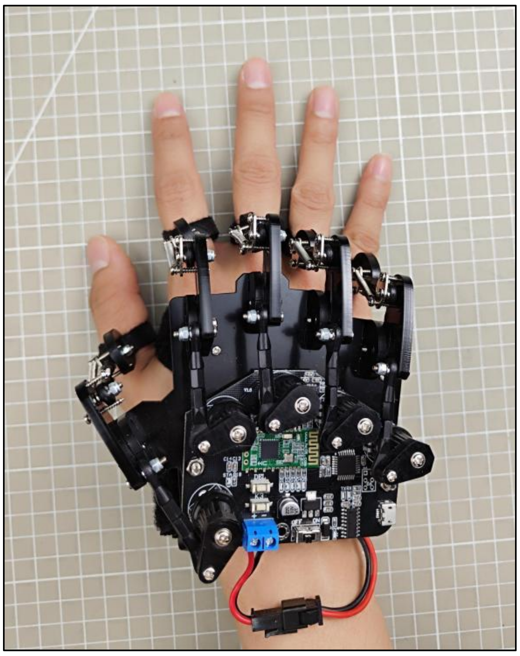
Tankbot Control

In this section, let’s use the wireless glove to control the Tankbot.

1. Getting Ready

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

Front



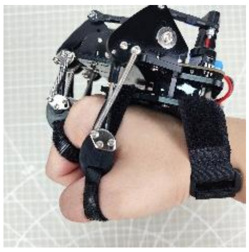
Back

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

2. Program Outcome

1) After connecting the wireless glove to Tankbot, set the wireless glove to control mode 2 (i.e., the control mode of the track robot car) via the K3 button, and LEDs D1 and D2 on it will be lit up.

Making specific gesture can control Tankbot’s movement. The specific control is as follows:

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

2) Set the wireless glove to control mode 3 (i.e., the robotic arm control mode of the PWM servo) via the K3 button, and LEDs D1 to D3 will be lit up.

Control the robotic arm servos by making specific gestures. The following table shows the available gestures and their corresponding servo numbers:

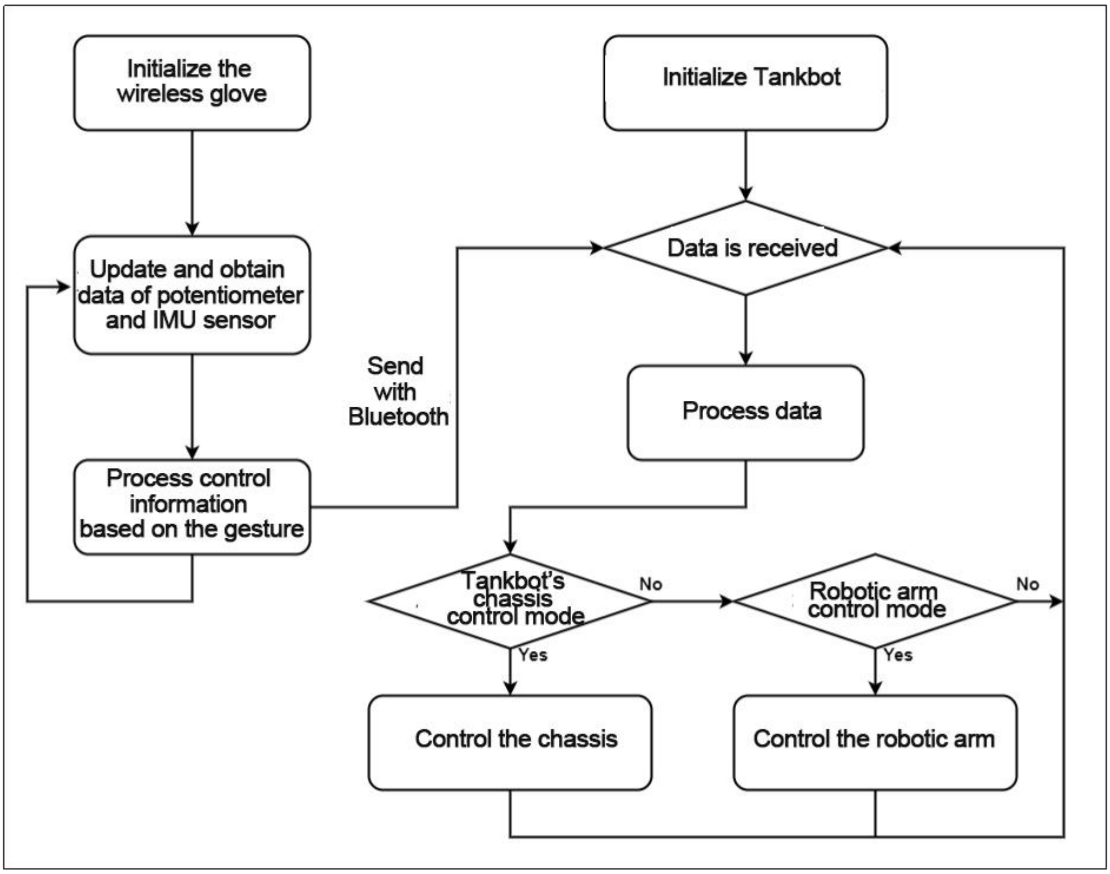
Initial Action	Description	Gesture Image	Control Servo
The palm faces down	Extend your index finger, and tilt the palm to the left or right.		Servo 1
	Extend your index and middle fingers, and tilt to the left or right.		Servo 2
	Extend your middle, ring, and little fingers, and tilt to the left or right.		Servo 3

