McKnum Wheel robotic arm Car

V1.11.15

1. Burn program code

1.1 Description

Before assembly, it is necessary to reset the angle of the servo and conduct an infrared remote control demonstration after the assembly is completed, so the infrared remote control program 5_IRID.ino is selected for burning .

1.2 Start the burning process

Open the code file (path: 2_Arduino_Code\5_IRID\5_IRID.ino)

3.3_cage	2023/11/9 16:50	文件夹
4.1_Ultrasonic_Sensor_Module	2023/11/9 16:50	文件夹
4.2_Ultrasonic_Obstacle_Avoidance_Robot_Car	2023/11/9 16:50	文件夹
4.3_Ultrasonic_Follow	2023/11/9 16:50	文件夹
5_IRID	2023/11/9 16:50	文件夹
6_BlueTooth	2023/11/9 16:50	文件夹

the Bluetooth module cannot be plugged in when uploading the program .

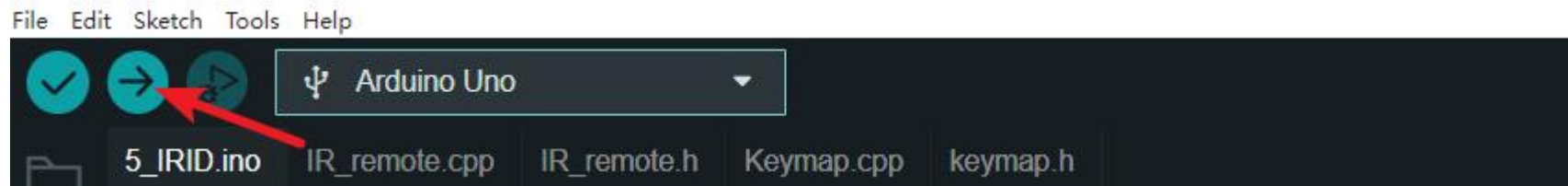Connect the Arduino board to the computer with a USB cable .
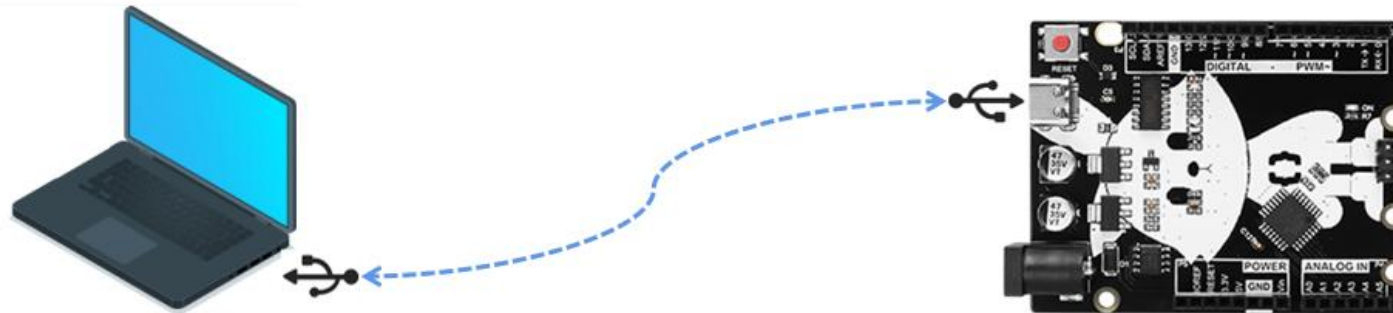
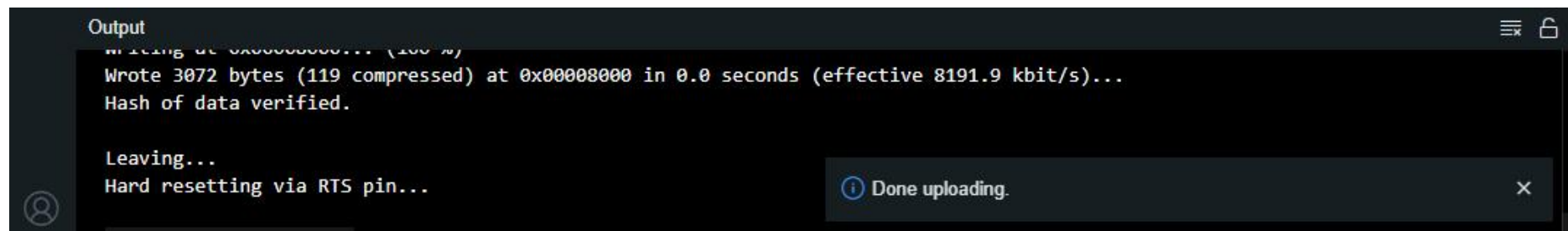Select Uno as the board type and COM5 as the serial port.
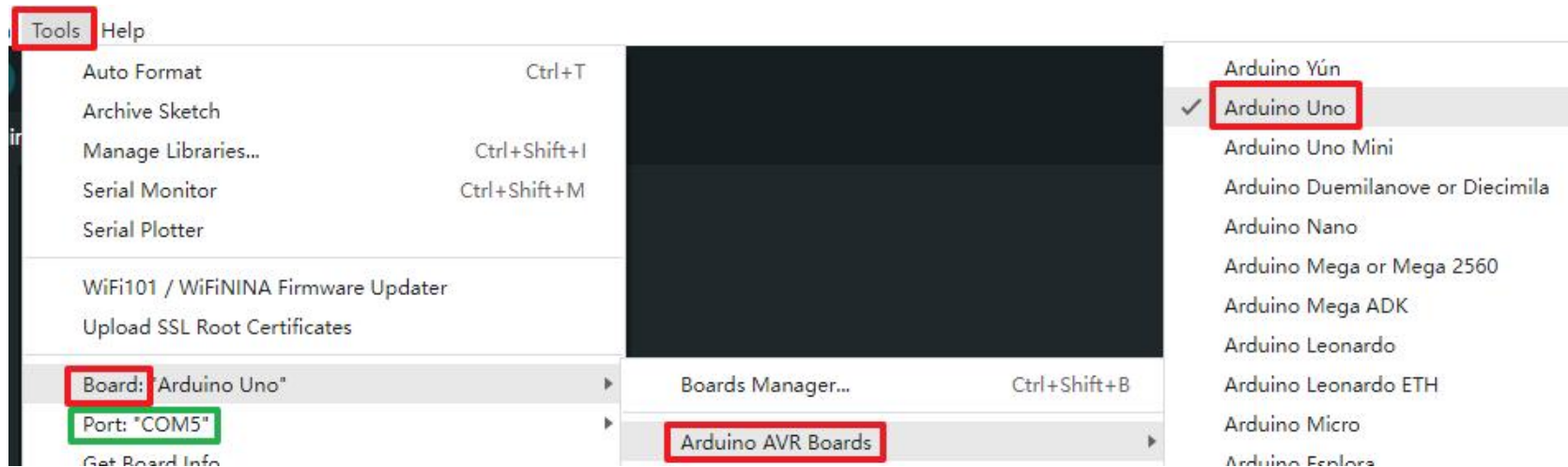
NOTE: Everyone 's serial port display will actually be different, although COM 5 is selected here , it may be COM3 or

COM4 on your computer.

After clicking the "Upload" button, the program starts uploading.

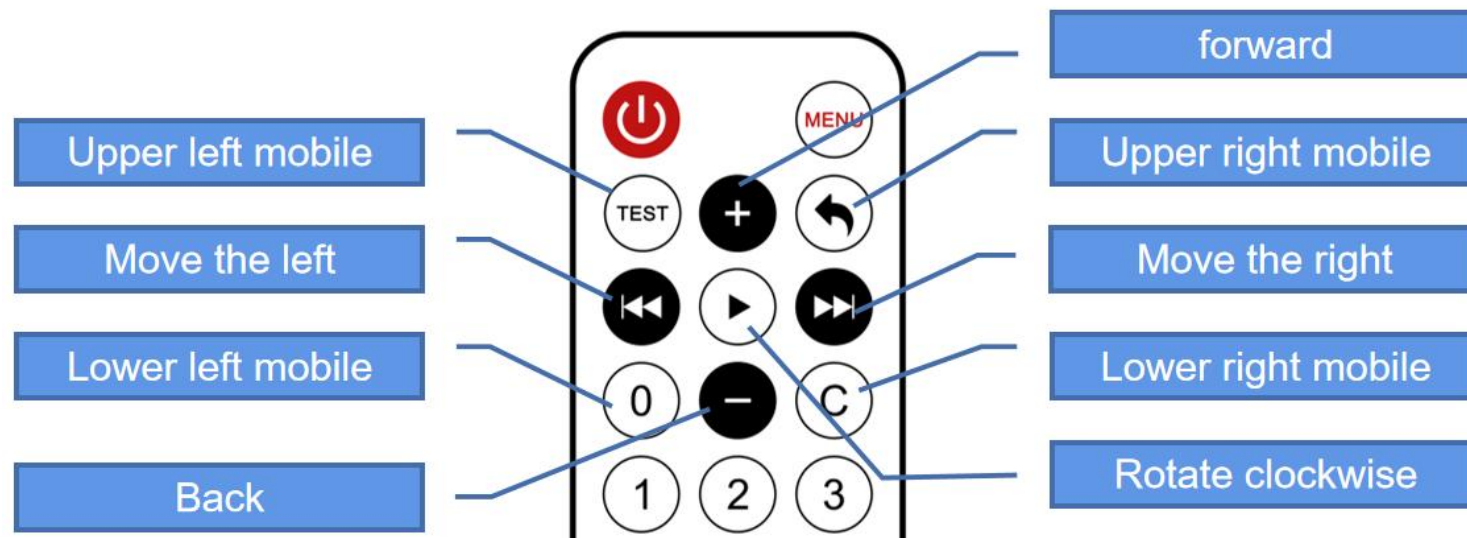After the upload is successful, it will prompt "Done uploading".

After you finish burning the code, please read the assembly manual or video to start assembling the car!

2. Infrared remote control car

2.1 Description

After assembling the car, you can use the infrared remote control to remotely control the robotic arm car. At the same time, please pay attention to whether the battery capacity of the car is sufficient and whether the infrared remote control has batteries installed. Aim the infrared remote control at the infrared receiver on the car and press the button to remotely control the car.

2.2 Infrared remote control command

Tracking mode

Claws open

Claws closed

Arm forward

Obstacle avoidance mode

Follow mode

Chassis left pendulum

Chassis right pendulum

Arm backwards
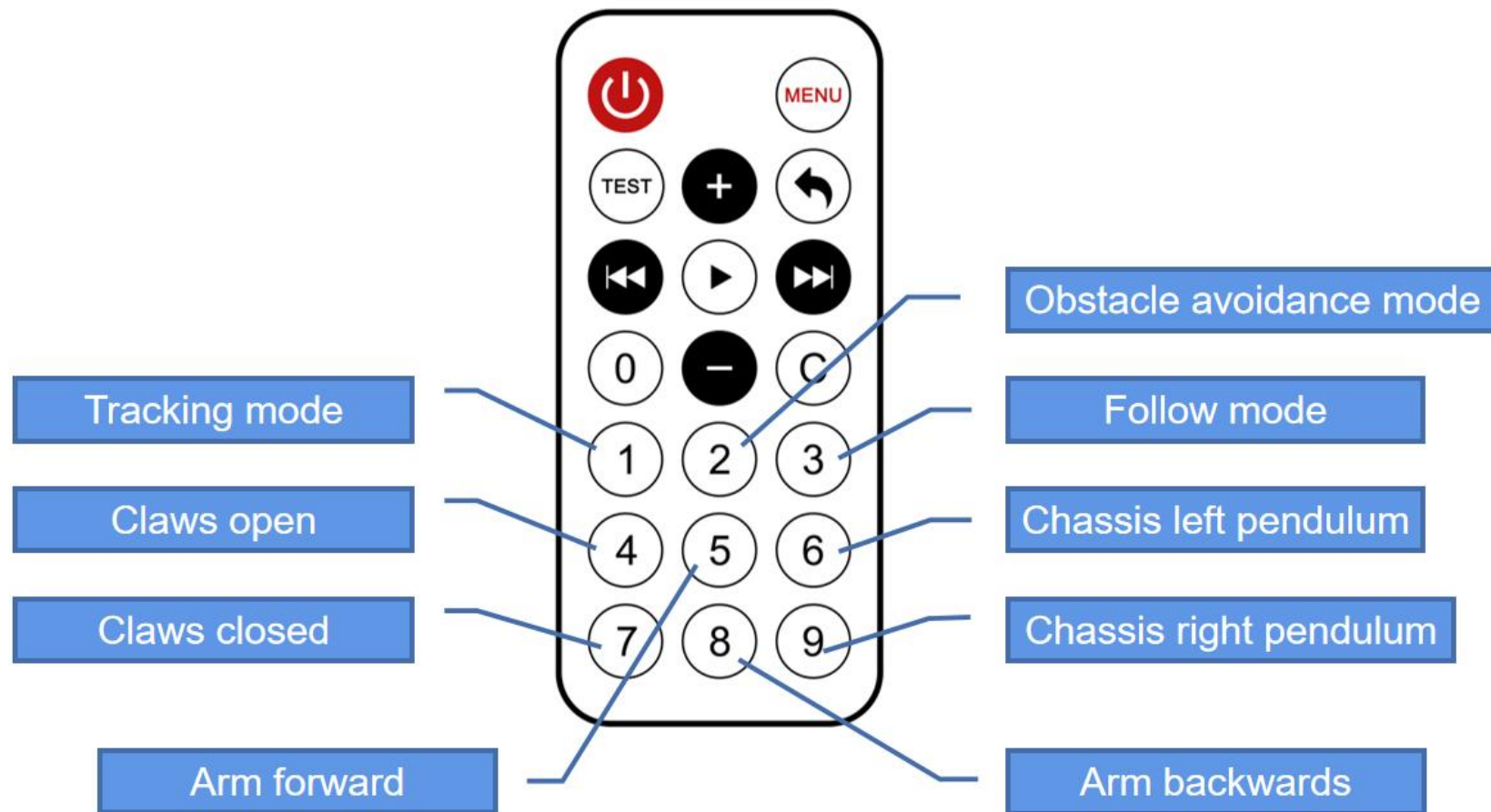
At this point, you only need to briefly learn how to control the car. The following tutorials will guide you from scratch to learn more about the device and programming related knowledge of this set.

3. The first program code-Blink

3.1 Description

In this section , you will learn how to program your control board to blink the built-in LED, as well as learn the basic

steps for downloading the program.

