Basic Routine (STM32 Version)

1. Working Principle

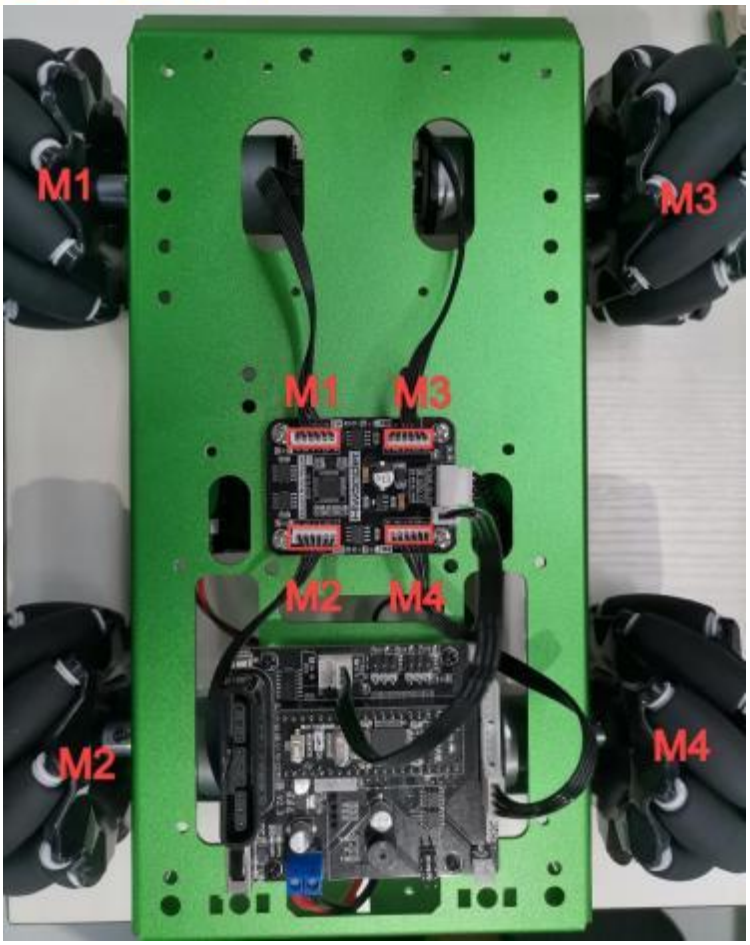
In this lesson, let's learn how to change the direction of movement with the button. When the button is pressed each time, the value increases by 1. Therefore, the direction of movement can be set according to different values. The following list shows the correspondence between the direction and the number of presses.

The number of presses	1	2	3	4	5
Movement direction	Go forward and backward	Turn	Move forward, backward, left, and right	Move diagonally	Drift

