5、Patrol function gameplay

1、Program function description

After the program starts, open the patrol route set by the dynamic parameter setter, click the "switch" of the GUI interface, the trolley 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 the controller program is turned on, you can also pause/resume the trolley movement through the R2 button.

2、Program code reference path

Raspberry Pi PI5 master needs to enter the docker container first, Orin master does not need to enter,

the location of the source code of this function is located at,

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_bringup/yahboomcar_bringup/patrol_a1_X3.py
```

3、The program starts

3.1、start the command

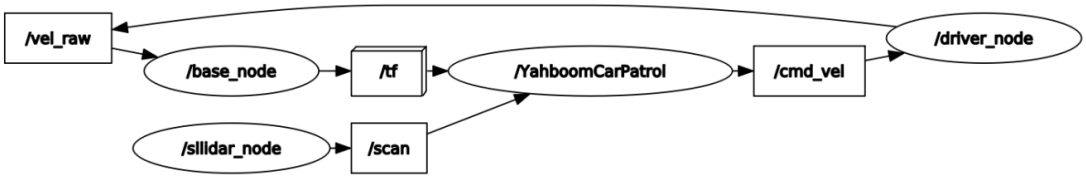
according to the actual model and radar model, the terminal input,

```
#Start the trolley chassis
ros2 launch yahboomcar_bringup yahboomcar_bringup_X3_launch.py
#Start the A1 radar
ros2 launch sllidar_ros2 sllidar_launch.py
#Start the S2 radar
ros2 launch sllidar_ros2 sllidar_s2_launch.py
#Start the patrol program X3 model
ros2 run yahboomcar_bringup patrol_a1_X3
#Start the handle, if needed
ros2 run yahboomcar_ctrl yahboom_joy_X3
ros2 run joy joy_node
```

3.2、View the topic communication node diagram

docker terminal input,

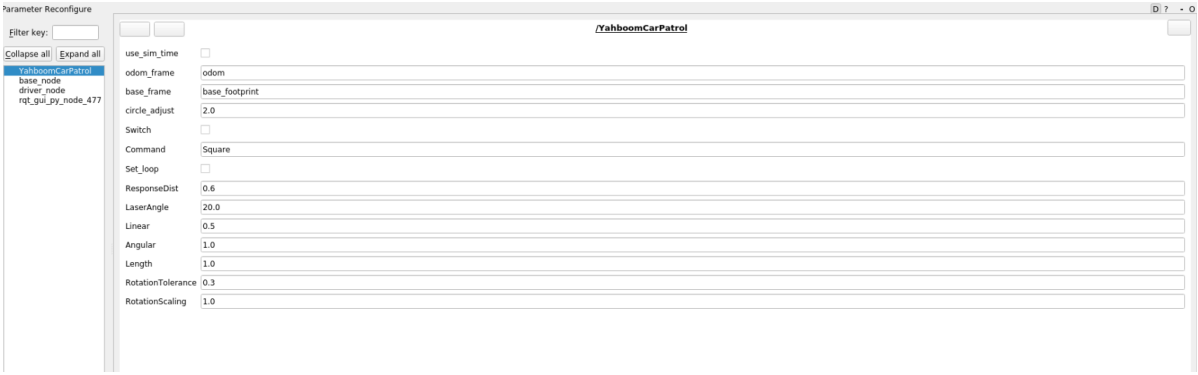
```
ros2 run rqt_graph rqt_graph
```



Set the size of parameters via dynamic parameter adjuster, terminal input,

```
ros2 run rqt_reconfigure rqt_reconfigure
```

The parameters of the dynamic parameter regulator are described as follows:



Parameter name	Parameter meaning
odom_frame	Odometer coordinate system
base_frame	Coordinate system
circle_adjust	Rotational angular velocity adjustment factor
Switch	Gameplay switch
Command	Patrol routes
Set_loop	Set up a loop
ResponseDist	Radar obstacle avoidance response range
LaserAngle	Radar scan angle
Linear	Line speed
Angular	angular velocity
Length	Straight-line test distance
RotationTolerance	Steering error tolerance
RotationScaling	Corner scale factor

After the program starts, enter any of the following routes in the Comand field in the GUI interface of the Dynamic Parameter Adjuster interface:

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

After selecting the route, click the blank parameter to write, and then click the Switch button to start the patrol movement. If the loop is set, you can loop through the previous route to patrol, and if the loop is false, it will stop when the patrol is completed.

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, take Triangle as an example, here to do the analysis,

```
#Set the patrol route and enter the self. Triangle function
self.command_src = "Triangle"
triangle = self.Triangle()
#Take part of self. Triangle code parsing
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)#随后调换方向，转向120，三角形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 goes through 3 loops, that is, to complete the triangle patrol,
mainly to see the self.advancing and self.Spin functions, these two functions
will return True after the execution is completed,
 def advancing(self,target_distance):
    #Here's how to get the xy coordinates, calculate the coordinates with the
previous moment, and calculate how far you have come
    #The way to get XY coordinates is to listen to the tf transformation of Odom
and base_footprint, which can 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, calculate how many degrees you
turned, and get the pose can refer to the self.get_odom_angle function, which is
also obtained by listening to 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
                #The circle_adjust here is the coefficient of the angle of
rotation, and the larger the length can be understood, the greater 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
```