Basic Routine (STM32 Version)

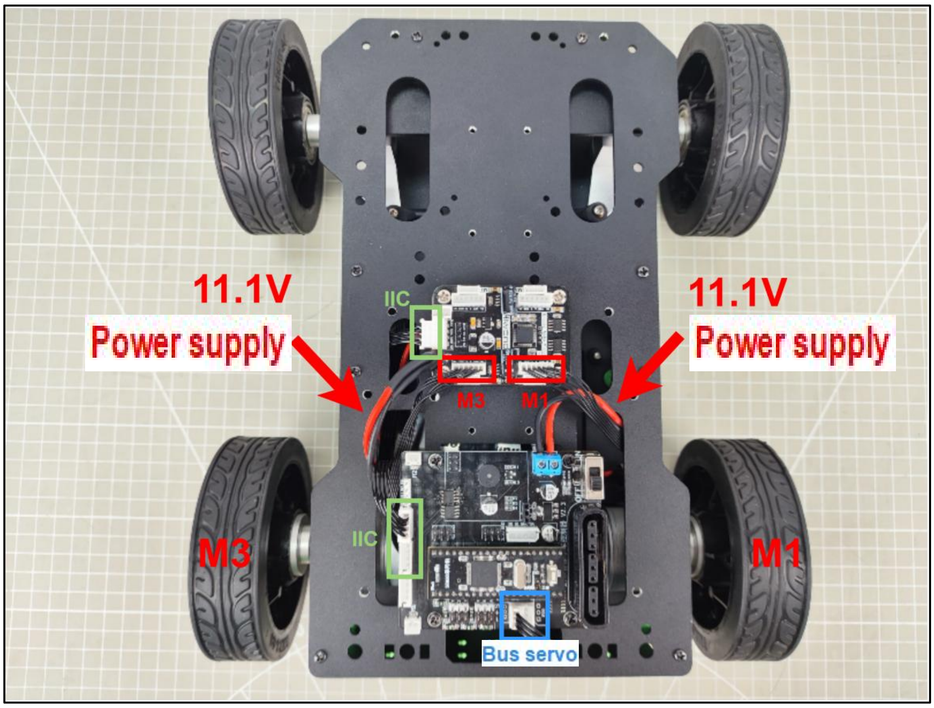
1. Working Principle

The Ackerman 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. Turning is achieved by setting different values for the bus servo. For example, when the bus servo rotates to the position of 500, it is in the neutral position of 90°, rotating to the position of 650 is left turn, and rotating to the position of 350 is right turn.

Then, you can change the direction of movement with the button. When the button on the controller is pressed, Ackerman chassis switches to motion state and moves forward, backward, left, or right.

2. Getting Ready

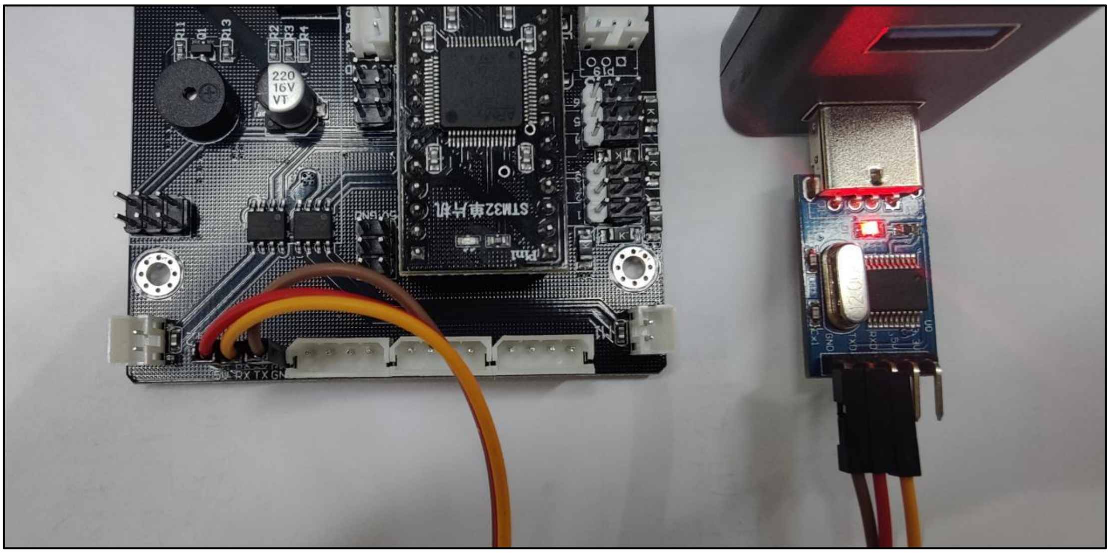
1) Connect the encoder motor to the M1 and M3 interfaces on the motor driver. Connect to the right for M1 and to the left for M3. Then, connect one end of the 4-pin cable to the I2C interface on the motor driver. In this program, a dual power supply of 11.1V is used.



