Basic Routine (STM32 Version)

1. Working Principle

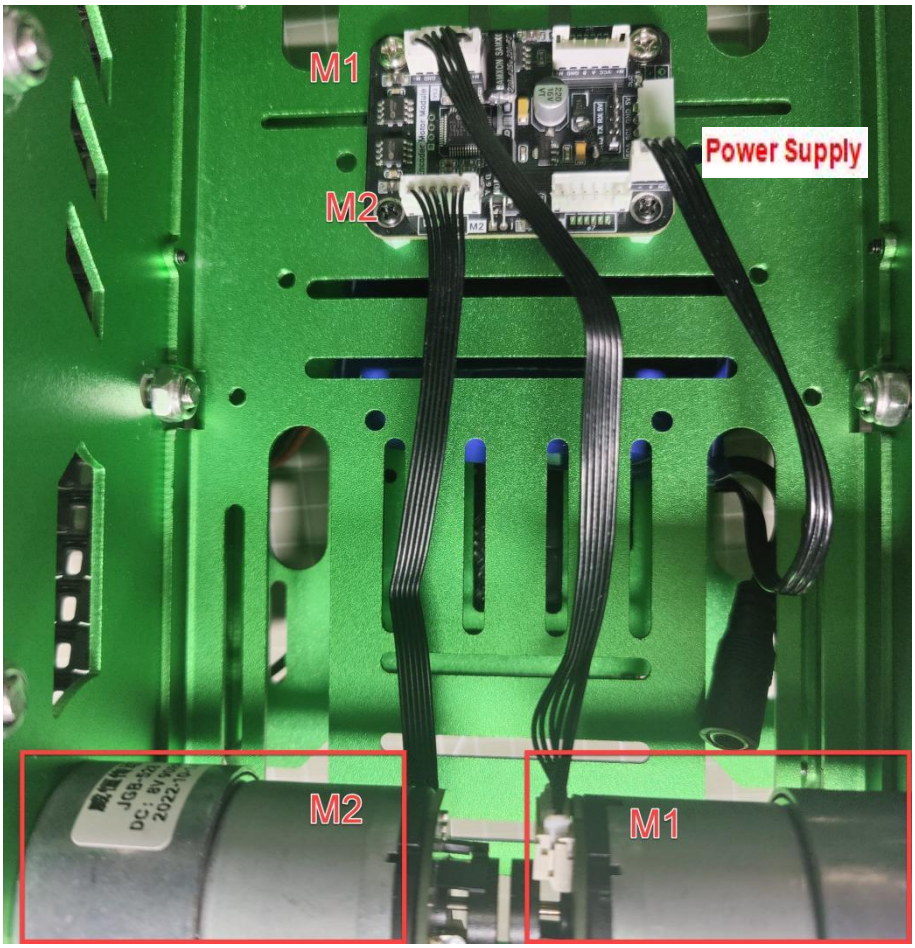
The tank car chassis is powered by the encoder motor. When the car is moving, the tracks will generate a opposite force due to the friction to drive the car. The rotation of the tank car can be controlled by setting the motor values. For example, When the value of a motor is positive, and the other one is negative, and they are a group of opposite number, the tank car will rotate. The range of motor value is between -100 and 100.

Then, you can 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.

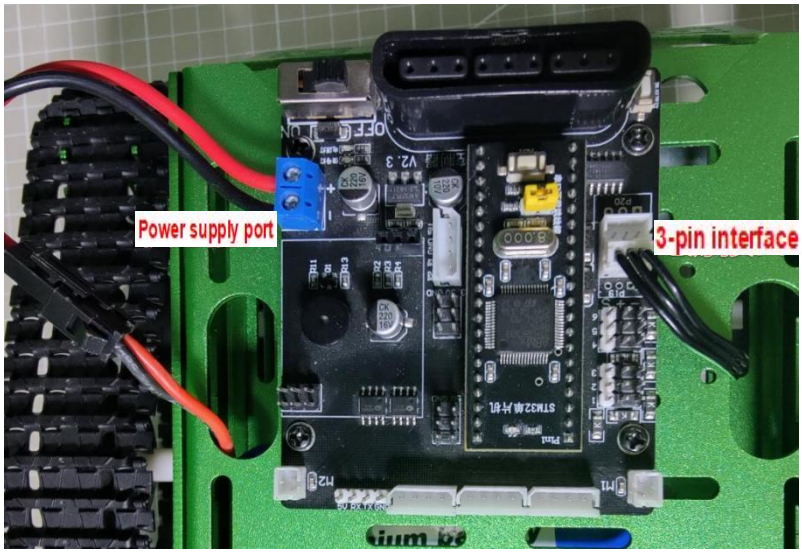
The number of presses	1	2	3	4	5
Movement direction	Go forward	Go backward	Turn left	Turn right	Stop

2. Getting Ready

1) Connect the encoder motors to M1 and M2 interfaces on the motor driver module. Then, connect one end of a 3-pin wire to its power interface. The wiring method refers to the following image.



