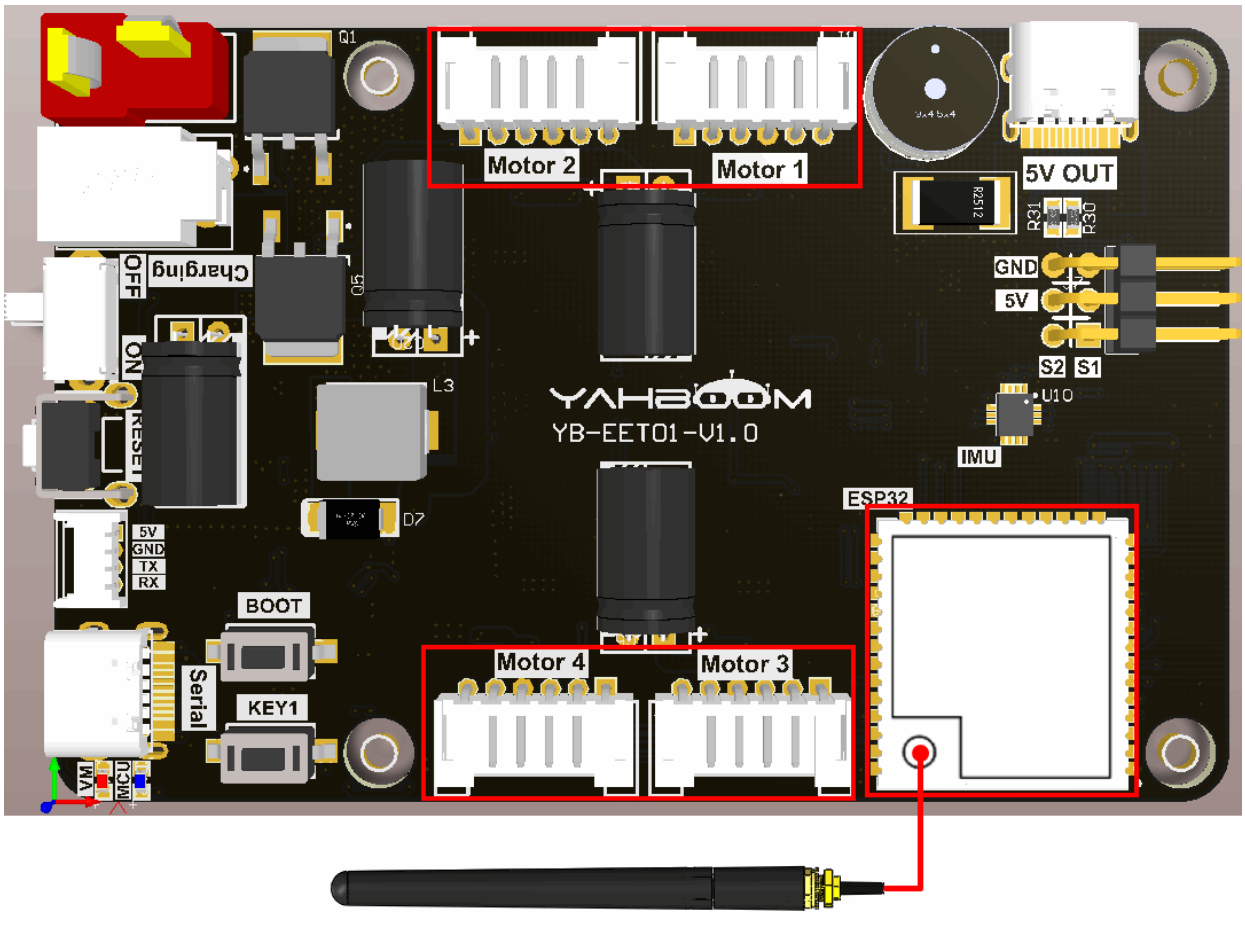
Subscribe speed control topics

1. Experimental purpose

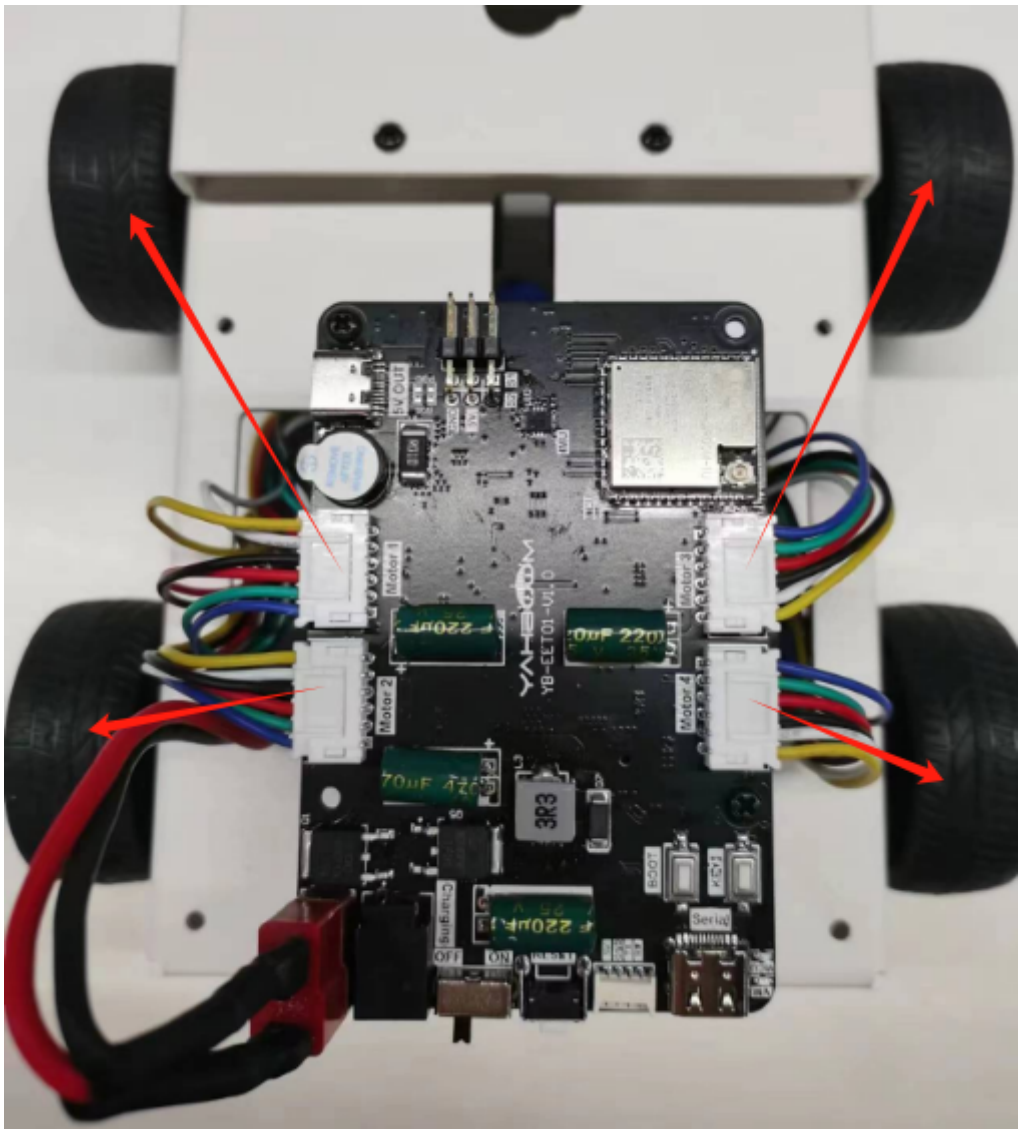
Learn ESP32-microROS components, access the ROS2 environment, and subscribe to the topic of controlling the speed of the car.