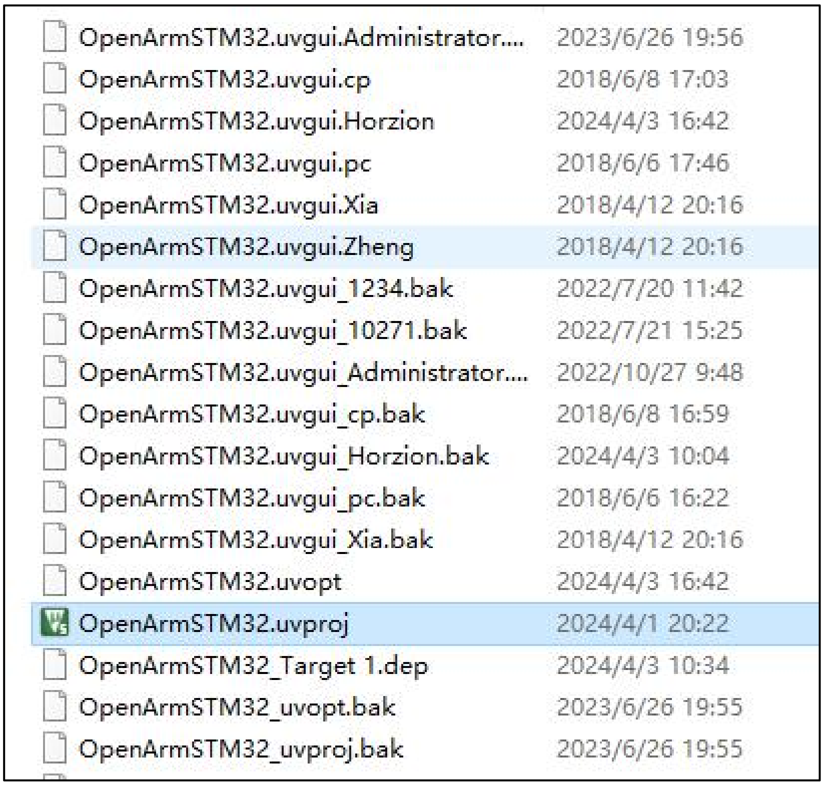
2） Please refer to “2.Software/2.2 STM32” to install and debug the Keil tool.

