

Parse Data Control Motor

Parse Data Control Motor

Device connection

Hardware connection

Software connection

Serial port parsing data

Control principle

Control pin

Data format (communication protocol)

Control the motor

Code Analysis

Experimental results

Parse the data in the specified data format through the serial port, and use the data to control the state of the car: Here we demonstrate the serial port control of the car state (controlling the car state is to control 4 motors). The specific control of each car state will be introduced in detail in the following tutorials!

The format of the serial port parsed data refers to the communication protocol of the WiFi camera module, and the following serial port control tutorials also refer to the communication protocol of the WiFi camera module!

Do not install the WiFi camera module here, otherwise it will occupy the serial port and abnormal information will appear.

Device connection

Hardware connection

Use Type-B data cable to connect Arduino Uno and computer.

Software connection

Open the "Arduino IDE" software and select the model and serial port number corresponding to the development board.

Serial port parsing data

Control principle

Receive data through the serial port and determine whether the data is in the specified format, parse and store the specified data, and use the stored data to control the state of the car.

Control pin

Use the serial port on the Arduino IDE to send and parse data.

Serial port	Arduino Uno
RX	0

Serial port	Arduino Uno
TX	1

Data format (communication protocol)

We only parse data in two data formats.

Data format	Description	Example
\$	Data header	
#	Data tail	
Data,Data,Data	Data type 1	Data represents only numbers: 1,0,0,0
Alphabet+Data	Data type 2	Alphabet+Data represents a letter plus the following three digits: A100

Example:

Data format 1: \$1,0,0,0#, \$1,2,0,0#, \$1,2,0,1#

Data format 2: \$A100#, \$A090#, \$B120#

Note:

- Data format 1 is mainly used to control the car status, RGB, buzzer, motor
- Data format 2 is mainly used to control the servo angle

The data needs to be sent strictly in the above format.

Control the motor

We use the first digit in data format 1 to control the car status.

Data packet	Header	Data: Control the car	Delimiter	Data: Control RGB	Delimiter	Data: Control the buzzer	Delimiter	Data	Tail
Car stops	\$	0	,	0	,	0	,	0	#
Car moves forward	\$	1	,	0	,	0	,	0	#
Car moves backward	\$	2	,	0	,	0	,	0	#
Car moves left	\$	3	,	0	,	0	,	0	#
Car moves right	\$	4	,	0	,	0	,	0	#

Data packet	Header	Data: Control the car	Delimiter	Data: Control RGB	Delimiter	Data: Control the buzzer	Delimiter	Data	Tail
Car rotates left	\$	5	,	0	,	0	,	0	#
Car rotates right	\$	6	,	0	,	0	,	0	#
Car stops	\$	Other	,	0	,	0	,	0	#

Data packet	Header	Servo number	Data: Control servo angle	Tail
Control servo to rotate 45 degrees	\$	A	045	#
Control servo to rotate 90 degrees	\$	A	090	#
Control servo to rotate 135 degrees	\$	A	135	#

Note: Due to the limited rotation angle of the camera, the control servo rotation range is 35°~145°. It will not rotate beyond this range and keep the boundary value.

Example:

Serial port controls the car to move forward: \$1,0,0,0#

Currently, the code can only control the car state, and other states cannot be controlled. New control functions will be gradually added in the following tutorials!

Code Analysis

Here we only briefly introduce the code content. For detailed code, please refer to the corresponding code file, which is provided in the download area!

- Include `wire`、`Adafruit_PWMServoDriver` library

```
#include <Wire.h> // 包含Wire(I2C)通讯库 Include wire library
#include <Adafruit_PWMServoDriver.h> // 包含Adafruit PWM servo driver库 Include Adafruit PWM servo driver library
```

