Robot kinematics analysis

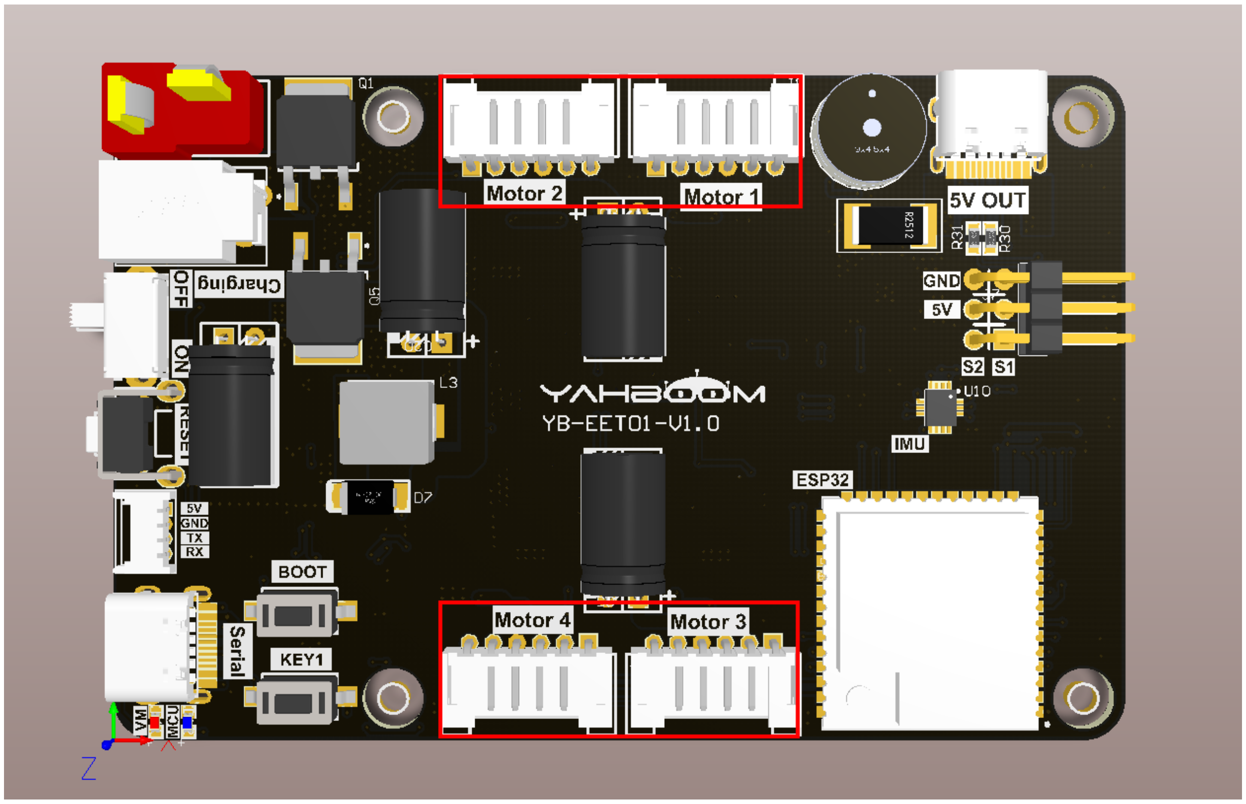
Robot kinematics analysis
- 1. Experimental purpose
- 2. Hardware connection
- 3. Establish kinematics model
- 4. Core code analysis
- 5. Compile, download and flash firmware
- 6. Experimental results

