

Release speed topic

Release speed topic

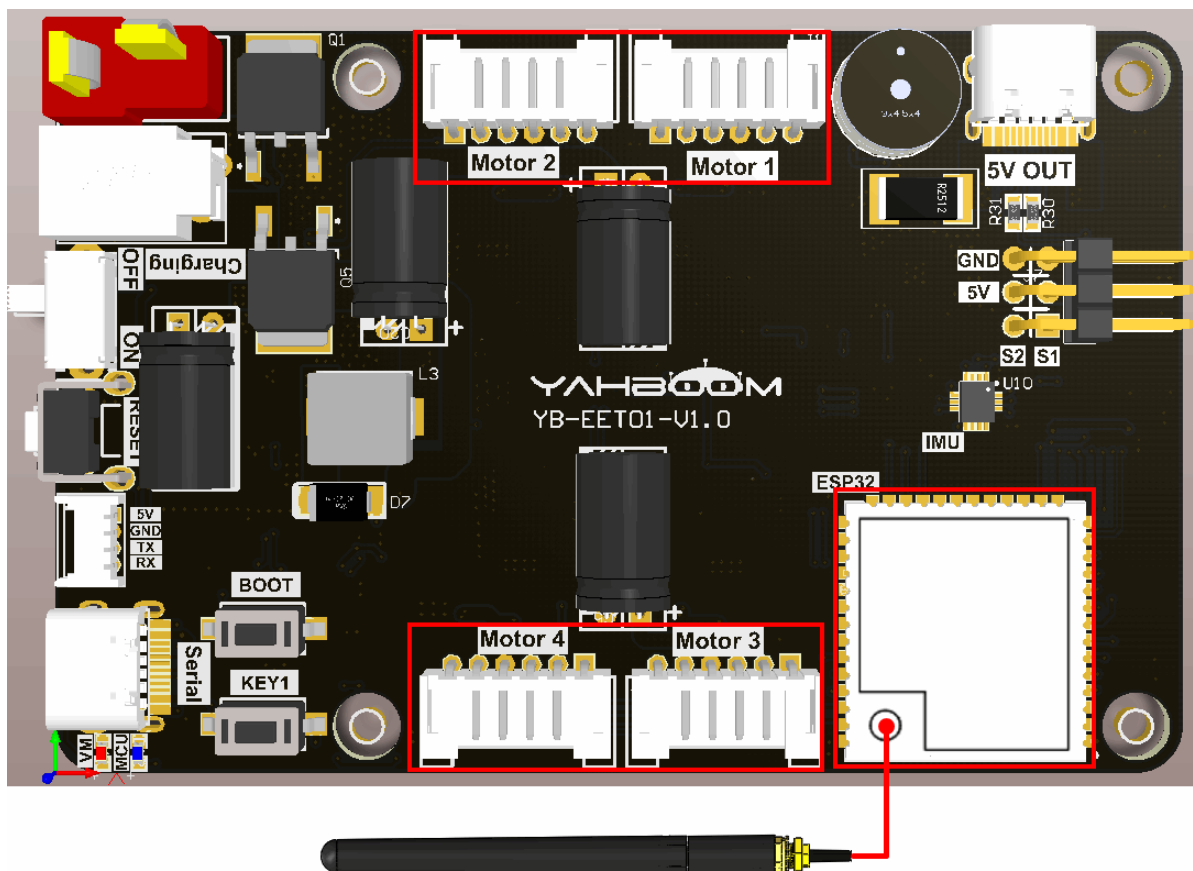
1. Experimental purpose
2. Hardware connection
3. Core code analysis
4. Compile, download and flash firmware
5. Experimental results