Main control board

There are multiple rows of connectors on both sides of the motherboard for connecting multiple electronic devices and

plug-in "modules" that extend their functionality.

It also has an LED that you can control from the sketch , which is built into the motherboard .
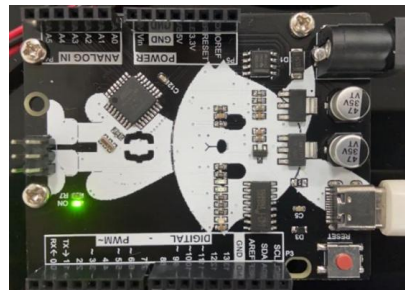
When you connect the motherboard to the USB plug, you may notice that its LED has blinked. This is because the board

comes with the "Blink" sketch pre-installed.

3.2 Make your own "Blink" sketch

In this section , we will reprogram the board using our own Blink sketch and then change its blink rate. Connect the board to your computer, set up the Arduino IDE and make sure you can find the correct serial port , and upload the program to test.

The Arduino IDE includes a number of example sketches that you can load and use , including a "blink" example sketch for making an "L" LED . In the IDE 's menu system File > Examples > 01.The " Blink " sketch you will find in Basics .
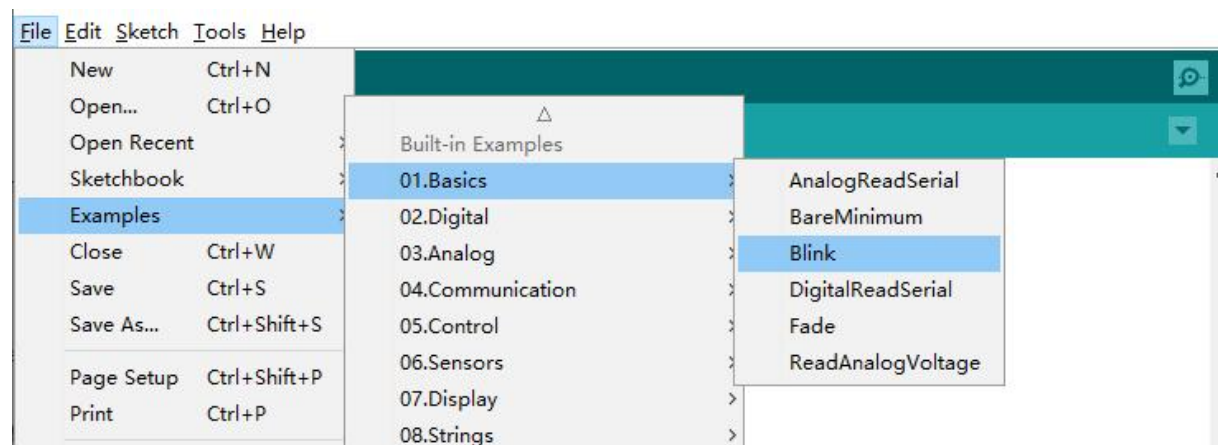
The example sketches included with the Arduino lDE are "read-only". That is, you can upload them to the UNOR3 board, but if you change them, you cannot save them as the same file. Since we're going to make changes to this sketch, the first thing you need to do is save your own copy.

From the Arduino IDE's File menu, select "Save As..." and save the sketch as "MyBlink".

You've saved your "flash" copy in your sketchbook, which means that if you ever want to find it again, just open it using the " File > Sketch book " menu option.
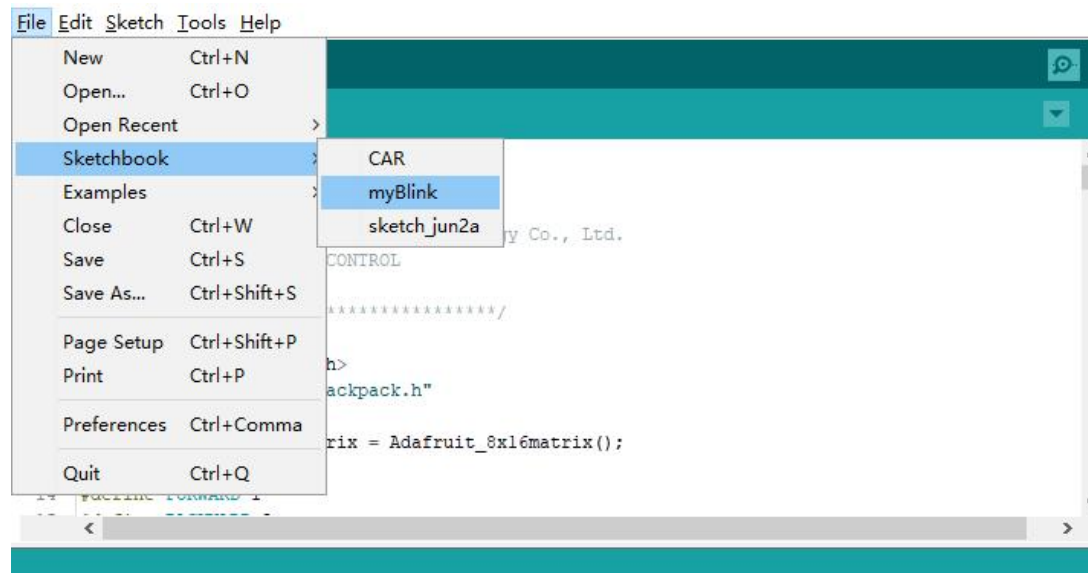
Connect the Arduino board to the computer with a USB cable .

Select Uno as the board type and COM5 as the serial port.

NOTE: Everyone 's serial port display will actually be different, although COM 5 is selected here , it may be COM3 or

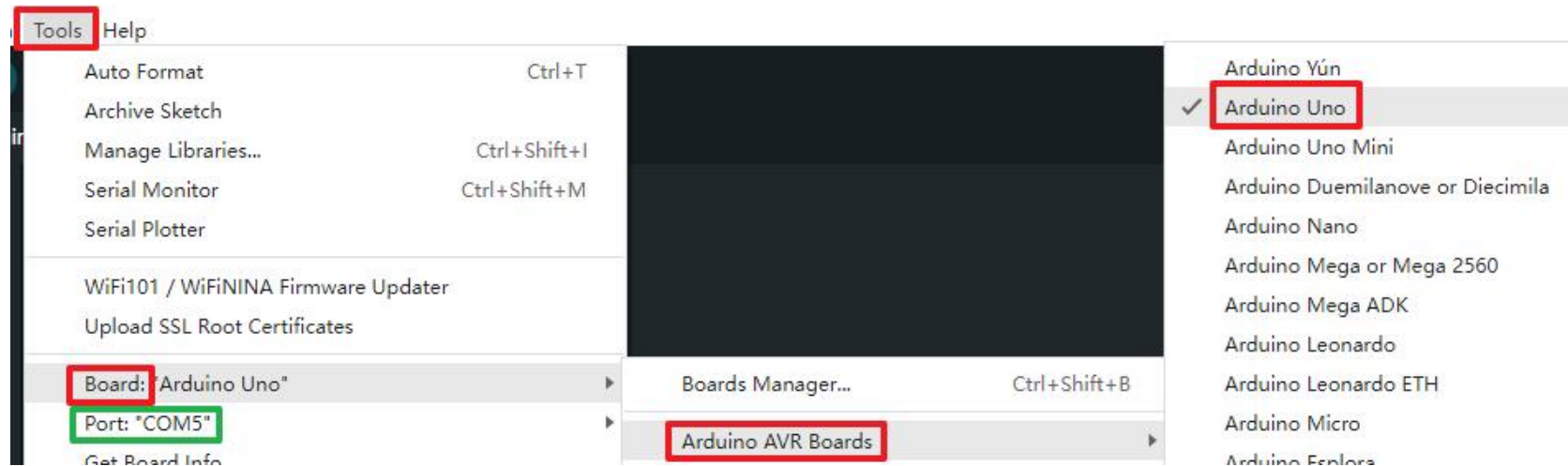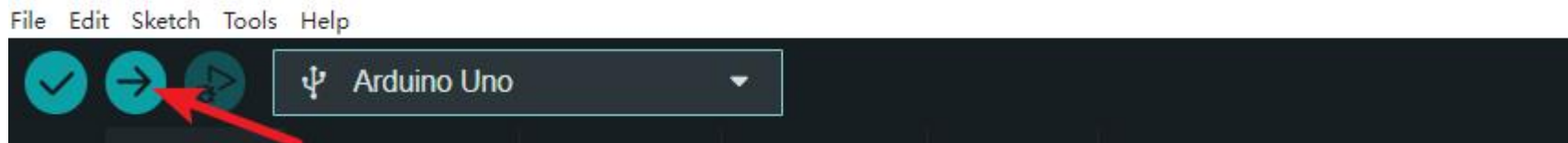COM4 on your computer.

After clicking the "Upload" button, the program starts uploading.

Once the upload is complete , the board LED should reboot and start blinking.

Note that a large part of this sketch consists of notes. These are not actual program instructions; instead, they simply explain how to make the program work. They are there for your ease of reading . Everything between " /* " and " */ " at the top of the sketch is a block comment that explains the purpose of the sketch.

Single-line comments begin with "//" and everything up to the end of the line is considered a comment.

The first part of the code is:

```
// the setup function runs once when you press reset or power the board
void setup () {
// initialize digital pin LED_BUILTIN as an output.
  pinMode (LED_BUILTIN, OUTPUT);
}
```

Every sketch requires a "set" function , which is a "Void setup()" function, this is executed when the reset button is

pressed. It is executed whenever the board resets for any reason, such as first power-up or after uploading a sketch .

The next step is to name the pin and set the output. Here, set " LED_BUILTIN " as the output port. On most Arduinos,

including UNO , pin 13 is the pin corresponding to the LED. To facilitate programming, the program has set the

LED_BUILTIN variable to this pin, so there is no need to rename pin 13 for direct use.

The sketch must also have a " loop " functionality. Unlike the "Set" function which only runs once, after a reset the

"Loop" function will start again as soon as it finishes running the command.

```
// the loop function runs over and over again forever
void loop () {
  digitalWrite (LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay ( 1000 ); // wait for a second
  digitalWrite (LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
  delay ( 1000 ); // wait for a second
}
```

Inside the loop function, the command first turns the LED pin on (high), then "delays for 1000 milliseconds (1 second),

then turns off the LED pin and pauses for one second."

You now want to make your LED blink faster. As you may have guessed, the key is to change the parameters in " delay

()".

```
31   // the loop function runs over and over again forever
32   void loop() {
33     digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
34     delay(1000);                       // wait for a second
35     digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
36     delay(1000);                       // wait for a second
37   }
```

This delay time is in milliseconds, so if you want the "LED" to flash twice as fast, change the value from "1000" to "500" .

This will pause for half a second on each delay, instead of one second. Upload the sketch again and you should see the

"LED" start blinking faster .

At this point, you have understood and mastered the basic Arduino programming knowledge and the basic steps for

downloading the program, which lays a good foundation for learning complex projects later.

4.Servo motor drive

4.1 Description

This section mainly focuses on understanding the properties and characteristics of the servo and learning related knowledge about the servo, mastering the debugging methods and circuit connections of the servo, and finally experiencing the working mode of the servo in Arduino programming.

4.2 Introduction to steering gear

The MG90S servo motor control pulse signal period is a 20MS pulse width modulation signal (PWM), the pulse width is from 0.5ms to 2.5ms , and the corresponding steering position changes linearly from 0 to 180 degrees.

In other words, if a certain pulse width is provided to the steering gear, its output shaft will maintain a certain corresponding

angle. No matter how the external torque changes, it will not change the output angle to a new corresponding position until another pulse signal is provided to it.

There is a reference circuit inside the steering gear, which generates a pulse signal with a period of 20ms and a width of 1.5ms . There is a comparator that compares the external signal with the reference signal to determine the direction and size, thereby generating a motor rotation signal.

4.3 Code analysis

Open the code file (path: 2_Arduino_Code\1.1_Servo_Angle\1.1_Servo_Angle.ino)

Import the servo library file and declare the three servo motor signal ports as 11/10/9.

```
1   #include    <Servo.h>
2
3   #define CLAW_PIN    11
4   #define ARM_PIN     10
5   #define BASE_PIN    9
```

Instantiate three servos and define rotation angle variables

```
 7    Servo clawservo;//Clamp servo
 8    Servo armservo;//Arm servo
 9    Servo baseservo;//Turntable servo
10
11    int pos = 0;
```

Initialize the servo motor angle

```
13    void setup()
14    {
15        clawservo.attach(CLAW_PIN);
16        armservo.attach(ARM_PIN);
17        baseservo.attach(BASE_PIN);
18        clawservo.write(135);
19        armservo.write(90);
20        baseservo.write(90);
```

The cycle is executed, and the clamp servo motor rotates reciprocally from 135° to 45°.

```
23    void loop()
24    {
25        for (pos = 135; pos >= 45; pos -= 1)
26        {
27            clawservo.write(pos);
28            delay(15);
29        }
30        for (pos = 45; pos <= 135; pos += 1)
31        {
32            clawservo.write(pos);
33            delay(15);
34        }
```

The robot arm servo motor rotates back and forth from 170° to 90°.

```
36        for (pos = 90; pos <= 170; pos += 1)
37        {
38            armservo.write(pos);
39            delay(15);
40        }
41
42        for (pos = 170; pos >= 90; pos -= 1)
43        {
44            armservo.write(pos);
45            delay(15);
46        }
```

The turntable servo motor rotates back and forth from 180° to 0°.

```
48        for (pos = 90; pos >= 0; pos -= 1)
49        {
50            baseservo.write(pos);
51            delay(15);
52        }
53        for (pos = 0; pos <= 180; pos += 1)
54        {
55            baseservo.write(pos);
56            delay(15);
57        }
58        for (pos = 180; pos >= 90; pos -= 1)
59        {
60            baseservo.write(pos);
61            delay(15);
```

5. Expansion board and motor driver

5.1 Description

This section mainly focuses on knowledge related to high-performance expansion boards and motor drives.

