Lidarpatrol

Note: The virtual machine needs to be in the same LAN as the car, and the ROS_DOMAIN_ID needs to be consistent. You can check [Must read before use] to set the IP and ROS_DOMAIN_ID on the board.

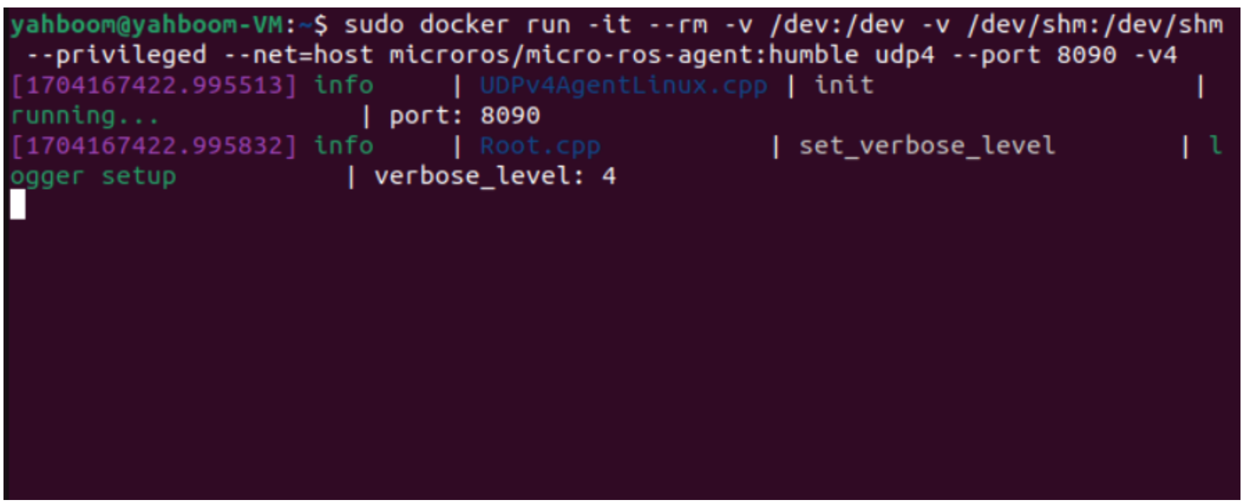
1. Program function description

The car connects to the agent, runs the program, sets the route to be patrolled in the dynamic parameter adjuster, and then clicks Start. The car will move according to the set patrol route. At the same time, the radar on the car will scan the set radar angle and whether there are obstacles within the set obstacle detection distance. If there are obstacles, it will stop and the buzzer will sound. If there are no obstacles, the car will stop and the buzzer will sound. If there are no obstacles, the car will stop and the buzzer will sound. Barriers continue to patrol.

2. Start and connect to the agent

Taking the supporting virtual machine as an example, enter the following command to start the agent:

```
sudo docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privileged --net=host
microros/micro-ros-agent:humble udp4 --port 8090 -v4
```



