Line Following Car

Line Following Car

- 1. Quick Start
- 2. Hardware Wiring
- 3. Usage Method
- 4. Observations and Results
- 5. Code Explanation

1. Quick Start

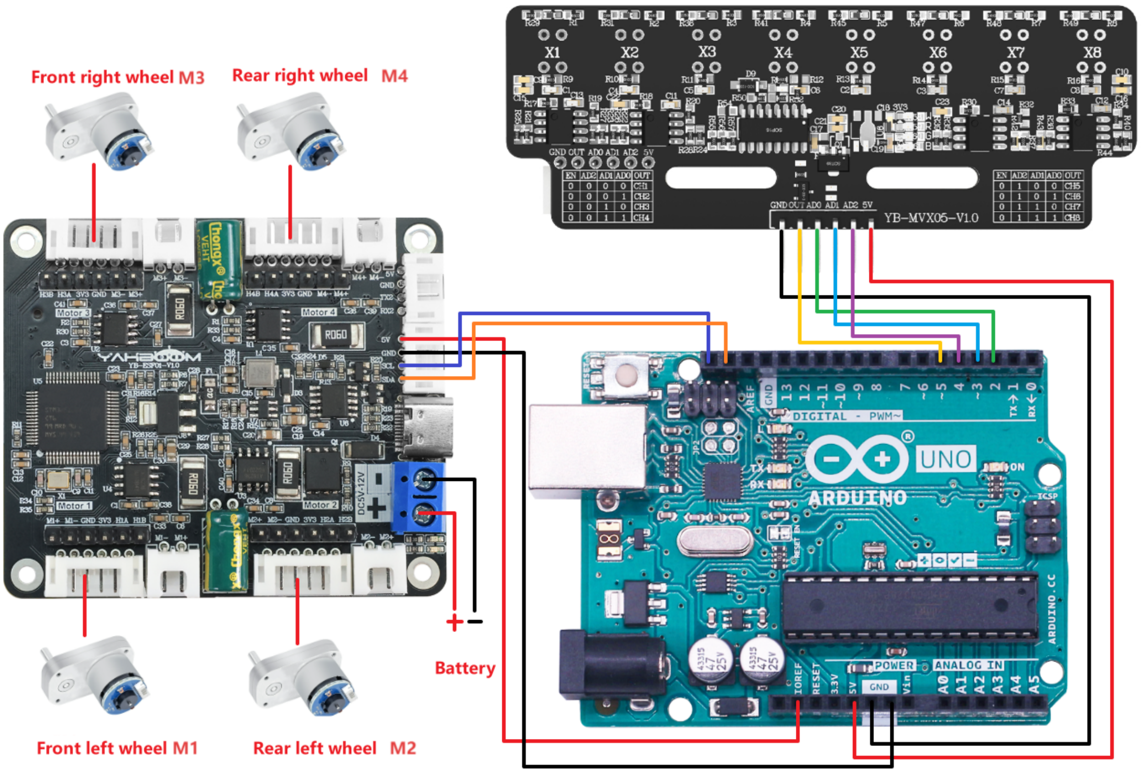
This tutorial explains how to use the Arduino UNO motherboard to obtain data from the eight-channel grayscale line following module and how to modify some parameters in the code to improve the car's performance on your own racetrack.

The hardware used in this tutorial includes an Arduino UNO, an eight-channel grayscale line following module (sold by Yabo), a V2 wheeled car chassis, an L-shaped 520 motor, a four-channel motor driver board, and a 12V battery pack. Refer to the wiring diagram below to connect the wires, then program the Arduino UNO. Power it on and start line following. If the results are unsatisfactory, you can skip to the code explanation section to modify the line following parameters and optimize the performance.

In addition, you need to modify the appropriate motor PID parameters. This example uses an L-shaped 520 motor with the following PID values: P: 1.9, I: 0.2, D: 0.8. Use a PC serial port assistant to communicate with the four-channel motor driver board and send commands to modify the PID values. If using other motors, you will need to test and determine suitable PID parameters for each motor.

This tuning process requires some experience in debugging small vehicles. Modules not compatible with Yabo are for reference only.

