Track Tracking Tutorial

1. Quick Start Guide

This tutorial explains how to use the ESP32-S3 motherboard to start track tracking on a racetrack after acquiring data from the eight-channel grayscale track tracking module, and how to modify some parameters in the code to improve the car's performance on your own racetrack.

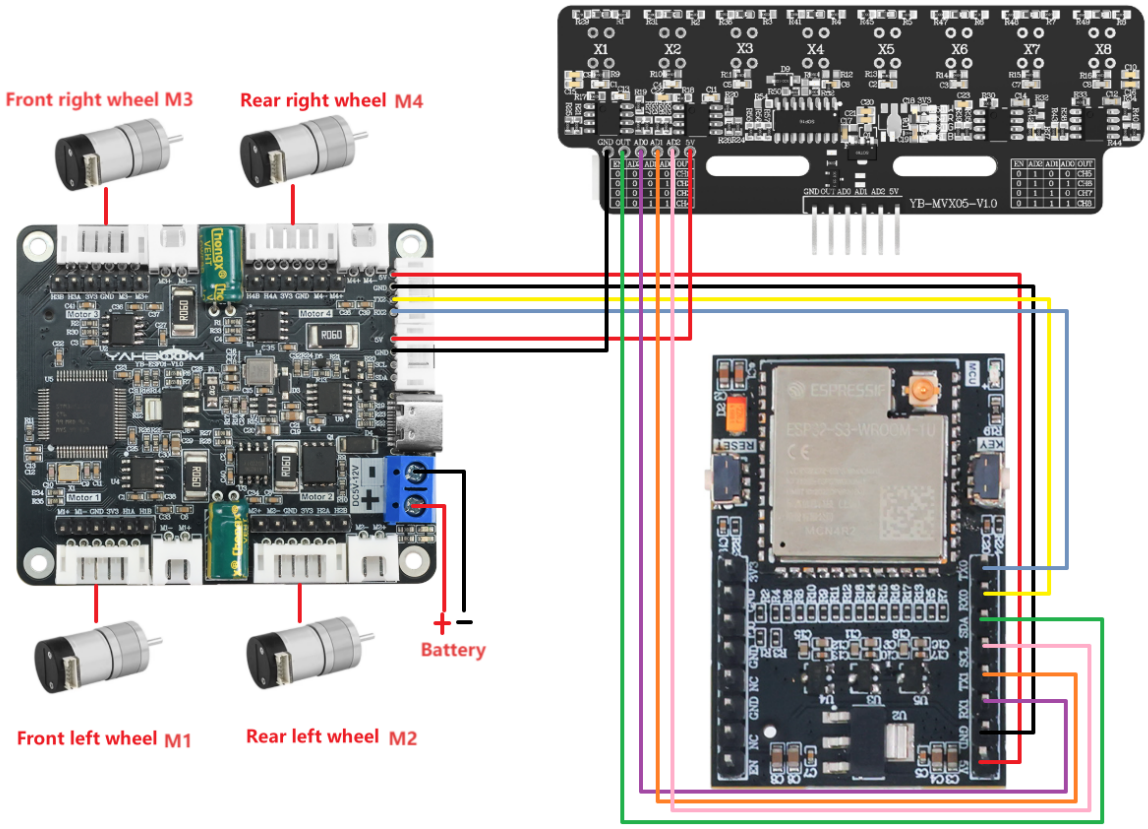
The hardware used in this tutorial consists of the following components sold by Yabo: ESP32-S3 WIFI image transmission module (lite), eight-channel grayscale track tracking module, V1 wheeled car chassis, 310 motor, four-channel motor driver board, and 7.4V battery pack. Because this code uses the programming serial port, you need to program the ESP32-S3 first, and then connect the wires. After powering on, you can start track tracking. If the results are unsatisfactory, you can skip to the code explanation section to modify the track tracking parameters and optimize the tracking effect.

In addition, appropriate motor PID parameters need to be modified. This case uses a 310 motor with PID values of P: 1.9, I: 0.2, D: 0.8. A PC serial port assistant is used to communicate with the four-channel motor driver board, sending commands to modify the PID values. If using other motors, suitable PID parameters need to be determined through testing.