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

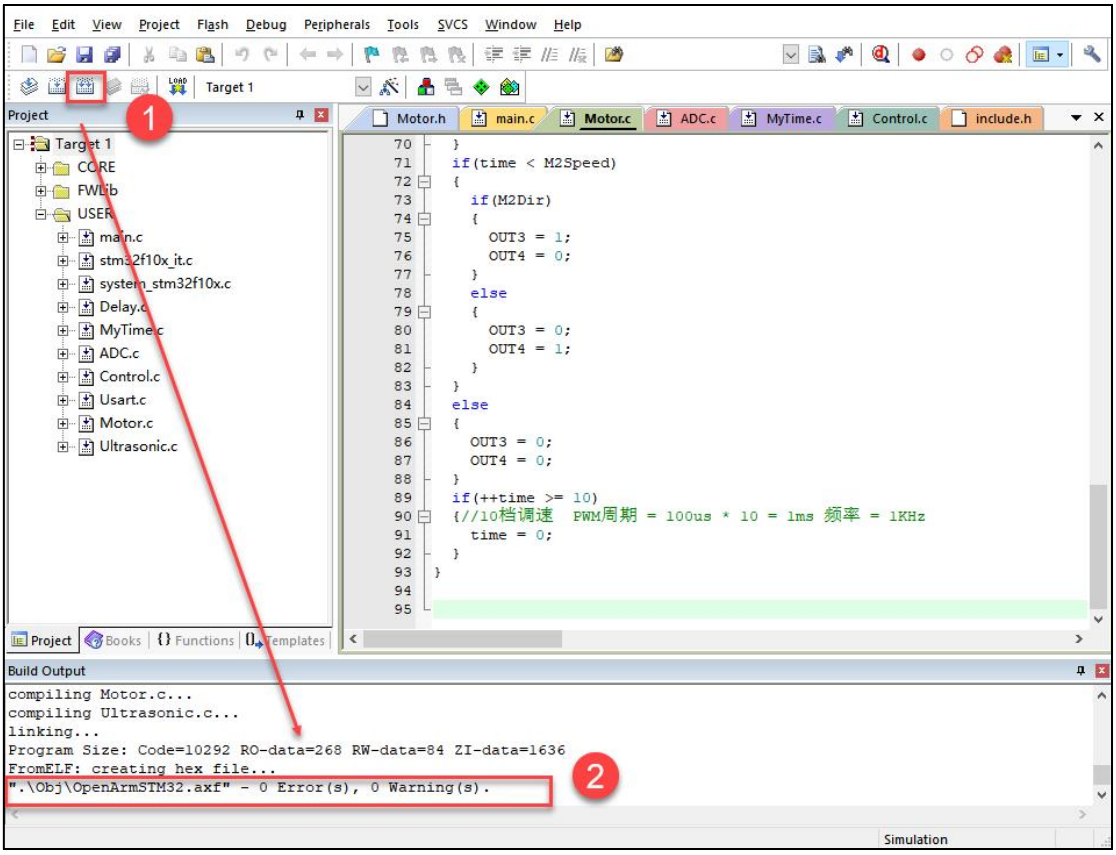


2） Double-click to open STM32 example program in “3.Program File/STM32 Version”.