	Extend four fingers, and tilt to the left or right.		Servo 4
	Extend all five fingers, and tilt to the left or right.		Servo 5
	Make a fist, and tilt to the left or right.		Servo 6

3. Brief Program Analysis

This section only focuses on the track robot car control part (including the chassis and the robotic arm) in the program. For the analysis of the track robot car 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 continuou
      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.

```
  if (turn_on && timer_init < millis())
  {
    switch (init_step)
    {
      case 0:
        digitalWrite(2, LOW);
        digitalWrite(3, LOW);
        digitalWrite(4, LOW);
        digitalWrite(5, LOW);
        digitalWrite(6, LOW);
        timer_init = millis() + 20;
        init_step++;
        break;
      case 1:
        digitalWrite(2, HIGH);
        digitalWrite(3, HIGH);
        digitalWrite(4, HIGH);
        digitalWrite(5, HIGH);
        digitalWrite(6, HIGH);
        timer_init = millis() + 200;
        init_step++;
        break;
```

The robotic arm’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.

```
void update_mpu6050()
{
  static uint32_t timer_u;
  if (timer_u < millis())
  {
    // put your main code here, to run repeatedly:
    timer_u = millis() + 20;
    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 Track Robot Car 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 Encapsulate Data Packet

Create a function to send data to Tankbot’s chassis to control it. The communication protocol is as follows: frame header (two 0x55 bytes), data length (four bytes), control command (0x32 for controlling the motors), speed of motor 1, speed of motor 2. The data packet should be sent to the track robot car via Bluetooth.

```
// send data to the smart car
void car_control(byte motor1, byte motor2)
{
  byte buf[6];
  buf[0] = buf[1] = 0x55;
  buf[2] = 0x04;
  buf[3] = 0x32;
  buf[4] = (byte)motor1;
  buf[5] = (byte)motor2;
  Bth.write(buf, 6);
}
```

3.2.3 Chassis Control Function

1) The “run2()” is a control function for the tank chassis. When the control mode is switched to mode 2, it will be executed.

```
// "run2" function controls the smart car
void run2()
{
```

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 uint8_t count = 0;
int act = 0;
static int last_act;
if (timer > millis())
  return;
timer = millis() + 100;
```

3) The gesture recognition is as follows:

- When data[2] is less than 600 (i.e., the middle finger is straightened), determine the Y-axis inclination angle based on the data of acceleration sensor on the wireless glove.
- When the Y-axis inclination angle is within the range of -90 to -30, it means the palm is tilted to the right. Control the Tankbot’s chassis to turn right.