This optimization process requires some experience in debugging small vehicles. Non-Yabo modules are for reference only.

2. Hardware Wiring

Since the ESP32 used in this case only has one 5V pin, the 5V from the eight-channel grayscale module needs to be connected to the four-channel motor driver board.



Eight-channel grayscale line-following module	ESP32-S3
AD0	RX1(35)
AD1	TX1(36)
AD2	SCL(37)
OUT	SDA(38)

ESP32-S3	Four-channel motor driver board
5V	5V
GND	GND
TX0(43)	RX2
RX0(44)	TX2

Eight-channel grayscale line-following module	Four-channel motor driver board
5V	5V
GND	GND

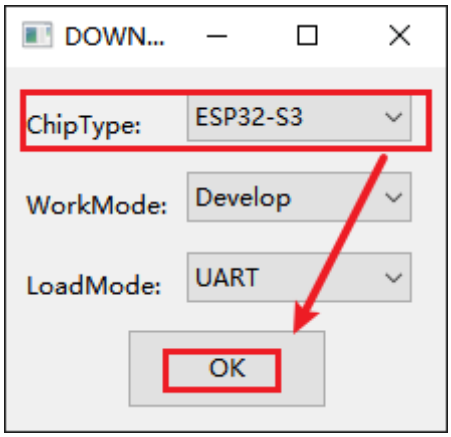
The serial port and IIC markings here are just markings on the ESP32 module board. In reality, only the pins connected to the four-channel motor driver board use the serial port function.

The eight-channel grayscale module sold by Yabo uses XH2.54 to 6-pin DuPont wire for connection:

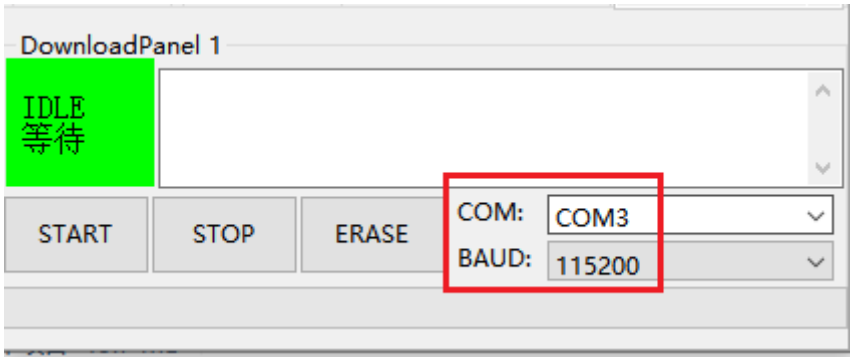
The connection method between the four-channel motor driver board sold by Yabo and the 310 motor can be found in the tutorial "Introduction and Usage of Common Motors" in the four-channel motor driver board documentation. Due to its length, it will not be elaborated here.