1. Experimental purpose

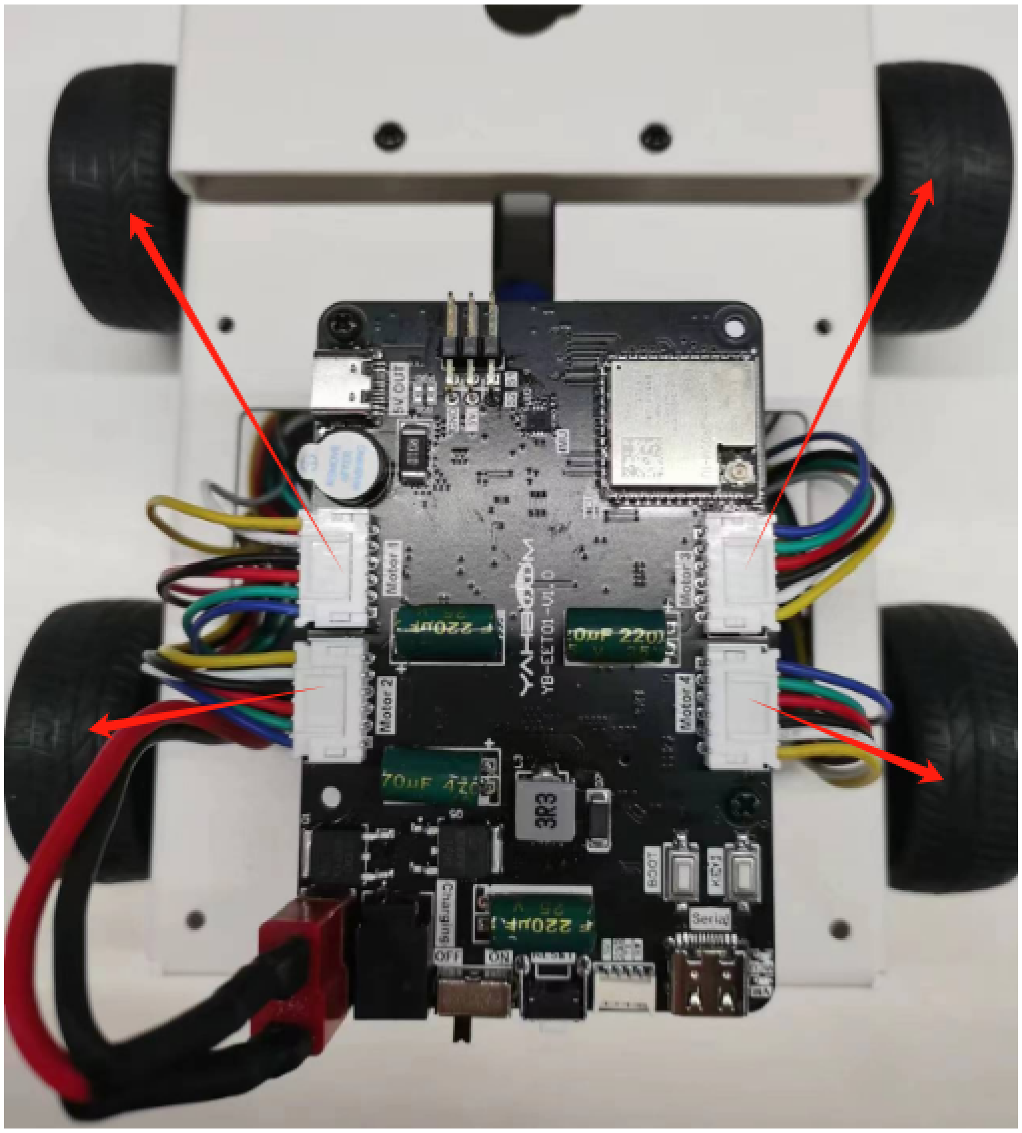
Use the encoded motor interface of the microROS control board to learn ESP32 to control the movement of the robot car and analyze the kinematic equations of the car. Since the real car has structural errors and factors such as resistance and friction during movement, it is more complicated. For the sake of simplicity, the analysis here is limited to the ideal state of the four-wheeled car.

2. Hardware connection

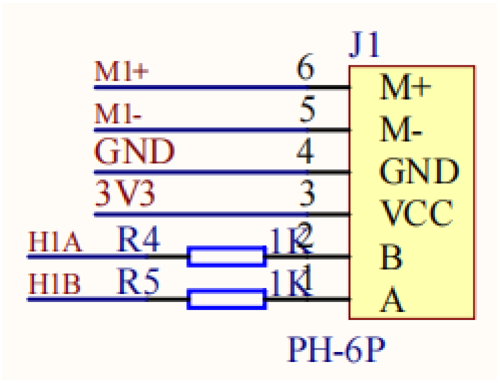
As shown in the figure below, the microROS control board integrates four encoder motor control interfaces. An additional encoder motor needs to be connected. The motor control interface supports 310 motors. A type-C data cable also needs to be connected to the computer and the microROS control board to burn firmware. 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, and H1A and H1B are the encoder pulse detection pins.