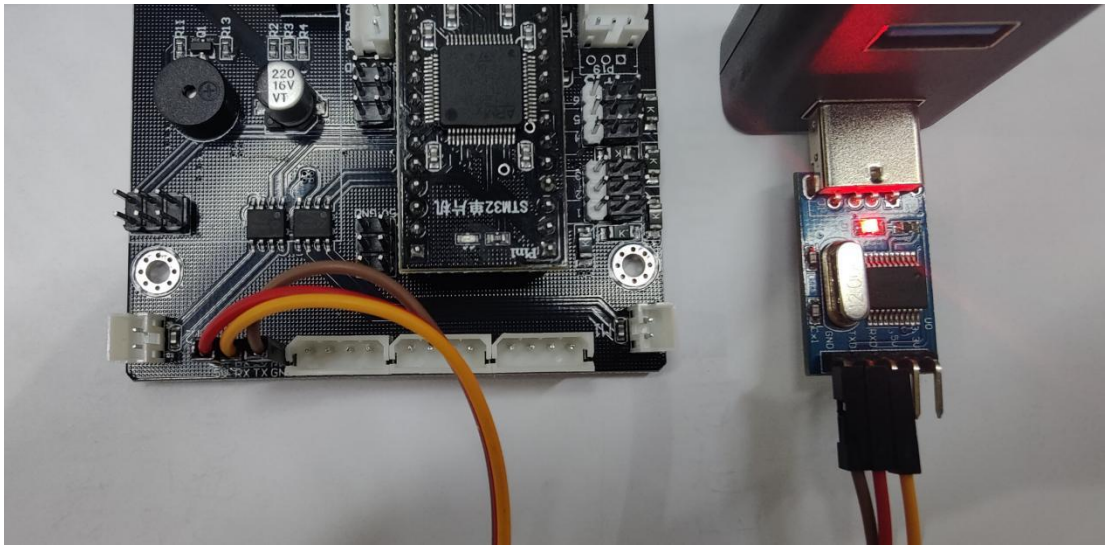
2) Connect the other end of the 3-pin wire to the power interface on the controller. The power supply port should be connected to the battery. Please refer to the relevant tutorial to connect the battery. The completed wiring is shown below:



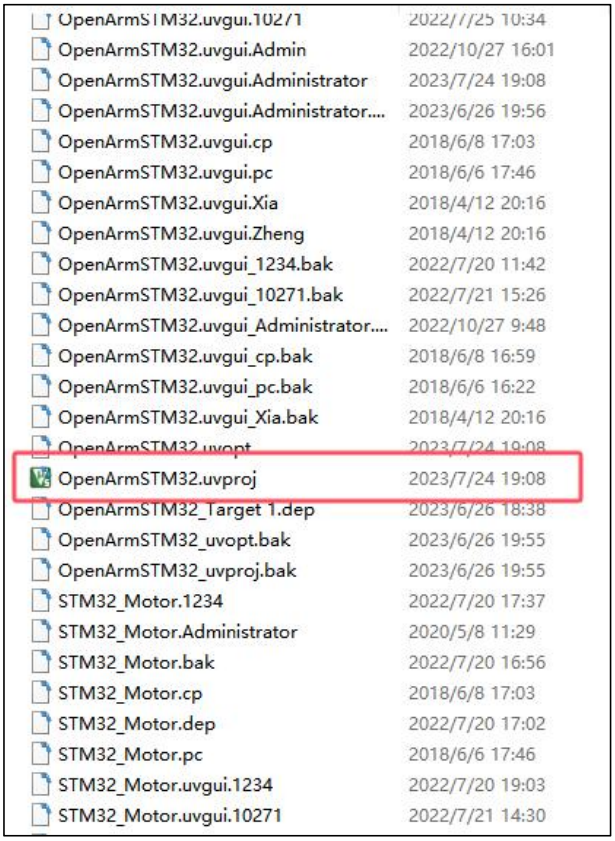
3) Please refer to “2.Software/2.2 STM32” to install and debug the Keil tool.

