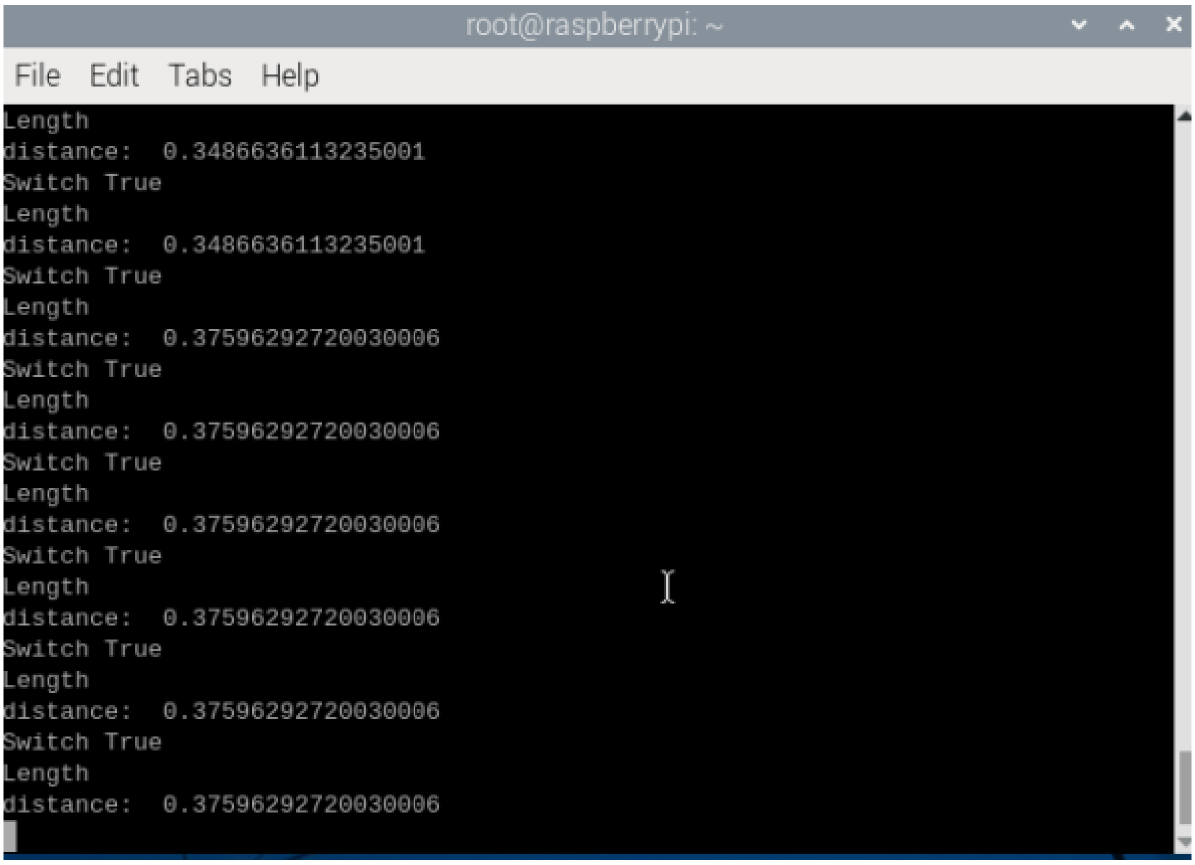
In the rqt interface, find the YahboomCarPatrol node. The [Command] node is the patrol route you set. Here, we'll use a square patrol route as an example. The different patrol routes are explained below. After setting the route in [Command], click Switch to start the patrol.



use_sim_time	☐
odom_frame	odom
base_frame	base_footprint
circle_adjust	2.0
Switch	✓
Command	Square
Set_loop	☐
ResponseDist	0.4
LaserAngle	20.0
Linear	0.25
Angular	2.0
Length	1.0
RotationTolerance	0.3
RotationScaling	1.0