- Define data such as arrays, flag bits, etc

```
// 定义数据，如数组，标志位等 Define data such as arrays, flag bits, etc
#define BUFFER_SIZE 20
volatile uint8_t Rx_Index = 0;
volatile uint8_t Data_Flag = 0;
char* Data_Array[BUFFER_SIZE];
```

```

char Data_Servo[BUFFER_SIZE];

int DataArraySize = 0;

// 定义电机控制引脚 Define motor control pins
#define Motor_L1_F_PIN 11 // 控制小车左前方电机前进 Control the motor on the left front of the car
#define Motor_L1_B_PIN 10 // 控制小车左前方电机后退 Control the motor back on the left front of the car
#define Motor_L2_F_PIN 8 // 控制小车左后方电机前进 Control car left rear motor forward
#define Motor_L2_B_PIN 9 // 控制小车左后方电机后退 Control the car left rear motor back
#define Motor_R1_F_PIN 13 // 控制小车右前方电机前进 Control the right front motor of the car to move forward
#define Motor_R1_B_PIN 12 // 控制小车右前方电机后退 Control the motor back on the right front of the car
#define Motor_R2_F_PIN 14 // 控制小车右后方电机前进 Control car right rear motor forward
#define Motor_R2_B_PIN 15 // 控制小车右后方电机后退 Control car right rear motor back

// 定义底层驱动芯片参数 Bottom-layer driver chip related parameters
#define Bottom_Layer_Driver_ADDR 0x40

// 定义PWM频率 Define PWM frequency
#define PWM_FREQUENCY 50

// 定义电机速度和小车状态 Define motor speed and trolley status
unsigned int uMotorSpeed = 50;
unsigned int uCurCarState = 0;
unsigned int uPreCarState = 0;

```

- Enumerate the common movement modes of omnidirectional cars

```

// 枚举全向小车的常见运动方式 Enumerate the common movement modes of omnidirectional cars
enum OmnidirectionalCar {
    STOP,
    FORWARD,
    BACKWARD,
    LEFT,
    RIGHT,
    LEFT_ROTATE,
    RIGHT_ROTATE,
};

```

- Create an instance of the Adafruit_PWMServoDriver class

```

// 创建Adafruit_PWMServoDriver类的实例 Create an instance of the Adafruit_PWMServoDriver class
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(Bottom_Layer_Driver_ADDR);

```

- Setting the Motor Speed

```

/**
 * @brief 设置单个电机速度 Setting the Motor Speed
 * @param motor_forward_pin: 控制电机前进引脚 Control the motor forward pin
 * @param motor_backward_pin: 控制电机后退引脚 Control the motor backward pin
 * @param motor_speed: 设置电机速度 Setting the Motor Speed
 * @retval 无 None
 */
void setMotorSpeed(uint16_t motor_forward_pin, uint16_t motor_backward_pin, int
motor_speed) {
    motor_speed = map(motor_speed, -255, 255, -4095, 4095);
    if (motor_speed >= 0) {
        pwm.setPWM(motor_forward_pin, 0, motor_speed);
        pwm.setPWM(motor_backward_pin, 0, 0);
    } else if (motor_speed < 0) {
        pwm.setPWM(motor_forward_pin, 0, 0);
        pwm.setPWM(motor_backward_pin, 0, -(motor_speed));
    }
}
}

```

- Set the car movement mode and speed

```

/**
 * @brief 设置小车运动方式和速度 Set the car movement mode and speed
 * @param Movement: 小车运动方式 Car movement
 * @param Speed: 小车运动速度 Car speed
 * @retval 无 None
 */
void setCarMove(uint8_t Movement, int Speed) {
    switch (Movement) {
        case STOP:
            Serial.println("Car State:STOP");
            setMotorSpeed(Motor_L1_F_PIN, Motor_L1_B_PIN, 0);
            setMotorSpeed(Motor_L2_F_PIN, Motor_L2_B_PIN, 0);
            setMotorSpeed(Motor_R1_F_PIN, Motor_R1_B_PIN, 0);
            setMotorSpeed(Motor_R2_F_PIN, Motor_R2_B_PIN, 0);
            break;
        case FORWARD:
            Serial.println("Car State:FORWARD");
            setMotorSpeed(Motor_L1_F_PIN, Motor_L1_B_PIN, Speed);
            setMotorSpeed(Motor_L2_F_PIN, Motor_L2_B_PIN, Speed);
            setMotorSpeed(Motor_R1_F_PIN, Motor_R1_B_PIN, Speed);
            setMotorSpeed(Motor_R2_F_PIN, Motor_R2_B_PIN, Speed);
            break;
        case BACKWARD:
            Serial.println("Car State:BACKWARD");
            setMotorSpeed(Motor_L1_F_PIN, Motor_L1_B_PIN, -Speed);
            setMotorSpeed(Motor_L2_F_PIN, Motor_L2_B_PIN, -Speed);
            setMotorSpeed(Motor_R1_F_PIN, Motor_R1_B_PIN, -Speed);
            setMotorSpeed(Motor_R2_F_PIN, Motor_R2_B_PIN, -Speed);
            break;
        case LEFT:
            Serial.println("Car State:LEFT");
            setMotorSpeed(Motor_L1_F_PIN, Motor_L1_B_PIN, -Speed);
            setMotorSpeed(Motor_L2_F_PIN, Motor_L2_B_PIN, Speed);