- When the Y-axis inclination angle is within the range of 35 to 90, it means the palm is tilted to the left. Control the Tankbot's chassis to turn left.

- When the absolute value of Y-axis inclination angle is less than 30, it means the palm is tilted forward. Control the Tankbot's chassis to move forward.

- When the Y-axis inclination angle is less than -130 or greater than 130, it means the palm is tilted backward. Control the Tankbot's chassis to move backward.

```
timer = millis() + 100;
if (data[2] < 600 && (radianY_last < -30 && radianY_last > -90))
{
  car_control(100, -100);
}
else if (data[2] < 600  && (radianY_last > 30 && radianY_last < 90))
{
 car_control(-100, 100);
}
else if (data[2] < 600 && abs(radianY_last) < 30 )
{
  car_control(100, 100);
}
else if (data[2] < 600 && (radianY_last < -130 ||  radianY_last > 130 ))
{
 car_control(-100, -100);
}
```

4) When the current gesture cannot be recognized, stop the Tankbot's chassis.

```
else
  car_control(0, 0);
```

3.2.2 Robotic Arm Control Function

1) The "run3()" is a control function for the Tankbot's robotic arm. When the control mode is switched to mode 3, it will be executed.

```
//control robotic arm (PWM servo)
void run3(int mode)
{
```

2) Use the "timer" to delay for 50ms.

```
static uint32_t timer;
static uint32_t step;
int act = 0;
static int last_act;
static uint8_t mode = 0;
static uint8_t mode1 = 0;
static uint8_t count = 0;
if (timer > millis())
  return;
timer = millis() + 50;
```

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 index finger, middle finger, and ring finger are bent, the rotation of servo 6 is controlled by tilting the wireless glove.
- When the thumb, index finger, middle finger, and ring finger are straightened, the rotation of servo 5 is controlled by tilting the wireless glove.
- When the index finger is straightened, and the middle finger and ring finger are bent, the rotation of servo 1 is controlled by tilting the wireless glove.
- When the index finger and middle finger are straightened and the ring finger is bent, the rotation of servo 2 is controlled by tilting the wireless glove.
- When the index finger is bent, and the middle finger, ring finger, and little finger are straightened, the rotation of servo 3 is controlled by tilting the wireless glove.
- When the thumb is bent, and the other four fingers are straightened, the rotation of servo 4 can be controlled by tilting the wireless glove.

```
if (data[1] < 1200 && data[2] < 1000 && data[3] < 1000)  // make a fist and tilt to control servo 6
{
    if (radianY_last < 90 && radianY_last > -90)
    {
        lsc.moveServo(6, median + radianY_last*10, 50);
        delay(50);
    }
}
else if ( data[0] > 1400 && data[1] > 1400 && data[2] > 1400 && data[3] > 1400) // stretch all five fingers and tilt to control servo5
{
    if (radianY_last < 90 && radianY_last > -90)
    {
        lsc.moveServo(5, median + radianY_last*10, 50);
        delay(50);
    }
}
else if (data[1] > 1400 && data[2] < 1000 && data[3] < 1000 ) // extend the index finger to control servo 1
{
    if (radianY_last < 90 && radianY_last > -90)
    {
        lsc.moveServo(1, median + radianY_last*10, 50);
        delay(50);
    }
}
```

```
else if (data[1] > 1400 && data[2] > 1400 && data[3] < 1000 ) // extend the index and middle fingers to control servo 2
{
    if (radianY_last < 90 && radianY_last > -90)
    {
        lsc.moveServo(2, median + radianY_last*10, 50);
        delay(50);
    }
}
else if (data[1] < 1400 && data[2] > 1200 && data[3] > 1000 ) // extend the middle, ring, and little fingers to control servo 3
{
    if (radianY_last < 90 && radianY_last > -90)
    {
        lsc.moveServo(3, median + radianY_last*10, 50);
        delay(50);
    }
}
else if (data[0] < 1400 && data[1] > 1400 && data[2] > 1400 && data[3] > 1400) // extend four fingers excepting the thumb to control servo 4
{
    if (radianY_last < 90 && radianY_last > -90)
    {
        lsc.moveServo(4, median + radianY_last*10, 50);
        delay(50);
    }
}
}
```

4) The function “moveServo()” that controls the servo is located in the “LobotServoController.cpp” file. It mainly encapsulates the ID of the servo, the position of the servo’s rotation, and the time of the rotation into a data packet, based on the communication protocol of the open-source 6-channel servo controller. Then, the data package is sent to Tankbot.

