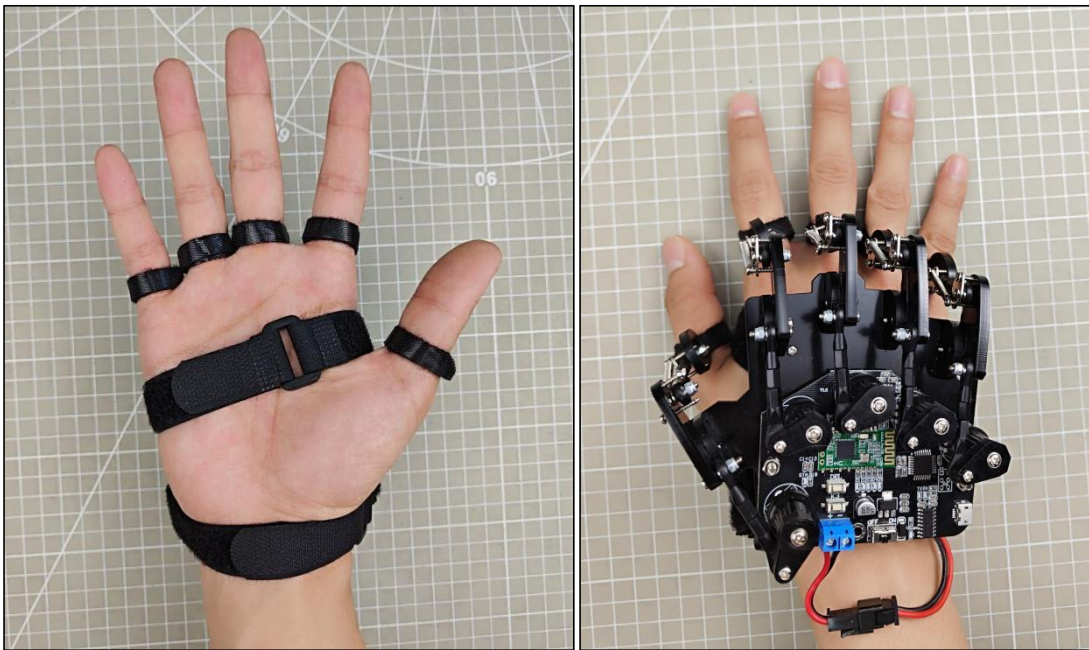


Spiderbot Control

In this section, let's use the wireless glove to control the Spiderbot.

1. Getting Ready

1) Please refer to “1.Tutorials\1.Wireless Glove Introduction and Wearing” to wear the wireless glove.



Front

Back




2) Please follow the device pairing instructions in “1.Tutorials\1.Wireless Glove Introduction and Wearing” to connect the wireless glove to the Spiderbot.



2. Program Outcome

1) Set the wireless glove to control mode 0 (i.e., the control mode of the bionic robot) via the K3 button, and all LEDs on it will not be lit up. The initial state of the wireless glove is mode 0.

2) Controlling the bionic robot to execute the action group is mainly

performed with gestures.