5.2 Introduction to expansion boards
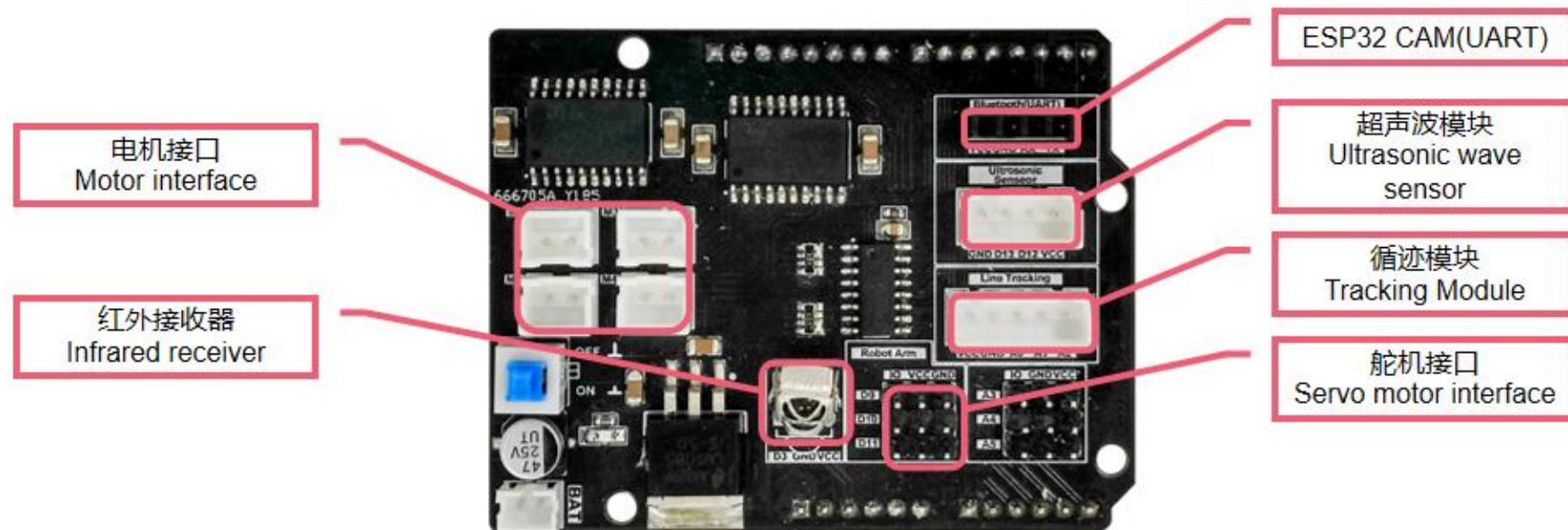
This expansion board extends the pins of the motherboard very well, and stably integrates the infrared receiver on the board. The motor, ultrasonic and tracking modules are all connected by plug-in terminals, which increases the firmness and reliability. Commonly used digital signal and analog signal ports have also been marked on the board and can be used for DIY.

5.3 Motor drive

Mecanum wheel:

Mecanum wheel is a wheel with a peripheral axle, generally divided into two types, one with a peripheral axle tilted to the left, and the other with a peripheral axle tilted to the right. These angled peripheral axles convert part of the wheel steering

force into a wheel normal force, so the car can achieve left and right translational motion.

Assembly points (top view):

The peripheral axle points to the center of the car
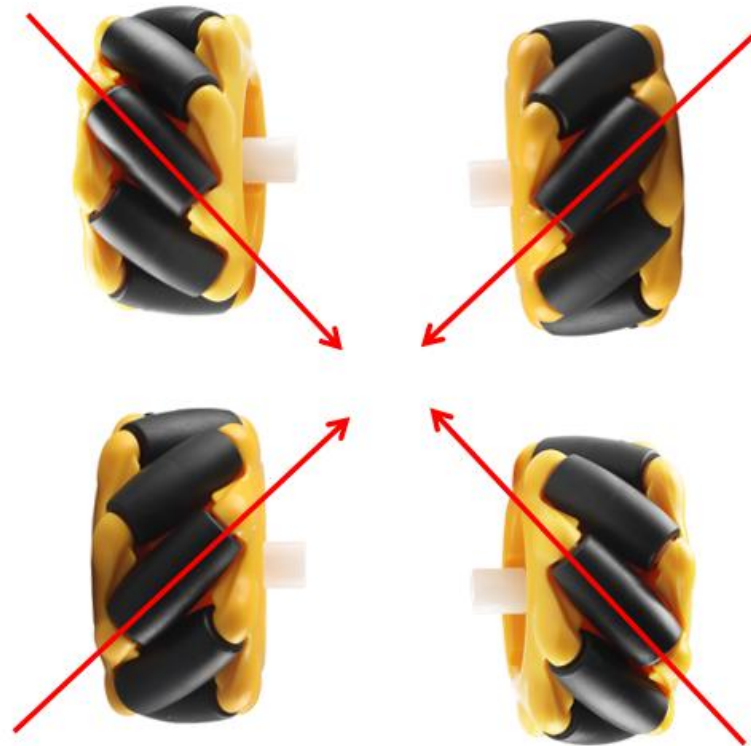
Movement principle:

Different rotation directions of the wheel correspond to left and right translation movements:

Left drift   Right drift
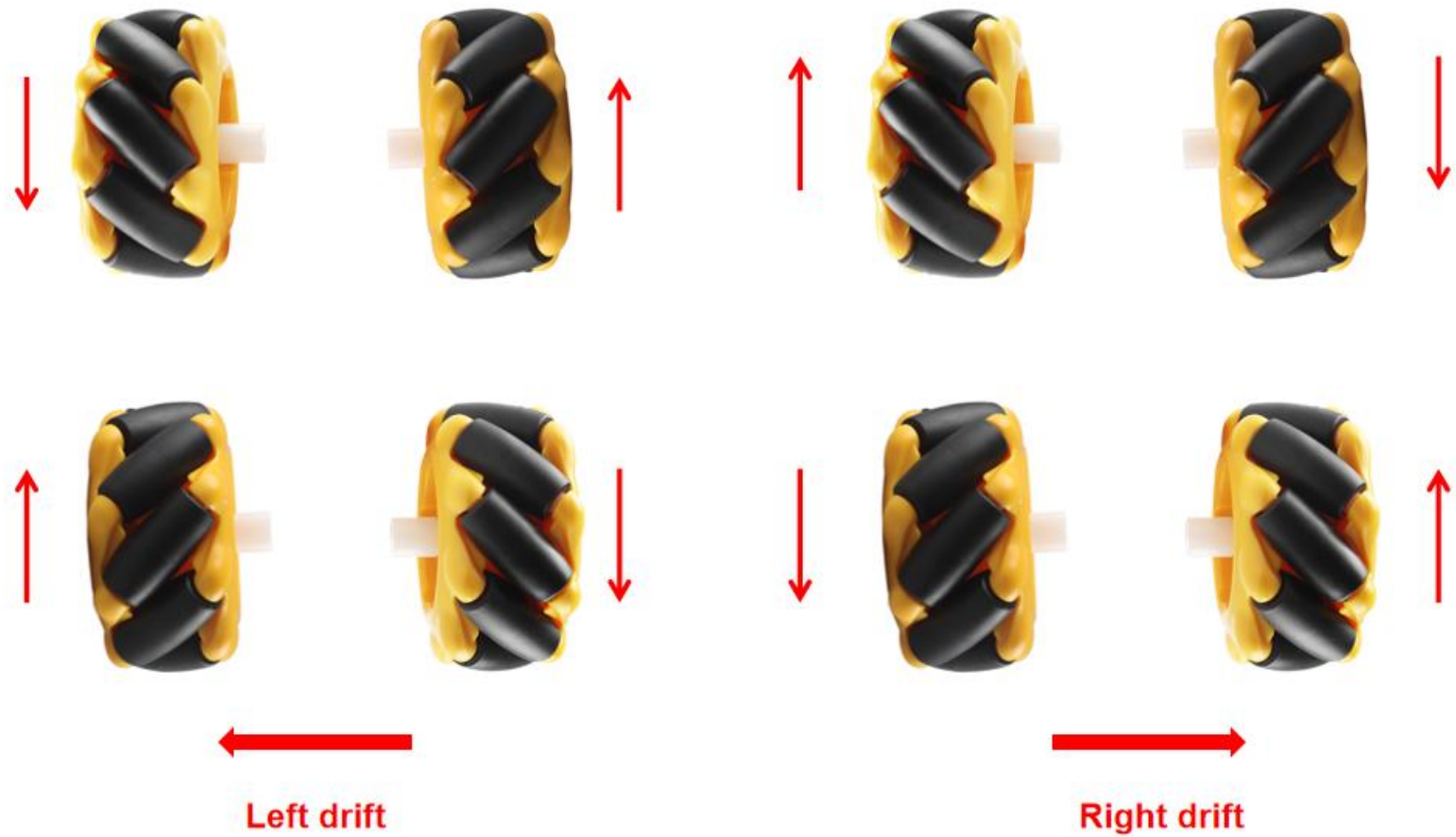
Different rotation directions of the wheel correspond to movement in the upper left direction and lower right direction:

Top left move
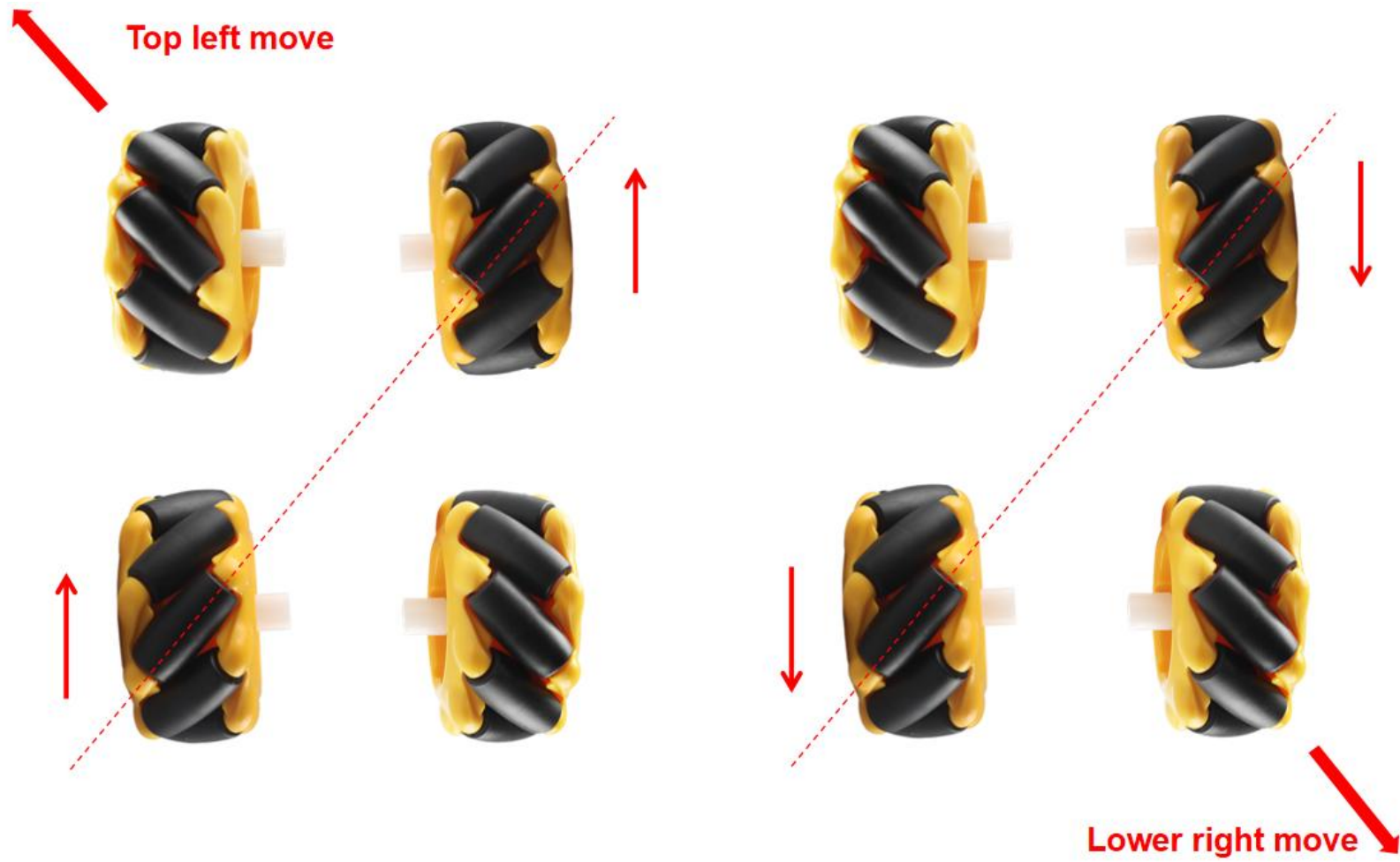
Lower right move

The different rotation directions of the wheel correspond to the movement in the lower left direction and the upper right direction:

Top right move

Lower left move
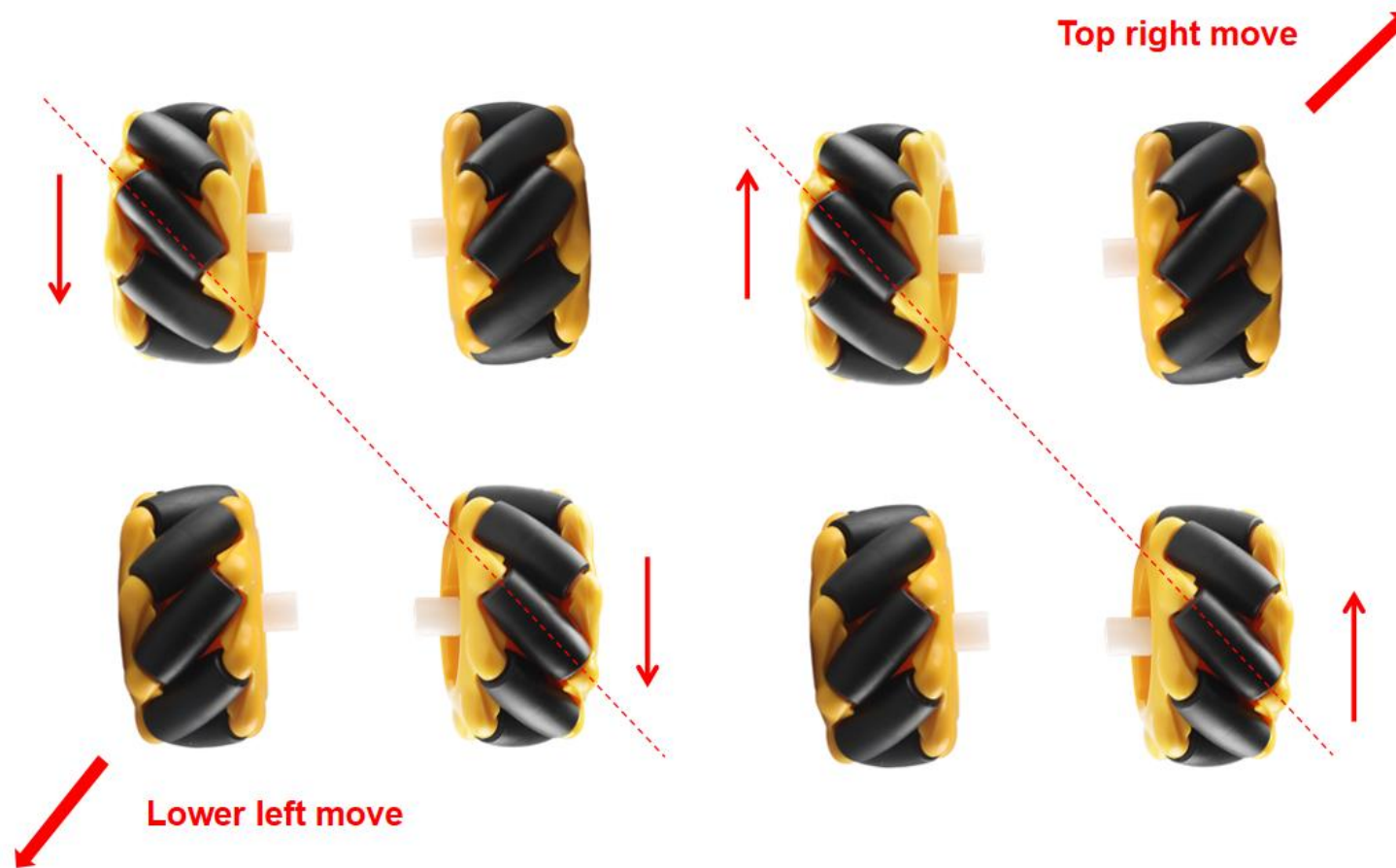
Different rotation directions of the wheel correspond to forward and backward movement:
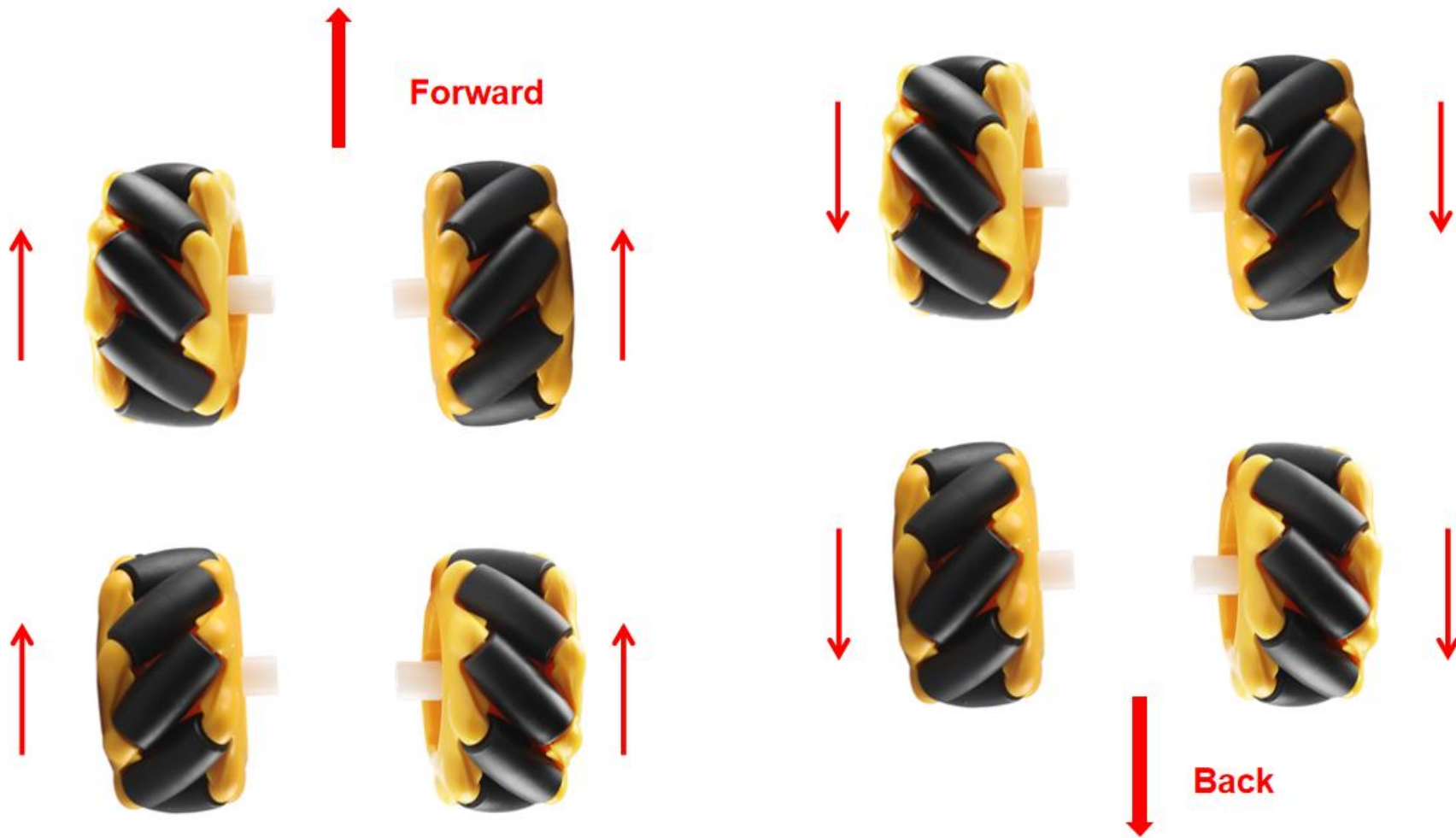
The different rotation directions of the wheel correspond to counterclockwise rotation and clockwise rotation:

Counterclockwise rotation

Rotate clockwise
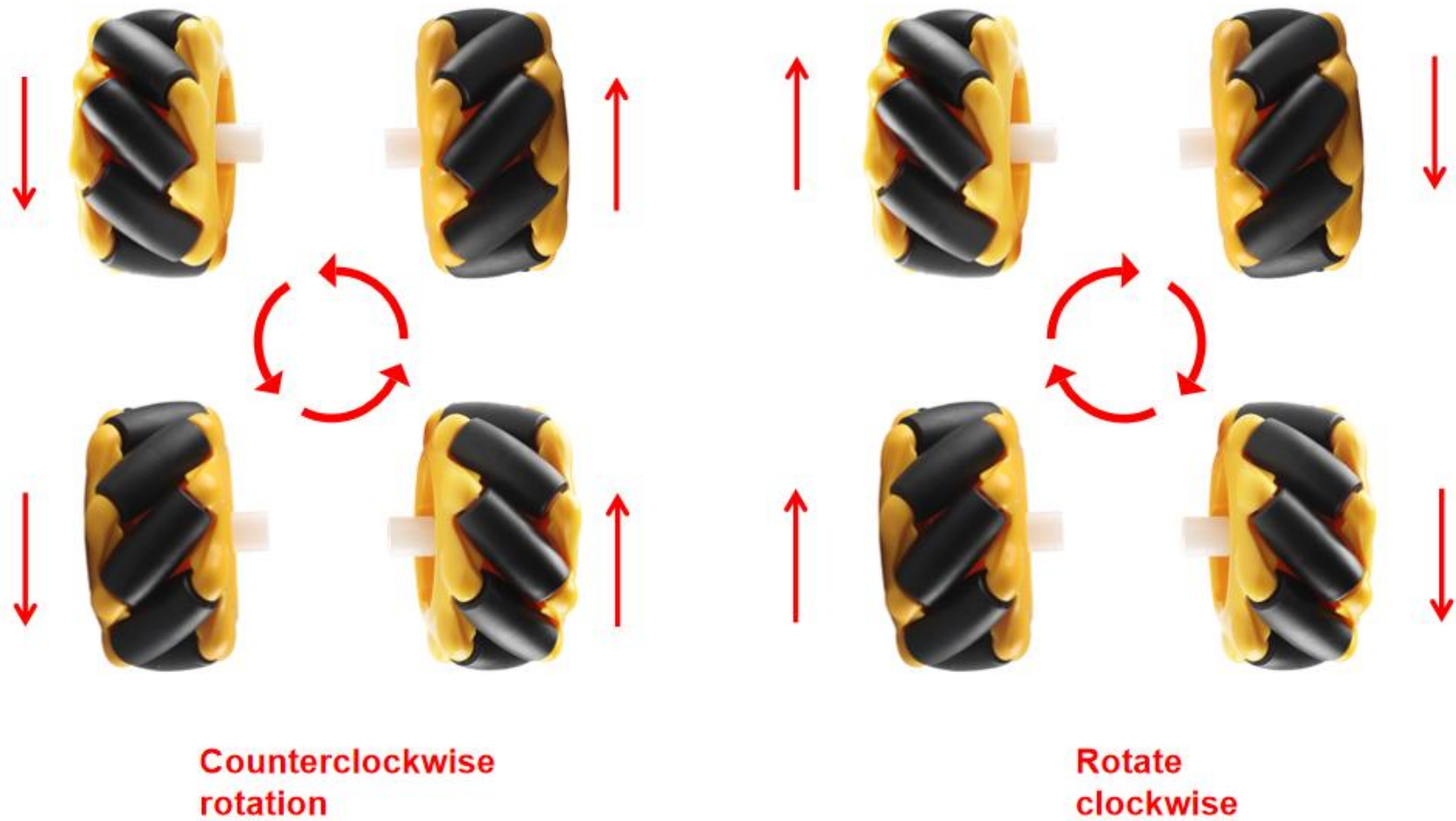
5.4 Code analysis

Open the program file (path: 2_Arduino_Code\2.1_Motor_Speed\2.1_Motor_Speed.ino)

Programming control principle:

The Pwm pin controls the wheel power (speed), and then controls the rotation direction of each motor through the 74HC595 chip pin.

Arduino's shiftOut function mainly acts on the 74HC595 chip;

main idea:

The high and low levels of each pin are controlled through decimal numbers 0 to 255 and 8-bit binary numbers;

Instructions:

```
76      digitalWrite(STCP_PIN, LOW);
77      shiftOut(DATA_PIN, SHCP_PIN, MSBFIRST, Dir);
78      digitalWrite(STCP_PIN, HIGH);
```

The shiftOut function has four parameters, and the first three parameters are defined and configured at the beginning. We only need to modify the value of value. At this time, the system will convert the decimal number into an 8-digit binary

number to control the high and low levels.

Define PWM pins and chip pins

```
2    // PWM control pin
3    #define PWM1_PIN          5
4    #define PWM2_PIN          6
5    // 74HCT595N Chip pins
6    #define SHCP_PIN          2                    // The displacement of the clock
7    #define EN_PIN            7                    // Can make control
8    #define DATA_PIN          8                    // Serial data
9    #define STCP_PIN          4                    // Memory register clock
```

Set the variable that stores the decimal encoded value

```
11   const int Forward      = 92;                  // forward
12   const int Backward     = 163;                 // back
13   const int Turn_Left    = 149;                 // left translation
14   const int Turn_Right   = 106;                 // Right translation
15   const int Top_Left     = 20;                  // Upper left mobile
16   const int Bottom_Left  = 129;                 // Lower left mobile
17   const int Top_Right    = 72;                  // Upper right mobile
18   const int Bottom_Right = 34;                  // The lower right move
19   const int Stop         = 0;                   // stop
20   const int Contrarotate = 172;                 // Counterclockwise rotation
21   const int Clockwise    = 83;                  // Rotate clockwise
```

Motor drive function, input two values, one is the encoding value to control the motor rotation direction and the PWM

power (speed) value

```
70    void Motor(int Dir, int Speed)
71    {
72        digitalWrite(EN_PIN, LOW);
73        analogWrite(PWM1_PIN, Speed);
74        analogWrite(PWM2_PIN, Speed);
75
76        digitalWrite(STCP_PIN, LOW);
77        shiftOut(DATA_PIN, SHCP_PIN, MSBFIRST, Dir);
78        digitalWrite(STCP_PIN, HIGH);
```

Ways to debug encoded values:

```
11    const int Forward      = 92;       // forward
12    const int Backward     = 163;      // back
13    const int Turn_Left    = 149;      // left translation
14    const int Turn_Right   = 106;      // Right translation
15    const int Top_Left     = 20;       // Upper left mobile
16    const int Bottom_Left  = 129;      // Lower left mobile
17    const int Top_Right    = 72;       // Upper right mobile
18    const int Bottom_Right = 34;       // The lower right move
19    const int Stop         = 0;        // stop
20    const int Contrarotate = 172;      // Counterclockwise rotation
21    const int Clockwise    = 83;       // Rotate clockwise
22
```

" Forward " variable value to 1 (converted to eight-digit binary to 0000 0001) as shown below , comment out other codes, and observe the motor rotation when calling the function.

```
11    const int Forward      = 1;                              // forward
12    const int Backward      = 163;                            // back
13    const int Turn_Left     = 149;                            // left translation

33    void loop()
34    {
35        /* Forward */
36        Motor(Forward, 250);
37        //delay(2000);
38        /* Backward */
```

You can see that only one motor is rotating, which means 0000 0001 is the code for the rotation direction of the motor. As shown in the figure below, when the variable is set to 2 (binary is 0000 0010), it represents another rotation state of another motor. When the variable is set to 4 (0000 0100 in binary), it represents another rotation state of another motor. In this way, we can infer the codes corresponding to 8 states in total for 4 motors * 2 forward and reverse rotation modes. Summarizing the data records, if you want the car to move forward, you must keep the four motors rotating forward, which is 01011100, which is converted into decimal 92. Finally, all corresponding codes for all motion states are obtained. In particular, the

potential controlling the same motor cannot be 1 (high level) at the same time, which will cause failure.