```

```

        setMotorSpeed(Motor_R1_F_PIN, Motor_R1_B_PIN, Speed);
        setMotorSpeed(Motor_R2_F_PIN, Motor_R2_B_PIN, -Speed);
        break;
    case RIGHT:
        Serial.println("Car State:RIGHT");
        setMotorSpeed(Motor_L1_F_PIN, Motor_L1_B_PIN, Speed);
        setMotorSpeed(Motor_L2_F_PIN, Motor_L2_B_PIN, -Speed);
        setMotorSpeed(Motor_R1_F_PIN, Motor_R1_B_PIN, -Speed);
        setMotorSpeed(Motor_R2_F_PIN, Motor_R2_B_PIN, Speed);
        break;
    case LEFT_ROTATE:
        Serial.println("Car State:LEFT_ROTATE");
        setMotorSpeed(Motor_L1_F_PIN, Motor_L1_B_PIN, -Speed);
        setMotorSpeed(Motor_L2_F_PIN, Motor_L2_B_PIN, -Speed);
        setMotorSpeed(Motor_R1_F_PIN, Motor_R1_B_PIN, Speed);
        setMotorSpeed(Motor_R2_F_PIN, Motor_R2_B_PIN, Speed);
        break;
    case RIGHT_ROTATE:
        Serial.println("Car State:RIGHT_ROTATE");
        setMotorSpeed(Motor_L1_F_PIN, Motor_L1_B_PIN, Speed);
        setMotorSpeed(Motor_L2_F_PIN, Motor_L2_B_PIN, Speed);
        setMotorSpeed(Motor_R1_F_PIN, Motor_R1_B_PIN, -Speed);
        setMotorSpeed(Motor_R2_F_PIN, Motor_R2_B_PIN, -Speed);
        break;
    default:
        Serial.println("Car State:STOP");
        setMotorSpeed(Motor_L1_F_PIN, Motor_L1_B_PIN, 0);
        setMotorSpeed(Motor_L2_F_PIN, Motor_L2_B_PIN, 0);
        setMotorSpeed(Motor_R1_F_PIN, Motor_R1_B_PIN, 0);
        setMotorSpeed(Motor_R2_F_PIN, Motor_R2_B_PIN, 0);
        break;
    }
}

```

- Receive data

Note: Receive data between \$ and #

```

/**
 * @brief 接收指定格式的数据  Receives data in the specified format
 * @param 无 None
 * @retval 接收指定数据 Received specified data
 */
char* recData() {
    static char Data_Buffer[BUFFER_SIZE];
    static char Rx_Buffer[BUFFER_SIZE];

    if (Serial.available() > 0) {
        char SerialData = Serial.read();
        if (SerialData == '$') {
            memset(Rx_Buffer, 0, sizeof(Rx_Buffer));
            memset(Data_Buffer, 0, sizeof(Data_Buffer));

            Data_Flag = 1;
            Rx_Index = 0;
        } else if (Data_Flag == 1 && SerialData == '#') {

```

```

    if (Rx_Index < BUFFER_SIZE) {
        memcpy(Data_Buffer, Rx_Buffer, Rx_Index);
        memset(Rx_Buffer, 0, sizeof(Rx_Buffer));
        Rx_Index = 0;
        Data_Flag = 0;
        // Serial.println(Data_Buffer);
        return Data_Buffer;
    } else {
        // Serial.println("Rx_Buffer is full!");
        memset(Rx_Buffer, 0, sizeof(Rx_Buffer));
        Rx_Index = 0;
        Data_Flag = 0;
    }
} else {
    if (Rx_Index < BUFFER_SIZE) {
        Rx_Buffer[Rx_Index++] = SerialData;
    } else {
        // Serial.println("Rx_Buffer is full!");
        memset(Rx_Buffer, 0, sizeof(Rx_Buffer));
        Rx_Index = 0;
        Data_Flag = 0;
    }
}
}
return nullptr;
}