Then, turn on the car switch and wait for the car to connect to the agent. The connection is successful, as shown in the figure below.

```
[1702630014.015846] info     | ProxyClient.cpp    | create_participant       | participant created     | client_key: 0x0B62A009, part
icipant_id: 0x000(1)
[1702630014.135363] info     | ProxyClient.cpp    | create_topic             | topic created           | client_key: 0x0B62A009, topi
c_id: 0x000(2), participant_id: 0x000(1)
[1702630014.223689] info     | ProxyClient.cpp    | create_publisher         | publisher created       | client_key: 0x0B62A009, publ
isher_id: 0x000(3), participant_id: 0x000(1)
[1702630014.415510] info     | ProxyClient.cpp    | create_datawriter        | datawriter created      | client_key: 0x0B62A009, data
writer_id: 0x000(5), publisher_id: 0x000(3)
[1702630014.428530] info     | ProxyClient.cpp    | create_topic             | topic created           | client_key: 0x0B62A009, topi
c_id: 0x001(2), participant_id: 0x000(1)
[1702630014.527190] info     | ProxyClient.cpp    | create_publisher         | publisher created       | client_key: 0x0B62A009, publ
isher_id: 0x001(3), participant_id: 0x000(1)
[1702630014.543889] info     | ProxyClient.cpp    | create_datawriter        | datawriter created      | client_key: 0x0B62A009, data
writer_id: 0x001(5), publisher_id: 0x001(3)
[1702630014.554490] info     | ProxyClient.cpp    | create_topic             | topic created           | client_key: 0x0B62A009, topi
c_id: 0x002(2), participant_id: 0x000(1)
[1702630014.737059] info     | ProxyClient.cpp    | create_publisher         | publisher created       | client_key: 0x0B62A009, publ
isher_id: 0x002(3), participant_id: 0x000(1)
[1702630014.755072] info     | ProxyClient.cpp    | create_datawriter        | datawriter created      | client_key: 0x0B62A009, data
writer_id: 0x002(5), publisher_id: 0x002(3)
[1702630014.818985] info     | ProxyClient.cpp    | create_topic             | topic created           | client_key: 0x0B62A009, topi
c_id: 0x003(2), participant_id: 0x000(1)
[1702630014.840001] info     | ProxyClient.cpp    | create_subscriber        | subscriber created      | client_key: 0x0B62A009, subs
criber_id: 0x000(4), participant_id: 0x000(1)
[1702630014.864010] info     | ProxyClient.cpp    | create_datareader        | datareader created      | client_key: 0x0B62A009, data
reader_id: 0x000(6), subscriber_id: 0x000(4)
[1702630014.959908] info     | ProxyClient.cpp    | create_topic             | topic created           | client_key: 0x0B62A009, topi
c_id: 0x004(2), participant_id: 0x000(1)
[1702630015.033537] info     | ProxyClient.cpp    | create_subscriber        | subscriber created      | client_key: 0x0B62A009, subs
criber_id: 0x001(4), participant_id: 0x000(1)
[1702630015.140350] info     | ProxyClient.cpp    | create_datareader        | datareader created      | client_key: 0x0B62A009, data
reader_id: 0x001(6), subscriber_id: 0x001(4)
[1702630015.158510] info     | ProxyClient.cpp    | create_topic             | topic created           | client_key: 0x0B62A009, topi
c_id: 0x005(2), participant_id: 0x000(1)
[1702630015.241039] info     | ProxyClient.cpp    | create_subscriber        | subscriber created      | client_key: 0x0B62A009, subs
criber_id: 0x002(4), participant_id: 0x000(1)
[1702630015.347393] info     | ProxyClient.cpp    | create_datareader        | datareader created      | client_key: 0x0B62A009, data
reader_id: 0x002(6), subscriber_id: 0x002(4)
```