1. Experimental purpose

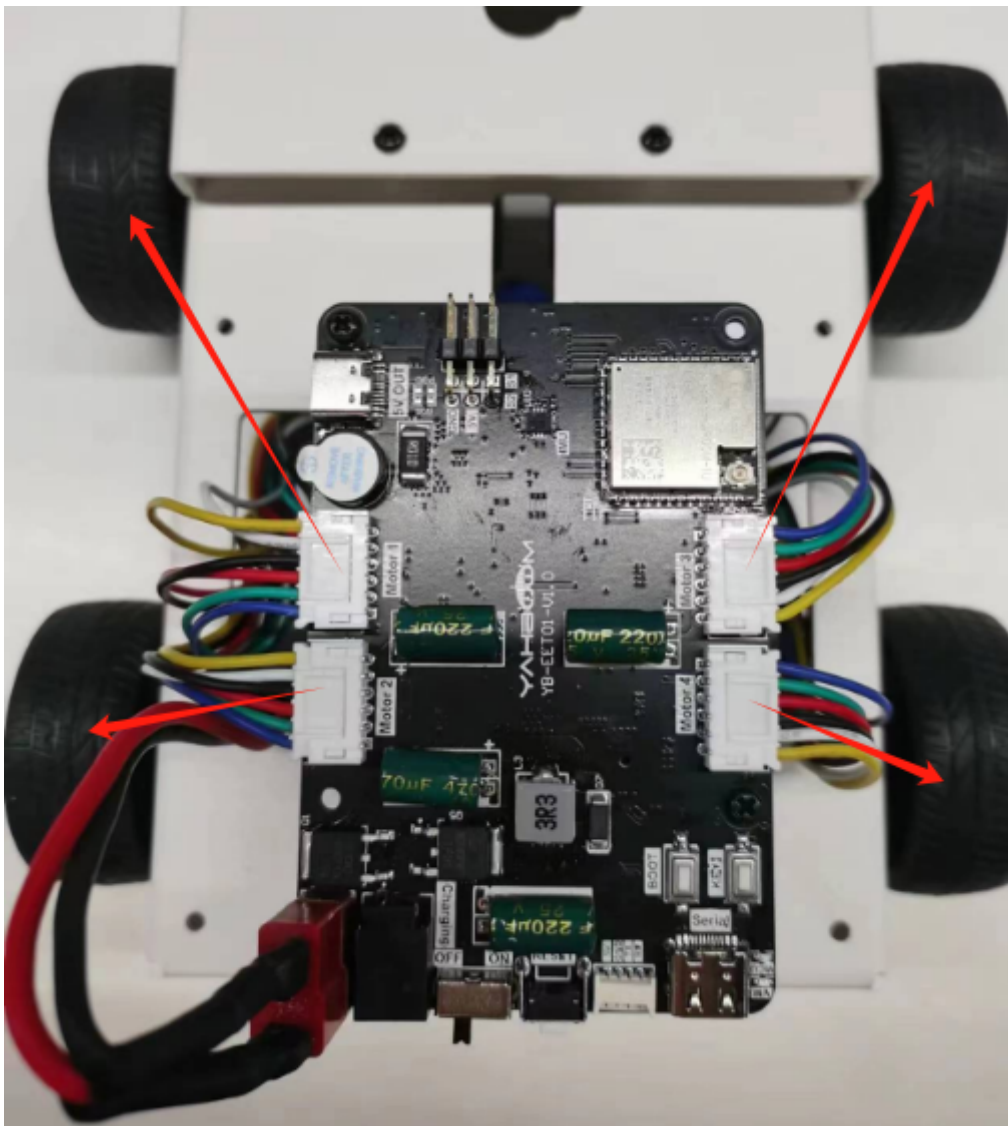
Learn ESP32-microROS components, access the ROS2 environment, and publish the topic of odom speed of robot cars.

2. Hardware connection

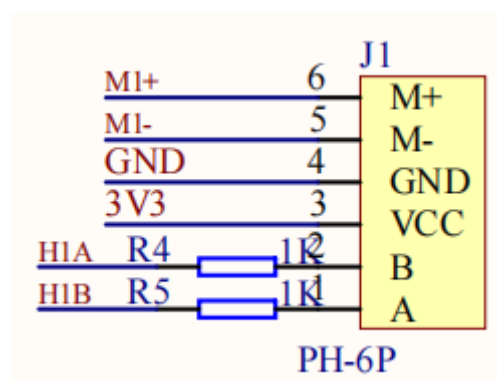
As shown in the figure below, the microROS control board integrates the ESP32-S3-WROOM core module with its own wireless WiFi function. The ESP32-S3 core module needs to be connected to the antenna, and then the four-way motors are connected to the motor interface. The type -C data cable connects the computer and microROS control board for 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.



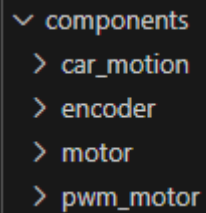
Note: If you are using the 310 motor and motor cable provided by Yabo Intelligence, connect the white wire shell end to the interface on the microROS control board, and the black wire shell end to the 310 motor interface.