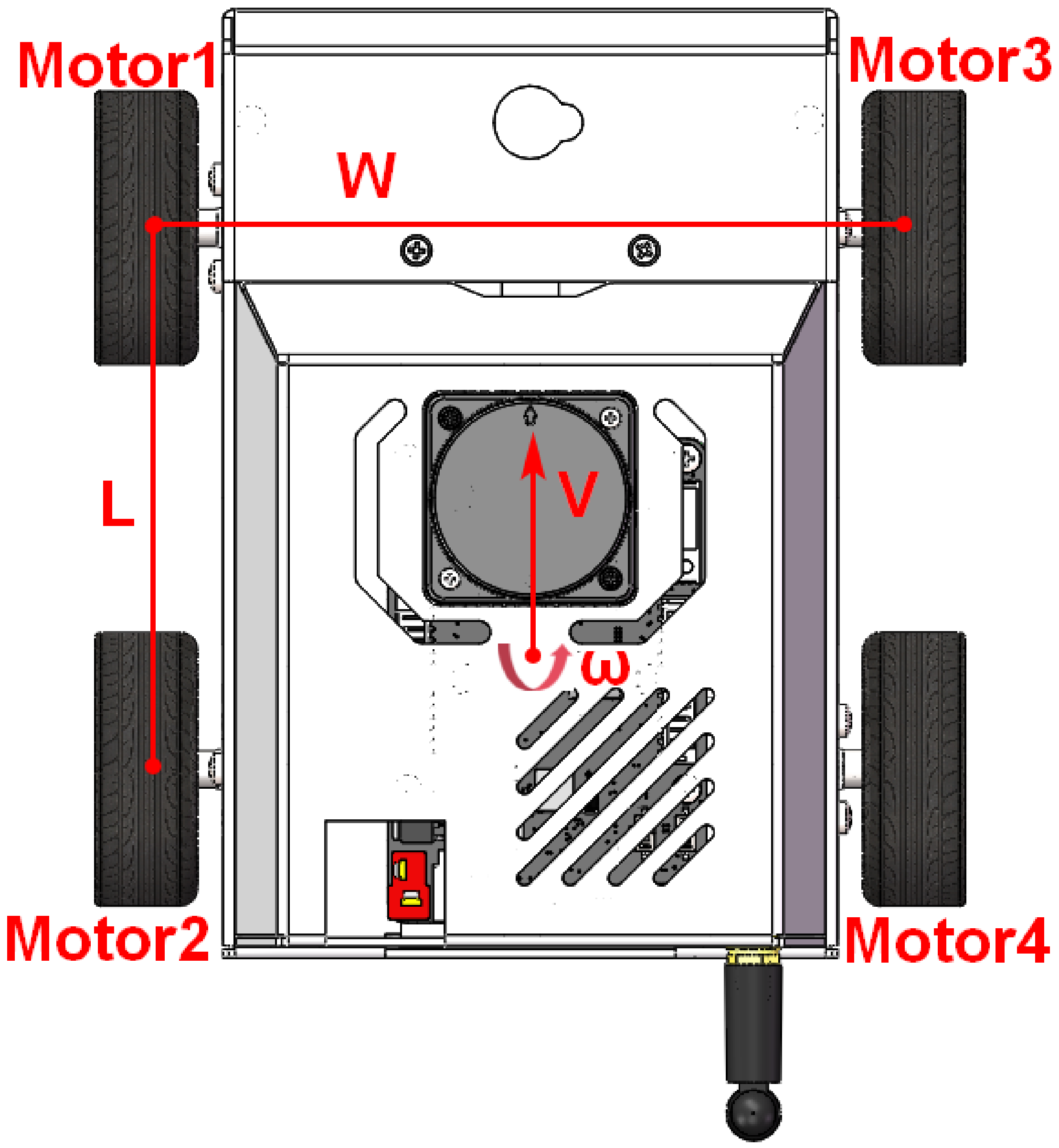


Note: If you are using the 310 motor and motor cable provided by Yabo Intelligence, connect the white wire shell end to the interface on the microROS control board, and the black wire shell end to the 310 motor interface.