2. Getting Ready

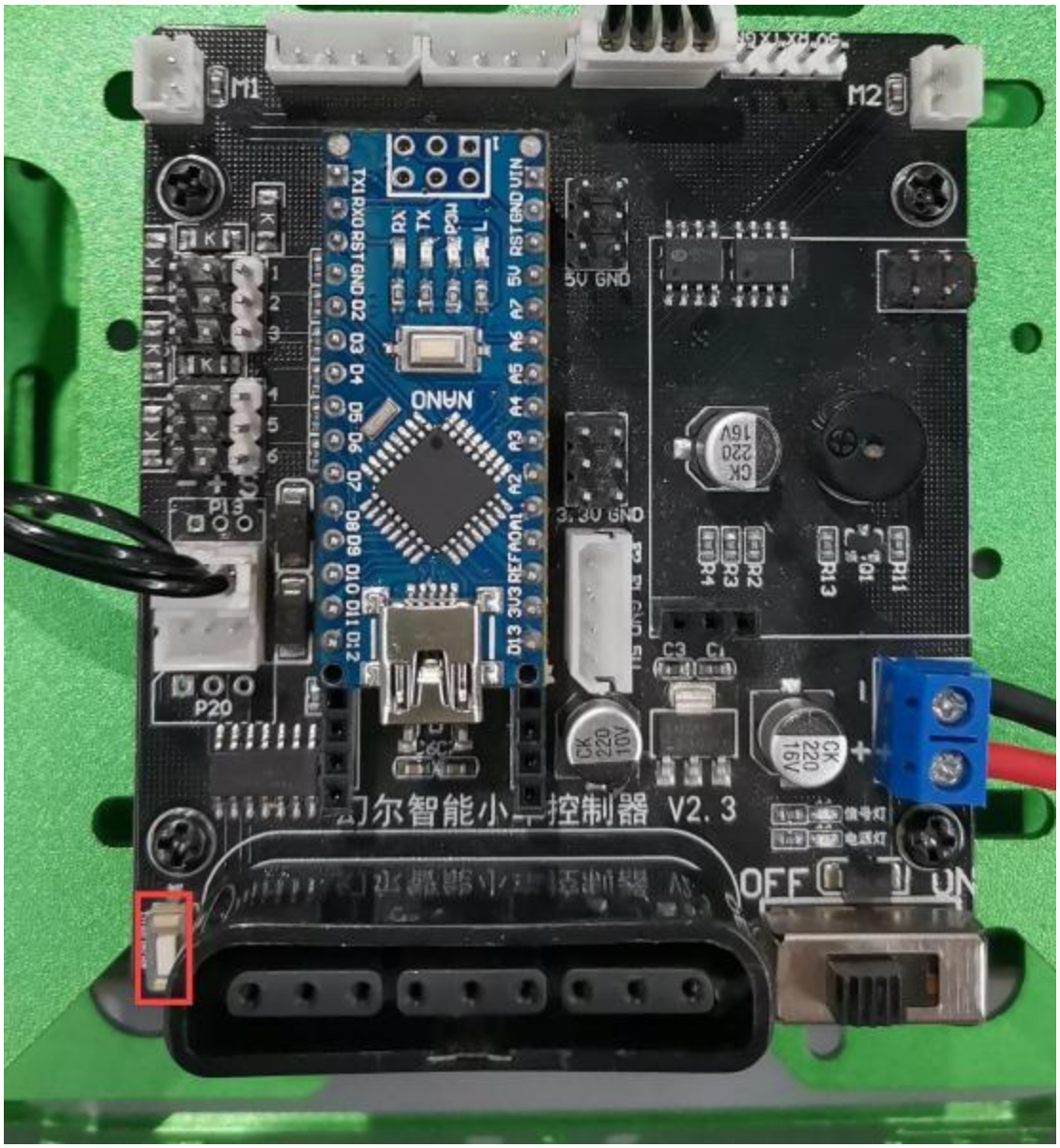
Before starting, please refer to the following figure to connect the encoder motors to the M1, M2, M3, and M4 interfaces on the encoder motor driver module.

Confirm the voltage of the motors you are using (8V or 12V).



Select the corresponding program for the voltage of your motors, depending on whether you are using the STM32 or Arduino as the controller. The program directory is labeled accordingly. The following image is an example of Arduino. Please refer to “2.Software/2.2 STM32” to install and debug the Keil tool.

The position of the button is as below:



3. Program Download

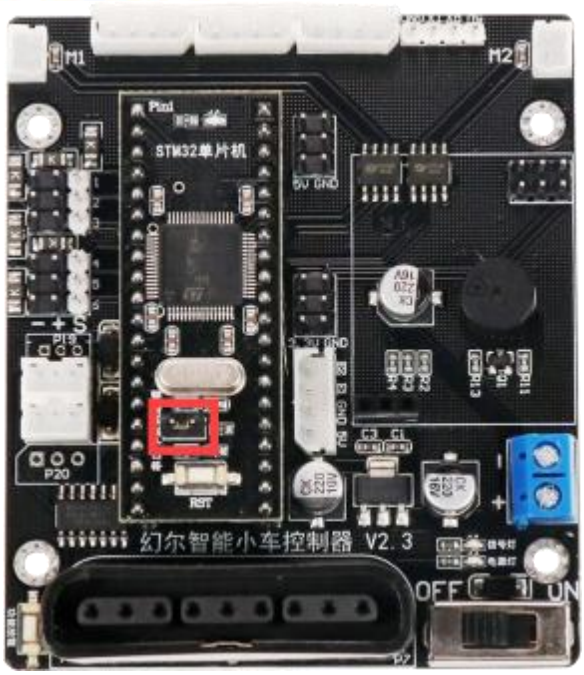
The program employs the 8V motors for demonstration. It is applicable to 12V motors.