3. Core code analysis

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

```
~/esp/Samples/microros_samples/odom_publisher
```

Based on twist_subscriber's subscription speed function, the function of publishing odom speed topics is added.



```
▼ components
  > car_motion
  > encoder
  > motor
  > pwm_motor
```

Initialize the odom information of the publishing speed, set the frame_id to "odom_frame", set the child_frame_id to "base_footprint", and then decide whether to add the ROS_NAMESPACE prefix based on whether ROS_NAMESPACE is empty.

```
void odom_ros_init(void)
{
    char* content_frame_id = "odom_frame";
    char* content_child_frame_id = "base_footprint";
    int len_namespace = strlen(ROS_NAMESPACE);
    int len_frame_id_max = len_namespace + strlen(content_frame_id) + 2;
    int len_child_frame_id_max = len_namespace + strlen(content_child_frame_id) +
2;
    char* frame_id = malloc(len_frame_id_max);
    char* child_frame_id = malloc(len_child_frame_id_max);
    if (len_namespace == 0)
    {
        // ROS命名空间为空字符
        // The ROS namespace is empty characters
        sprintf(frame_id, "%s", content_frame_id);
        sprintf(child_frame_id, "%s", content_child_frame_id);
    }
    else
    {
        // 拼接命名空间和frame id
        // Concatenate the namespace and frame id
        sprintf(frame_id, "%s/%s", ROS_NAMESPACE, content_frame_id);
        sprintf(child_frame_id, "%s/%s", ROS_NAMESPACE, content_child_frame_id);
    }
    msg_odom.header.frame_id =
micro_ros_string_utilities_set(msg_odom.header.frame_id, frame_id);
    msg_odom.child_frame_id =
micro_ros_string_utilities_set(msg_odom.child_frame_id, child_frame_id);
    free(frame_id);
    free(child_frame_id);
}
```

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();
RCHECK(rcl_init_options_init(&init_options, allocator));
// 修改ROS域ID
// change ros domain id
RCHECK(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
RCHECK(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 = rcl_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 "odom_publisher", 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(rcl_node_init_default(&node, "odom_publisher", ROS_NAMESPACE,
&support));

```

To create the publisher "odom_raw", you need to specify the publisher information as nav_msgs/msg/Odometry type.

```

RCCHECK(rcl_publisher_init_default(
    &publisher_odom,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(nav_msgs, msg, Odometry),
    "odom_raw"));

```

Create a timer for the publisher with a publishing frequency of 20HZ.

```

const unsigned int timer_timeout = 50;
RCCHECK(rcl_timer_init_default(
    &timer_odom,
    &support,
    RCL_MS_TO_NS(timer_timeout),
    timer_odom_callback));

```

Create an executor, where the three parameters are the numbers controlled by the executor, which should be greater than or equal to the number of subscribers and publishers added to the executor. and add the publisher's timer to the executor.

```

    rcl_executor_t executor;
    int handle_num = 2;
    RCCHECK(rcl_executor_init(&executor, &support.context, handle_num,
&allocator));

    RCCHECK(rcl_executor_add_timer(&executor, &timer_odom));
    RCCHECK(rcl_executor_add_subscription(
        &executor,
        &twist_subscriber,
        &twist_msg,
        &twist_Callback,
        ON_NEW_DATA));

```

The main function of odom's timer callback function is to update odom data and send the data.

```

void timer_odom_callback(rcl_timer_t *timer, int64_t last_call_time)
{
    RCLC_UNUSED(last_call_time);
    if (timer != NULL)
    {
        struct timespec time_stamp = get_timespec();
        unsigned long now = get_millisecond();
        float vel_dt = (now - prev_odom_update) / 1000.0;
        prev_odom_update = now;
        Motion_Get_Speed(&car_motion);
        odom_update(
            vel_dt,
            car_motion.Vx,
            car_motion.Vy,
            car_motion.Wz);
        msg_odom.header.stamp.sec = time_stamp.tv_sec;
        msg_odom.header.stamp.nanosec = time_stamp.tv_nsec;
        RCSOFTCHECK(rcl_publish(&publisher_odom, &msg_odom, NULL));
    }
}

```

The speed read from the robot car, and the odom information is updated based on the car speed.

