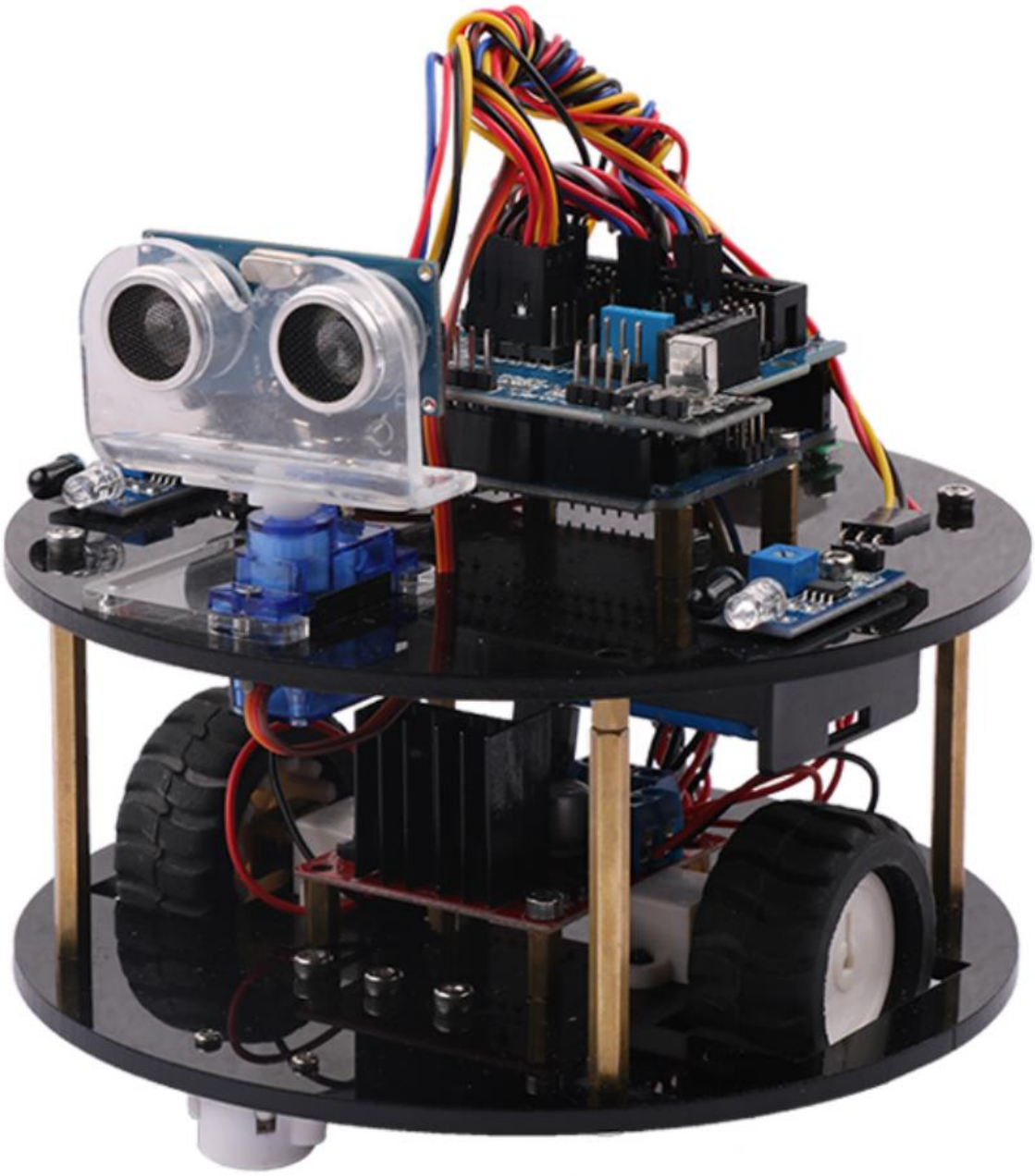
Beetle-Bot
Instruction Manual



Github url https://github.com/keywish/keywish-beetle-bot

Data	Version	Description	Author
2018/7/24	V-1.0	Create	Carl.du
2018/7/30	V-1.1	Review	Ken.chen