3.1 STM32 Program Download

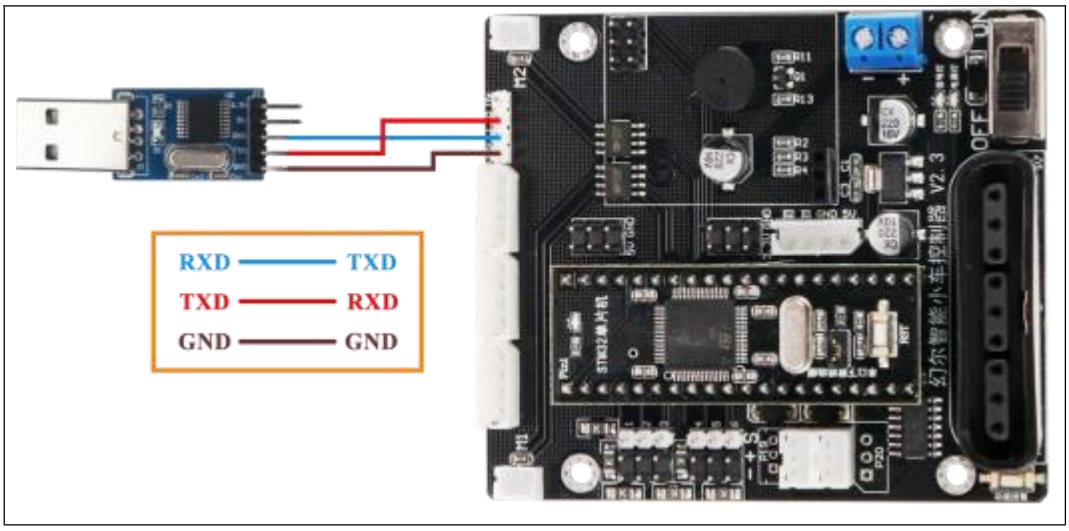
Follow the following steps to flash the program to the Mecanum wheel chassis controller:

1) Remove the jumper cap on the controller to start the flashing.

Please remove the jumper cap when the power is off.



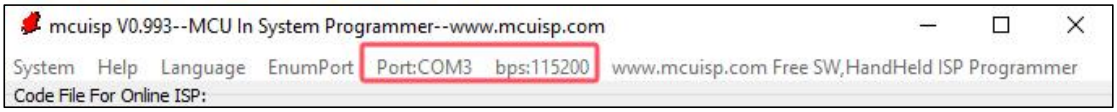
2) Connect the USB downloader to the controller with a DuPont wire. Then, connect the USB downloader to the computer.



3) Switch the controller on. Start the Mecanum wheel chassis.