3. Start the program

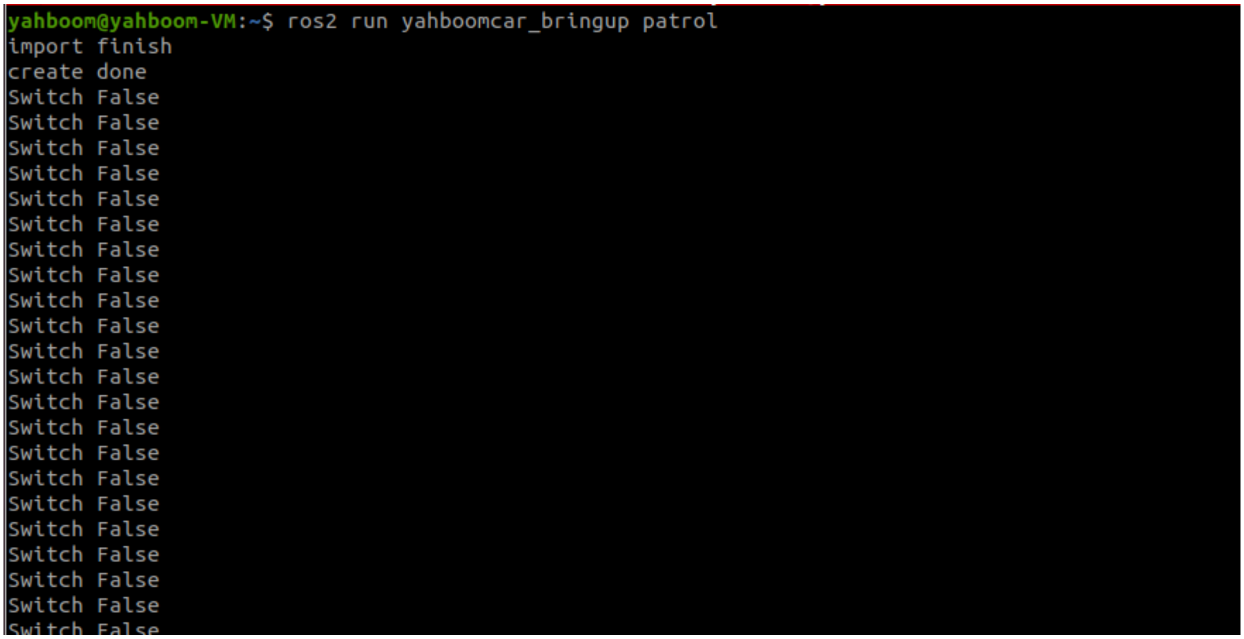
First, start the car's underlying data processing program. This program will release the TF transformation of odom->base_footprint. With this TF change, you can calculate "how far the car has gone" and input it at the terminal.

```
ros2 run yahboomcar_base_node base_node_X3
```

```
[INFO] [imu_filter_madgwick_node-1]: process started with pid [6638]
[INFO] [ekf_node-2]: process started with pid [6640]
[INFO] [static_transform_publisher-3]: process started with pid [6642]
[INFO] [joint_state_publisher-4]: process started with pid [6644]
[INFO] [robot_state_publisher-5]: process started with pid [6646]
[INFO] [static_transform_publisher-6]: process started with pid [6658]
[static_transform_publisher-3] [WARN] [1702865272.944043208] []: Old-style arguments are deprecated; see --help for new-style argume
nts
[static_transform_publisher-6] [WARN] [1702865272.984740987] []: Old-style arguments are deprecated; see --help for new-style argume
nts
[static_transform_publisher-3] [INFO] [1702865272.991057276] [base_link_to_base_imu]: Spinning until stopped - publishing transform
[static_transform_publisher-3] translation: ('-0.002999', '-0.003000', '0.031701')
[static_transform_publisher-3] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-3] from 'base_link' to 'imu_frame'
[static_transform_publisher-6] [INFO] [1702865273.005707993] [static_transform_publisher_JH06Gexf4GRodmgs]: Spinning until stopped -
 publishing transform
[static_transform_publisher-6] translation: ('0.000000', '0.000000', '0.050000')
[static_transform_publisher-6] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-6] from 'base_footprint' to 'base_link'
[robot_state_publisher-5] [WARN] [1702865273.013202438] [kdl_parser]: The root link base_link has an inertia specified in the URDF,
but KDL does not support a root link with an inertia.  As a workaround, you can add an extra dummy link to your URDF.
[robot_state_publisher-5] [INFO] [1702865273.013312806] [robot_state_publisher]: got segment base_link
[robot_state_publisher-5] [INFO] [1702865273.013516195] [robot_state_publisher]: got segment imu_Link
[robot_state_publisher-5] [INFO] [1702865273.013524175] [robot_state_publisher]: got segment jq1_Link
[robot_state_publisher-5] [INFO] [1702865273.013528144] [robot_state_publisher]: got segment jq2_Link
[robot_state_publisher-5] [INFO] [1702865273.013531665] [robot_state_publisher]: got segment radar_Link
[robot_state_publisher-5] [INFO] [1702865273.013535185] [robot_state_publisher]: got segment yh_Link
[robot_state_publisher-5] [INFO] [1702865273.013538763] [robot_state_publisher]: got segment yq_Link
[robot_state_publisher-5] [INFO] [1702865273.013542135] [robot_state_publisher]: got segment zh_Link
[robot_state_publisher-5] [INFO] [1702865273.013545612] [robot_state_publisher]: got segment zq_Link
[imu_filter_madgwick_node-1] [INFO] [1702865273.030399479] [imu_filter]: Starting ImuFilter
[imu_filter_madgwick_node-1] [INFO] [1702865273.031826501] [imu_filter]: Using dt computed from message headers
[imu_filter_madgwick_node-1] [INFO] [1702865273.031858361] [imu_filter]: The gravity vector is kept in the IMU message.
[imu_filter_madgwick_node-1] [INFO] [1702865273.032488302] [imu_filter]: Imu filter gain set to 0.100000
[imu_filter_madgwick_node-1] [INFO] [1702865273.032525566] [imu_filter]: Gyro drift bias set to 0.000000
[imu_filter_madgwick_node-1] [INFO] [1702865273.032531441] [imu_filter]: Magnetometer bias values: 0.000000 0.000000 0.000000
[imu_filter_madgwick_node-1] [INFO] [1702865273.053298796] [imu_filter]: First IMU message received.
[joint_state_publisher-4] [INFO] [1702865273.282975810] [joint_state_publisher]: Waiting for robot_description to be published on th
e robot_description topic...
```