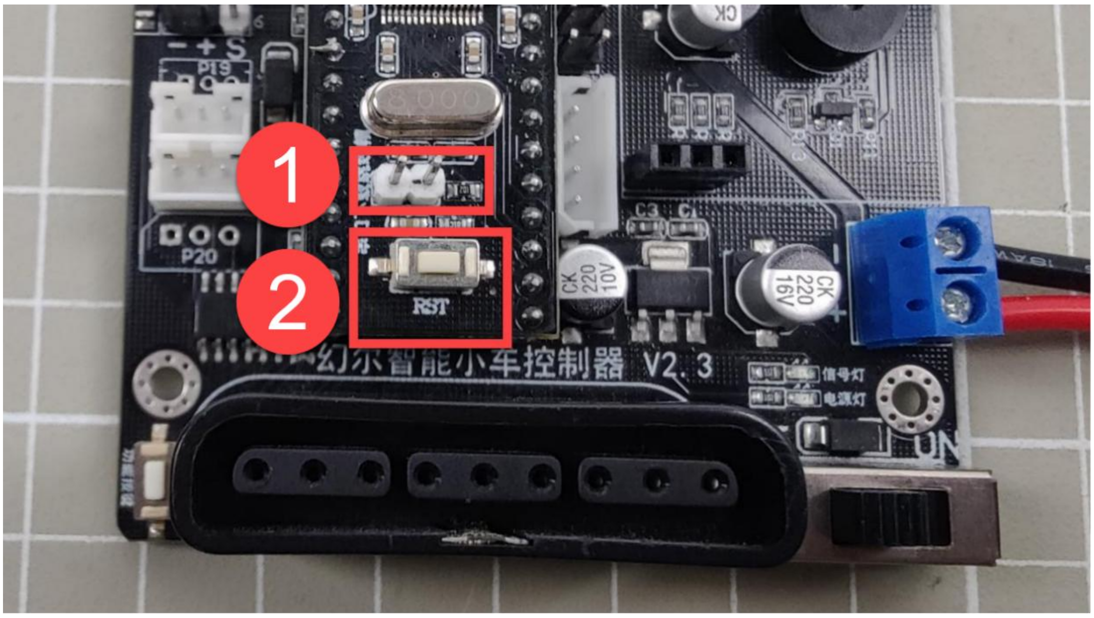


OpenArmSTM32.uvgui.Administrator....	2023/6/26 19:56
OpenArmSTM32.uvgui.cp	2018/6/8 17:03
OpenArmSTM32.uvgui.Horzion	2024/4/3 16:42
OpenArmSTM32.uvgui.pc	2018/6/6 17:46
OpenArmSTM32.uvgui.Xia	2018/4/12 20:16
OpenArmSTM32.uvgui.Zheng	2018/4/12 20:16
OpenArmSTM32.uvgui_1234.bak	2022/7/20 11:42
OpenArmSTM32.uvgui_10271.bak	2022/7/21 15:25
OpenArmSTM32.uvgui_Administrator....	2022/10/27 9:48
OpenArmSTM32.uvgui_cp.bak	2018/6/8 16:59
OpenArmSTM32.uvgui_Horzion.bak	2024/4/3 10:04
OpenArmSTM32.uvgui_pc.bak	2018/6/6 16:22
OpenArmSTM32.uvgui_Xia.bak	2018/4/12 20:16
OpenArmSTM32.uvopt	2024/4/3 16:42
OpenArmSTM32.uvproj	2024/4/1 20:22
OpenArmSTM32_Target 1.dep	2024/4/3 10:34
OpenArmSTM32_uvopt.bak	2023/6/26 19:55
OpenArmSTM32_uvproj.bak	2023/6/26 19:55

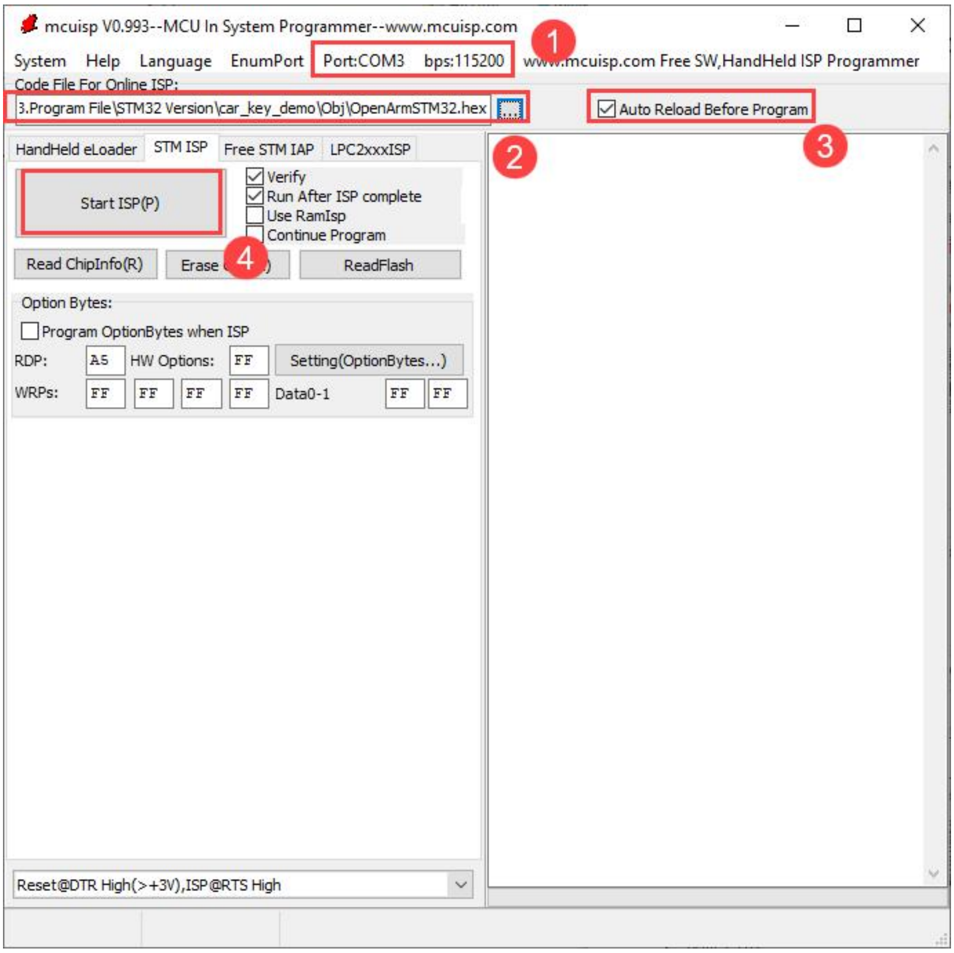
3) After opening, click the "Compile" button to generate an executable file with all the code.