The specific corresponding coding table is as follows:

8 bit binary	1000 0000	0100 0000	0010 0000	0001 0000	0000 1000	0000 0100	0000 0010	0000 0001		
The decimal system	128	64	32	16	8	4	2	1		
State of the wheel	Upper left wheel back	Left upper wheel forward	Lower left wheel back	Lower left wheel forward	Lower right wheel forward	Upper right wheel forward	Upper right wheel back	Lower right wheel back	8 bit binary	decimal system
Forward	0	1	0	1	1	1	0	0	01011100	92
Back	1	0	1	0	0	0	1	1	10100011	163
left translation	1	0	0	1	0	1	0	1	10010101	149
Right translation	0	1	1	0	1	0	1	0	01101010	106
Upper left mobile	0	0	0	1	0	1	0	0	00010100	20
Lower left mobile	1	0	0	0	0	0	0	0	10000001	129
Upper right mobile	0	1	0	0	1	0	0	0	01001000	72
Lower right mobile	0	0	1	0	0	0	1	0	00100010	34
Counterclockwise	1	0	1	0	1	1	0	0	10101100	172
Rotate clockwise	0	1	0	1	0	0	1	1	01010011	83

Pass the value in the rightmost column into the variable Dir of the function shifOut to get different motion states.

```
76      digitalWrite(STCP_PIN, LOW);
77      shiftOut(DATA_PIN, SHCP_PIN, MSBFIRST, Dir);
78      digitalWrite(STCP_PIN, HIGH);
```

6. Tracking module

6.1 Description

This section mainly studies the line-following sensor module and its application.

6.2 Tracking module

tracking sensor is an infrared tracking sensor commonly used in manufacturing tracking smart cars. The tracker sensor uses ITR20001/T infrared reflection sensor. The infrared emitting diode of the ITR2001/T sensor continuously emits infrared rays. When the emitted infrared rays are reflected by objects, they are received by the infrared receiver and output an analog value. The output simulation value is related to object distance and object color. The position of the tracking line is determined by calculating the analog values of the three outputs.

The tracking sensor is located on the front of the car and consists of an infrared transmitting tube and an infrared receiving tube. The former is an LED that can transmit infrared light, and the latter is a photoresistor used to receive infrared light. Black surfaces have different light reflectivity than white surfaces. Therefore, the intensity of reflected infrared light received by a car on a black road is different from that on a white road, and the movement changes. Based on the principle of voltage division between series resistors, the path of motion is determined by inferring the color of the route beneath the car from the voltage of the sensor .

**Get the measurement values of the three-way tracking sensor:**

Open the code file (path: 2_Arduino_Code \ 3.1_Line_Tracking_Sensor \ 3.1_Line_Tracking_Sensor. I no)

Connect the car motherboard to the computer and burn the code. Use black tape to stick a black line on the table for testing (line width is about 1.5cm). Then place the car on the black line, click on the serial port monitor in the upper right corner of the Arduino IDE, and observe the test values.
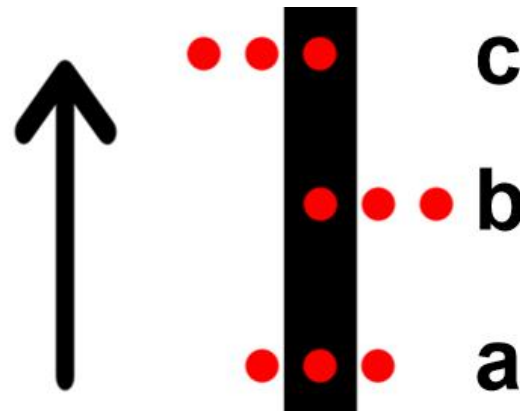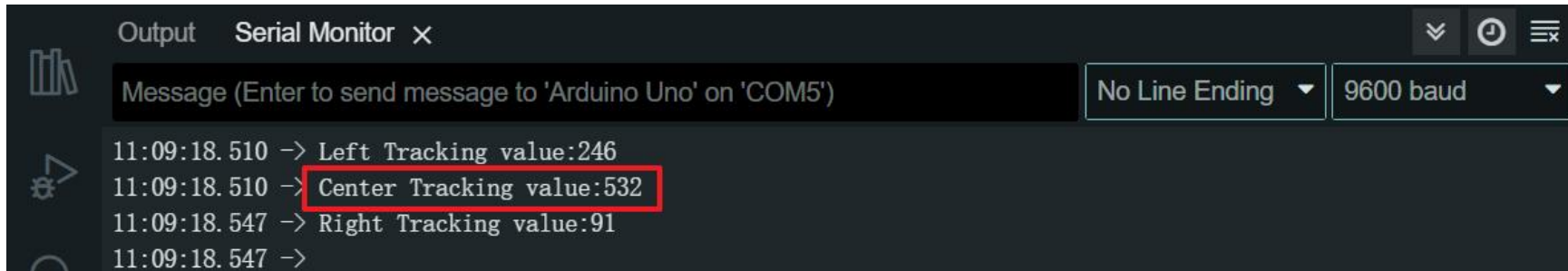
As shown in (b) above, the value obtained when the sensor on the left side of the tracking module detects the black line:
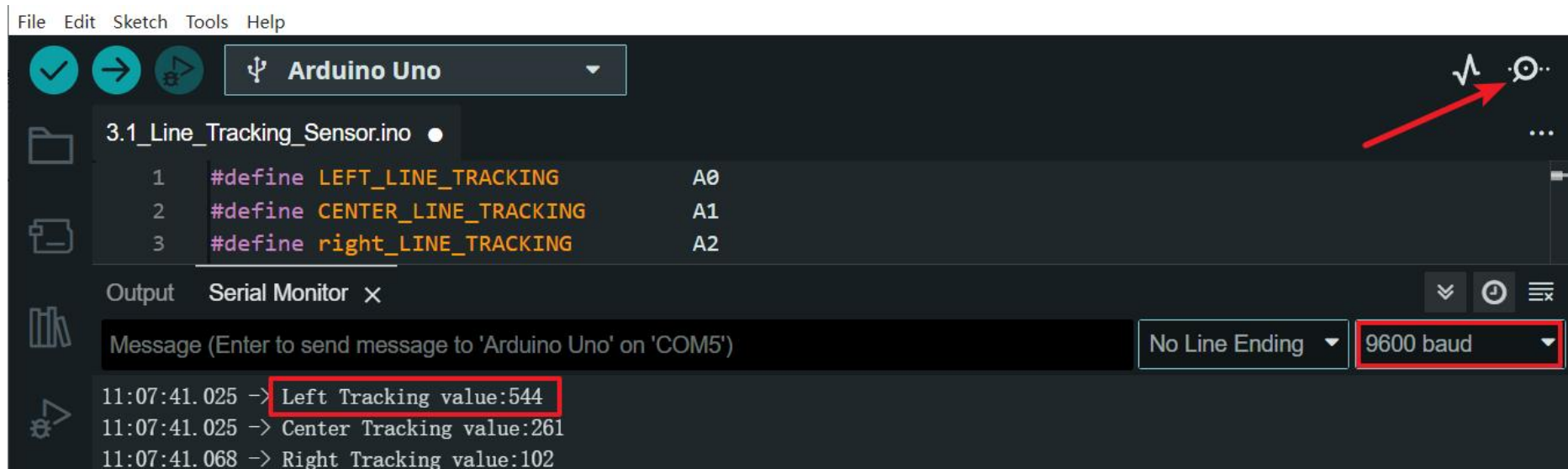
As shown in (a) above, the value obtained when the sensor in the middle of the tracking module detects the black line:

As shown in (c) above, the value obtained when the sensor on the right side of the tracking module detects the black line:

The result can be obtained: when the sensor detects a black line, the value is about 500~550, and when the sensor does not detect a black line, the value is less than 300. (Different materials, colors, tapes, and light affect the test results. The tests here do not represent all results)

6.3 Code analysis

Define the left, middle and right sensor pins A0/A1/A2 of the tracking module

```
1   #define LEFT_LINE_TRACKING          A0
2   #define CENTER_LINE_TRACKING        A1
3   #define right_LINE_TRACKING         A2
```

Three pin ports are set up as inputs

```
5   void setup()
6   {
7       Serial.begin(9600);
8       pinMode(LEFT_LINE_TRACKING, INPUT);
9       pinMode(CENTER_LINE_TRACKING, INPUT);
10      pinMode(right_LINE_TRACKING, INPUT);
11  }
```

The analog value reads the value obtained by the tracking module and then prints it out.

```
18  void Infrared_Tracing()
19  {
20      int Left_Tra_Value;
21      int Center_Tra_Value;
22      int Right_Tra_Value;
23      Left_Tra_Value = analogRead(LEFT_LINE_TRACKING);
24      Center_Tra_Value = analogRead(CENTER_LINE_TRACKING);
25      Right_Tra_Value = analogRead(right_LINE_TRACKING);
26      Serial.print("Left Tracking value:");
27      Serial.println(Left_Tra_Value);
28      Serial.print("Center Tracking value:");
29      Serial.println(Center_Tra_Value);
30      Serial.print("Right Tracking value:");
31      Serial.println(Right_Tra_Value);
32      Serial.println("");
33      delay(1000);
```

7. Tracking car

7.1 Description

Master the principle of line following and realize the function of the three-way tracking car through Arduino programming.

7.2 Tracking principle
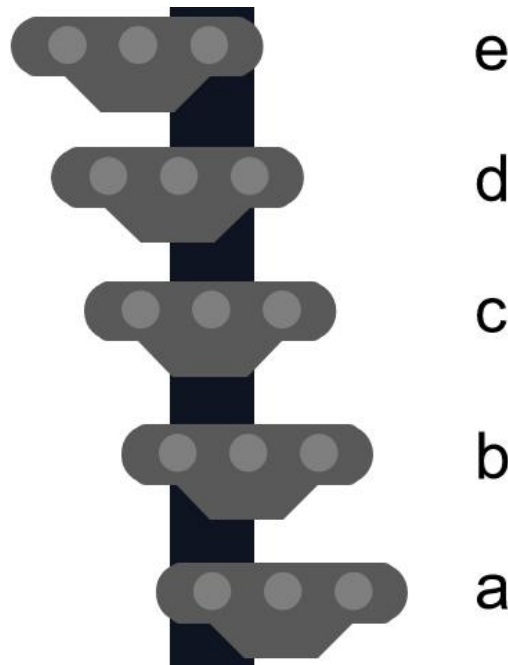
a→ Only the left sensor of the tracking module detects the black line. At this time, the car needs to turn to the left at a larger angle ;

b→ The left and middle sensors of the tracking module detect black lines at the same time , and the car needs to turn to the left at a smaller angle ;

c → Only the middle sensor of the tracking module detects the black line, and the car can drive straight at this time ;

d → The right and middle sensors of the tracking module detect black lines at the same time , and the car needs to turn to the right at a smaller angle ;

e → Only the right sensor of the tracking module detects the black line. At this time, the car needs to turn to the right at a larger angle ;

Combining the above information, we can see the tracking principle of the tracking car . After the car is started, the tracking module only needs to sense the black lines on the road and take appropriate actions as

needed. There are many more complex algorithms, such as PID. Therefore, after implementing the tracking function, you can learn more algorithms to control the car yourself.

Prepare the insulating tape (black tape) lines:

First, we need to make a runway ourselves. We can stick black tape on a flat and clean floor. It's best to let the trajectory angle change slowly and not change too much at once. Because if the angle of the turn is too large, the car may run off the track. However, if you want to make it more difficult, you can make the angle of the turn larger. The size of the runway is generally not less than 40*60 cm.

（1）Curved parts of the line should be transitioned as smoothly as possible, otherwise the car is more likely to overrun the track.

（2）Line tracing scenes can be made from black and white tape to design different walking paths.

（3）In addition to line tracing, we can also develop other procedural line tracing principles. For example, confining a car to a certain area and letting it move around will be covered later .
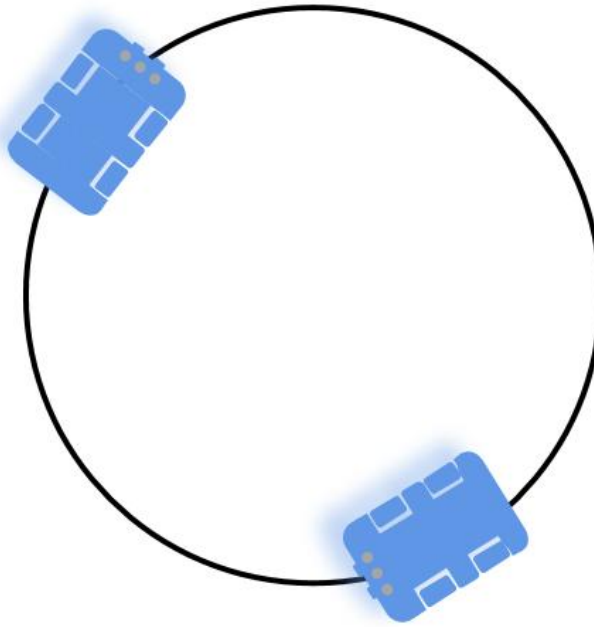
7.3 Code analysis

Open the code file (path: 2_Arduino_Code\3.2_Line_Tracking_Smart_Car\3.2_Line_Tracking_Smart_Car.ino)

Define the three-way tracking measurement numerical variable and the reference value variable "Black_Line" used to compare whether a black line is detected.

```
26    int Left_Tra_Value;
27    int Center_Tra_Value;
28    int Right_Tra_Value;
29    int Black_Line = 500;
```

Compare the three-channel tracking sensor value with the reference value Black_Line to obtain the five situations described

in the tracking principle:

Case c only the middle sensor detects the black line:

```
if (Left_Tra_Value < Black_Line && Center_Tra_Value >= Black_Line && Right_Tra_Value < Black_Line)
{
    Motor (Forward, 175 );
}
```

In case b, the left and middle sensors detect black lines at the same time :

```
else if (Left_Tra_Value >= Black_Line && Center_Tra_Value >= Black_Line && Right_Tra_Value < Black_Line)
{
    Motor (Contrarotate, 165 );
}
```

In case a, only the left sensor detects the black line:

```
else if (Left_Tra_Value >= Black_Line && Center_Tra_Value < Black_Line && Right_Tra_Value < Black_Line)
{
    Motor (Contrarotate, 190 );
```

```
}
```

In case e, only the right sensor detects the black line:

```
    else if (Left_Tra_Value < Black_Line && Center_Tra_Value < Black_Line && Right_Tra_Value >= Black_Line)
{

    Motor (Clockwise, 190 );

}
```

In case d, the right and middle sensors detect black lines at the same time :

```
    else if (Left_Tra_Value < Black_Line && Center_Tra_Value >= Black_Line && Right_Tra_Value >= Black_Line)
{

    Motor (Clockwise, 165 );

}
```

Coupled with the situation where all sensors detect black lines:

```
    else if (Left_Tra_Value >= Black_Line && Center_Tra_Value >= Black_Line && Right_Tra_Value >= Black_Line)
{

    Motor (Stop, 0 );

}
```

Execute the Motor() function with parameters

```
76    void Motor(int Dir, int Speed)
```

8. Draw the ground as a prison car

8.1 Description

This book mainly deepens the understanding of the tracking module through an interesting "drawing the ground as a prison" project, and learns to apply the tracking module more flexibly in different scenarios.