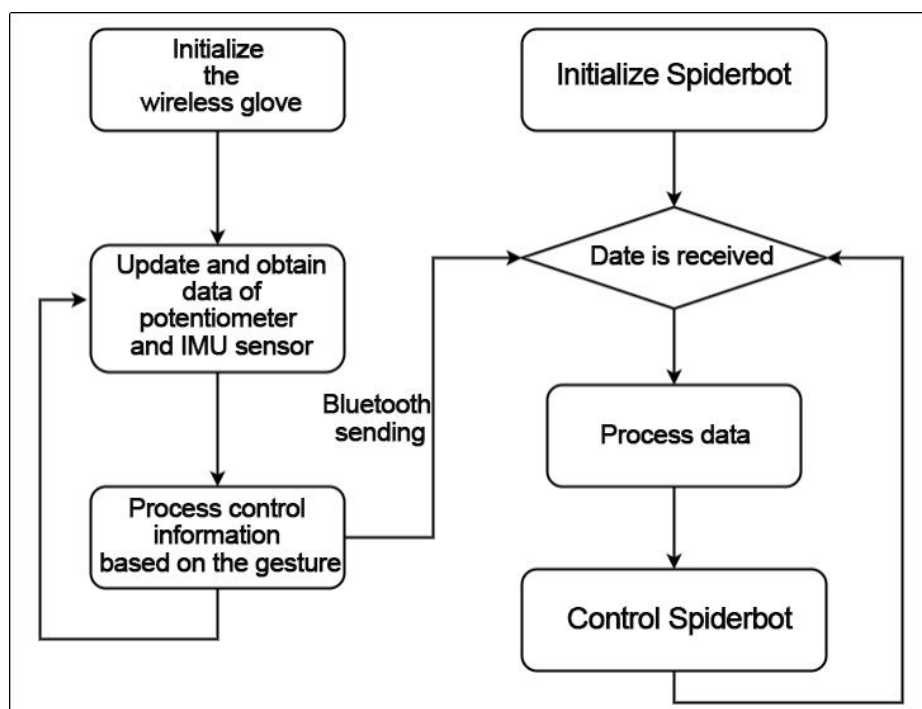
Description	Gesture Image	Action
Tilt your hand to the right at an angle greater than 35 degrees and less than 90 degrees, with the index finger extended and the ring finger bent.		Turn right
Tilt your hand to the left at an angle greater than 35 degrees and less than 90 degrees, with the middle finger extended and the ring finger bent.		Turn left
Make a fist with the palm facing down.		Stop
Stretch your hand with the palm facing down.		Go forward

Bend your middle finger and extend the other four fingers, with the palm facing up. (the same hand gesture as Spider-Man)		Go backward
Stretch your hand with the palm facing up.		Stop

3. Brief Program Analysis

This section only focuses on the analyzing of the robotic arm control program. For the analysis of the bionic robot program, please refer to the corresponding tutorials.

The implementation logic of the program can be referred to the following flowchart:



3.1 Basic Program Analysis

The source code is located at “2.Software Tools\2.Source Code\lehand\lehand.ino”.

3.1.1 Import Library File

Import the software serial library, robotic control signal library, MPU6050 library, and I2C library files required for this program.

```
#include <SoftwareSerial.h> //software serial library
#include "LobotServoController.h" //robot control signal library
#include "MPU6050.h" //MPU6050 library
#include "Wire.h" //I2C library
```

3.1.2 Define Pins and Create Objects

1) Set the RX and TX pins of the Bluetooth module on the wireless glove to 11 and 12 respectively. Limit the range of its five finger potentiometers to 0-255, and set the servo position of the robotic arm to 1500 for later mapping processing.

Set the potentiometer calibration flag to True, initialize the Bluetooth communication serial port, and create the control object for the robotic arm.

```
// RX and TX pins of the Bluetooth
#define BTH_RX 11
#define BTH_TX 12

// create the minimum and maximum store values of the potentiometers
float min_list[5] = {0, 0, 0, 0, 0};
float max_list[5] = {255, 255, 255, 255, 255};
// data variables read by each finger
float sampling[5] = {0, 0, 0, 0, 0};
// finger-related servo variables
float data[5] = {1500, 1500, 1500, 1500, 1500};
uint16_t ServePwm[5] = {1500, 1500, 1500, 1500, 1500};
uint16_t ServoPwmSet[5] = {1500, 1500, 1500, 1500, 1500};
// potentiometer calibration flag
bool turn_on = true;

// initialize Bluetooth communication serial port
SoftwareSerial Bth(BTH_RX, BTH_TX);
// the control object of the robot
LobotServoController lsc(Bth);
```

2) Define a mapping function that takes five float parameters: “in”, “left_in”, “right_in”, “left_out”, and “right_out”. This function maps the input value “in” from the range of “left_in to right_in” to the range of “left_out to right_out”, and returns the mapped value.

```
// float parameter mapping function
float float_map(float in, float left_in, float right_in, float left_out, float right_out)
{
    return (in - left_in) * (right_out - left_out) / (right_in - left_in) + left_out;
}
```

3) Create MPU6050 related variables, such as ax, ay, az, which are used to store the original data of the accelerometer. MPU6050 is a widely used six-axis motion tracking sensor, which includes a 3-axis gyroscope and a 3-axis accelerometer.

```
// MPU6050 related variables
MPU6050 accelgyro;
int16_t ax, ay, az;
int16_t gx, gy, gz;
float ax0, ay0, az0;
float gx0, gy0, gz0;
float ax1, ay1, az1;
float gx1, gy1, gz1;

// accelerometer calibration variable
int ax_offset, ay_offset, az_offset, gx_offset, gy_offset, gz_offset;
```

Variables gx, gy, and gz are used to store the original data of the gyroscope; ax0, ay0, and az0 are the calibrated values of the accelerometer data; gx0, gy0, and gz0 are the calibrated values of the gyroscope data, etc.