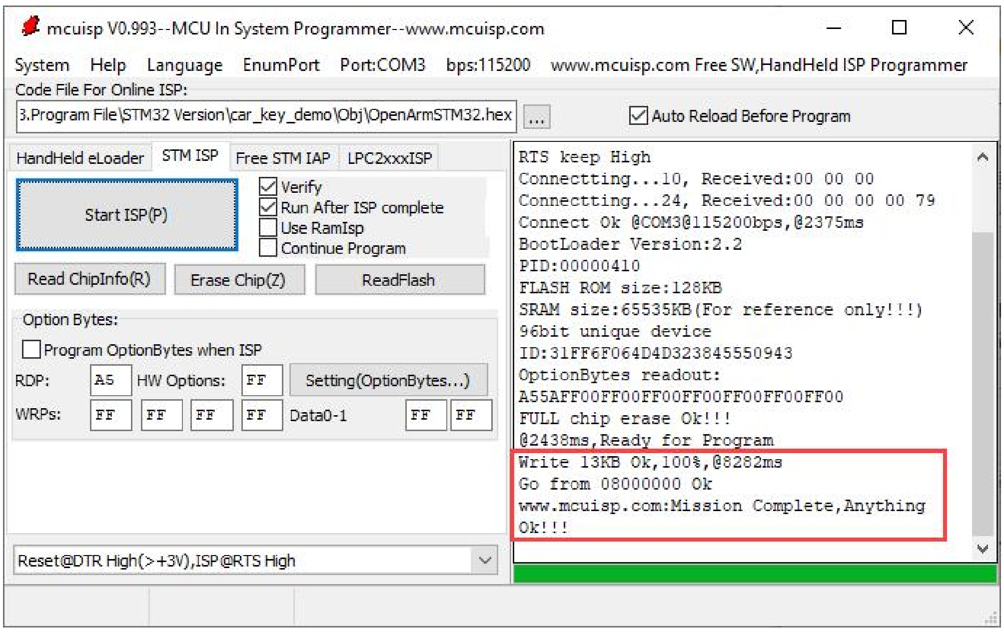


4) Remove the jumper cap on STM32 development board, and press RST reset button.