Drawing the ground as a prison: As the name suggests, it is to circle an area on the ground and allow the car robot to run freely within the circle without rushing out of the area, just like staying in a dungeon.

8.2 Circle the dungeon area on the ground

Prepare black tape and paste an area with a radius of no less than 40cm on a clean and flat ground. Note that the circled area must be closed, otherwise the car will rush out of the gap.
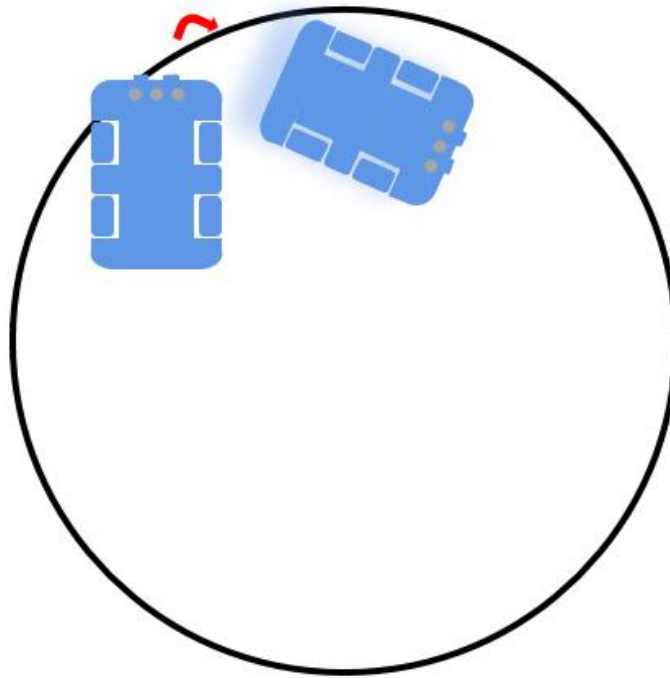
8.3 Code analysis

Open the code file (path: 2_Arduino_Code\3.3_cage\3.3_cage.ino)

Comparing the three-channel tracking sensor value with the reference value Black_Line, 5 situations are obtained:

None of the three sensors detected the black line, that is, they were all smaller than Black_Lin.

```
    if (Left_Tra_Value < Black_Line && Center_Tra_Value < Black_Line && Right_Tra_Value < Black_Line)
{

    Motor (Forward, 150 );
}
```

Only the left sensor detects the black line, that is, the left side of the car is almost beyond the black line. At this time, you

need to back up and then turn right (clockwise) to adjust the direction of walking.

```
else if (Left_Tra_Value >= Black_Line && Center_Tra_Value < Black_Line && Right_Tra_Value < Black_Line)
{

    Motor (Backward, 150 );
    delay ( 200 );
    Motor (Clockwise, 160 );
    delay ( 500 );
}
```

When the left and middle sensors detect black lines, you need to step back and then turn right to adjust the direction of

walking at a larger angle;

```
    else if (Left_Tra_Value >= Black_Line && Center_Tra_Value >= Black_Line && Right_Tra_Value < Black_Line)
{

    Motor (Backward, 150 );
    delay ( 200 );
    Motor (Clockwise, 160 );
    delay ( 600 );
}
```

The same goes for the sensor on the right. After detecting the black line, it will back up and then turn left (counterclockwise)

to adjust the direction of walking;

```
    else if (Left_Tra_Value < Black_Line && Center_Tra_Value < Black_Line && Right_Tra_Value >= Black_Line)
{
    Motor (Backward, 150 );
    delay ( 200 );
    Motor (Contrarotate, 160 );
    delay ( 500 );
}else if (Left_Tra_Value < Black_Line && Center_Tra_Value >= Black_Line && Right_Tra_Value >= Black_Line){
    Motor (Backward, 150 );
    delay ( 200 );
    Motor (Contrarotate, 160 );
    delay ( 600 );
}
```

Otherwise, when the three-way tracking module detects the black line, it will rotate clockwise by default to adjust the

direction.

```
    else{
    Motor (Backward, 150 );
    delay ( 600 );
    Motor (Clockwise, 160 );
    delay ( 500 );
}
```

9. Ultrasonic ranging

9.1 Description

This section mainly focuses on understanding the working principle of the ultrasonic module, mastering the connections of the ultrasonic circuit diagram, and learning how to measure the distance of the ultrasonic module through programming.

9.2 Ultrasonic sensor

Sound waves are produced by vibrations and can travel at different speeds in different media. Ultrasonic waves have the advantages of strong directivity, slow energy loss, and long propagation distance in media, and are often used for distance measurement. For example, distance meters, liquid level measuring instruments, etc. can all be realized through ultrasonic waves.

Electrical parameters	HC-SR04 Ultrasonic module
Working voltage	DC-5V
Working current	15mA
Working frequency	40KHz
Maximum range	4m
Minimum range	2cm
Measuring angle	15 °
Input trigger signal	10 US TTL pulse
Output echo signal	Output TTL level signal, proportional to the range
Size	45*20*15

Ultrasonic ranging is a non-contact detection method

Especially when used in airborne ranging, due to the slow wave speed in the air, the echo signal contained along the

direction of structural information propagation is easy to detect and has very high resolution, so its accuracy is higher than other methods; while ultrasonic sensors have a simple structure , small size, reliable signal processing and other characteristics. The use of ultrasonic detection is often faster, more convenient, simpler to calculate, easier to achieve real-time control, and can meet industrial practical requirements in terms of measurement accuracy.

There are many methods of ultrasonic distance measurement. The principle of this system in ultrasonic measurement is to detect the transmission time of ultrasonic waves from the ultrasonic transmitter through the gas medium to the receiver. Multiply this time by the speed of sound in the gas to find the distance the sound travels.

The ultrasonic transmitter emits ultrasonic waves in a certain direction, and the MCU starts timing at the same time. The ultrasonic waves are launched in the air and return immediately when encountering obstacles on the way. The ultrasonic receiver stops timing immediately after receiving the reflected waves.

T recorded by the timer , the distance (s) from the launch point to the obstacle can be calculated .

Formula: S = VT/2

Four factors limit the maximum measurable distance of an ultrasound system: the amplitude of the ultrasound wave, the

texture of the reflector, the angle between the reflected and incident sound waves, and the sensitivity of the receiving transducer. The ability of the receiving transducer to directly receive the acoustic pulse will determine the minimum measurable distance.

The trigger signal input terminal (TRIG) will input a high-level signal of more than 10 microseconds. After receiving the signal, the ultrasonic transmitter will automatically send eight 40Hz square waves. At the same time, the timer will start. When the sensor receives the echo, it stops timing and outputs the echo signal.

Based on the time interval, the distance can be calculated by the formula: distance = (high level time * speed of sound) / 2 .

9.3 Code analysis

Open the code file (path: 2_Arduino_Code\4.1_Ultrasonic_Sensor_Module)

Define ultrasonic control pins

```
1   // Ultrasonic control pin
2   const int Trig = 12;
3   const int Echo = 13;
```

Get the ranging distance function getDistance()

```
16    float getDistance()
17    {
18        digitalWrite(Trig, LOW);
19        delayMicroseconds(2);
20        digitalWrite(Trig, HIGH);
21        delayMicroseconds(10);
22        digitalWrite(Trig, LOW);
23        float distance = pulseIn(Echo, HIGH) / 58.00;
24        delay(10);
25        return distance;
26    }
```

The pulseIn function is actually a function that measures pulse width. The default unit is us. In other words, what pulseIn measures is the time elapsed from the transmission to the reception of the ultrasonic wave.

The propagation speed of sound in dry, 20 degrees Celsius air is approximately 343 m / s , which means that it takes 29.15 microseconds to propagate 1 centimeter. Transmitting plus receiving takes double the time, so it's about 58.3 microseconds, with a value of 58.

Printout of ultrasonic measured distances

```
11  void loop()
12  {
13      Ultrasonic_Sensor_Module();
14  }
15
16  void Ultrasonic_Sensor_Module()
17  {
18      int Distance = 0;
19      Distance = getDistance();
20      Serial.print("Distance:");
21      Serial.print(Distance);
22      Serial.println("CM");
23      delay(100);
```

10. Ultrasonic obstacle avoidance car

10.1 Description

This section mainly focuses on learning and consolidating the practical application of ultrasonic waves. Based on the content of the previous section, we will learn the principles of ultrasonic obstacle avoidance and realize the obstacle avoidance function of the obstacle avoidance car through programming.

10.2 Principle of obstacle avoidance

When the distance detected by the ultrasonic wave is less than the set distance, it is judged that the car has encountered an obstacle, and then the obstacle avoidance program is triggered , causing the car to retreat and then turn left or right to avoid the obstacle, thereby realizing automatic driving of the car to avoid obstacles.
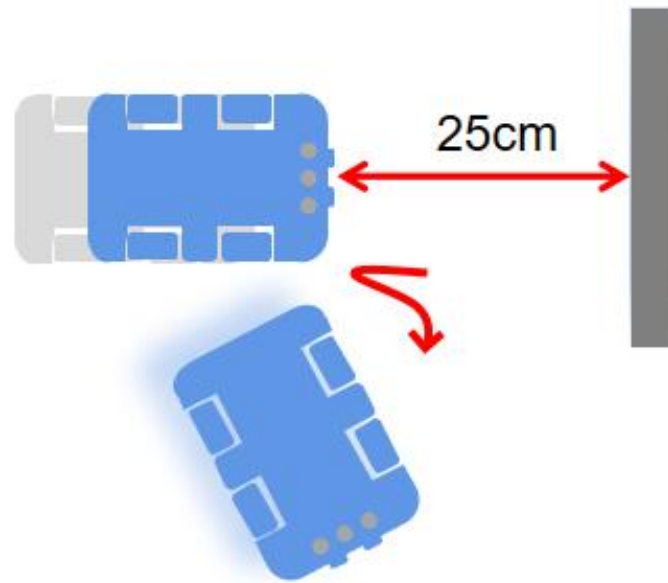
10.3 Code analysis

Open the code file (path: 2_Arduino_Code\4.2_Ultrasonic_Obstacle_Avoidance_Robot_Car)

Save the ultrasonic measurement distance obtained by the SR04 function into the variable Avoidance_distance. The first "if" condition determines whether the distance is less than or equal to 25. If the distance between the car and the object in front is less than or equal to 25, it will be determined whether it is less than 15. If yes, the internal Motor function will be

executed to let the car stop, retreat and then turn clockwise. Otherwise Just turn counterclockwise. If the first "if"

condition is not met, keep going straight.

```
52    Avoidance_distance = SR04(Trig, Echo);
53    if (Avoidance_distance <= 25)
54    {
55        if (Avoidance_distance <= 15)
56        {
57            Motor(Stop, 0);
58            delay(100);
59            Motor(Backward, 180);
60            delay(600);
61            Motor(Clockwise, 180);
62            delay(200);
63        }else
64        {
65            Motor(Stop, 0);
66            delay(100);
67            Motor(Backward, 180);
68            delay(300);
69            Motor(Contrarotate, 180);
70            delay(600);
71        }
72    }else
73    {
74        Motor(Forward, 180);
```

11.Follow the car

11.1 Description

After learning about the application of ultrasonic waves in practical obstacle avoidance, this section will let us learn about a following system. Through ultrasonic ranging in front of the car, it always follows the object in front at a certain distance.

11.2 Schematic diagram of following principle
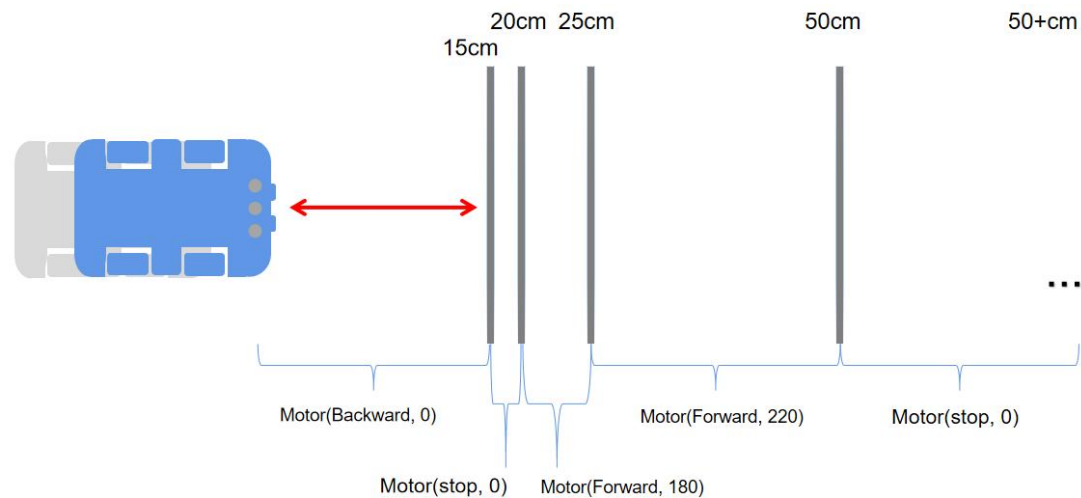
11.3 Combine the above picture with code analysis

file (path: 2_Arduino_Code \ 4.3_Ultrasonic_Follow)

When the ultrasonic wave in front of the car detects that the distance to the object in front is 20~25cm, the car moves

forward with an analog power of 180;

```
else if ( 20 <= Avoidance_distance && Avoidance_distance <= 25 )
{
    Motor (Forward, 180 );
}
```

When the distance between the car and the object in front is 25~50cm, the car moves forward with an analog power of

220 values;

```
else if ( 25 <= Avoidance_distance && Avoidance_distance <= 50 )
{
    Motor (Forward, 220 );
}
```

When the distance between the car and the object in front is less than 15, the car retreats with 200 analog power;

```
if (Avoidance_distance < 15 ){
    Motor (Backward, 200 );
}
```

Otherwise, the car will stop when the distance between the car and the object in front exceeds 50cm or there is no object;

```
    else {
        Motor (Stop, 0 );
}
```

12. Infrared remote control car

12.1 Description

Infrared remote control is a widely used remote control method. The car is already equipped with an infrared receiver, thus allowing it to be controlled using an infrared remote control. This section mainly focuses on understanding the infrared remote control and receiver , the principle of infrared remote control and learning the implementation ideas of infrared remote control programming. We have upgraded the function and added the operation of the robotic arm controlled by the servo motor. Come and experience the charm of controlling the robotic arm.

infrared receiver

Infrared remote control
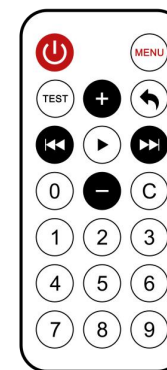
Principle of infrared remote control

The universal infrared remote control system consists of two parts: sending and receiving. The sending part is composed of infrared remote control, and the receiving part is composed of infrared receiving tube. The signal sent by the infrared remote control is a series of binary pulse codes. In order to avoid interference from other infrared signals during wireless transmission, it is generally necessary to modulate at a given carrier frequency and then transmit through an infrared emitting phototransistor. The infrared receiving tube filters out other noise waves, receives only the signal of a given frequency, and restores it to a demodulated binary pulse code. The built-in receiving tube converts the light signal sent by the infrared light-emitting diode, amplifies the signal through the amplifier in the IC, and restores the original code sent by the remote control through automatic gain control, band-pass filtering, demodulation, and wave formation, and outputs the signal through the infrared receiving module Pins identify the circuits that enter an appliance.