3. Usage Method

1. Open the flashing tool `flash_download_tool`, select the ESP32-S3 chip, and then click OK.

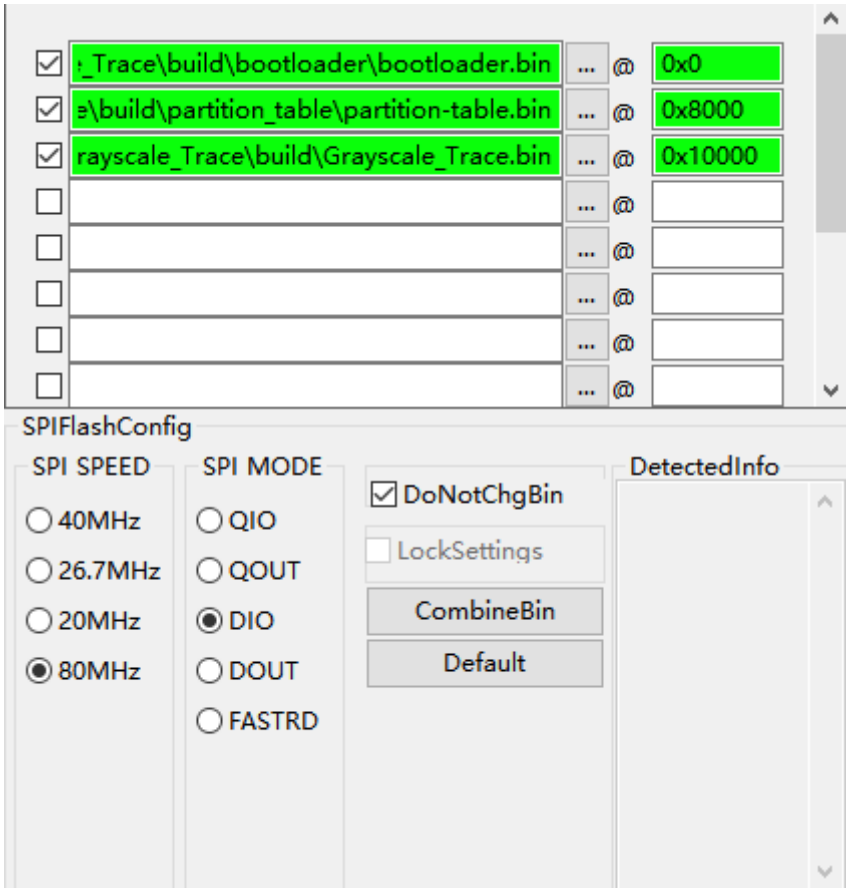


2. Connect the ESP32-S3 to the computer using a data cable, and then select the COM port and baud rate in the software.



3. In the `SPIDownload` window, select the three ESP32S3 firmware files to be burned. The path is usually in the `build` folder under the root directory of the code project. The format is a .bin file. Remember to enter the firmware address. Make sure the box to the left of ".bin file" is checked, and both the file and address are green.

Check "DoNotChgBin". Leave other configurations as default.

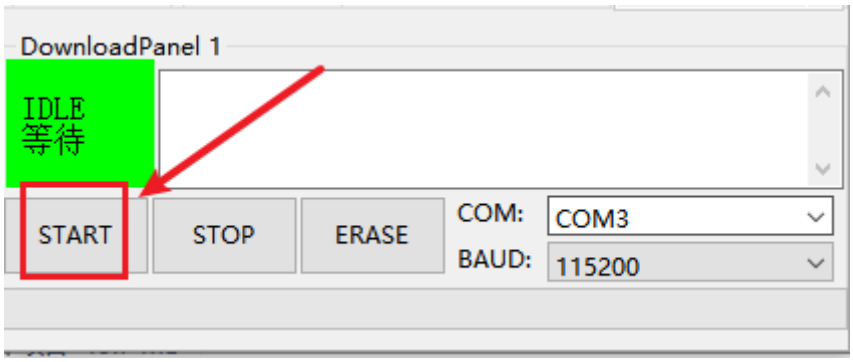


The file and address mapping is shown in the table below:

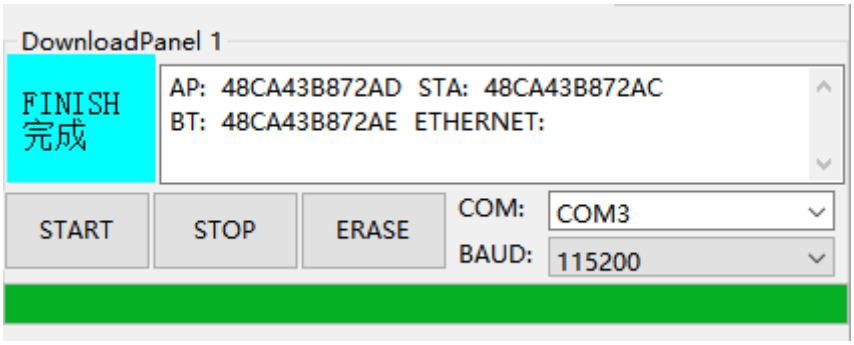
Firmware Name	Firmware Address	Remarks
bootloader.bin	0x0000	Boot file
partition-table.bin	0x8000	Partition table file
Grayscale_Trace.bin	0x10000	Function file

4. Click the START button, and the tool will automatically begin flashing the firmware.

Note: If flashing the firmware does not start automatically, please press and hold the IO0 key, then press the EN key, and release them simultaneously to manually enter flashing mode, and then click the START button.