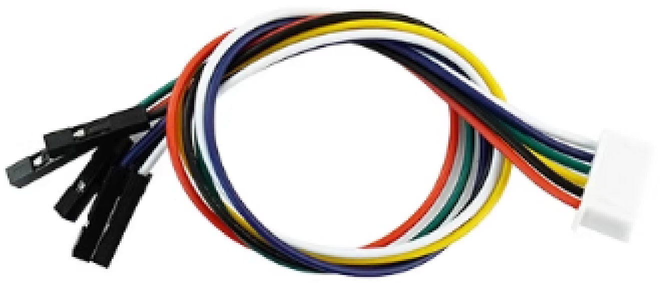
2. Hardware Wiring



Eight-channel grayscale line-following module	Arduino UNO
5V	5V
GND	GND
AD0	2
AD1	3
AD2	4
OUT	5

Arduino UNO	Four-channel motor driver board
IOREF(5V)	5V
GND	GND
SCL	SCL
SDA	SDA

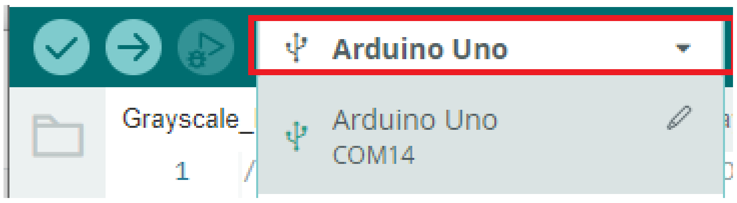
The standard cable for the eight-channel grayscale trace module sold by Yabo is an XH2.54 to 6-pin DuPont cable. One end of the XH2.54 ribbon cable should be connected to the eight-channel grayscale module, and the other end of the DuPont cable can be connected normally as shown in the picture above:



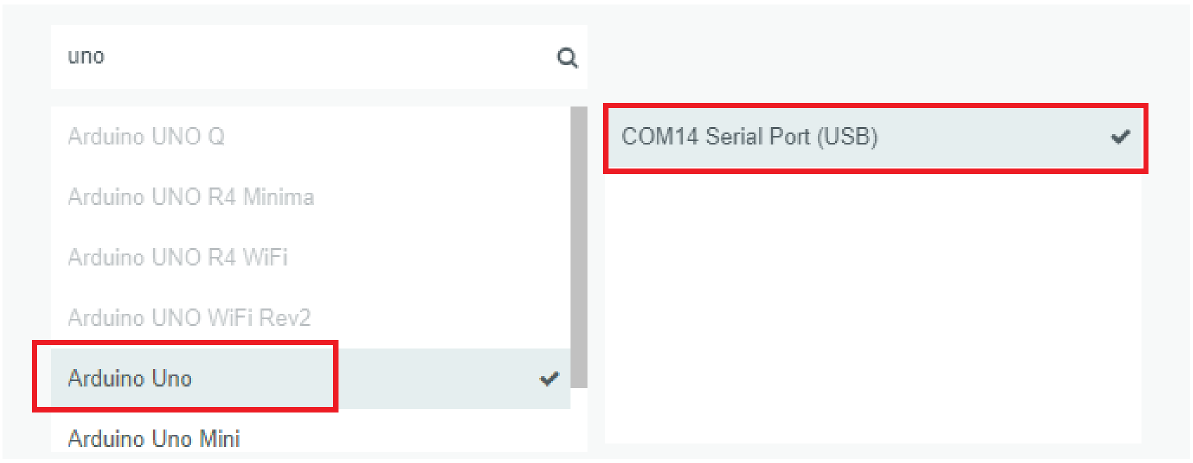
The connection method between the four-channel motor driver board sold by Yabo and the L-type 520 motor can be found in the tutorial "Introduction and Usage of Common Motors" in the four-channel motor driver board datasheet. Due to its length, it will not be elaborated here.

3. Usage Method

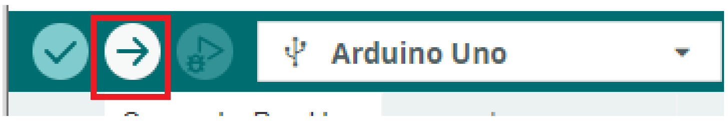
1. Open the `Grayscale_Trace.ino` file in the `Grayscale_Trace` folder using the Arduino IDE.
2. Connect the Arduino UNO to the computer using the data cable.
3. In the top left corner of the Arduino IDE menu bar, click the port selection box and then select the serial port of the Arduino you are connecting to.