3. Program Download

1) Connect controller to computer with USB to TTL converter. The GND, TX and RS on controller are connected to the GND, RX and TX on TTL converter.

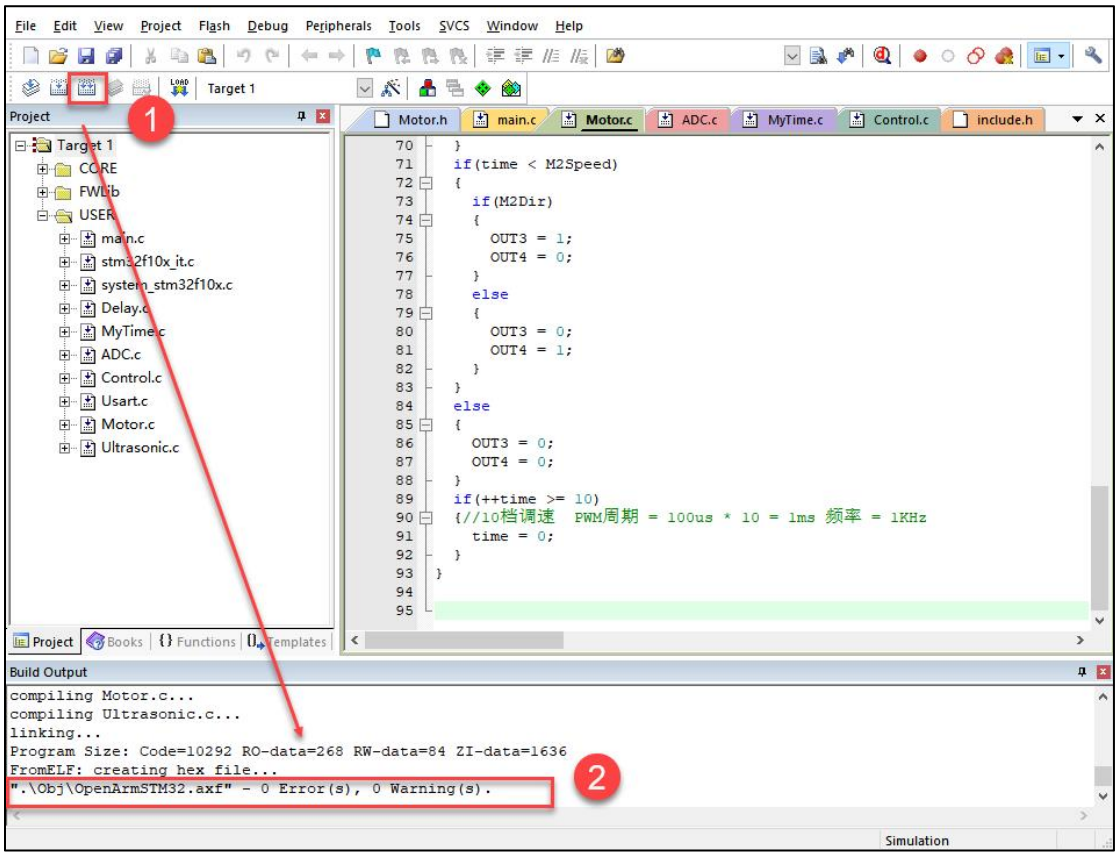


2) Double-click to open STM32 example program in “3.Program File/STM32 Version”.