There are some related variables for accelerometer calibration, which are used to calibrate the sensor data. After reading the original data, these offsets will be subtracted from them to obtain more accurate results.

3.1.3 Initialization

1) The serial baud rate is set to 9600. The function button K3 and the potentiometers on the wireless glove are set to input mode, and the LEDs D1 to D5 on it are set to output mode.


```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  // initialize function button)
  pinMode(7, INPUT_PULLUP);
  // configure each finger's potentiometer
  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(A2, INPUT);
  pinMode(A3, INPUT);
  pinMode(A6, INPUT);
  // configure LEDs
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
}
```

2) Initialize Bluetooth serial communication and set the baud rate to 9600. Two AT commands are sent to the Bluetooth module: one to set the Bluetooth to master mode, and the other to perform a soft reset of the Bluetooth module. This ensures that the Bluetooth module starts up in the correct mode.

```
// configure Bluetooth
Bth.begin(9600);
Bth.print("AT+ROLE=M"); // set Bluetooth to master mode
delay(100);
Bth.print("AT+RESET"); // perform a soft reset of the Bluetooth module
delay(250);

// configure MPU6050
Wire.begin();
Wire.setClock(20000);
accelgyro.initialize();
accelgyro.setFullScaleGyroRange(3); // set the range of angular velocity
accelgyro.setFullScaleAccelRange(1); // set the range of acceleration
delay(200);
accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz); // obtain current data of each axis for calibration
ax_offset = ax; // calibration data for the X-axis acceleration
ay_offset = ay; // calibration data for the Y-axis acceleration
az_offset = az - 8192; // calibration data for the Z-axis acceleration
gx_offset = gx; // calibration data for the X-axis angular velocity
gy_offset = gy; // calibration data for the Y-axis angular velocity
gz_offset = gz; // calibration data for the Z-axis angular velocity
```

In the configuration of the MPU6050, I2C communication is initialized and I2C clock frequency is set. After initializing the MPU6050 sensor, the program sets the range of angular velocity and acceleration.

The current acceleration and angular velocity data is read and used for subsequent calibration. Next, the read data is stored in the corresponding

offset variables, which will be used for calibration in subsequent data readings.

3.1.4 Obtain Data

The angle data of finger bending is obtained mainly through the finger function defined in the main function.

- 1) Firstly, two static variables “timer_sampling” and “timer_init” are defined, and their values remain unchanged between function calls. The “init_step” variable is used to track the initialization steps. The “sampling []” and “data []” arrays are used to store the read sensor data. The “min_list []” and “max_list []” arrays may be used to store the minimum and maximum measurement values of each finger.
- 2) The code reads the sensor data of the fingers via a “for” loop. For each finger, the code reads the analog value, adds it to an accumulated value, and divides it by 2 to obtain the average value. This reduces the impact of occasional reading errors or outliers.
- 3) Then, the “float_map” function is used to map the average value to the range of servo pulse width 500 to 2500 corresponding to the robotic arm, and the mapped value is limited between 500 and 2500.

```
// read potentiometer data of each finger
void finger() {
    static uint32_t timer_sampling;
    static uint32_t timer_init;
    static uint8_t init_step = 0;
    if (timer_sampling <= millis())
    {
        for (int i = 14; i <= 18; i++)
        {
            if (i < 18)
                sampling[i - 14] += analogRead(i); // read data of each finger
            else
                sampling[i - 14] += analogRead(A6); // Read data of little finger. I2C uses A4 and A5 ports, therefore, it cannot read continuous
            sampling[i - 14] = sampling[i - 14] / 2.0; // obtain the average value between the previous and current measurement values
            data[i - 14] = float_map( sampling[i - 14], min_list[i - 14], max_list[i - 14], 2500, 500); // Map the measured value to 500-2500,
            data[i - 14] = data[i - 14] > 2500 ? 2500 : data[i - 14]; // limit the maximum value to 2500
            data[i - 14] = data[i - 14] < 500 ? 500 : data[i - 14]; // limit the minimum value to 500
        }
        timer_sampling = millis() + 10;
    }
}
```