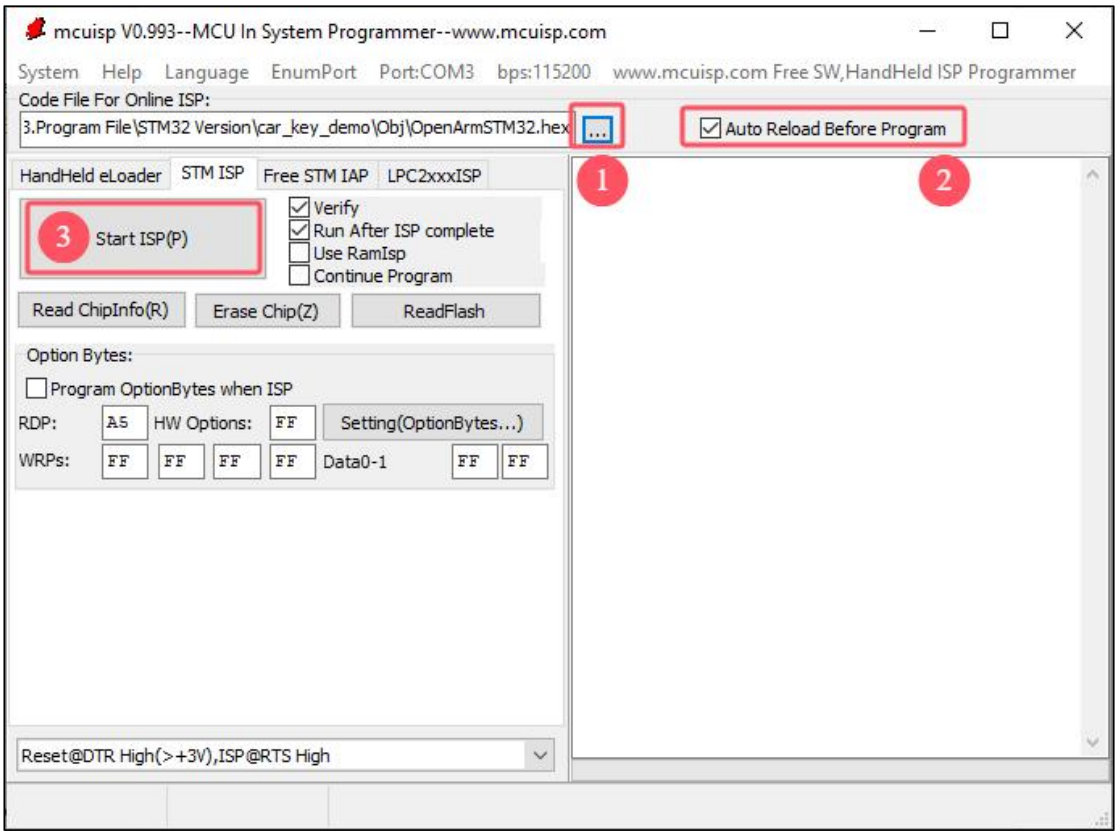
4) Open the program download tool  located in the “2.Software”.

5) Select the serial port and baud rate 115200 to write the program to the development board.



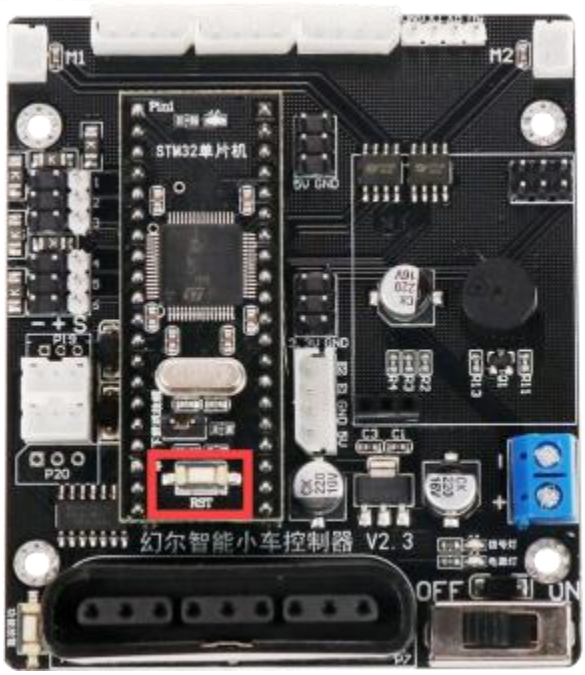
Note: If you need to know how to query the controller port, click "Port:" to view.

6) Click ... to select the program file to be flashed. After selecting, check "Reload file before programming". Click "Start ISP".



7) The program file “OpenArmSTM32.hex” is located in “3.Program File/ /STM32 Version/car_key_demo/Obj”.

8) After the program is flashed, reconnect the jumper cap to the controller. Press the "RST" button on the controller. Restart the Mecanum wheel chassis.



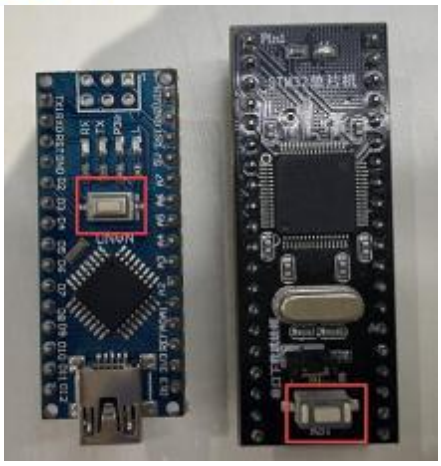
4. Program Outcome

After the program is downloaded, turn the controller on. Then, click the function button on the controller to switch different game.

The following list shows the correspondence between the direction and the number of presses.