5. After downloading, a blue FINISH indicator will appear. At this point, power off and restart the microcontroller or press the EN key to restart the program and close the programming tool. Finally, use a serial port assistant to open this port to see the printed information.



4. Phenomenon and Results

After powering on, the program starts running. If all eight LEDs on the eight-channel grayscale line-following module are either fully lit or all off, the car will not move to prevent it from running around on the wrong map. When the LEDs on the line-following module are inconsistent, the car begins line-following.

The LED lights on the patrol line need to be opposite to those on the track outside the line. In other words, if the LED lights on the patrol line are off, the corresponding LED lights on the track outside the line need to be on for the patrol to function correctly.

5. Code Explanation

```
//#########################################################
// Important Configuration Parameters
//#########################################################
// Select the corresponding motor parameter configuration according to the type
of motor you use
#define MOTOR_TYPE 2
//1: 520 motor 2: 310 motor 3: Speed code disc TT Motor 4: TT DC reduction motor
Motor 5: L type 520 motor
//#########################################################
// Place the grayscale line-following module on the track and observe the light
facing the line. If the light is on, the value is 1, LINE_RAW_VALUE=1; if the
light is off, the value is 0, LINE_RAW_VALUE=0
#define LINE_RAW_VALUE 1
//#########################################################
```

The code begins with two macro definitions: `MOTOR_TYPE` and `LINE_RAW_VALUE`.

- `MOTOR_TYPE`: Modify `MOTOR_TYPE` according to the motor you purchased. The motors adapted in the code are all currently sold by Yabo.
- `LINE_RAW_VALUE`: Place the grayscale line-following module on the track and observe the light directly facing the line. If the light is on, set `LINE_RAW_VALUE=1`; if the light is off, set the value to 0, `LINE_RAW_VALUE=0`.

```
// Grayscale_Trace.c
void line_following_init(line_following_t* controller) {
    controller->kp = 270.0f;   // Scale factor
    controller->ki = 0.5f;      // Scale factor
    controller->kd = 50.0f;    // Differential factor

    controller->base_speed = 450;   // Base speed
    controller->max_speed = 800;    // Maximum speed
```

```
    controller->sensor_weights[0] = -5.0f;
    controller->sensor_weights[1] = -4.0f;
    controller->sensor_weights[2] = -2.0f;
    controller->sensor_weights[3] = -1.0f;
    controller->sensor_weights[4] = 1.0f;
    controller->sensor_weights[5] = 2.0f;
    controller->sensor_weights[6] = 4.0f;
    controller->sensor_weights[7] = 5.0f;
    //...
}
```

`line_following_init` function

Key modifications for optimizing line following effect:

PID parameters

- `kp`: Increasing `kp` makes the response faster, but it is prone to oscillation; decreasing `kp` reduces oscillation, but the response will be slower and may deviate from the line.
- `ki`: Increasing `ki` can eliminate steady-state error faster, but it is prone to overshoot and oscillation; decreasing `ki` can reduce the risk of overshoot, but the error elimination will be slower or even impossible to completely eliminate.
- `kd`: Increasing `kd` can improve system stability and reduce oscillation and overshoot, but too much will make it sensitive to noise and cause jitter; decreasing `kd` can reduce the sensitivity to noise, but the ability to suppress oscillation is weakened, and the response may be delayed or overshoot.

  Speed Parameters
- `base_speed`: Determines the vehicle's forward speed during smooth line following, affecting overall efficiency and cornering response.
- `max_speed`: Limits the maximum rotational speed of the vehicle's left and right wheels to prevent excessive acceleration deviation.