- 4) Initialize the wireless glove via the “if” function.

```

108 if (turn_on && timer_init < millis())
109 {
110     switch (init_step)
111     {
112     case 0:
113         digitalWrite(2, LOW);
114         digitalWrite(3, LOW);
115         digitalWrite(4, LOW);
116         digitalWrite(5, LOW);
117         digitalWrite(6, LOW);
118         timer_init = millis() + 20;
119         init_step++;
120         break;
121     case 1:
122         digitalWrite(2, HIGH);
123         digitalWrite(3, HIGH);
124         digitalWrite(4, HIGH);
125         digitalWrite(5, HIGH);
126         digitalWrite(6, HIGH);
127         timer_init = millis() + 200;
128         init_step++;
129         break;

```

The robot's pan-tilt can be controlled by rotating your wrist with the wireless glove. It is necessary to obtain and update the data from the MPU6050 on the wireless glove in real-time.

5) Create a static variable "timer_u" to store the time elapsed since the last function call. Use an "if" statement to check if enough time has passed, and if so, execute the code inside the "if" statement. Use the "accelgyro.getMotion6" statement to obtain initial acceleration and angular velocity data from the MPU6050 sensor.

```

224 void update_mpu6050()
225 {
226     static uint32_t timer_u;
227     if (timer_u < millis())
228     {
229         // put your main code here, to run repeatedly:
230         timer_u = millis() + 20;
231         accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

```

- ◆ The next three sets of filtering formulas, such as "ax0 = ((float)(ax)) * 0.3 + ax0 * 0.7", are used to filter the original data to reduce noise.
- ◆ The acceleration data is converted to multiples relative to the acceleration of gravity. This is achieved by subtracting an offset value, such as "ax_offset" and "ay_offset", and dividing by the constant 8192.0.


```
ax0 = ((float)(ax)) * 0.3 + ax0 * 0.7; // filter the read value
ay0 = ((float)(ay)) * 0.3 + ay0 * 0.7;
az0 = ((float)(az)) * 0.3 + az0 * 0.7;
ax1 = (ax0 - ax_offset) / 8192.0; // calibrate and convert to the multiples of the gravity acceleration
ay1 = (ay0 - ay_offset) / 8192.0;
az1 = (az0 - az_offset) / 8192.0;

gx0 = ((float)(gx)) * 0.3 + gx0 * 0.7; // filter the read value of angular velocity
gy0 = ((float)(gy)) * 0.3 + gy0 * 0.7;
gz0 = ((float)(gz)) * 0.3 + gz0 * 0.7;
gx1 = (gx0 - gx_offset); // calibrate angular velocity
gy1 = (gy0 - gy_offset);
gz1 = (gz0 - gz_offset);
```

- ◆ Use the “atan2” function and the calibrated acceleration data to calculate the inclination angles for the X and Y axes. The tilt angles are in radians and converted to degrees. In addition, a complementary filter is used to smooth the inclination angle readings. The complementary filter combines data from the gyroscope and accelerometer to obtain more accurate inclination angle readings.

```
// complementary calculation for x-axis inclination angle
radianX = atan2(ay1, az1);
radianX = radianX * 180.0 / 3.1415926;
float radian_temp = (float)(gx1) / 16.4 * 0.02;
radianX_last = 0.8 * (radianX_last + radian_temp) + (-radianX) * 0.2;

// complementary calculation for y-axis inclination angle
radianY = atan2(ax1, az1);
radianY = radianY * 180.0 / 3.1415926;
radian_temp = (float)(gy1) / 16.4 * 0.01;
radianY_last = 0.8 * (radianY_last + radian_temp) + (-radianY) * 0.2;
}
```

3.2 Program Analysis for bionic robot Control

3.2.1 Main Function

- 1) The “loop()” is the main function of this program. When entering it, it calls to update the finger potentiometer data and updates the inclination angle data of the acceleration sensor.

```
void loop() {
    finger(); // update data of finger potentiometers
    update_mpu6050(); // update data of inclination sensor
```