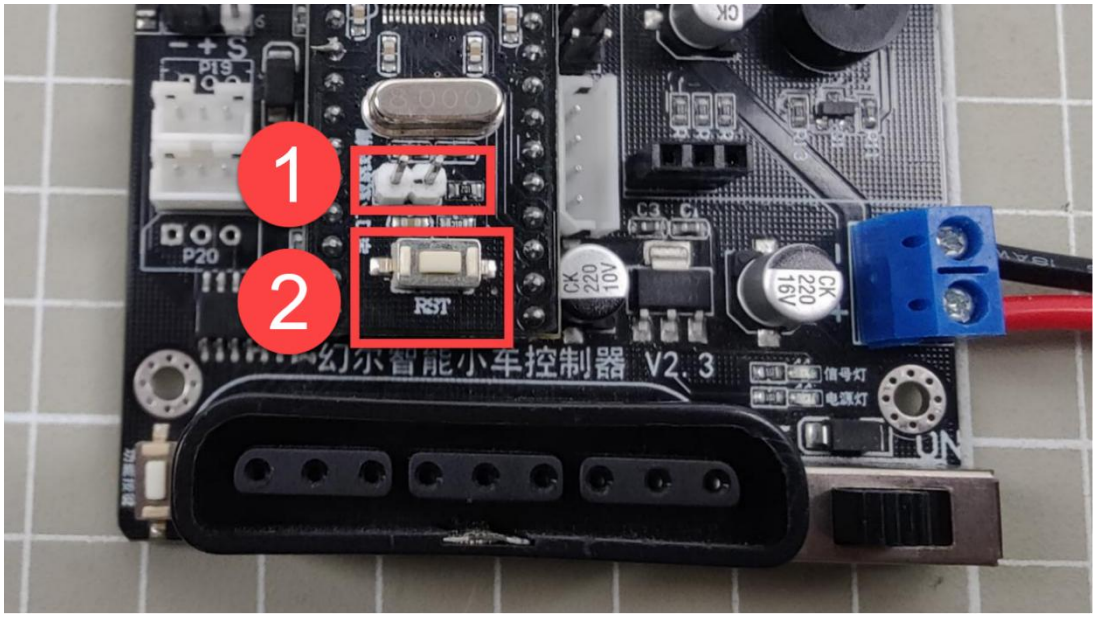


3) After opening, click the "Compile" button to generate an executable file with

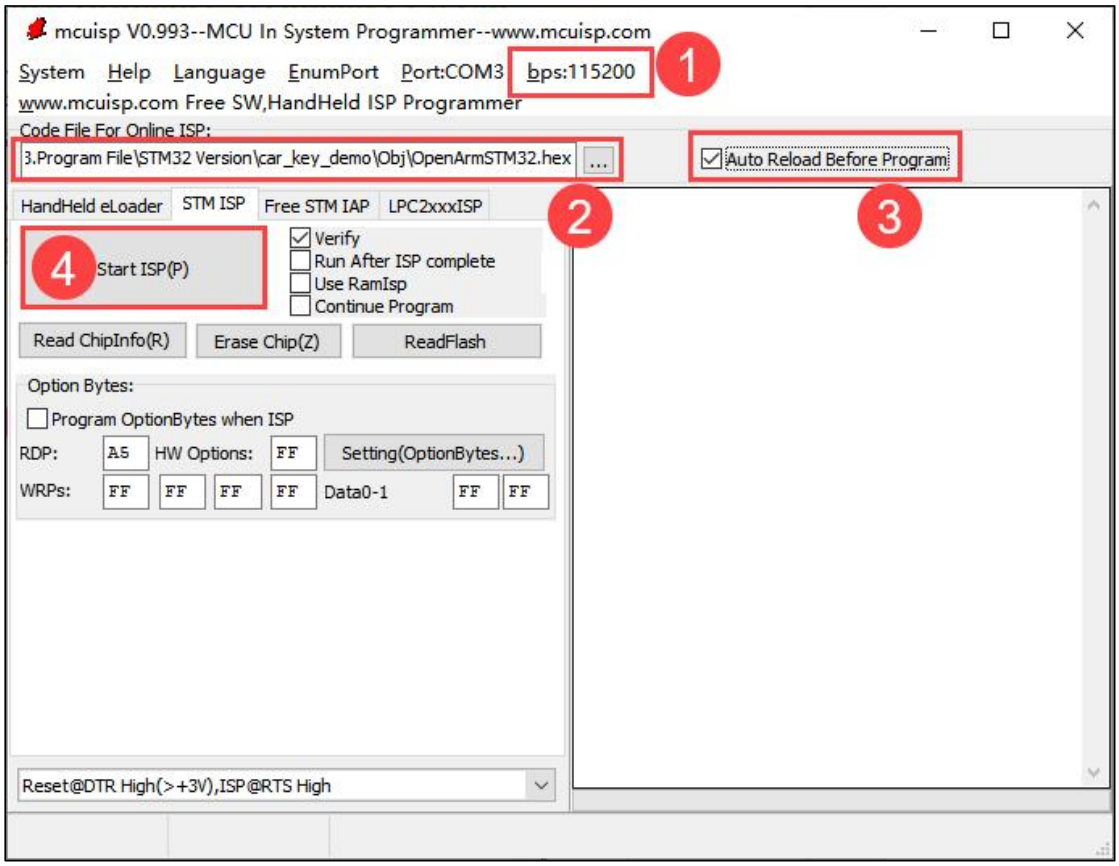
all the code.