```

- Separate the data into a new array

```

/**
 * @brief 将数据分离到一个新的数组中 Separate the data into a new array
 * @param dataBuffer: 要分离的数据 The data to be separated
 * @param dataArray: 分离后的数据 The data after separation
 * @param dataArraySize: 分离数据的大小 The size of the separated data
 * @retval 无 None
 */
void splitDataAndSave(char* dataBuffer, char** dataArray, int& dataArraySize) {
    char* token = strtok(dataBuffer, ",");
    int index = 0;
    while (token != NULL) {
        dataArray[index] = token;
        token = strtok(NULL, ",");
        index++;
    }
    dataArraySize = index;
}

```

- Parse the serial port data after processing

Dump format 2 data and print format 1/2 data

```

/**
 * @brief 处理后解析串口数据 Parse the serial port data after processing
 * @param dataBuffer: 分离后的数据 The data after separation

```

```

* @retval 无 None
*/
void parseSerialData(char* dataBuffer) {
    if (dataBuffer != nullptr) {
        // Serial.println(dataBuffer);
        if (isalpha(dataBuffer[0])) {
            memcpy(Data_Servo, dataBuffer + 1, 3);
            Serial.print("Servo angle:");
            Serial.println(strtol(Data_Servo, NULL, 10));
        } else {
            splitDataAndSave(dataBuffer, Data_Array, DataArraySize);
            // Serial.print("Car state:");
            // Serial.println(strtol(Data_Array[0], NULL, 10));
            uCurCarState = strtol(Data_Array[0], NULL, 10); // Control the car status
            // Serial.print("RGB state:");
            // Serial.println(strtol(Data_Array[1], NULL, 10));
            // Serial.print("Buzzer state:");
            // Serial.println(strtol(Data_Array[2], NULL, 10));
            // Serial.print("No define:");
            // Serial.println(strtol(Data_Array[3], NULL, 10));
        }
        memset(Data_Servo, 0, sizeof(Data_Servo));
        memset(Data_Array, 0, sizeof(Data_Array));
    }
}

```

- Initialization Code

```

void setup() {
    Serial.begin(115200);           // 初始化串口波特率115200 Initialize serial
communication at 115200 bps
    wire.begin();                  // 初始化I2C通讯 Initialize I2C communication
    delay(1000);                   // 如果小车功能异常，可以增加这个延时 If the function
is abnormal, you can increase the delay
    pwm.begin();                   // PWM初始化 Initialize the Pulse Width
Modulation (PWM) library
    pwm.setPWMFreq(PWM_FREQUENCY); // 设置PWM频率 Set the PWM frequency
    setCarMove(STOP, 0);           // 设置小车停止状态 Set the car to stop state
}

```

- Looping code

```

void loop() {
    parseSerialData(recData()); // 处理后解析串口数据 Parse the serial port data
after processing
    // 打印小车状态和控制小车 Print the car status and control the car
    if (uCurCarState != uPreCarState) {
        setCarMove(uCurCarState, uMotorSpeed);
        uPreCarState = uCurCarState;
    }
}