3. Establish kinematics model

The four-wheel drive car is abstracted into the model in the figure below, where W represents the interval between the center points of the left and right motors; L represents the interval between the center points of the front and rear motors; the motors M1, M2, M3, and M4 represent the four drive motors of the car; V represents the line of the car Speed (forward and backward speed), ω represents the car angular speed (rotation speed).

Ideally, by controlling the linear velocity V and angular velocity ω, it is transformed into controlling four motors to control the movement of the four-wheel drive car.



Let V~m1~, V~m2~, V~m3~, and V~m4~ be the speed values of motors M1, M2, M3, and M4, that is, the rotation speed of the wheels. V~x~ is the linear speed of the car, V~ When x~ is positive, it means forward, when V~x~ is negative, it means backward, V~z~ is the angular speed of the car, when V~z~ is positive, it means left, and when V~z~ is negative, it means right.   A is half the distance W between the center points of the left and right motors of the car A=$\frac{W}{2}$, B is half the distance L between the center points of the front and rear motors of the car B=$\frac{L}{2}$

When the car moves forward or backward.

V~m1~=V~x~

V~m2~=V~x~

V~m3~=V~x~

V~m4~=V~x~

When the car rotates around the geometric center point,

V~m1~=-V~z~*(A+B)

V~m2~=-V~z~*(A+B)

V~m3~=V~z~*(A+B)

V~m4~=V~z~*(A+B)

According to the above formula, we can get the following information

V~m1~=V~x~-V~z~*(A+B)

V~m2~=V~x~-V~z~*(A+B)

V~m3~=V~x~+V~z~*(A+B)

V~m4~=V~x~+V~z~*(A+B)

4. Core code analysis

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

```
~/esp/Samples/esp32_samples/car_motion
```

According to the analysis of the robot kinematic model, the function to control the robot is obtained. V_x represents the control linear velocity [-1.0, 1.0], V_y is invalid, and V_z represents the control angular velocity [-5.0, 5.0].

```
void Motion_Ctrl(float V_x, float V_y, float V_z)
{
    line_v = V_x;
    angular_v = V_z;
    speed_L1_setup = line_v - angular_v * ROBOT_APB;
    speed_L2_setup = line_v - angular_v * ROBOT_APB;
    speed_R1_setup = line_v + angular_v * ROBOT_APB;
    speed_R2_setup = line_v + angular_v * ROBOT_APB;
    Motor_Set_Speed(speed_L1_setup, speed_L2_setup, speed_R1_setup, speed_R2_setup);
}
```

By reading the speed of the four motors, the kinematic speed of the robot is calculated.

```
void Motion_Get_Speed(car_motion_t* car)
{
    float speed_m1 = 0, speed_m2 = 0, speed_m3 = 0, speed_m4 = 0;
    Motor_Get_Speed(&speed_m1, &speed_m2, &speed_m3, &speed_m4);

    car->Vx = (speed_m1 + speed_m2 + speed_m3 + speed_m4) / 4;
    car->Vy = 0;
    car->Wz = -(speed_m1 + speed_m2 - speed_m3 - speed_m4) / 4.0f / ROBOT_APB;
    if(car->Wz == 0) car->Wz = 0;
}
```

Call the Motor_Init function in app_main to initialize the encoder motor, then set the speed of the car to 0.5m/s, print the current speed value of the car every 100 milliseconds, and the car will stop after 5 seconds.

```
void app_main(void)
{
    printf("hello yahboom\n");
    ESP_LOGI(TAG, "Nice to meet you!");
    int count = 0;
    vTaskDelay(pdMS_TO_TICKS(1000));


    Motor_Init();
    Motion_Ctrl(0.5, 0, 0);
    car_motion_t micro_car;

    while (1)
    {
        Motion_Get_Speed(&micro_car);
        ESP_LOGI(TAG, "Read Motion:%.2f, %.2f, %.2f", micro_car.Vx, micro_car.Vy,
micro_car.Wz);
        vTaskDelay(pdMS_TO_TICKS(100));
        count++;
        if(count >= 50)
        {
            Motion_Stop(STOP_COAST);
        }
    }
}
```

5. Compile, download and flash firmware

Note: Since the connected motor will rotate after the firmware is burned, please lift the car into the air first to prevent the car from moving around on the desktop.