Grayscale Line Following Module Weight Parameters

- `sensor_weights`: Modifies the weights on the grayscale line following module. The first value refers to the X1 light at the module's outermost edge, and the second value refers to the X2 light. Increasing the first value results in a more dramatic response when the X1 light is on, while decreasing it results in a smoother response.

```
// app_motor_uart.c
void Set_Motor(int MOTOR_TYPE)
{
    if(MOTOR_TYPE == 1)
    {
        send_motor_type(1);
        //...
    }
    //...
}
void Contrl_Speed(int16_t M1_speed,int16_t M2_speed,int16_t M3_speed,int16_t
M4_speed)
{
```

```
sprintf((char*)send_buff,"$spd:%d,%d,%d,%d#",M1_speed,M2_speed,M3_speed,M4_speed
);
    Send_Motor_ArrayU8(send_buff, strlen((char*)send_buff));
}
```

`app_motor_uart` (Motor Driver Module)

- `Set_Motor`: Configures the relevant parameters of the motor (type, reduction ratio, magnetic core wire, wheel diameter, dead zone) according to the preset motor type (MOTOR_TYPE).
- `Contrl_Speed`: Sends commands to the motor driver board via UART to control the speed of the four motors in closed-loop encoder control.

```
// grayscale_sensor.c
void Grayscale_Sensor_Read_All(uint16_t* sensor_values)
{
    uint8_t i;
    for (i = 0; i < GRAYSCALE_SENSOR_CHANNELS; i++)
    {
        _select_channel(i);
        _delay_us(50);
        sensor_values[i] = Read_OUT_value();
    }
}
```

`grayscale_sensor` (Grayscale sensor module)

- `Grayscale_Sensor_Read_All` : Selects all sensor channels sequentially, reads and returns a list of output values for each channel.

```
// trace_task.c
bool check_sensors_safe(line_following_t* controller, uint16_t* sensor_values) {
    uint16_t first_value = sensor_values[0];
    //...
    return false;
}

float calculate_error(line_following_t* controller, uint16_t* sensor_values,
uint16_t line_raw_value) {
    float weighted_sum = 0.0f;
    int active_sensors = 0;
    //...
    return error;
}

float pid_control(line_following_t* controller, float error) {
    if (fabsf(error) < 0.6f) {
        error = 0.0f;
    }
    //...
    return output;
}

void differential_speed_control(line_following_t* controller, float pid_output,
```

```
                                int16_t* left_speed, int16_t* right_speed) {
    //...
}

void follow_line(line_following_t* controller, uint16_t* sensor_values,
                 uint16_t line_raw_value) {
    if (controller->motor_locked) {
        if (check_sensors_safe(controller, sensor_values)) {
            controller->motor_locked = false;
        } else {
            Contrl_Speed(0, 0, 0, 0);
            return;
        }
    }

    float error = calculate_error(controller, sensor_values, line_raw_value);
    float pid_output = pid_control(controller, error);
    int16_t left_speed, right_speed;
    differential_speed_control(controller, pid_output, &left_speed,
&right_speed);

    Contrl_Speed(left_speed, left_speed, right_speed, right_speed);
}
```

`trace_task` (Line-following task module)

- `check_sensors_safe`: Checks if all grayscale sensors display the same value, used to determine if the vehicle is in a safe state at startup.
- `calculate_error`: Calculates the error value of the vehicle's deviation from the line center based on the values read by the grayscale sensors and preset weights.
- `pid_control`: Executes the PID control algorithm and calculates the control output based on the input error value.
- `differential_speed_control`: Calculates the differential speed between the left and right wheels based on the PID control output, thereby adjusting the vehicle's steering.
- `follow_line`: The main line-following function, responsible for reading sensor data, performing safety checks, calculating errors, and performing PID control and differential control to drive the motors.

```
// Grayscale_Trace.c
void app_main(void)
{
    uart0_init();
    Grayscale_Sensor_Init();
    Set_Motor(MOTOR_TYPE);
    send_motor_PID(1.9,0.2,0.8);
    line_following_init(&g_line_controller);

    xTaskCreate(Grayscale_ReadData_Task, "GrayscaleReadData", 4096, NULL, 3,
NULL);
    xTaskCreate(Trace_Task, "Trace", 4096, NULL, 4, NULL);
}
```

`main` (Main Program)

- `app_main`: Program entry point. Initializes the UART, grayscale sensor, motor parameters, and line-following controller sequentially. Two independent FreeRTOS tasks were then created: `Grayscale_ReadData_Task` for continuously reading sensor data, and `Trace_Task` for executing the line-following logic. This multi-tasking architecture allows sensor data acquisition and line-following control to be processed in parallel without blocking each other.