The number of presses	1	2	3	4	5
Movement direction	Go forward and backward	Turn	Move forward, backward, left, and right	Move diagonally	Drift

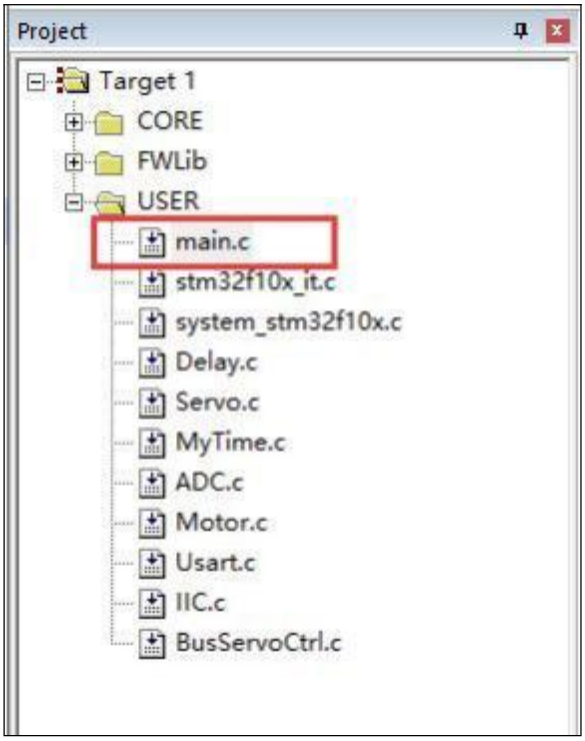
If you need to run the program again, press the reset button on the microcontroller. The reset buttons for Arduino Nano and STM32 are shown in the following image:



5. Code Analyze

Let's analyze the program for the 8V motor. The program for the 12V motor is mainly different in the motor setting section. Click to jump to the motor position setting for details. Additionally, since the analysis approach for the STM32 and Arduino programs is similar, you can check the code images based on your own needs.

The source code for the STM32 program is located in "3.Program Files/STM32 Version/car_key_demo/Obj/OpenArmSTM32.hex". The code analysis mainly focuses on the "main.c".



5.1 Define address for data transmission

The program uses I2C communication to send data to the encoder motor.

The relevant code for Arduino is shown below:

```
#include "include.h"

#define CAM_DEFAULT_I2C_ADDRESS       (0x34)      //I2C address
#define MOTOR_TYPE_ADDR               20         //Encoder motor type setting register address
#define MOTOR_FIXED_SPEED_ADDR        51         //Speed register address; belongs to closed-loop control
#define MOTOR_ENCODER_POLARITY_ADDR   21         //Motor encoder direction polarity address
#define MOTOR_FIXED_PWM_ADDR          31         //Fixed PWM control address, belongs to open-loop control
#define MOTOR_ENCODER_TOTAL_ADDR      60         //Total pulse value of each of the 4 encoding motors
#define ADC_BAT_ADDR                  0          //Voltage address
```

The relevant code for STM32 is shown below:

```
#define CAM_DEFAULT_I2C_ADDRESS       (0x34)
#define MOTOR_TYPE_ADDR               20
#define MOTOR_FIXED_SPEED_ADDR        51
#define MOTOR_ENCODER_POLARITY_ADDR   21
#define MOTOR_FIXED_PWM_ADDR          31
#define MOTOR_ENCODER_TOTAL_ADDR      60
#define ADC_BAT_ADDR                  0
```

5.2 Define motor type

Define the motor type. You can choose the appropriate motor type based on your own development needs. The encoder motor is used in this program. Therefore, the “MOTOR_TYPE_JGB” type will be selected.