- 2) In the part that detects the K3 button state, “key_state” is the flag of the button. During the process of detecting the button, the flag is used to check the state of the button. “digitalRead(7)” reads the state of the 7th interface (i.e. the

button interface). If a button press is detected, a delay of 30ms is used to prevent mechanical jitter of the button, and then the “key_state” is set to false. If the button is detected to be released (i.e. the button interface reads false) and the value of “key_state” is false, proceed to switch the control mode.

```
// if K3 button is pressed
if(key_state == true && digitalRead(7) == true)
{
    delay(30);
    if(digitalRead(7) == true)
        key_state = false;
}
if (digitalRead(7) == false && key_state == false)
{
    delay(30);
    // If K3 is pressed, switch the control mode and display corresponding numbers of LEDs
    if (digitalRead(7) == false)
    {
        key_state = true;
    }
}
```

3) The following program part controls the LED lights to light up, based on the “mode” variable. The “mode” represents the current control mode. When the control mode is 0, all LED lights will be turned off. If it is 1, the LED light on the interface 2 will be turned on, and so on.

```
if (mode == 6)
{
    mode = 0;
}
else
    mode++;
if (mode == 0)
{
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);
    digitalWrite(5, HIGH);
    digitalWrite(6, HIGH);
}
```

4) Next, the corresponding control function is called based on the control mode. “run()” is the control function for the bionic robot, “run1()” is for the robotic hand, “run2()” is for the tank chassis, and “run3()” is for the robotic arm.

```
if (mode == 0)
{
    run(); // bionic robots such as hexapod and humanoid robots
}
if (mode == 1 || mode == 4) { // Mode 1 is for the left robotic hand, and mode 2 is for the right robotic hand
    run1(mode); // robotic hand
}
if (mode == 2)
{
    run2(); // smart car
}
if (mode == 3 || mode == 5)
{
    run3(mode); // robotic arm; Mode 3 drives PWM servo and mode 5 drives bus servo
}
```

3.2.2 Bionic Robot Control Function

1) The “run()” is a control function for the bionic robot. When the control mode is switched to mode 0, it will be executed.

```
/*"run()" function controls bionic robots such as hexapod and humanoid robots
void run()
{
    // ...
}
```

2) The following part creates some basic variables. The “timer” is the marker for the running time. Each time the function “run2()” is entered, the current running time is added with 100ms and assigned to “timer”. Delay operation is achieved by comparing “timer” with “millis()”.

```
static uint32_t timer;
static uint32_t step;
static int act;
static int last_act;
static uint8_t count = 0;
if (timer > millis())
{
    return;
}
timer = millis() + 80;
```

3) The servo is controlled by recognizing gestures. Data [5] represents the position of the five finger potentiometers. When the position of a potentiometer is closer to 1500, it indicates that the corresponding finger is more straightened. The specific gesture controls are as follows:

- ◆ When the Y-axis inclination angle is maintained within the range of -90 to -30, the palm tilts to the right. If the value of data[3] is less than 1200 and data[2] is greater than 2000, with the index finger extended and the ring finger bent, the robot turns right.

- ◆ When the Y-axis inclination angle is maintained within the range of 35 to 90, the palm tilts to the left. If the index finger is extended and the ring finger is bent, the robot turns left.
- ◆ When the Y-axis inclination angle is maintained within the range of -15 to 15, the palm faces downward. If the value of data[2] is less than 600 and the index finger is bent, we detect that the fist is clenched and stop the robot.
- ◆ When the palm is facing downward and the index finger is extended, the robot moves forward.
- ◆ When the Y-axis inclination angle is less than -130 or greater than 130, the palm faces upward. If the value of data[3] is less than 1200 and data[4] is greater than 2000, with the index finger bent and the little finger extended, the robot moves backward.
- ◆ When the palm is facing upward and the index finger is extended with a value of data[2] greater than 2000, indicating an open hand, the robot stops.

```

if (radianY_last < -35 && radianY_last > -90 && data[3] < 1200 && data[2] > 2000) // If the palm's inclination angle to the right
{
    act = TURN_RIGHT; //右转(turn right)
}
if (radianY_last < 90 && radianY_last > 35 && data[3] < 1200 && data[2] > 2000) // If the palm's inclination angle to the left
{
    act = TURN_LEFT; //左转(turn left)
}
if ((radianY_last < 15 && radianY_last > -15) && data[2] < 600) // If making a fist with the palm facing down, and the index finger
{
    act = STOP;
}
if ((radianY_last < 15 && radianY_last > -15) && data[2] > 2100 && data[3] > 2100) // If the hand is stretched with the palm facing down
{
    act = GO_FORWARD;
}
if ((radianY_last < -130 || radianY_last > 130) && data[2] < 1200 && data[4] > 2000) // If the hand gesture of Spider-Man is made
{
    act = GO_BACK;
}
if ((radianY_last < -130 || radianY_last > 130) && data[2] > 2000) // If the hand is stretched with the palm facing upward, the
{
    act = STOP;
}
    
```