```

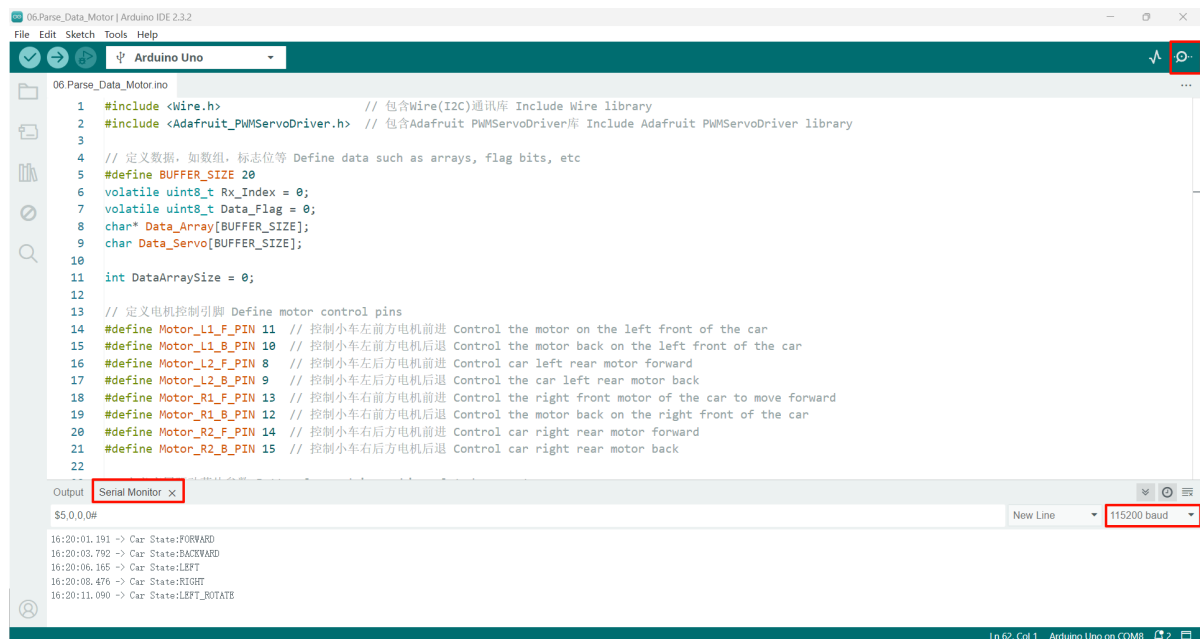

Experimental results

After compiling the program successfully, upload the code to the Arduino Uno development board.

After the program is started, send data in the specified format, and the serial port will return the parsed data and the car status!

If there is no display content, you can check whether the serial port baud rate is consistent with the code setting, and then press the RESET button on the development board.

The drive motor needs an external battery pack and the expansion board switch is turned on to drive normally.
The burning program cannot use other programs to occupy the serial port or an external serial communication module (for example: WiFi camera module), otherwise the program cannot be burned or an error message will be prompted!



The screenshot shows the Arduino IDE interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. The toolbar shows icons for compiling, uploading, and opening the serial monitor. The code editor displays the following code:

```
1 #include <Wire.h> // 包含Wire(I2C)通讯库 Include Wire library
2 #include <Adafruit_PWMServoDriver.h> // 包含Adafruit PWM Servo Driver库 Include Adafruit PWM Servo Driver library
3
4 // 定义数据, 如数组, 标志位等 Define data such as arrays, flag bits, etc
5 #define BUFFER_SIZE 20
6 volatile uint8_t Rx_Index = 0;
7 volatile uint8_t Data_Flag = 0;
8 char* Data_Array[BUFFER_SIZE];
9 char Data_Servo[BUFFER_SIZE];
10
11 int DataArraySize = 0;
12
13 // 定义电机控制引脚 Define motor control pins
14 #define Motor_L1_F_PIN 11 // 控制小车左前方电机前进 Control the motor on the left front of the car
15 #define Motor_L1_B_PIN 10 // 控制小车左前方电机后退 Control the motor back on the left front of the car
16 #define Motor_L2_F_PIN 8 // 控制小车左后方电机前进 Control car left rear motor forward
17 #define Motor_L2_B_PIN 9 // 控制小车左后方电机后退 Control the car left rear motor back
18 #define Motor_R1_F_PIN 13 // 控制小车右前方电机前进 Control the right front motor of the car to move forward
19 #define Motor_R1_B_PIN 12 // 控制小车右前方电机后退 Control the motor back on the right front of the car
20 #define Motor_R2_F_PIN 14 // 控制小车右后方电机前进 Control car right rear motor forward
21 #define Motor_R2_B_PIN 15 // 控制小车右后方电机后退 Control car right rear motor back
22
```

The Serial Monitor window is open, showing the output:

```
$5.0,0.04
16:20:01.191 -> Car State:FORWARD
16:20:03.792 -> Car State:BACKWARD
16:20:06.165 -> Car State:LEFT
16:20:08.476 -> Car State:RIGHT
16:20:11.090 -> Car State:LEFT_ROTATE
```

The baud rate is set to 115200 baud.