4. If this is the first time connecting, there is a COM port number, but no Arduino model is specified, manually select the model.



5. Next, click the "Upload" button on the toolbar. The program will begin compiling and burning to the Arduino UNO.



6. After programming, disconnect the Arduino from the computer, place the car on the track, and connect the battery to power the car.

4. Observations and Results

After powering on, the program starts running. If all eight LEDs on the eight-channel grayscale line-following module are either all on 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 to follow the line.

The LEDs on the line-following module need to be opposite to those on the track outside the line. That is, if the LED on the line is off, the corresponding LED on the track outside the line needs to be on for normal line following.

5. Code Explanation

```
//#########################################################
// Important Configuration Parameters
//#########################################################
// Select the corresponding motor parameter configuration according to the type
of motor you use
#define MOTOR_TYPE 5
                //1:520 motor 2:310 motor 3:speed code disc TT motor 4:TT DC
reduction 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.ino
void line_following_init(line_following_t* controller) {
    controller->kp = 100.0f;    //  proportional gain
    controller->ki = 0.5f;      // integral gain
    controller->kd = 50.0f;     // derivative gain

    controller->last_error = 0.0f;  //  last error
    controller->integral = 0.0f;    //  integral accumulation

    controller->base_speed = 350;   // base speed
    controller->max_speed = 700;    //  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;

    controller->motor_locked = true;  //  lock motors on power-up
}
```

Key modifications to optimize line-following performance:

PID parameters

kp: Increasing `kp` makes the response faster, but it is prone to oscillation; decreasing `kp` reduces oscillation, but the response becomes slower and may deviate from the line.

ki: Increasing `ki` eliminates steady-state error faster, but it easily causes overshoot and oscillation; decreasing `ki` reduces the risk of overshoot, but eliminates error more slowly or even fails to completely eliminate it.

kd: Increasing `kd` improves system stability and reduces wobbling and overshoot, but excessive `kd` increases sensitivity to noise and causes jitter; decreasing `kd` reduces noise sensitivity, but weakens wobbling suppression, potentially leading to lag 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 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 is for the X2 light. Increasing the value results in a more dramatic response when the X1 light is on, while decreasing it results in a smoother response.

```
// bsp_motor_iic.cpp
void Set_Motor(int MOTOR_TYPE)
{
    if(MOTOR_TYPE == 1)
    {
        Set_motor_type(1);
        delay(100);
        Set_Pluse_Phase(30);
        delay(100);
        Set_Pluse_line(11);
        delay(100);
        Set_Wheel_dis(67.00);
        delay(100);
        Set_motor_deadzone(1900);
        delay(100);
    }
    //...
}

void control_speed(int16_t m1,int16_t m2 ,int16_t m3,int16_t m4)
{
    static uint8_t speed[8];
    speed[0] = (m1>>8)&0xff;
    speed[1] = (m1)&0xff;
    //...
    i2cWrite(Motor_model_ADDR,SPEED_Control_REG,8,speed);
}
```

- `Set_Motor`: Configures the relevant parameters of the motor (type, reduction ratio, magnetic core, wheel diameter, dead zone) according to the preset motor type (MOTOR_TYPE).
- `control_speed`: Sends commands to the motor driver board via I2C to control the speed of the four motors in a closed-loop encoder manner.

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

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

```
// trace_task.cpp
void follow_line(line_following_t* controller, uint16_t* sensor_values,
                 uint8_t motor_type, uint16_t line_raw_value)
{
    if (controller->motor_locked) {
        if (check_sensors_safe(controller, sensor_values)) {
            controller->motor_locked = false;
        } else {
            control_pwm(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);

    if (motor_type == 1 || motor_type == 2 || motor_type == 3 || motor_type ==
5) {
        control_speed(left_speed, left_speed, right_speed, right_speed);
    } else if (motor_type == 4) {
        control_pwm(left_speed, right_speed, left_speed, right_speed);
    }
}
```

- `check_sensors_safe` : Checks if all grayscale sensors display the same value, used to determine if the car is in a safe state at startup.

- `calculate_error` : Calculates the error value of the car'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 car's steering.
- `follow_line` : The main function for line following, responsible for reading sensor data, performing safety checks, calculating errors, and using PID and differential control to drive the motor.