- 4) Next, control the robot to call the actual action group, based on the action group number recorded by "act".

```
if (act == STOP)
{
    if (count != 1) {
        count = 1;
        lsc.stopActionGroup(); // stop current action group
        lsc.runActionGroup(0, 1); // run specified action group
        return;
    }
}
```

```
if (act == GO_FORWARD)
{
    if (count != 2) {
        count = 2;
        lsc.stopActionGroup();
        lsc.runActionGroup(1, 0);
        return;
    }
}
```

```
if (act == GO_BACK)
{
    if (count != 3) {
        count = 3;
        lsc.stopActionGroup();
        lsc.runActionGroup(2, 0);
        return;
    }
}
```

```
if (act == TURN_LEFT)
{
    if (count != 4) {
        count = 4;
        lsc.stopActionGroup();
        lsc.runActionGroup(3, 0);
        return;
    }
}
```

```
if (act == TURN_RIGHT)
{
    if (count != 5) {
        count = 5;
        lsc.stopActionGroup();
        lsc.runActionGroup(4, 0);
        return;
    }
}
```

3. Program Download

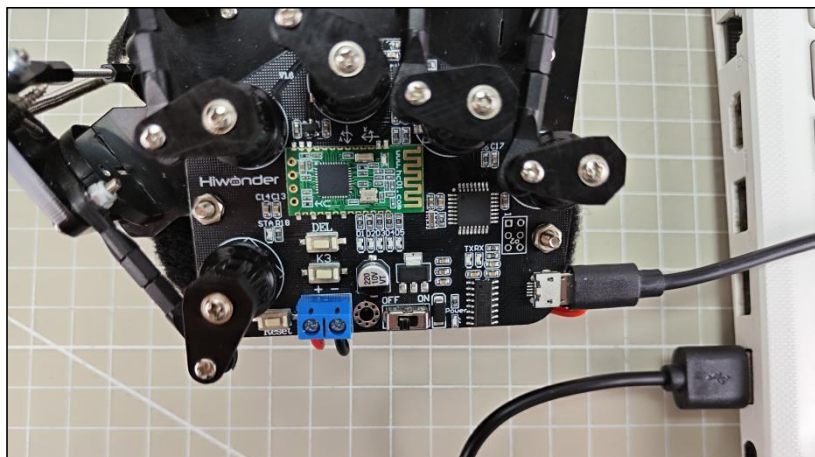
Note: The program for the wireless glove has already been downloaded before delivery.

This part is for reference only.

- 1) Locate and open the program file located in “2.Software Tools\2.Source Code\lehand\lehand.ino”.


helper_3dmath.h	2024/7/5 16:32	C He
I2Cdev.cpp	2024/7/5 16:32	C++
I2Cdev.h	2024/7/5 16:32	C He
lehand.ino	2024/7/12 15:18	INO
LobotServoController.cpp	2024/7/5 16:32	C++
LobotServoController.h	2024/7/5 16:32	C He
MPU6050.cpp	2024/7/5 16:32	C++
MPU6050.h	2024/7/5 16:32	C He
MPU6050_6Axis_MotionApps20.h	2024/7/5 16:32	C He
MPU6050_9Axis_MotionApps41.h	2024/7/5 16:32	C He

- 2) Connect the wireless glove to the computer with the micro USB cable.



- 3) Click the “Select Board”, the software will automatically detect the current Arduino port. Then click to connect.



- 4) Click  to download the program to Arduino, and then wait for the download to complete.