2. Hardware connection

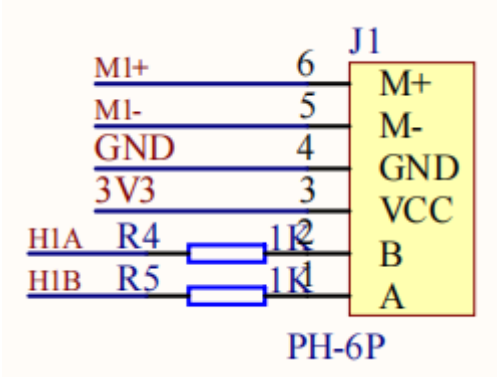
As shown in the figure below, the microROS control board integrates an active buzzer and the ESP32-S3-WROOM core module, which has its own wireless WiFi function. The ESP32-S3 core module needs to be connected to the antenna, and then the four-way motors are connected to the motor interface. On the computer, you also need to connect the type-C data cable to the computer and the microROS control board for the firmware burning function.



The corresponding names of the four motor interfaces are: left front wheel->Motor1, left rear wheel->Motor2, right front wheel->Motor3, right rear wheel->Motor4.



Motor interface line sequence, there is a detailed line sequence silk screen on the back of the microROS control board. Here we take Motor1 as an example. M1+ and M1- are the interfaces for controlling the rotation of the motor. GND and VCC are the power supply circuits of the encoder. H1A and H1B are the encoder pulses. Detection pin.



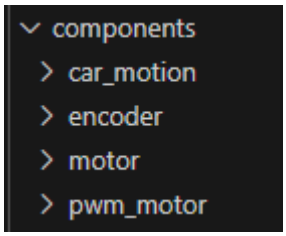
Note: If you are using the 310 motor and motor cable provided by Yabo Intelligence, connect the white wire shell end to the interface on the microROS control board, and the black wire shell end to the 310 motor interface.

3. Core code analysis

The virtual machine path corresponding to the program source code is as follows

```
~/esp/Samples/microros_samples/twist_subscriber
```

Since the 310 encoder motor is used this time, and the components with the encoder motor have been made in the previous routine, the relevant components of the encoder motor need to be copied to the components directory of the project. Call Motor_Init at the beginning of the program to initialize the motor.



Get the WiFi name and password to connect from the IDF configuration tool.

```
#define ESP_WIFI_SSID      CONFIG_ESP_WIFI_SSID
#define ESP_WIFI_PASS      CONFIG_ESP_WIFI_PASSWORD
#define ESP_MAXIMUM_RETRY  CONFIG_ESP_MAXIMUM_RETRY
```

The uros_network_interface_initialize function will connect to WiFi hotspots based on the WiFi configuration in IDF.

```
ESP_ERROR_CHECK(uros_network_interface_initialize());
```

Then obtain ROS_NAMESPACE, ROS_DOMAIN_ID, ROS_AGENT_IP and ROS_AGENT_PORT from the IDF configuration tool.

```
#define ROS_NAMESPACE      CONFIG_MICRO_ROS_NAMESPACE
#define ROS_DOMAIN_ID      CONFIG_MICRO_ROS_DOMAIN_ID
#define ROS_AGENT_IP       CONFIG_MICRO_ROS_AGENT_IP
#define ROS_AGENT_PORT     CONFIG_MICRO_ROS_AGENT_PORT
```

Initialize the configuration of microROS, in which ROS_DOMAIN_ID, ROS_AGENT_IP and ROS_AGENT_PORT are modified in the IDF configuration tool according to actual needs.

```
    rcl_allocator_t allocator = rcl_get_default_allocator();
    rclc_support_t support;

    // 创建rcl初始化选项
    // Create init_options.
    rcl_init_options_t init_options = rcl_get_zero_initialized_init_options();
    RCCHECK(rcl_init_options_init(&init_options, allocator));
```

```
    // 修改ROS域ID
    // change ros domain id
    RCCHECK(rcl_init_options_set_domain_id(&init_options, ROS_DOMAIN_ID));

    // 初始化rmw选项
    // Initialize the rmw options
    rmw_init_options_t *rmw_options =
rcl_init_options_get_rmw_init_options(&init_options);

    // 设置静态代理IP和端口
    // Setup static agent IP and port
    RCCHECK(rmw_uros_options_set_udp_address(ROS_AGENT_IP, ROS_AGENT_PORT,
rmw_options));
```