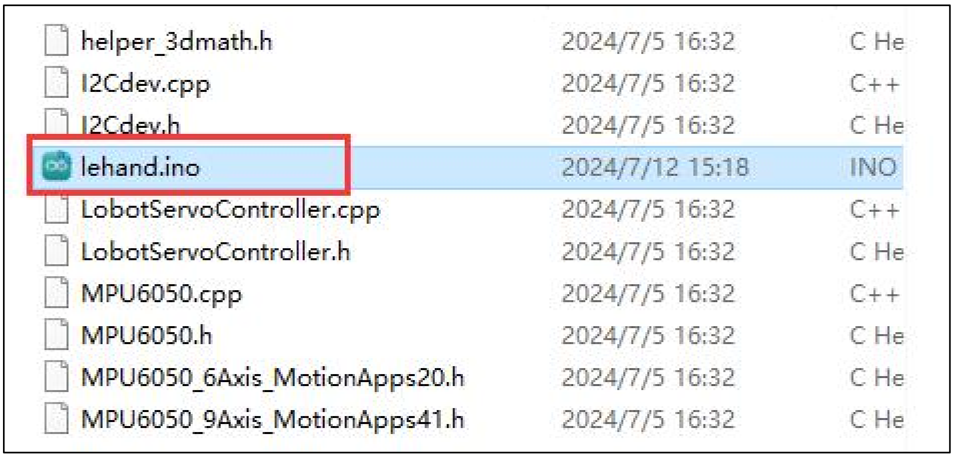
```
void LobotServoController::moveServo(uint8_t servoID, uint16_t Position, uint16_t Time)
{
  uint8_t buf[11];
  if (servoID > 31 || !(Time > 0)) {
    return;
  }
  buf[0] = FRAME_HEADER;
  buf[1] = FRAME_HEADER;
  buf[2] = 8;
  buf[3] = CMD_SERVO_MOVE;
  buf[4] = 1;
  buf[5] = GET_LOW_BYTE(Time);
  buf[6] = GET_HIGH_BYTE(Time);
  buf[7] = servoID;
  buf[8] = GET_LOW_BYTE(Position);
  buf[9] = GET_HIGH_BYTE(Position);

  SerialX->write(buf, 10);
}
```

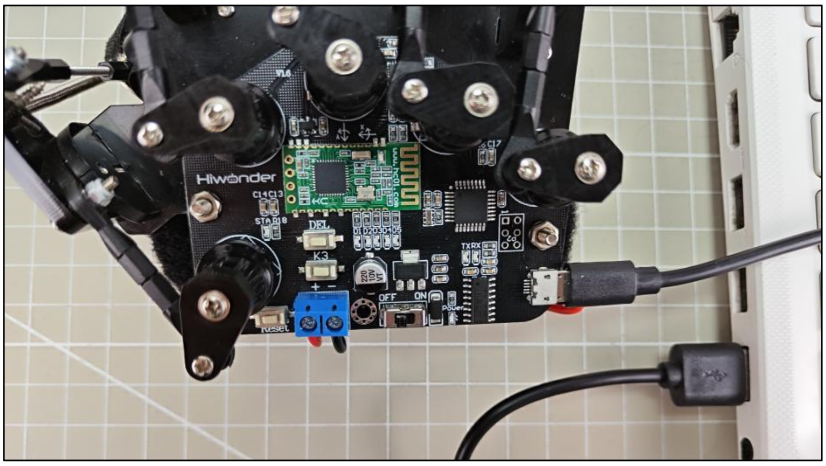
4. Program Download

Note: The program for the wireless glove has already been downloaded before delivery. This part is for reference only.

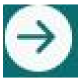
1) Locate and open the program file located in “2.Software Tools\2.Source Code\lehand\lehand.ino”.



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.