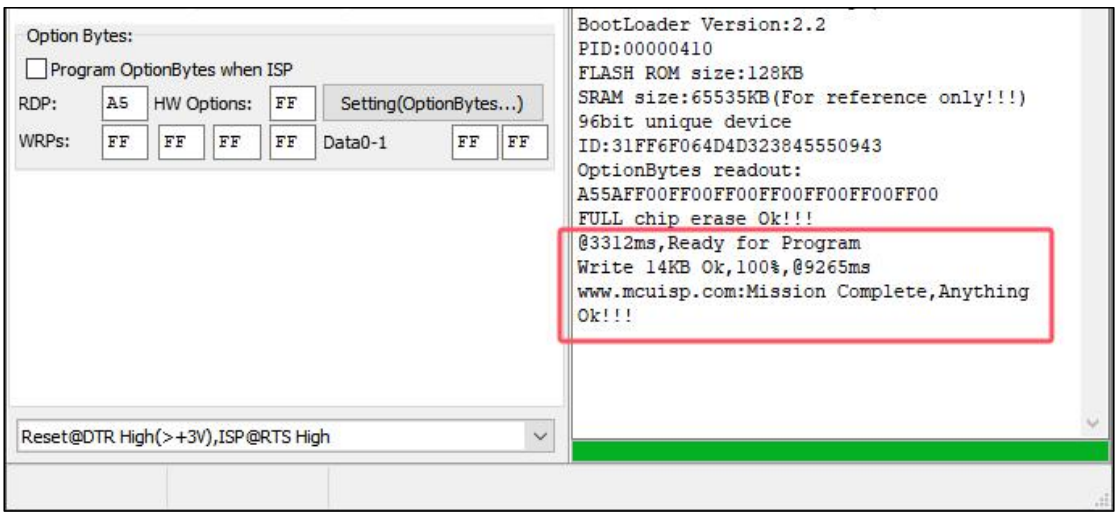


4) Remove the jumper cap on STM32 development board, and press RST reset button.