Try to connect to the proxy. If the connection is successful, go to the next step. If the connection to the proxy is unsuccessful, you will always be in the connected state.

```
    while (1)
    {
        ESP_LOGI(TAG, "Connecting agent: %s:%s", ROS_AGENT_IP, ROS_AGENT_PORT);
        state_agent = rclc_support_init_with_options(&support, 0, NULL,
&init_options, &allocator);
        if (state_agent == ESP_OK)
        {
            ESP_LOGI(TAG, "Connected agent: %s:%s", ROS_AGENT_IP, ROS_AGENT_PORT);
            break;
        }
        vTaskDelay(pdMS_TO_TICKS(500));
    }
```

Create the node "twist_subscriber", in which ROS_NAMESPACE is empty by default and can be modified in the IDF configuration tool according to actual conditions.

```
    rcl_node_t node;
    RCCHECK(rclc_node_init_default(&node, "twist_subscriber", ROS_NAMESPACE,
&support));
```

To create the subscriber "cmd_vel", you need to specify the ROS topic information as geometry_msgs/msg/Twist type.

```
    RCCHECK(rclc_subscription_init_default(
        &twist_subscriber,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(geometry_msgs, msg, Twist),
        "cmd_vel"));
```

Add subscribers to the executor, where the handle_num parameter of the executor represents the number added to the executor.

```
    rclc_executor_t executor;
    int handle_num = 1;
    RCCHECK(rclc_executor_init(&executor, &support.context, handle_num,
&allocator));

    RCCHECK(rclc_executor_add_subscription(
        &executor,
        &twist_subscriber,
        &twist_msg,
        &twist_Callback,
        ON_NEW_DATA));
```

When the microros subscriber receives the topic data, the twist_Callback callback function is triggered to control the movement of the robot based on the received value.

```
void twist_Callback(const void *msgin)
{
    ESP_LOGI(TAG, "cmd_vel:%.2f, %.2f, %.2f", twist_msg.linear.x,
twist_msg.linear.y, twist_msg.angular.z);
    Motion_Ctrl(twist_msg.linear.x, 0, twist_msg.angular.z);
}
```

Call rclc_executor_spin_some in the loop to make microros work normally.

```
    while (1)
    {
        rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100));
        usleep(1000);
    }
```

4. Compile, download and flash firmware

Use a Type-C data cable to connect the virtual machine/computer and the microROS control board. If the system pops up, choose to connect to the virtual machine.

Activate the ESP-IDF development environment. Note that every time you open a new terminal, you need to activate the ESP-IDF development environment before compiling the firmware.

```
source ~/esp/esp-idf/export.sh
```

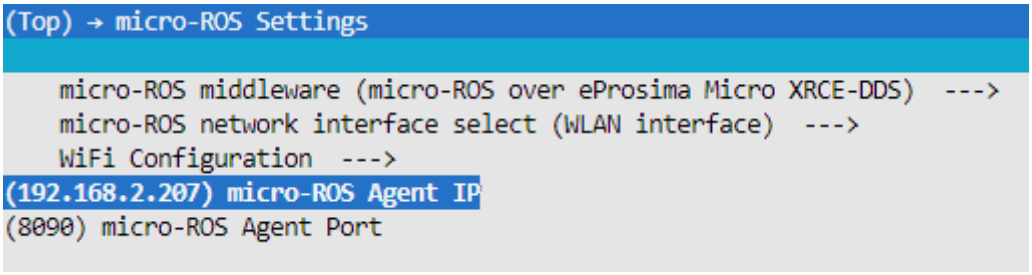
Enter the project directory

```
cd ~/esp/Samples/microros_samples/twist_subscriber
```

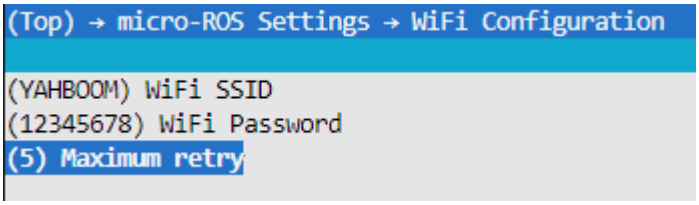
Open the ESP-IDF configuration tool.

```
idf.py menuconfig
```