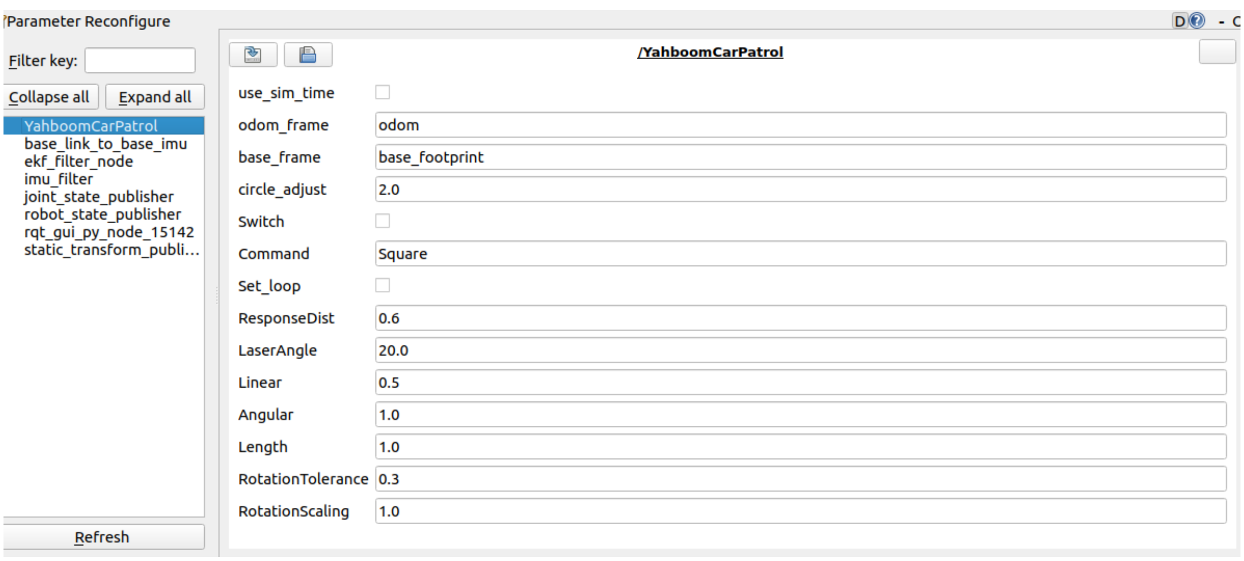
Then, start the radar patrol program and enter in the terminal,

```
ros2 run yahboomcar_bringup patrol
```



Open the parameter adjuster, give the patrol route, and enter the terminal.

```
ros2 run rqt_reconfigure rqt_reconfigure
```



Note: There may not be the above nodes when you first open it. You can see all nodes after clicking Refresh. The displayed YahboomCarPatrol is the patrol node.