The relevant code for Arduino is shown below:

```
//motor type specific address
#define MOTOR_TYPE_WITHOUT_ENCODER    0    //Motor without encoder, 44 pulses per magnetic ring rotation, reduction ratio: 90, default
#define MOTOR_TYPE_TT                 1    //TT encoder motor
#define MOTOR_TYPE_N20                2    //N20 encoder motor
#define MOTOR_TYPE_JGB                3    //44 pulses per magnetic ring rotation, reduction ratio: 90, default
```

The relevant code for STM32 is shown below:

```
//motor type specific address
#define MOTOR_TYPE_WITHOUT_ENCODER    0
#define MOTOR_TYPE_TT                 1
#define MOTOR_TYPE_N20                2
#define MOTOR_TYPE_JGB                3
```

5.3 Send data

Arduino sends data to the motor driver module via the “ WireWriteDataArray()” function. The relevant code is shown below:

```
38 bool WireWriteDataArray( uint8_t reg,uint8_t *val,unsigned int len)
39 {
40     unsigned int i;
41
42     Wire.beginTransmission(I2C_ADDR);
43     Wire.write(reg);
44     for(i = 0; i < len; i++) {
45         Wire.write(val[i]);
46     }
47     if( Wire.endTransmission() != 0 ) {
48         return false;
```

STM32 sends data to the motor driver module via the “int8_ I2C_Write_Len()” function.

The relevant code is shown below:

```
55  int8_t I2C_Write_Len(int8_t Reg,int8_t *Buf,int8_t Len)
56 {
57   uint8_t i;
58   IIC_start();
59   IIC_send_byte((CAM_DEFAULT_I2C_ADDRESS << 1) | 0);
60   if(IIC_wait_ack() == 1)
61   {
62     IIC_stop();
63     return 1;
64   }
65   IIC_send_byte(Reg);
66   if(IIC_wait_ack() == 1)
67   {
68     IIC_stop();
69     return 1;
70   }
71   for(i =0;i<Len;i++)
72   {
73     IIC_send_byte(Buf[i]);
74     if(IIC_wait_ack() == 1)
75     {
76       IIC_stop();
77       return 1;
78     }
79   }
80   IIC_stop();
81   return 0;
```

The parameter meanings of these two functions are shown below (the STM32 version is in parentheses):

The first parameter "uint8_t reg" ("int8_t Reg") represents the location where the data is sent;

The second parameter "uint8_t *val" ("uint8_t *Buf") represents the data information to be sent;

The third parameter "unsigned int len" ("int8_t Len") represents the data length.

5.4 Initialization

Initialize the port and motor.

The relevant code for Arduino is shown below:

```
uint8_t MotorType = MOTOR_TYPE_JGB;
uint8_t MotorEncoderPolarity = 0;
void setup()
{
  Wire.begin();
  Serial.begin(9600);
  printf_begin();
  delay(200);
  WireWriteDataArray(MOTOR_TYPE_ADDR,&MotorType,1);
  delay(5);
  WireWriteDataArray(MOTOR_ENCODER_POLARITY_ADDR,&MotorEncoderPolarity,1);
```

"WireWriteDataArray(MOTOR_TYPE_ADDR,&MotorType, 1)" is the motor type.

"WireWriteDataArray(MOTOR_ENCODER_POLARITY_ADDR,&MotorEncoderPolarity, 1)" is the motor polarity. The "uint8_t MotorEncoderPolarity = 0" is defined, which means that the polarity is set to 0.

Note: Do not set the polarity to 1. When the polarity is set to 1, all motors will rotate clockwise by default. Subsequent parameter settings will be invalid.

The relevant code for STM32 is shown below:

```
  SystemInit();
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
  InitDelay(72);
  InitTimer2();
  IIC_init();
  Usart1_Init();
  InitLED();
  DelayMs(200);
  I2C_Write_Len(MOTOR_TYPE_ADDR,&MotorType,4);
  DelayMs(5);
  I2C_Write_Len(MOTOR_ENCODER_POLARITY_ADDR,&MotorEncoderPolarity,1);
  DelayMs(5);
```

"I2C_Write_Len(MOTOR_TYPE_ADDR,&MotorType,4)" is the motor type.

"I2C_Write_Len(MOTOR_ENCODER_POLARITY_ADDR,&MotorEncoderPolarity,1)" is the motor polarity. The "int8_t MotorEncoderPolarity = 0" is defined, which means that the polarity is set to 0.

Note: Do not set the polarity to 1. When the polarity is set to 1, all motors will rotate clockwise by default. Subsequent parameter settings will be invalid.

5.5 Button Detection

The button detection program for Arduino is as below:

The button state is detected. Different functions will be triggered based on the accumulated number of button presses.