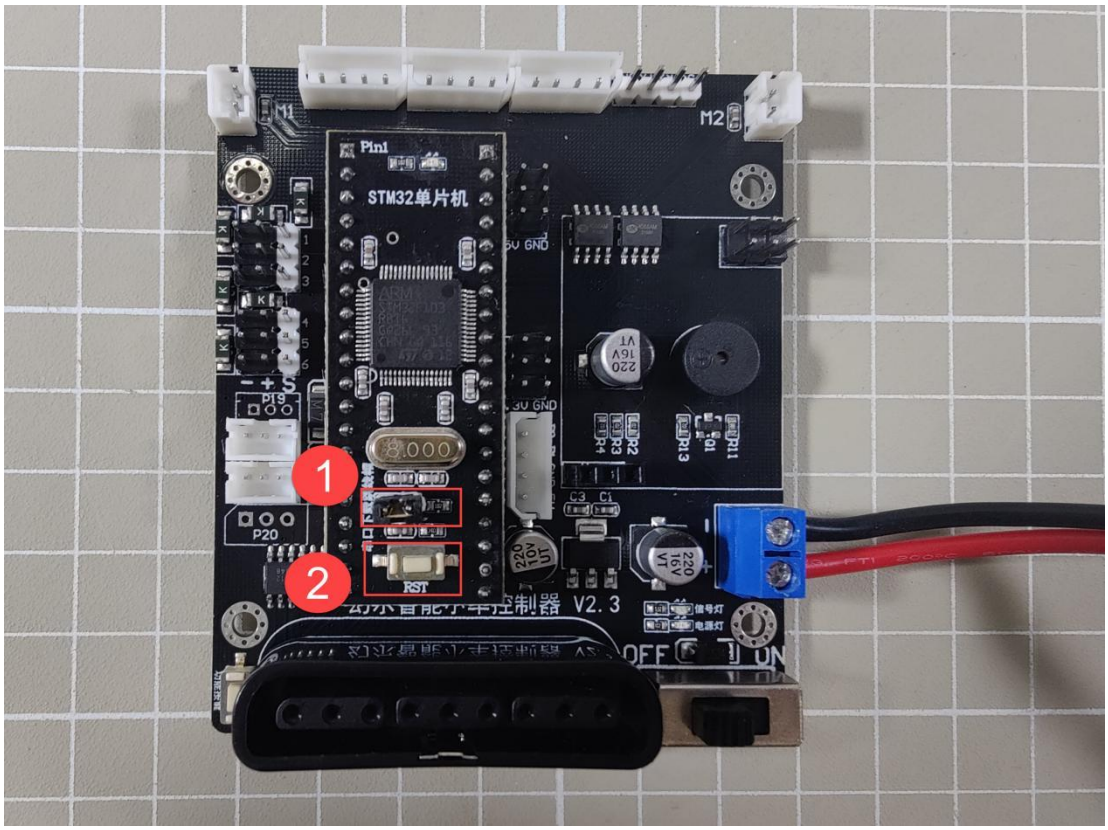


5) Open the program download tool  , select the serial port and baud rate 115200 to write the program to the development board. Please refer to the following image in details.





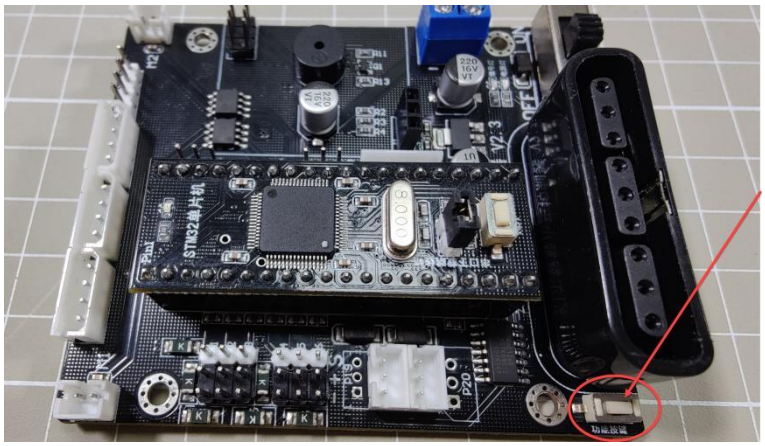
6) After writing, insert the jumper cap and press RST button to run the program.



4. Project Outcome

Note: When switching functions, wait at least 1 second before pressing the button again.

Press the function button on the controller, as shown in the following figure:



The corresponding direction for each press is shown below:

The	1	2	3	4	5

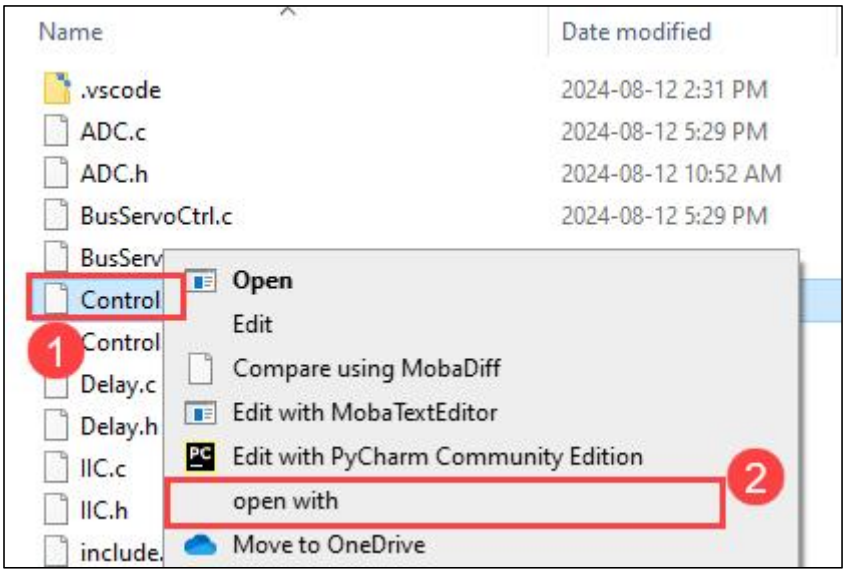
number of presses					
Movement direction	Go forward	Go backward	Turn left	Turn right	Stop