Table of Contents

Chapter1 Introduction 4
1.1 Writing Purpose 4
1.2 Product Introduction 4
Chapter2 Preparations 7
2.1 Development environment Arduino IDE 8
Chapter3 Experiments 17
3.1Beetle Bot Assembly 17
3.1.1 Base board universal wheel installation 17
3.1.2 Install motor and wheel 18
3.1.3 Motor Driver Board Installation 20
3.1.4 Tracing module and copper column installation 23
3.1.5 Battery box and Keywish Uno R3 motherboard installation 25
3.1.6 Installation of the Servo and ultrasonic 29
3.1.7 Infrared obstacle avoidance module installation 32
3.1.8 Voltage display module installation 33
3.1.9 Welding power cord 35
3.1.10 Whole Assembly 36
3.1.11 Expansion board wiring diagram 38
3.2 Beetle Bot Module experiment 40
3.2.1 Walking Principle of the Car 40
3.2.2 Infrared Obstacle Avoidance 46
3.2.3 Infrared Tracing 56
3.2.4 Ultrasonic Obstacle Avoidance 66
3.2.5 Infrared Remote Control 78
3.2.6 Mobile Phone Bluetooth Control 87
3.2.7 PS2 Handle (Optional) 99

Chapter1 Introduction

1.1 Writing Purpose

The purpose of this manual is to create a fast, practical and convenient development learning platform for the vast number of electronic enthusiasts and let them grasp the Arduino and its extended system design methods and design principles, as well as the corresponding hardware debugging methods.

This manual will lead you to learn every function of "Beetle-Bot" step by step and open a new "Beetle-Bot" journey for you. It is divided into two parts:

1, Preparation chapter, which mainly introduces the use of common Arduino development software and some downloading and debugging skills.

2, Experiment chapter, which contains hardware and software, the former mainly introduces the function and principle of each module; the latter mainly introduces each part of the program and leads you to understand and grasp the principle of Arduino and the car development through written examples step by step.

This manual is a specifications for "Beetle-Bot" , the file whose format is PDF which is in the CD along with our product requires the corresponding software to open. It contains detailed schematic diagrams and complete source codes for all instances, the codes won't have any mistake under our strict test. In addition, the library files used in the source codes are put into the corresponding path, you only need to see corresponding phenomenon of the car and personally experience the process of experiment by downloading the source codes to Arduino via the serial port emulator.

This manual is very suitable for students and electronic enthusiasts to learn, all course videos will be synchronized to https://github.com/keywish/keywish-beetle-bot， please real-time synchronization of the latest information

1.2 Product Introduction

"Beetle-Bot" is a multifunctional car based on the Arduino UNO and L298N motor. Compared with the traditional car, "Beetle-Bot" is also equipped with wireless control (Bluetooth, infrared, WIFI and so on); ultrasonic; infrared. It can trace and avoid obstacles automatically, of course, makers can also automatically control the car with wireless and make full use of each module, as well as integrate all kinds of related sensors to make the car more intelligent, which is more challenging. "Beetle-Bot" has various types of information, technical manuals, routines, etc., which can teach you step by step. Each electronic fan can use it easily to achieve their desired function.