The encoding scheme that matches the infrared remote control protocol is: NEC protocol. Next, let us understand what the NEC protocol is.

(1) 8 address bits, 8 sequence bits address bits and sequence bits are transmitted twice to ensure reliability

(3) Pulse position modulation

(4) The carrier frequency is 38 kHz

(5) The time for each bit is 1.125 ms or 2.25 ms

12.3 Infrared remote control settings

In the car experiment, we need to control the car to move in all directions and move the robotic arm , which means we need

remote control buttons and settings to send and receive information.

Obstacle avoidance mode

Tracking mode

Follow mode

Claws open

Chassis left pendulum

Claws closed

Chassis right pendulum

Arm forward

Arm backwards
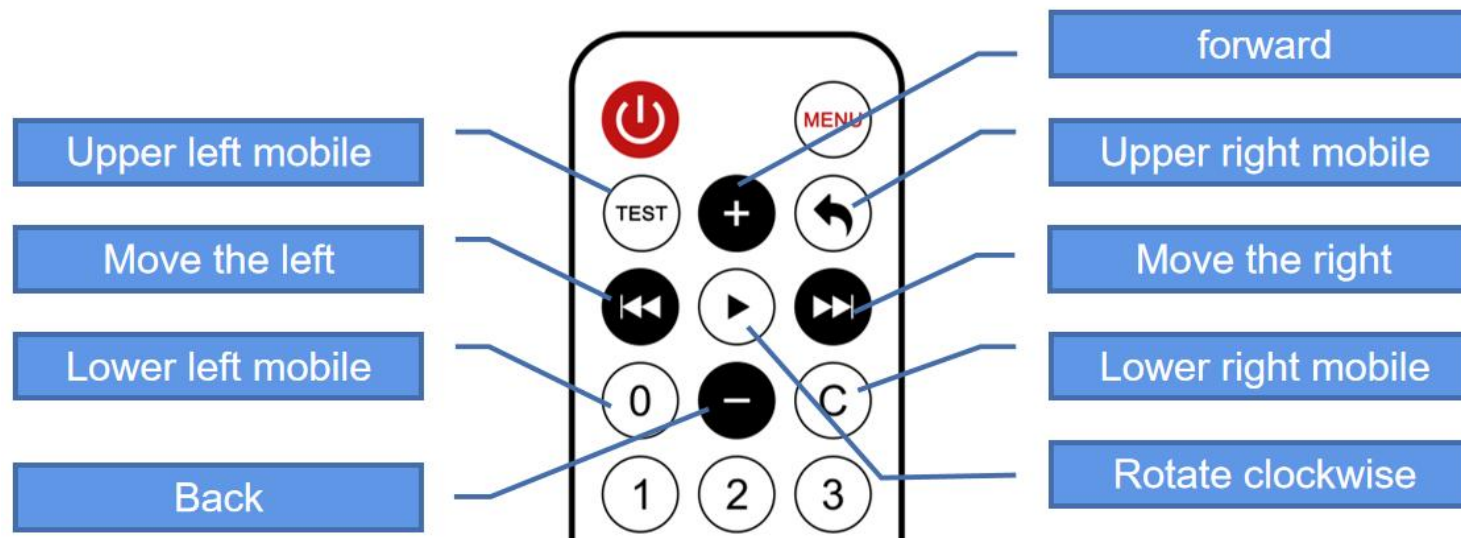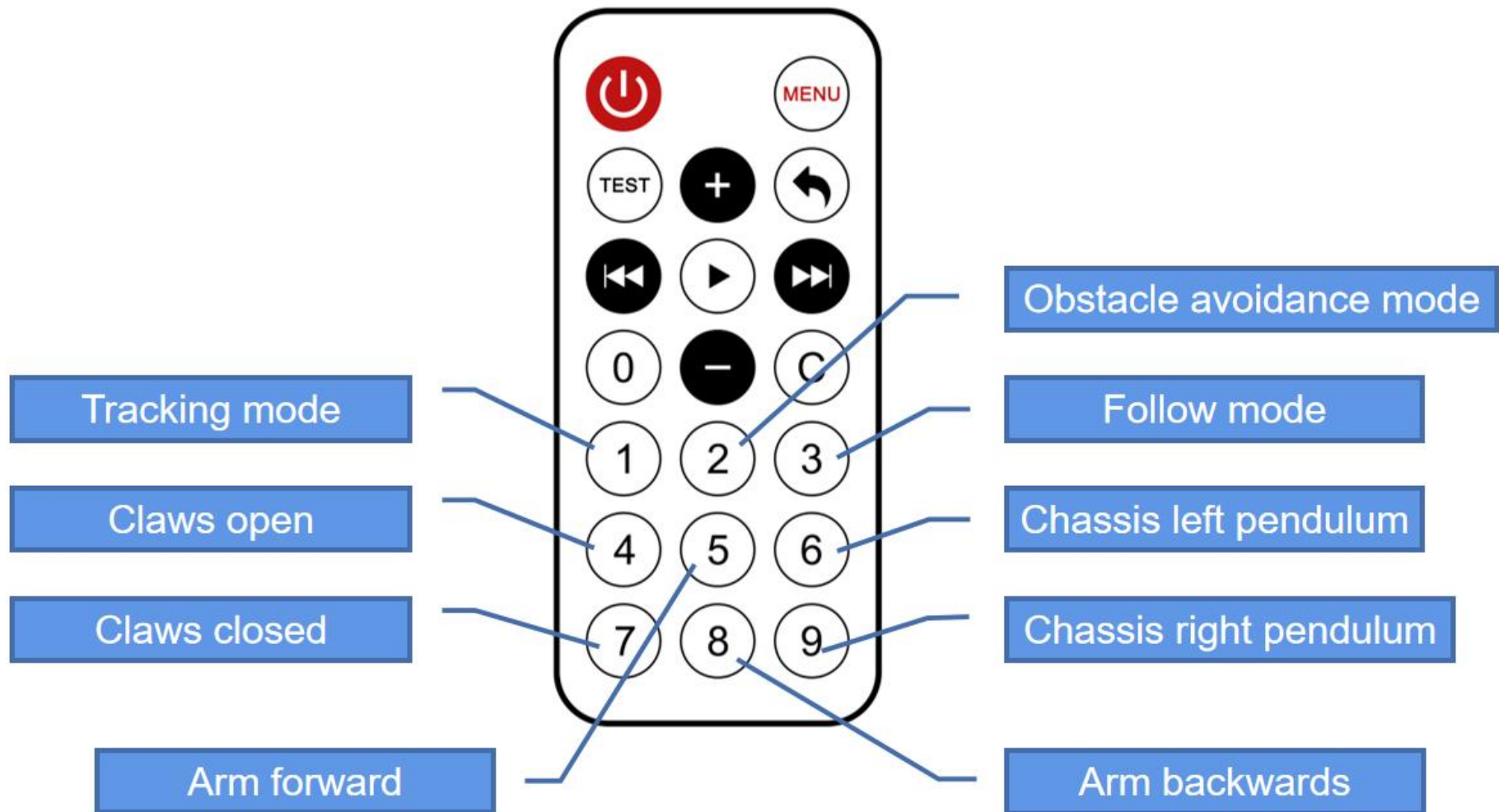
12.4 Code analysis:

Open the code file with *Arduino* IDE (path: " *2_Arduino_Code \ 5_IRID \ 5_IRID.ino* ")

Define the pins of the infrared receiver

```
// Infrared receiving control pin
#define RECV_PIN        3
```

Enable infrared module

```
IRremote IR ( RECV_PIN );
```

Initialize tracking, obstacle avoidance and following modes to off

```
boolean Line_tracking_Function_flag = false ;
boolean Avoidance_Function_flag = false ;
boolean Following_Function_flag = false ;
```

Receive the information sent by the infrared remote control for analysis and judgment, and trigger the car to implement various actions, including switching between the other two control modes. Among them, IR.getIrKey() and IR.getCode() are well encapsulated in the IR_remote.cpp file, and you can get the corresponding data by calling them directly.

```
switch ( IR . getIrKey ( IR . getCode (), IR_TYPE_EM))
{
        case EM_IR_KEYCODE_UP: // Forward
            Motor (Forward, 200 );
            delay ( 200 );
            break ;
        case EM_IR_KEYCODE_DOWN: // Backward
            Motor (Backward, 200 );
            delay ( 200 );
            break ;
```

5_IRID.ino | IR_remote.cpp | IR_remote.h | Keymap.cpp | keymap.h | launch.json

```
485      *      None
486      */
487     unsigned char IRremote::getCode()
488     {
489       irIndex = 0;
490       loop();
491       return irRead;
492     }
493     String IRremote::getKeyMap(byte keycode, byte ir_type)
494   > {···
503     }
504
505     byte IRremote::getIrKey(byte keycode, byte ir_type)
506     {
507         byte i;
```

The functions corresponding to each function

car moving

```
void Motor ( int Dir , int Speed )
```

Tracking function

```
void Line_tracking_Function ()
```

follow function

```
void Following_Function ()
```

Obstacle avoidance function

```
void Avoidance_Function ()
```

13.Bluetooth remote control

13.1 Description

Bluetooth remote control is a very convenient and efficient control method. The car is already equipped with a Bluetooth

module , so it can be controlled through a Bluetooth APP . This section mainly focuses on understanding the principles of

Bluetooth remote control, being able to wirelessly control your car in a specific space and learning about the programming

implementation ideas of Bluetooth remote control.

13.2 BT05 Bluetooth
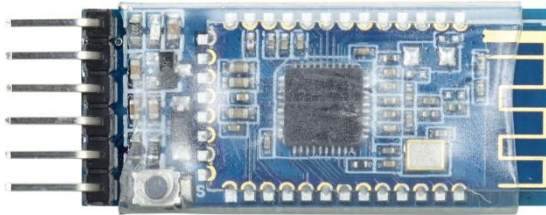
Bluetooth is a wireless technology standard used in various fields such as industry, science and medicine to exchange data

at short distances between different devices using short-wave ultra - high frequency radio waves in the radio frequency band

(2.400 to 2.485 GHz) . The car is equipped with a BT05 Bluetooth, and a Bluetooth APP for remote control is prepared in the data package: TSCINBUNY.apk. This Bluetooth has 4 pins and must be connected correctly to function, otherwise the Bluetooth will be damaged. Note that one side of the Bluetooth has been marked, and the pin placement is as follows:

The bluetooth module	Expansion board
RXD	D13
TxD	D12
GND	GND
VCC	VCC

Because Bluetooth will occupy the RX/TX port, unplug the Bluetooth before uploading the code!

13.3 Add library files

Before uploading the code, make sure the "MsTimer2" library file is installed. If not, please refer to the tutorial document in the 1_Get_start folder.

1_Get_start-准备	2023/11/10 9:39	文件夹
2_Arduino_Code-代码文件	2023/11/9 16:50	文件夹
3_APP	2023/11/9 16:56	文件夹

13.4 Code analysis

Open the code file (path: 2_Arduino_Code\6_BlueTooth\6_BlueTooth.ino)

In the previous section, we defined components such as motors and sensors accordingly. In this Bluetooth remote control program, we mainly focus on the analysis of Bluetooth remote control and communication.

Define and assign Bluetooth data packets to send and receive

```
byte TX_package [ 5 ] = { 0xA5 , 0 , 0 , 0 , 0x5A };
byte RX_package [ 10 ] = { 0 };
```

Define the ultrasonic module and tracking module variables, the initial angle of the servo motor, define the memory action and the Boolean values of each automatic mode

```
39    int UT_distance = 0;
40    int Serialcount = 0;
41    int claw_degrees = 135;
42    int arm_degrees = 90;
43    int base_degrees = 90;
44    boolean menory_action_flag;
45    boolean Line_tracking_Function_flag = false;
46    boolean Avoidance_Function_flag = false;
47    boolean Following_Function_flag = false;
48    boolean Jail_Function_flag = false;
```

Define the action positions and action step numbers of the three servo motors of the robotic arm

```
int actions_count = 0 ;
int auto_count;
int claw_read_degress [ 20 ] = { 0 , 0 , 0 };
int arm_read_degress [ 20 ] = { 0 , 0 , 0 };
int base_read_degress [ 20 ] = { 0 , 0 , 0 };
```

Define the structure data for receiving Bluetooth

```
typedef struct
{
byte mode1; // Bit0: free mode;Bit1: tracking mode;Bit2: Obstacle avoidance mode;Bit3: Follow mode;
// Bit4: Dungeon Mode;Bit5: Save button;Bit6: Automatic button;Bit7: empty
byte mode2; // Bit0: fluke;Bit1: closed claw;Bit2: clockwise rotation;Bit3: reverse;
    char x_axis = 0 ; // Store variables on the X axis
    char y_axis = 0 ; // Store the variables on the Y axis
```

```
byte C_speed = 127 ; // Speed of storage cart
    char x_Base = 0 ; // Store the steering gear on the X axis
    char y_Arm = 0 ; // Store the steering gear on the Y-axis
} rx_buffer ;

rx_buffer RX_buffer;
```

Initialize and set Timer2Isr() and setup()

```
void Timer2Isr ()
{
    sei ();
UT_distance = SR04 (Trig, Echo);
}
void setup (){
    pinMode (SHCP_PIN, OUTPUT);
    pinMode (EN_PIN, OUTPUT);
```

Communication between Bluetooth remote control APP and smart car

```
    TX_Information (UT_distance, auto_count); // Send ultrasonic data
    RX_Information (); // Receiving Bluetooth data
```