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/esp32_samples/car_motion
```

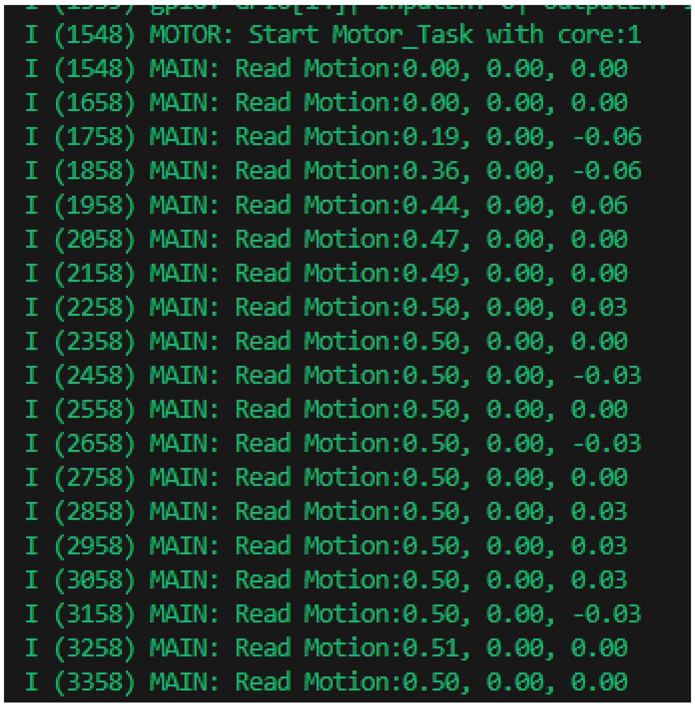
Compile, flash, and open the serial port simulator

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

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

6. Experimental results

The serial port simulator prints the "hello yahboom" greeting. At this time, the car speed will be printed every 100 milliseconds. And the car starts to move at the preset speed. Motion_Ctrl(0.5, 0, 0) is a function to adjust the speed. The car's movement speed can be adjusted according to actual needs. Theoretically, there will be a certain error between the car speed and the actual setting. If there is not much difference between the read speed and the set speed, it means that the motor is normal. The car will automatically stop after 5 seconds of movement. If you need to let the motor move again, please press the reset button on the microROS control board.



```
I (1548) MOTOR: Start Motor_Task with core:1
I (1548) MAIN: Read Motion:0.00, 0.00, 0.00
I (1658) MAIN: Read Motion:0.00, 0.00, 0.00
I (1758) MAIN: Read Motion:0.19, 0.00, -0.06
I (1858) MAIN: Read Motion:0.36, 0.00, -0.06
I (1958) MAIN: Read Motion:0.44, 0.00, 0.06
I (2058) MAIN: Read Motion:0.47, 0.00, 0.00
I (2158) MAIN: Read Motion:0.49, 0.00, 0.00
I (2258) MAIN: Read Motion:0.50, 0.00, 0.03
I (2358) MAIN: Read Motion:0.50, 0.00, 0.00
I (2458) MAIN: Read Motion:0.50, 0.00, -0.03
I (2558) MAIN: Read Motion:0.50, 0.00, 0.00
I (2658) MAIN: Read Motion:0.50, 0.00, -0.03
I (2758) MAIN: Read Motion:0.50, 0.00, 0.00
I (2858) MAIN: Read Motion:0.50, 0.00, 0.03
I (2958) MAIN: Read Motion:0.50, 0.00, 0.03
I (3058) MAIN: Read Motion:0.50, 0.00, 0.03
I (3158) MAIN: Read Motion:0.50, 0.00, -0.03
I (3258) MAIN: Read Motion:0.51, 0.00, 0.00
I (3358) MAIN: Read Motion:0.50, 0.00, 0.00
```