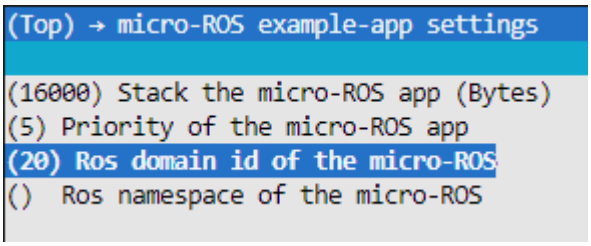
Open micro-ROS Settings, fill in the IP address of the agent host in micro-ROS Agent IP, and fill in the port number of the agent host in micro-ROS Agent Port.



Open micro-ROS Settings->WiFi Configuration in sequence, and fill in your own WiFi name and password in the WiFi SSID and WiFi Password fields.



Open the micro-ROS example-app settings. The Ros domain id of the micro-ROS defaults to 20. If multiple users are using it at the same time in the LAN, the parameters can be modified to avoid conflicts. Ros namespace of the micro-ROS is empty by default and does not need to be modified under normal circumstances. If non-empty characters (within 10 characters) are modified, the namespace parameter will be added before the node and topic.



After modification, press S to save, and then press Q to exit the configuration tool.

Compile, burn, and open the serial port simulator.

```
idf.py build flash monitor
```

If you need to exit the serial port simulator, press **Ctrl+]**.

5. Experimental results

Note: The motor of the car controlled by ROS2 will rotate. Please lift the car into the air first to prevent the car from moving randomly on the table.

After powering on, ESP32 tries to connect to the WiFi hotspot, and then tries to connect to the proxy IP and port.

If the agent is not turned on in the virtual machine/computer terminal, please enter the following command to turn on the agent. If the agent is already started, there is no need to start the agent again.

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

```
:~$ docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privile
ged --net=host microros/micro-ros-agent:humble udp4 --port 8090 -v4
[1705475406.254095] info     | UDPv4AgentLinux.cpp | init                    |
running...             | port: 8090
[1705475406.254622] info     | Root.cpp           | set_verbose_level       | l
ogger setup            | verbose_level: 4
```

After the connection is successful, a node and a subscriber are created.

```
I (2051) MAIN: Connecting agent: 192.168.2.207:8090
I (2059) main_task: Returned from app_main()
I (2070) MAIN: Connected agent: 192.168.2.207:8090
```

At this time, you can open another terminal on the virtual machine/computer and view the /twist_subscriber node.

```
ros2 node list
ros2 node info /twist_subscriber
```

```
:~$ ros2 node list
/twist_subscriber
:~$ ros2 node info /twist_subscriber
/twist_subscriber
  Subscribers:
    /cmd_vel: geometry_msgs/msg/Twist
  Publishers:

  Service Servers:

  Service Clients:

  Action Servers:

  Action Clients:
```

Publish data to the /cmd_vel topic and control the robot car to walk forward at 0.5m/s.

```
ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.5, y: 0.0, z:
0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
```

Publish data to the /cmd_vel topic and control the robot car to rotate at 1.5rad/s.

```
ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.0, y: 0.0, z:
0.0}, angular: {x: 0.0, y: 0.0, z: 1.5}}"
```

Publish data to the /cmd_vel topic and control the robot car to stop.

```
ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.0, y: 0.0, z:
0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
```

```
[redacted]:~$ ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear
: {x: 0.5, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
publisher: beginning loop
publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=0.5, y
=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))

[redacted]:~$ ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear
: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.5}}"
publisher: beginning loop
publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=0.0, y
=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=1.5))

[redacted]:~$ ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear
: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
publisher: beginning loop
publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=0.0, y
=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
```

You can see the printed information on the serial port simulator, indicating that the subscription is successful.