```

void odom_update(float vel_dt, float linear_vel_x, float linear_vel_y, float
angular_vel_z)
{
    float delta_heading = angular_vel_z * vel_dt; // radians
    float cos_h = cos(heading_);
    float sin_h = sin(heading_);
    float delta_x = (linear_vel_x * cos_h - linear_vel_y * sin_h) * vel_dt; // m
    float delta_y = (linear_vel_x * sin_h + linear_vel_y * cos_h) * vel_dt; // m

    // calculate current position of the robot
    x_pos_ += delta_x;
    y_pos_ += delta_y;
    heading_ += delta_heading;

    // calculate robot's heading in quaternion angle
    // ROS has a function to calculate yaw in quaternion angle
    float q[4];
    odom_euler_to_quat(0, 0, heading_, q);
}

```

```

// robot's position in x,y, and z
msg_odom.pose.pose.position.x = x_pos_;
msg_odom.pose.pose.position.y = y_pos_;
msg_odom.pose.pose.position.z = 0.0;

// robot's heading in quaternion
msg_odom.pose.pose.orientation.x = (double)q[1];
msg_odom.pose.pose.orientation.y = (double)q[2];
msg_odom.pose.pose.orientation.z = (double)q[3];
msg_odom.pose.pose.orientation.w = (double)q[0];

msg_odom.pose.covariance[0] = 0.001;
msg_odom.pose.covariance[7] = 0.001;
msg_odom.pose.covariance[35] = 0.001;

// linear speed from encoders
msg_odom.twist.twist.linear.x = linear_vel_x;
msg_odom.twist.twist.linear.y = linear_vel_y;
msg_odom.twist.twist.linear.z = 0.0;

// angular speed from encoders
msg_odom.twist.twist.angular.x = 0.0;
msg_odom.twist.twist.angular.y = 0.0;
msg_odom.twist.twist.angular.z = angular_vel_z;

msg_odom.twist.covariance[0] = 0.0001;
msg_odom.twist.covariance[7] = 0.0001;
msg_odom.twist.covariance[35] = 0.0001;
}

```

Call `rclcpp_executor_spin_some` in the loop to make microros work normally.

```

while (1)
{
    rclcpp_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/odom_publisher
```

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.

```
(Top) → micro-ROS Settings
micro-ROS middleware (micro-ROS over eProsima Micro XRCE-DDS) --->
micro-ROS network interface select (WLAN interface) --->
WiFi Configuration --->
(192.168.2.207) micro-ROS Agent IP
(8090) 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.

```
(Top) → micro-ROS Settings → WiFi Configuration
(YAHBOOM) WiFi SSID
(12345678) WiFi Password
(5) Maximum retry
```

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.

```
(Top) → micro-ROS example-app settings
(16000) Stack the micro-ROS app (Bytes)
(5) Priority of the micro-ROS app
(20) Ros domain id of the micro-ROS
() Ros namespace of the micro-ROS
```

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

Compile, flash, and open the serial port simulator.

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

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

5. Experimental results

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 --privileged --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 | 1
ogger_setup | verbose_level: 4
```

After the connection is successful, a node, a publisher 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 in the virtual machine/computer and view the /odom_publisher node.

```
ros2 node list
ros2 node info /odom_publisher
```

```
~$ ros2 node list
/odom_publisher
~$ ros2 node info /odom_publisher
/odom_publisher
Subscribers:
  /cmd_vel: geometry_msgs/msg/Twist
Publishers:
  /odom_raw: nav_msgs/msg/Odometry
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}}"
```

Subscribe to the data of the /odom_raw topic,

```
ros2 topic echo /odom_raw
```

Press Ctrl+C to end the command

```

:~$ ros2 topic echo /odom_raw
header:
  stamp:
    sec: 209
    nanosec: 224000000
  frame_id: odom_frame
child_frame_id: base_footprint
pose:
  pose:
    position:
      x: 38.30796432495117
      y: -0.9879703521728516
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: -0.06556393206119537
      w: 0.997848391532898

```

```

twist:
  twist:
    linear:
      x: 0.49662500619888306
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: -0.031521592289209366

```

Check the frequency of the /odom_raw topic. It is normal if it is about 20hz.

```
ros2 topic hz /odom_raw
```

Press Ctrl+C to end the command

```

:~$ ros2 topic hz /odom_raw
WARNING: topic [/odom_raw] does not appear to be published yet
average rate: 19.561
  min: 0.025s max: 0.075s std dev: 0.01553s window: 21
average rate: 19.866
  min: 0.020s max: 0.084s std dev: 0.01700s window: 42
average rate: 19.837
  min: 0.020s max: 0.084s std dev: 0.01616s window: 63
average rate: 19.790
  min: 0.020s max: 0.084s std dev: 0.01513s window: 84
average rate: 20.008
  min: 0.017s max: 0.115s std dev: 0.01590s window: 105

```

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}}"
```