The received remote control commands are analyzed and then different functional modes are executed, such as tracking,

obstacle avoidance, dungeon, etc.

```
121     switch (RX_buffer.mode1)
122     {
123         case 0x02:        //tracking mode
124             Motor(Stop, 0);
125             Line_tracking_Function();
126             delay(10);
127             break;
128         case 0x04:        //Obstacle avoidance mode
129             Motor(Stop, 0);
130             Avoidance_Function();
```

4 functions with different functional modes

```
154    void Line_tracking_Function()      // tracking mode
155  > {...
192    }
193
194    void Avoidance_Function()          // Obstacle avoidance mode
195  > {...
291    }
292
293    void Following_Function()          // Follow the pattern
294  > {...
324    }
325
326    void Jail_Function()               // Dungeon mode
327  > {...
474    }
```

The action memory function auto_doit() initializes the variable actions_count of the recorded steps to zero, and then starts

recording the automated program steps.

```
void auto_doit () // Automatic mode
{
    if ( 0 != auto_count)
{
menory_action_flag = true ;
}
actions_count = 0 ;
```

The free control mode function controls movement in all directions and each function button to control the action of the

robotic arm based on the information sent by the APP and stored in the RX_buffer.

```
void free_mode () // free mode
{
    if ( RX_buffer . x_axis >= - 30 && RX_buffer . x_axis <= 30 && RX_buffer . y_axis >= 30 ) //Forward
{
        Motor (Forward, RX_buffer . C_speed );
        delay ( 5 );
}
```

Read the saved action step record, the number of records is less than or equal to 19

```
void read_degress ()
{
    if (actions_count <= 19 )
{
        claw_read_degress [( int )((actions_count + 1 ) - 1 )] = clawservo . read ();
        delay ( 50 );
        RX_Information ();
```

Execute the previously recorded action function auto_do(), initialize the variable actions_count of the recorded steps to zero, and then repeat the automation program steps.

```
void auto_do ()
{
    if ( 0 != auto_count)
{
menory_action_flag = true ;
}
actions_count = 0 ;
```

Like the previous section on infrared remote control, motor drive, ultrasonic module and Bluetooth sending and receiving functions are also required.

```
879
880    void Motor(int Dir, int Speed)        // Motor drive
881  > {...
889    }
890
891    float SR04(int Trig, int Echo)        // Ultrasonic distance measurement
892  > {...
901    }
902
903    void TX_Information(byte dis, byte act)     // Sending data packets
904  > {...
912    }
913
914    void RX_Information(void)                   // Receiving data packet
915  > {...
958    }
```

13.5 Install and set up APP

Open folder 3_APP and install " TSCINBUNY .apk " to your phone

Next , we use an Android phone to demonstrate how to control the ZHIYI smart robotic arm car through this application :

Enter the professional debugging interface and click the add button "+"

You need to fill in the project name for the project name.

➕ 添加调试工程

工程名称:

mypro|

工程类型:

◯ 竖屏视图

◉ 横屏视图

取消　　　确定

Click OK to see the built project

● Tx: 0B/s　　● Rx: 0B/s　　● Err: 0B/s

mypro
今天 18:09打开过　　▶

Click on the project name and the options to modify the project will be displayed.

First configure the communication settings , click the "+" sign, add a Boolean value , and then enter the Boolean value

name.

You can see that the created Boolean values will be displayed in this column. Use the same method to create 11 more

Boolean variables. They are tracking, Avoid, follow, dungeon, save, auto, empty, claws_open, claws_closed,

count-clockwise, clockwise

Click the "+" sign in the byte value column, add the byte value "byte" , and then enter the byte value name

Take setting a variable X that controls the left and right movement direction as an example, and change the name to X

You can see that the created byte values will be displayed in this column. Use the same method to create 4 more byte value variables. respectively Y, speed, Base, Arm

Click the second option at the bottom to create two receive byte values, namely: Distance, Actions
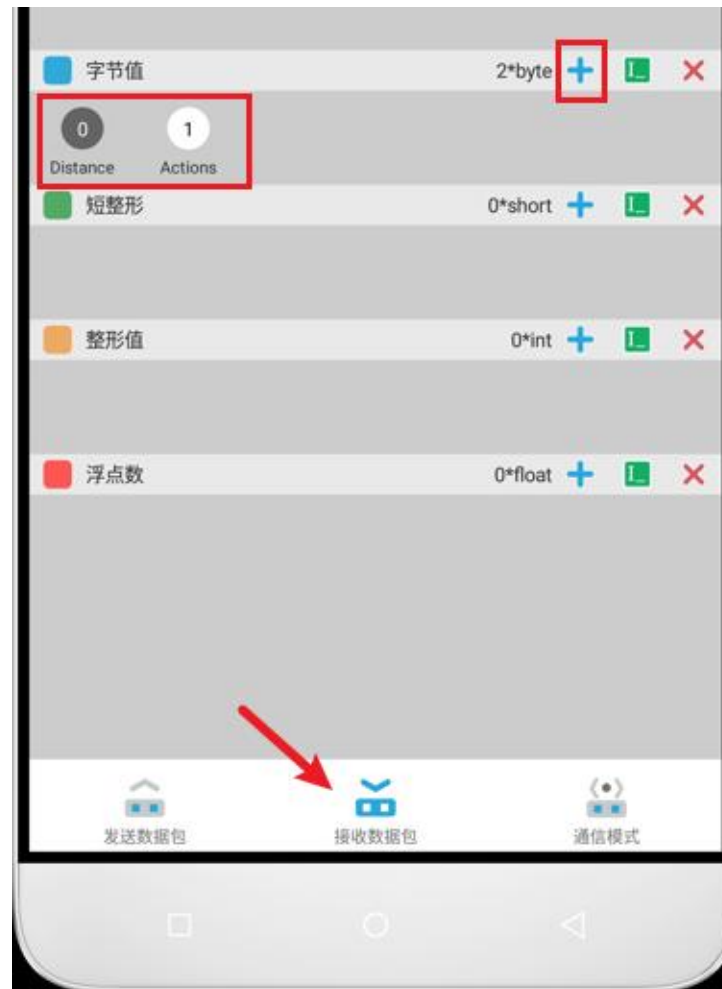
After all the above are added, return to the layout " Edit Controls " to add controls .

Click the "+" sign in the upper right corner to create a new control, taking the slider control that controls speed as an example;

添加一个控件

按钮

开关

Text 文本

Text 可编辑文本

能量槽

滑动条

摇杆

Y-X二维曲线图

Select the data type to be connected and the variable values just set, byte and speed

Then set the upper and lower limits and click OK to complete;

Add another joystick control to control the direction of the car:

Select the connection data type byte and variable name X/Y , and then check the release automatic reset :

Add another same joystick control to control the opening and closing of the claws, and select Base and Arm as variables.

Add a text control to display the ultrasonic distance :

Use the same method to add Actions;

Next add a control button and click the green button in the upper right corner to link the data:
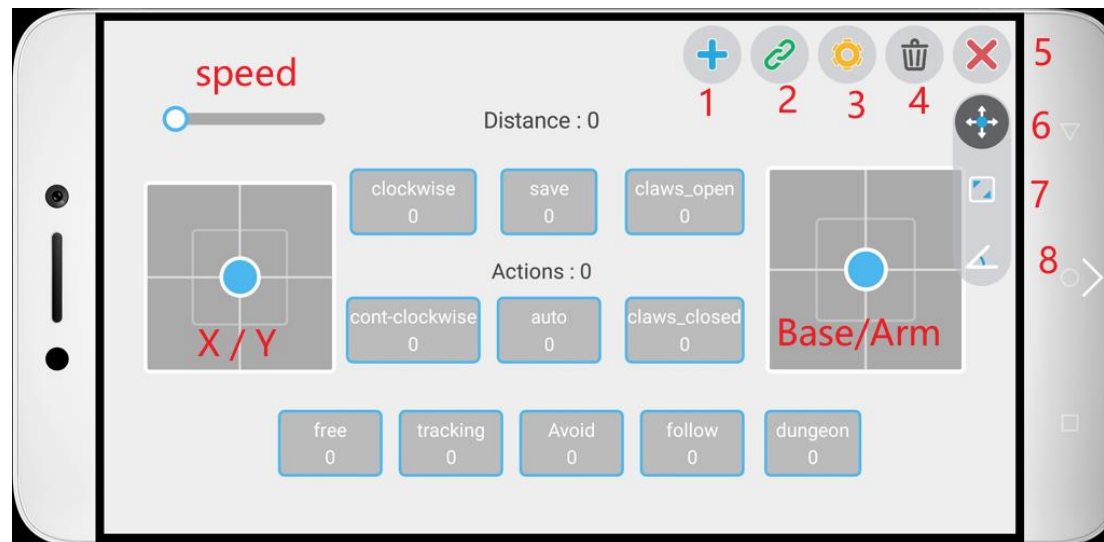
设定按下时的数值:

1

设定松开时的数值:

0

如果输入框中数据为空，则在相应时刻变量值不会改变。

OK

Use the same method to add 11 more controls , namely: tracking, Avoid, follow, dungeon, save, auto, empty, claws_open, claws_closed, count-clockwise, clockwise. Please drag and place these controls according to your own convenience.

The functions of each icon button in the upper right corner of the screen:

1Add various controls,

2 connection variables

3 means setting control parameters

4 is to delete the control

5Exit the current control layout

6Movement control

7 Zoom in and out controls

8 control rotation angle

When the controls are all adjusted, start connecting via Bluetooth, go to Device Connections and click Search.

Find " BT05 " and click "Add Device". If a password is required, enter the password as 1234 or 0000. (If you find that connecting to Bluetooth is slow in future use, first "forget" the Bluetooth in the phone settings. Then search for and connect to Bluetooth in the APP.)

"+" again

When a red "x" appears, it means the Bluetooth connection is successful :
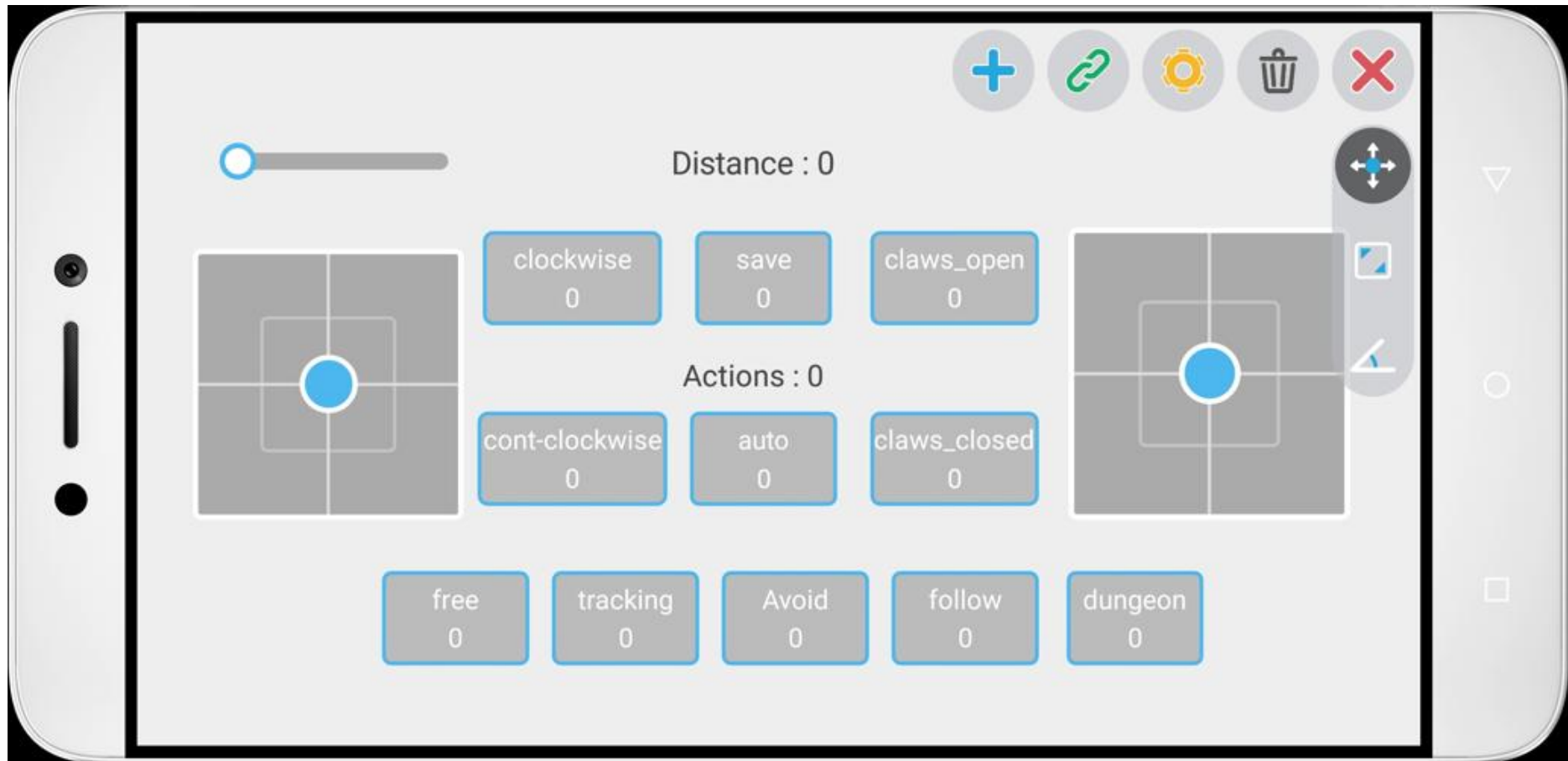
After the connection is successful, click Start and operate the car :

The final operation interface is as follows

free: free control, tracking: line following, Avoid: obstacle avoidance, follow: follow, dungeon: dungeon mode

Notice:

In free mode, the ultrasonic detection distance can be displayed in real time, but in other modes it is not displayed in real time;

Other modes are automatically executed. If you want to stop, you need to switch back to free mode;

When operating, pay attention to observe that the servo motor cannot be left in an unfinished state for a long time to prevent heating and damage;

Memory function operation: (Every time a servo motor is controlled by remote control, an action must be saved, and a maximum of 19 steps can be recorded)

The initial robot arm state is remote controlled to state 1, press the save button to record 1;

When remote control from state 1 to state 2, press the save button to record 2;

Then remote control from state 2 to state 3, and then press the save button to record 3;

This goes back and forth until the end action. It should be noted that the end action must be consistent with the initial action, so that continuous and consistent actions can be maintained when the automation execution is started.