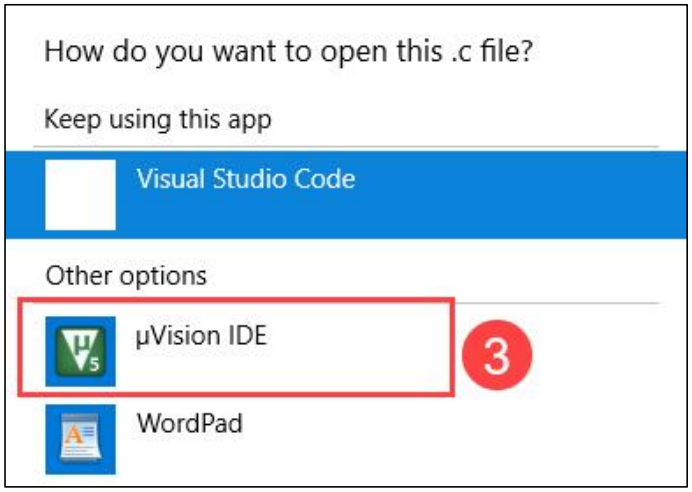
5. Function Extension

◆ **Modify motor speed**

The source code of the program is located at “1.Tutorial/2.Development Tutorial/2.2 Car Chassis Control/01 Tank Car Chassis Control/3.Program File/STM32 Version/car_key_ demo\USER\Control.c”.

Locate and open the program file.





Control.c

```c
#include "include.h"

static u8 key_num = 0;

/****************************************
*Name:     gerKey
*Function:  determine if the button is pressed
*Input:    null
*Return:   TRUE/FALSE
*Author:     ROC
****************************************/
bool gerKey(void)
{
  if(!KEY)
  {
    DelayMs(10);
    if(!KEY)
    {
      return TRUE;
    }
    return FALSE;
  }
  else
    return FALSE;
}
```

The motor speed ranges from -100 to 100. When the value is negative, the motor rotates counterclockwise and the car moves forward; when the value is positive, the motor rotates clockwise and the car moves backward. If the left and right motor values are different, the car turns.

```c
  switch(key_num)
  {
    case 0:
    MotorControl(0,0);    //forward
    break;

    case 1:
    MotorControl(-100,-100);    //forward
    break;

    case 2:
    MotorControl(100,100);    //backward
    break;

    case 3:
    MotorControl(100,-100);    //turn left
    break;

    case 4:
    MotorControl(-100,100);    //turn right
    break;

    default:
    key_num=0;
    break;
  }
}
```

For example, if you want to reduce the speed of the car, such as slowing it down to half of its previous speed, modify it as shown in the figure.

```
55      case 1:
56      MotorControl(-100,-100);    //forward
57      break;
```

After modifying the code, click  in the upper left corner. Follow “3.Program Download” to download the program.

◆ **Modify function**

If you want to modify the function corresponding to the number of times the button is pressed, locate the “case” statement for MotorControl in “Control.c”. 1 represents forward, 2 represents backward, 3 represents left turn, and 4 represents right turn, corresponding to pressing the button for the first time, the second time, and so on, as shown in the figure.

```
55      case 1:
56      MotorControl(-100,-100);    //forward
57      break;
58
59      case 2:
60      MotorControl(100,100);    //backward
61      break;
62
63      case 3:
64      MotorControl(100,-100);    //turn left
65      break;
66
67      case 4:
68      MotorControl(-100,100);    //turn right
69      break;
```

For example, if you want to switch the functions of moving forward and turning left, modify it as shown below.

```
51      case 0:
52      MotorControl(0,0);    //forward
53      break;
54
55      case 1:
56      //MotorControl(-100,-100);    //forward
57      MotorControl(100,-100);    //turn left
58      break;
59
60      case 2:
61      MotorControl(100,100);    //backward
62      break;
63
64      case 3:
65      //MotorControl(100,-100);    //turn left
66      MotorControl(-100,-100);    //forward
67      break;
68
69      case 4:
70      MotorControl(-100,100);    //turn right
71      break;
72
73      default:
74      key_num=0;
75      break;
76  }
```

6. Code Analyze

The control of encoder motor and geared motor is different. The geared motor sends data by directly defining the pin output high or low level, while the encoder motor transfers data through the I2C interface. Below is a specific analysis of the control code for the encoder motor.

The path to the source code of the program is “3.Program File/STM32 Version/car_key_demo/OpenArmSTM32.uvproj”.

◆ Define address for data transmission

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

◆ 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 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_JGB37_520_12V_110RPM   3    //44 pulses per magnetic ring rotation, reduction ratio: 90, default
```

◆ Send data

The data is sent to the motor driver module via the “int8_ I2C_Write_Len()” function.