Define variables for the number of presses and set the initial value to 0.

```
129 int key_num = 0;
```

Pull-up pin to change the voltage to determine whether the button is pressed.

```
103 void setup()
104 {
105   pinMode(A7, INPUT_PULLUP);
106   Serial.println("Init key Is OK!");
```

Use “if()” statement to determine whether the button is pressed. If the button is pressed, increment the counter variable by 1. If the counter variable is greater than 4, set it to 0.

```
131 void loop()
132 {
133   while(1)
134   {
135    int key = analogRead(A7);
136    if (key == 0)
137    {
138     delay(10);
139     if (key == 0)
140     {
141       key_num++;
142       run = 1;
143       delay(500);
144       if(key_num > 5)
145       {
146          key_num = 0;
147        }
148      }
149    }
```

If the number of button presses is different, it will trigger different game.

```
159     case 0:
160     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p0, 4);
161     break;
162
163     case 1:           //forward and backward
164     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p1, 4);
165     DelayMs(1800);
166     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p2, 4);
167     DelayMs(1800);
168     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p0, 4);
169     DelayMs(1800);
170     break;
171
172     case 2:         //turn
173     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p5, 4);
174     DelayMs(1800);
175     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p6, 4);
176     DelayMs(1800);
177     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p0, 4);
178     break;
179
180     case 3:     //forward, backward, left, and right
181     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p1, 4);
182     DelayMs(1800);
183     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p2, 4);
184     DelayMs(1800);
185     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p3, 4);
186     DelayMs(1800);
187     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p4, 4);
188     DelayMs(1800);
189     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p0, 4);
190     break;
```

The button detection program for STM32 is as below:

Define variables for the number of presses and set the initial value to 0.

```
132   static u8 key_num=0 ;
```

Change the voltage to determine whether the button is pressed.

```
1  #ifndef _ADC_H_
2  #define _ADC_H_
3
4  #define ADC_BAT   13    //AD detection channel for battery voltage
5  #define BUZZER    PCout(9)
6  #define LED PCout(15)
7  #define KEY PCin(0)   //button
8
9  void InitADC(void);
10 void InitBuzzer(void);
11 void InitLED(void);
12 void InitKey(void);
13
14 void CheckBatteryVoltage(void);
15 u16 GetBatteryVoltage(void);
16 void Buzzer(void);
17 uint16 GetADCResult(BYTE ch);
18
19
20 #endif
```

Use “if()” statement to determine whether the button is pressed. If the button is pressed, increment the counter variable by 1. If the counter variable is greater than 4, set it to 0.

```
146     while(gerKey())
147     {
148     key_num++;
149     if(key_num > 5)
150       {
151     key_num = 0;
152       }
153     LED=0;
154     DelayMs(1000);
155     LED=1;
156     TaskTimeHandle();   //ADC detection
157     switch(key_num)
158   {
159     case 0:
160     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
161     break;
```

If the number of button presses is different, it will trigger different game.

```
159     case 0:
160     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
161     break;
162
163     case 1:           //forward and backward
164     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,4);
165     DelayMs(1800);
166     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p2,4);
167     DelayMs(1800);
168     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
169     DelayMs(1800);
170     break;
171
172     case 2:         //turn
173     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p5,4);
174     DelayMs(1800);
175     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p6,4);
176     DelayMs(1800);
177     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
178     break;
179
180     case 3:     //forward, backward, left, and right
181     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,4);
182     DelayMs(1800);
183     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p2,4);
184     DelayMs(1800);
185     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p3,4);
186     DelayMs(1800);
187     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p4,4);
188     DelayMs(1800);
189     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
190     break;
```

5.6 Call function

```
116 int8_t p0[4]={0,0,0,0};
117 int8_t p1[4]={-50,50,50,-50};
118 int8_t p2[4]={50,-50,-50,50};
119 int8_t p3[4]={50,50,50,50};
120 int8_t p4[4]={-50,-50,-50,-50};
121 int8_t p5[4]={50,-50,50,-50};
122 int8_t p6[4]={-50,50,-50,50};
123 int8_t p7[4]={0,50,0,50};
124 int8_t p8[4]={0,50,50,0};
125 int8_t p9[4]={-50,0,0,-50};
126 int8_t p10[4]={0,-50,-50,0};
127 int8_t p11[4]={50,0,0,50};
128 int32_t EncodeTotal[4];
129 int key_num = 0;
130 int run = 0;
131 void loop()
```

The “WireWriteDataArray()” function is used to send data to control the movement of the tank chassis. A speed value is set for the motor, using "-50, -50" as an example.