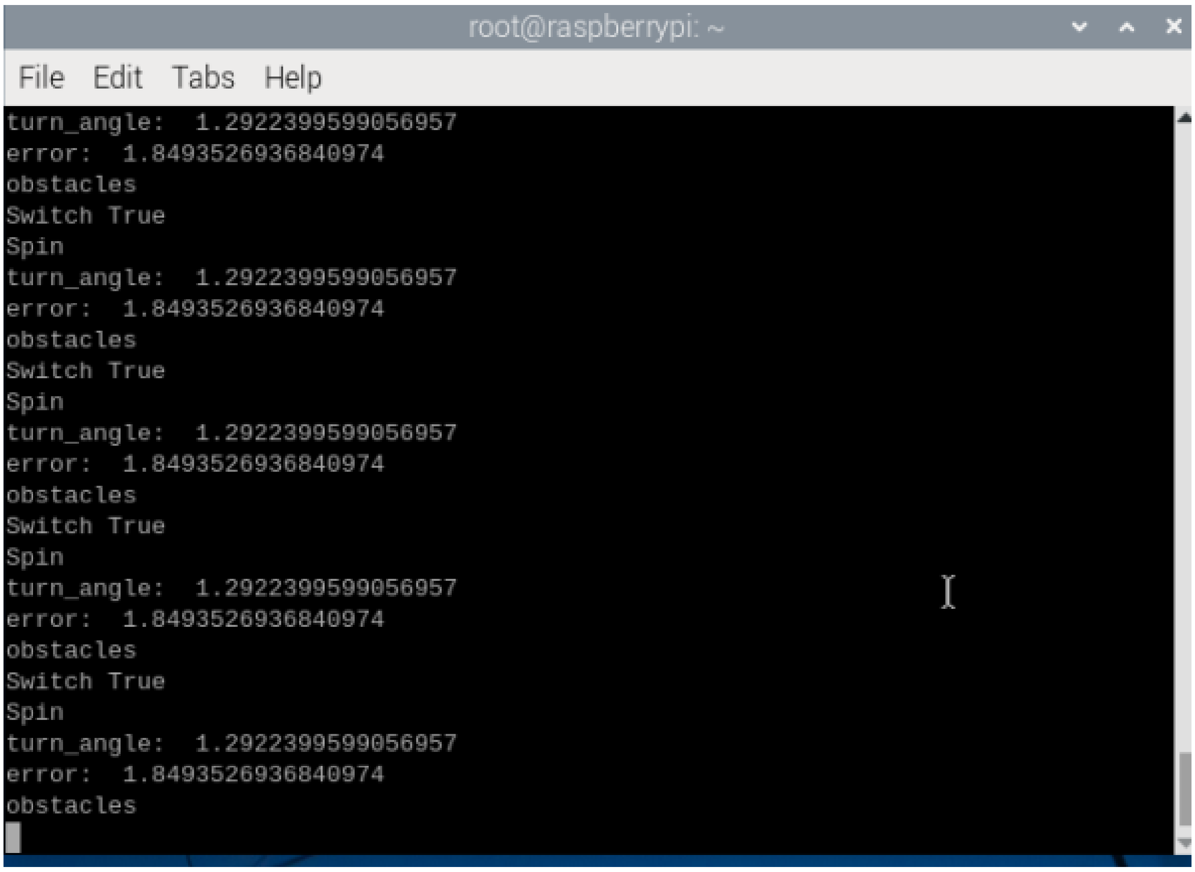
(System message might be shown here when necessary)

Here, we've chosen [Square] as an example. You can customize the patrol state. First, go straight, then turn 90 degrees, then go straight again, then turn 90 degrees again, and so on, until the patrol route is a square.



```
Length
distance:  0.3486636113235001
Switch True
Length
distance:  0.3486636113235001
Switch True
Length
distance:  0.37596292720030006
Switch True
Length
distance:  0.37596292720030006
Switch True
Length
distance:  0.37596292720030006
Switch True
Length
distance:  0.37596292720030006
Switch True
Length
distance:  0.37596292720030006
Switch True
Length
distance:  0.37596292720030006
```