Product Features

- Three groups of black line infrared tracing module

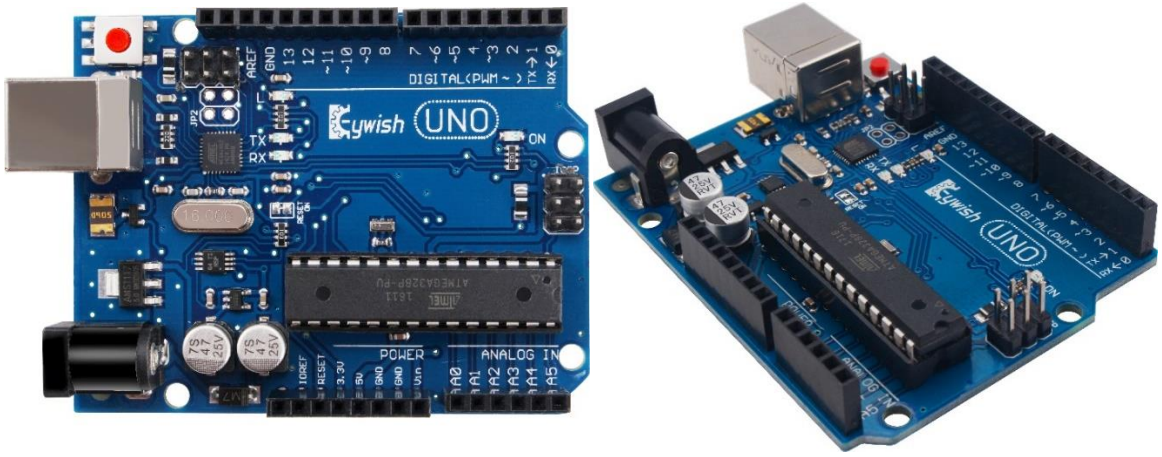
- Two groups of infrared obstacle avoidance module
- Ultrasonic obstacle avoidance
- Four DC motor drive
- Two 2000mZh, 3.7V rechargeable lithium battery with longer endurance
- Remaining capacity of battery real-time detection
- Infrared remote control
- Bluetooth app control
- PS2 handle control (optional)
- Support handle control of nRF24L01 (optional)

Product device list:

 1 x Keywish UNO R3 Controller Board	 1 x L298N motor drive board	 1 x Arduino extension board	 1 x 18650 li-battery box 2 x 18650 li-battery	 The charger	 1 x USB Cable
 1 x Ultrasound Module	 1 x SG90 servo	 1 x JDY-16 Bluetooth module	 1 x Infrared remote	 2 x Infrared obstacle avoidance module	 3 x Infrared line tracking module
 1 x Power indicator digital tube	 1 x Infrared receiver module	 2 X Universal wheel	 2 X Motor Fixing frame	 2 x DC Motor 2 x Motor line	 2 X wheel
1 x Ultrasonic Holder	1 x SG90 Servo fixed plate	 1 x Power DC head	 1 x Cross wrench	 1 x M3 Allen wrench	 1 x 3.0-7.5mm Phillips screwdriver
 6 X M3*55 Dual channel copper posts	 8 X M3*10+6mm Single channel copper pillar	 3 X M3*5+6mm Single channel copper pillar	 14 x M3*12mm screw	 11 x M3*7mm screw	 6 x M3*8 Flat-headscrew
 2 X M2*14 1 X M2*8 Flat-headscrew	 27 x M3 nut	 3 x M2 nut	 20cm Male to Female Dupont Line (7pin)	 3 X black wire 3 X White wire	Several cable ties

About Arduino uno r3

In "Beetle-Bot", we used the Arduino uno r3 as the main control board, which has 14 digital input/output pins (6 of which can be used as PWM output), 6 analog inputs, and a 16 MHz ceramic resonator, 1 USB connection, 1 power socket, 1 ICSP head and 1 reset button. It contains everything that supports the microcontroller; You just need to connect it to a computer via a USB cable or start with an AC-DC adapter or battery.