```
//Loop to send an array of data (addr: address  buf: data content  leng: data length)
uint8_t I2C_Write_Len(uint8_t Reg,uint8_t *Buf,uint8_t Len)//I2C write data
{
  uint8_t i;
  IIC_start();                                  //After the start signal, a 7-bit slave address and 1-bit direction b
  IIC_send_byte((CAM_DEFAULT_I2C_ADDRESS << 1) | 0);  //Send device address and write command
  if(IIC_wait_ack() == 1)                       //Wait for the response. If it fails, send stop signal and return 1
  {
    IIC_stop();
    return 1;
  }
  IIC_send_byte(Reg);                           //send register address
  if(IIC_wait_ack() == 1)                       //Wait for the response. If it fails, send stop signal and return 1
  {
    IIC_stop();
    return 1;
  }
  for(i =0;i<Len;i++)                           //Loop len times to write data
  {
    IIC_send_byte(Buf[i]);                      //Send the 8-bit data of the i-th bit
    if(IIC_wait_ack() == 1)                     //Wait for the response. If it fails, send stop signal and return 1
    {
      IIC_stop();
      return 1;
    }
  }
  IIC_stop();                                   //Send stop signal
  return 0;                                     //Return 0 to confirm successful transmission
}
```

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.

◆ Initialization

Initialize the port and motor.

```
static u8 key_num=0 ;
SystemInit();                                   //System clock is initialized to 72M   SYSCLK_FREQ_72MHz
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);  //Set NVIC interrupt group 2: 2-bit preemption priority, 2-
InitDelay(72);                                  //Initialize delay function
InitTimer2();                                   //Timer 2 initialization
IIC_init();                                     //IIC initialization
Usart1_Init();                                  //Serial port initialization
InitLED();                                      //Initialize LED indicator
DelayMs(200);
I2C_Write_Len(MOTOR_TYPE_ADDR,&MotorType,1);      //Write the motor type number to the motor type address
DelayMs(5);
I2C_Write_Len(MOTOR_ENCODER_POLARITY_ADDR,&MotorEncoderPolarity,1);   //Set motor polarity
DelayMs(5);
while(1)
 {
```

"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.

◆ Button Detection

After that, the button is checked. 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.

```
126   static u8 key_num=0 ;
```

Change the voltage to determine whether the button is pressed.

```
uint8 BuzzerState = 0;
uint16 BatteryVoltage;

void InitKey(void)
{
  GPIO_InitTypeDef  GPIO_InitStructure;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;       //
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
```

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.

```
    while(gerKey())
    {
    key_num++;
    if(key_num > 4)
      {
      key_num = 0;
      }
    LED=0;
    DelayMs(1000);
    LED=1;
    TaskTimeHandle();  //ADC detection
    switch(key_num)
  {
    case 0:
    I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,2);
    DelayMs(1800);
    break;
```

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

```
    switch(key_num)
  {
    case 0:
    I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,2);
    DelayMs(1800);
    break;

    case 1:           //forward
    I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,2);
    DelayMs(1800);
    break;

    case 2:        //backward
    I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p2,2);
    DelayMs(1800);
    break;

    case 3:     //turn left
    I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p3,2);
    DelayMs(1800);
    break;

    case 4:   //turn right
    I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p4,22);
    DelayMs(1800);
    break;

    default:
    break;
```

◆ **Call function**

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.

```
113  int8_t p0[4]={0,0};                    //stop
114  int8_t p1[4]={-50,-50};               //forward
115  int8_t p2[4]={50,50};                //backward
116  int8_t p3[4]={50,-50};                //turn left
117  int8_t p4[4]={-50,50};                //turn right
```

“p1” represents the speed data to be sent. “-50” and “-50” respectively represent the speed values of M1 and M2 motors. When the speed value is positive, the motor rotates clockwise; when the speed value is negative, the motor rotates counterclockwise; when the value is 0, the motor stops rotating.

The “WireWriteDataArray()” function is used to control the motor to rotate at the speed values set above, as shown in the following code:

```
160      case 1:              //forward
161      I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,2);
162      DelayMs(1800);
163      break;
```

Take "I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,2);" as an example:

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), which means that both M1 and M2 motors rotate in counterclockwise at a speed of 50. If all motor speed values are 0, the car stops moving.

The third parameter "2" represents the data length.