If the vehicle encounters an obstacle while driving, it will stop, a buzzer will sound, and the terminal will print information.



```
turn_angle:  1.2922399599056957
error:  1.8493526936840974
obstacles
Switch True
Spin
turn_angle:  1.2922399599056957
error:  1.8493526936840974
obstacles
Switch True
Spin
turn_angle:  1.2922399599056957
error:  1.8493526936840974
obstacles
Switch True
Spin
turn_angle:  1.2922399599056957
error:  1.8493526936840974
obstacles
Switch True
Spin
turn_angle:  1.2922399599056957
error:  1.8493526936840974
obstacles
```

4. Core Code

4.1. patrol_A1.py

This program has the following main functions:

- Subscribes to the lidar topic and obtains surrounding lidar data;
- Obtains and pre-processes the nearest lidar data;
- Publishes the vehicle's forward speed, distance, and turning angle based on the following strategy.

Some of the core code is as follows:

Creates subscribers for lidar and remote control data.

```
self.sub_scan =
self.create_subscription(LaserScan,"/scan",self.LaserScanCallback,1)
self.sub_joy =
self.create_subscription(Bool,"/JoyState",self.JoyStateCallback,1)
```

Create speed and buzzer data publisher

```
self.pub_cmdVel = self.create_publisher(Twist,"cmd_vel",5)
self.pub_Buzzer = self.create_publisher(UInt16,'/beep',1)
```

Monitor the TF transformation of odom and base_footprint, calculate the current XY coordinates and rotation angle,

```
def get_position(self):
    try:
        now = rclpy.time.Time()
        trans =
self.tf_buffer.lookup_transform(self.odom_frame,self.base_frame,now)
        return trans
    except (LookupException, ConnectivityException, ExtrapolationException):
        self.get_logger().info('transform not ready')
        raise
        return
self.position.x = self.get_position().transform.translation.x
self.position.y = self.get_position().transform.translation.y

def get_odom_angle(self):
    try:
        now = rclpy.time.Time()
        rot =
self.tf_buffer.lookup_transform(self.odom_frame,self.base_frame,now)
        #print("oring_rot: ",rot.transform.rotation)
        cacl_rot = PyKDL.Rotation.Quaternion(rot.transform.rotation.x,
rot.transform.rotation.y, rot.transform.rotation.z, rot.transform.rotation.w)
        #print("cacl_rot: ",cacl_rot)
        angle_rot = cacl_rot.GetRPY()[2]
        return angle_rot
    except (LookupException, ConnectivityException, ExtrapolationException):
        self.get_logger().info('transform not ready')
        raise
        return
self.odom_angle = self.get_odom_angle()
```

There are two important functions, linear advancing and rotation Spin. All patrol routes are just a combination of linear and rotation actions.

```
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;
        self.Switch  =
rclpy.parameter.Parameter('Switch',rclpy.Parameter.Type.BOOL,False)
        all_new_parameters = [self.Switch]
        self.set_parameters(all_new_parameters)
        return True
.....
```

spin

```
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 or self.Switch==False :
        self.pub_cmdVel.publish(Twist())
        self.turn_angle = 0.0
        return True
    if self.Joy_active or self.warning > 10:
        if self.moving == True:
            self.pub_cmdVel.publish(Twist())
            self.moving = False
            b = UInt16()
            b.data = 1
            self.pub_Buzzer.publish(b)
            print("obstacles")
.....
```

Now that we have the functions of walking in a straight line and rotating, we can arrange and combine them according to the patrol route. Taking a square as an example,

```
def Square(self):
    if self.index == 0:
    print("Length")
    #首先直线行走1米 First walk 1 meter in a straight line
    step1 = self.advancing(self.Length)
    #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")
        #旋转90度 Rotate 90 degrees
        step2 = self.Spin(90)
        #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)
.....
```