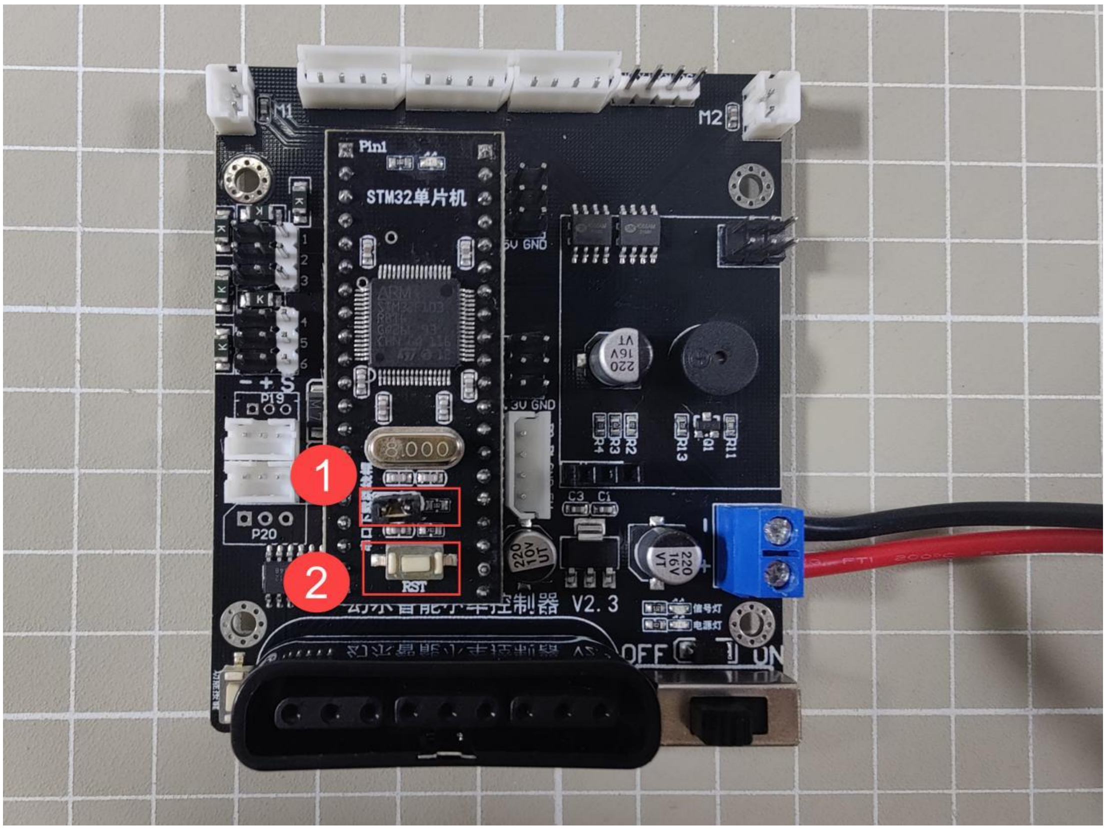


5) Open the program download tool  , select the serial port and baud rate 115200 to write the program to the development board. Please refer to the following image in details.





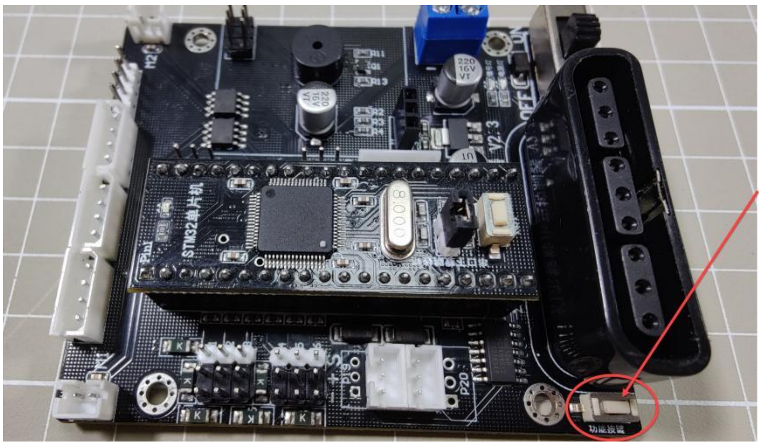
After writing, insert the jumper cap and press RST button to run the program.



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:



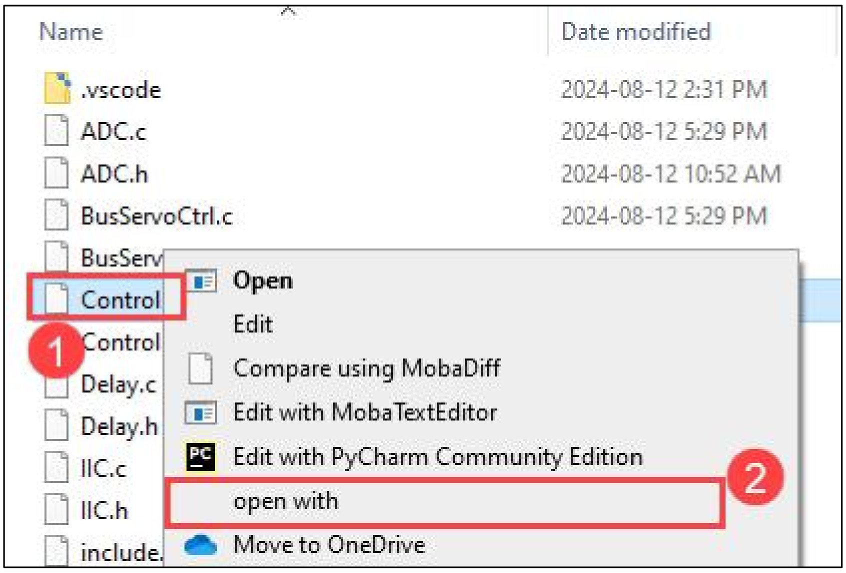
5. Function Extension

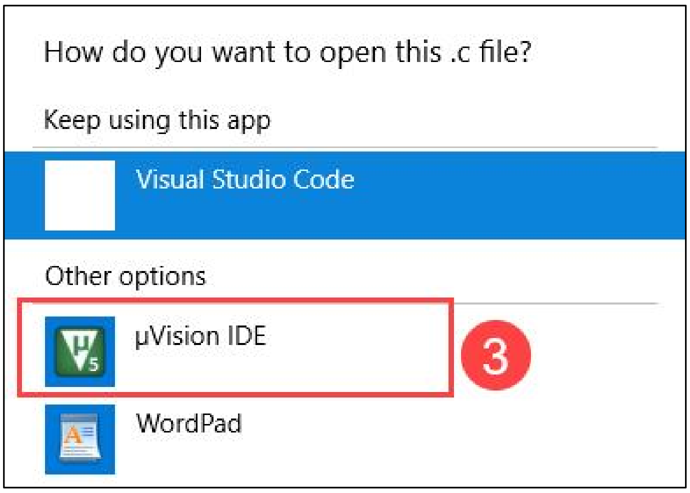
- **Modify motor speed**

The source code of the program is located at “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. The array of length 4 represents the speed of M1 to M4 motors. The “BusServoCtrl()” function controls the bus servo to achieve Ackerman turning.

```c
      switch(key_num)
      {
        case 0:
        //stop
        I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
        DelayMs(1800);
        break;

        case 1:
        DelayMs(3600);
        //forward
        BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,500,1000);
        I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,4);
        DelayMs(1800);

        I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
        DelayMs(1800);

        //backward
        BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,500,1000);
        I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p2,4);
        DelayMs(1800);

        I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
        DelayMs(1800);

        //turn left
        BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,650,1000);
        I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,4);
        DelayMs(1800);

        BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,500,1000);
        I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
        DelayMs(1800);
```

For example, if you want to increase the speed of the car, such as increase it to 50. Modify it as shown in the figure.

```
int8_t p0[4]={0,0,0,0};                        //stop
int8_t p1[4]={-25,0,25,0};                     //forward
int8_t p2[4]={25,0,-25,0};                     //backward
```

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

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.

◆ **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:

```
#define MOTOR_TYPE_WITHOUT_ENCODER        0      //Motor without encoder, 44 pulses per magnetic ring rotation,
#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, d
```

◆ **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
  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
  {
    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
  {
    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
    {
      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.

```
int main(void)
{
  static u8 key_num = 1 ;
  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
  InitBusServoCtrl();                            //initialize bus servo
  DelayMs(200);
  I2C_Write_Len(MOTOR_TYPE_ADDR,&MotorType,4);     //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 1.

```
static u8 key_num = 1 ;
```

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 1, set it to 0.

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

```
        switch(key_num)
        {
          case 0:
          //stop
          I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
          DelayMs(1800);
          break;

          case 1:
          DelayMs(3600);
          //forward
          BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,500,1000);
          I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,4);
          DelayMs(1800);

          I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
          DelayMs(1800);

          //backward
          BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,500,1000);
          I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p2,4);
          DelayMs(1800);

          I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
          DelayMs(1800);

          //turn left
          BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,650,1000);
          I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,4);
          DelayMs(1800);

          BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,500,1000);
          I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
          DelayMs(1800);

          BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,650,1000);
          I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p2,4);
          DelayMs(1800);

          BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,500,1000);
          I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
          DelayMs(1800);

          //turn right
          BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,350,1000);
          I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,4);
          DelayMs(1800);

          BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,500,1000);
          I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p0,4);
          DelayMs(1800);

          BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,350,1000);
          I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p2,4);
          DelayMs(1800);
```

◆ Call function

The "WireWriteDataArray()" function is used to send data to control the movement of the Ackerman chassis. A speed value is set for the motor, using "int8_t p1[4]={-25,0,25,0}" as an example.

```
int8_t p0[4]={0,0,0,0};                  //stop
int8_t p1[4]={-25,0,25,0};             //forward
int8_t p2[4]={25,0,-25,0};             //backward
```

"p1" represents the speed data to be sent. "-25", "0", "25", and "0" respectively represent the speed values of M1 to 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:

```
case 1:
DelayMs(3600);
//forward
BusServoCtrl(1,SERVO_MOVE_TIME_WRITE,500,1000);
I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,4);
DelayMs(1800);
```

1) Take “I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,4);” 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=（-25，0，25，0）, which means that M1 rotates counterclockwise at a speed of 25; M3 rotates clockwise at a speed of 25. If all motor speed values are 0, the car stops moving.

The third parameter "4" represents the data length.

2) The “BusServoCtrl()” function is used for Ackerman chassis steering. For example, in the code "BusServoCtrl(1, SERVO_MOVE_TIME_WRITE, 500, 1000);":

The first parameter "1" represents the ID number of the bus servo to be controlled.

The second parameter "SERVO_MOVE_TIME_WRITE" represents the use of servo speed control mode.

The third parameter "500" represents the position to which the bus servo needs to rotate, with a value range of 0-1000.

The fourth parameter "1000" represents that the bus servo will rotate to the set angle within 1000ms.