4. Start patrolling

In the rqt interface, find the YahboomCarPatrol node. The [Command] inside is the set patrol route. Here we take the square patrol route as an example. The following will explain what types of patrol routes there are. After setting the route in [Command], click Switch to start patrolling.

```
distance:  0.6335016035139822
Switch True
Length
distance:  0.6335016035139822
Switch True
Length
distance:  0.7284602368836334
Switch True
Length
distance:  0.7284602368836334
Switch True
Length
distance:  0.7284602368836334
Switch True
Length
distance:  0.8070901854409693
Switch True
Length
distance:  0.8070901854409693
Switch True
Length
distance:  0.8070901854409693
Switch True
Length
```

Taking a square as an example, first walk in a straight line, then rotate 90 degrees, then walk in a straight line, then rotate 90 degrees, and so on, until the completed route is a square. If you encounter an obstacle while walking, the buzzer will stop and sound.

```
Length
distance:  5.138314275626346
obstacles
Switch True
Length
distance:  5.121662891248637
obstacles
Switch True
Length
distance:  5.121662891248637
obstacles
Switch True
Length
distance:  5.09916912722615
obstacles
Switch True
Length
```

As shown in the picture above, obstacles will be printed when encountering obstacles.

Other parameters of the rqt interface are described as follows:

- odom_frame: The coordinate system name of the odometer
- base_frame: The name of the base coordinate system
- circle_adjust: When the patrol route is circular, this value can be used as a coefficient to adjust the size of the circle. See the code for details.

- Switch：Game switch
- Command：There are several types of patrol routes: [LengthTest]-straight line patrol, [Circle]-circle patrol, [Square]-square patrol, [Triangle]-triangle patrol
- Set_loop：The development of re-patrol will continue patrolling in a circular pattern with the set route after setting it up.
- ResponseDist：Obstacle detection distance
- LaserAngle：Radar detection angle
- Linear：Linear speed
- Angular：Angular velocity
- Length：straight line distance
- RotationTolerance：Tolerance value allowed for rotation error
- RotationScaling：rotation scaling factor

After modifying the above parameters, you need to click on the blank space to transfer the parameters into the program.

5. Code analysis

Source code reference path (taking the supporting virtual machine as an example):

```
/home/yahboom/yahboomcar_ws/src/yahboomcar_bringup/yahboomcar_bringup
```

patrol.py, the core code is as follows,

Create radar and remote control data subscribers

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

Creation speed, 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()
```

Two important functions are linear advancing and rotation Spin. All patrol routes are nothing more than a combination of linear motion and rotation.

advancing

```
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
    else:
        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")
            else:
                #print("Go")
                b = UInt16()
                b.data = 0
                self.pub_Buzzer.publish(UInt16())
                self.pub_cmdVel.publish(move_cmd)
                self.moving = True
                return False
```

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")
    else:
        b = UInt16()
        b.data = 0
        self.pub_Buzzer.publish(UInt16())
        if self.Command == "Square" or self.Command == "Triangle":
            move_cmd.angular.z = copysign(self.Angular, self.error)
        elif self.Command == "Circle":
            length = self.Linear * self.circle_adjust / self.Length
            move_cmd.linear.x = self.Linear
            move_cmd.angular.z = copysign(length, self.error)
        self.pub_cmdVel.publish(move_cmd)
    self.moving = True
```

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

```
def Square(self):
    if self.index == 0:
    print("Length")
    #First walk in a straight line for 1 meter
    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")
        #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)
.....
```