Set a speed value to the motor. Take “int8_t p1[4]={-50,50,50,-50}” as an example. As shown above:

“p1” represents the speed data to be sent. “-50”, “50”, “50” and “-50” respectively represent the speed values of M1 to M4 motors. When the speed value is positive, the motors rotates clockwise; when the speed value is negative, the motor rotates counterclockwise. If the encoder motor is 12V, the motor rotates clockwise. When the value is 0, the motor stops rotating.

The “WireWriteDataArray()” function sends data to control the rotation of the Mecanum chassis. Take “WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,p1,4)” in case 1 as an example.

```
163     case 1:           //forward and backward
164     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p1, 4);
165     DelayMs(1800);
166     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p2, 4);
167     DelayMs(1800);
168     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p0, 4);
169     DelayMs(1800);
170     break;
```

The first parameter "MOTOR_FIXED_SPEED_ADDR" represents that the data will be sent to the encoder motor driver;

The second parameter "p1" represents the speed value to be sent, p1=(-50，50，50，-50), which means that M1 and M4 motors rotate counterclockwise at a speed of 50; M2 and M3 motors rotate clockwise at a speed of 50.

If all motor speed values are 0, the car stops moving.

STM32 uses the “I2C_Write_Len()” function to send data to control the rotation of the Mecanum chassis. Take “I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,4)” in case 1 as an example.

```
163     case 1:           //forward and backward
164     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p1, 4);
165     DelayMs(1800);
166     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p2, 4);
167     DelayMs(1800);
168     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p0, 4);
169     DelayMs(1800);
170     break;
171
172     case 2:         //turn
173     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p5, 4);
174     DelayMs(1800);
175     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p6, 4);
176     DelayMs(1800);
177     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p0, 4);
178     break;
179
180     case 3:     //forward, backward, left, and right
181     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p1, 4);
182     DelayMs(1800);
183     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p2, 4);
184     DelayMs(1800);
185     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p3, 4);
186     DelayMs(1800);
187     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p4, 4);
188     DelayMs(1800);
189     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p0, 4);
190     break;
191
192     case 4:   //move diagonally
193     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p8, 4);
194     DelayMs(1800);
195     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p9, 4);
196     DelayMs(1800);
197     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p10, 4);
198     DelayMs(1800);
199     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p11, 4);
200     DelayMs(1800);
201     I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR, p0, 4);
202     break;
```

The first parameter "MOTOR_FIXED_SPEED_ADDR" represents that the data will be sent to the encoder motor driver;

The second parameter "p1" represents the speed value to be sent, p1=(-50，50，50，-50), which means that M1 and M4 motors rotate counterclockwise at a speed of 50; M2 and M3 motors rotate clockwise at a speed of 50.

The third parameter "4" represents the data length.

```
int8_t p0[4]={0, 0, 0, 0};                          //stop
int8_t p1[4]={-50, 50, 50, -50};                    //forward
int8_t p2[4]={50, -50, -50, 50};                    //backward
int8_t p3[4]={50, 50, 50, 50};                      //move to the left
int8_t p4[4]={-50, -50, -50, -50};                  //move to the right
int8_t p5[4]={50, -50, 50, -50};                    //turn right
int8_t p6[4]={-50, 50, -50, 50};                    //turn left
int8_t p7[4]={0, 50, 0, 50};                        //drift left
int8_t p8[4]={0, 50, 50, 0};                        //left front
int8_t p9[4]={-50, 0, 0, -50};                      //right front
int8_t p10[4]={0, -50, -50, 0};                     //right rear
int8_t p11[4]={50, 0, 0, 50};                       //left rear
```

Set a speed value to the motor. Take “int8_t p1[4]={-50,50,50,-50} ” as an example.

“p1” represents the speed data to be sent. “-50”, “50”, “50” and “-50” respectively represent the speed values of M1 to M4 motors. When the speed value is positive, the motors rotates clockwise; when the speed value is negative, the motor rotates counterclockwise. If the encoder motor is 12V, the motor rotates clockwise. When the value is 0, the motor stops rotating.