Technical specifications:

Working voltage: 5V

Input voltage: USB powered or external 7V~12V DC input

Output voltage: 5V DC output and 3.3V DC output and external power input

Microprocessor: ATmega328 (Chip data sheet is in the documentation)

Bootloader： Arduino Uno

Clock frequency: 16 MHz

Support USB interface protocol and power supply (without external power supply)

Support ISP download function

Digital I/O port: 14 (4 PWM output ports)

Analog input port: 6

DC Current I/O Port: 40mA

DC Current 3.3V Port: 50mA

Flash memory: 32 KB (ATmega328) (0.5 KB for bootloader)

SRAM ： 2 KB (ATmega328)

EEPROM： 1 KB (ATmega328)

Size：75x55x15mm

2.1 Development environment Arduino IDE

2.1.1 Install the IDE

AduinoIDE is an open source software and hardware tool written by open source software such as Java, Processing, and avr-gcc. It is an integrated development environment that runs on a computer. It can write and transfer programs to the board. The major feature of the IDE is cross-platform compatibility for Windows, MaxOSX, and Linux. Only a simple code base is needed, and the creators can create personalized home internet solutions through the platform, such as remote home monitoring and constant temperature control and so on.

In this tutorial, we use the version is 1.6.0, download address is:https://www.arduino.cc/en/Main/OldSoftwareReleases#previous， After opening the link, we can see the interface as shown in Figure 2.1.1. In this interface, we can see the different versions of the IDE and different operating environments. Everyone can download according to their own computer system, of course, There will be a downloaded installation package on our companion CD, but only the Windows version, because this tutorial is all running under Windows system.

1.8.1	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit Linux ARM	Source code on Github
1.8.0	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit Linux ARM	Source code on Github
1.6.13	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit Linux ARM	Source code on Github
1.6.12	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit Linux ARM	Source code on Github
1.6.11	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit Linux ARM	Source code on Github
1.6.10	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit Linux ARM	Source code on Github
1.6.9	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit Linux ARM	Source code on Github
1.6.8	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit	Source code on Github
1.6.7	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit	Source code on Github

Figure 2.1.1 ArduinoIDE download interface

After the downloading, we will get a compressed package as shown in Figure 2.1.2. The compressed package will be decompressed. After decompression, the files in Figure 2.1.3 are extracted. The “drivers” is the driver software. When the “Arduino.exe” is installed, it will be Install the driver automatically. Because the installation of "arduino.exe" is very simple, it will not be explained here. It is recommended to exit the

anti-virus software during the installation process, otherwise it may affect the installation of the IDE. After the installation is complete, click "arduino.exe" again to enter the IDE programming interface.

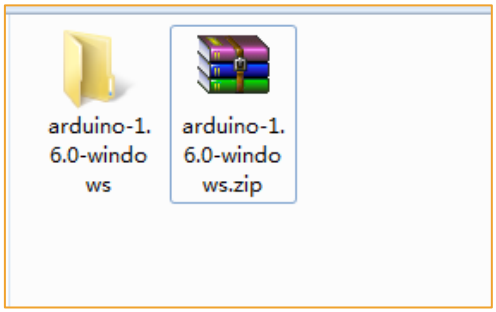


Figure 2.1.2 Arduino IDE Installation Package

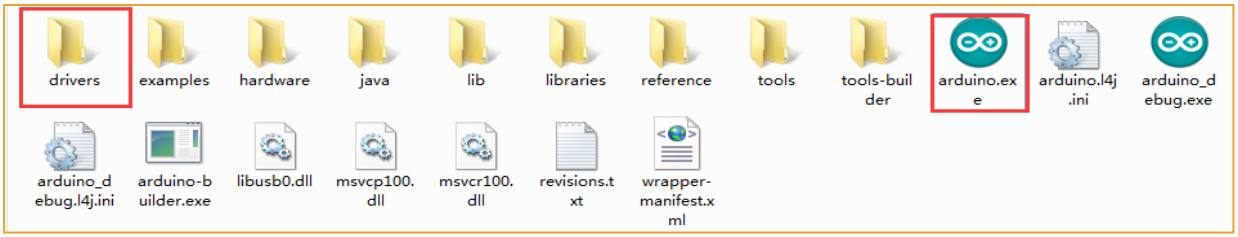


Figure 2.1.3 Extracted files

When finish the installation of the IDE，connect to the Arduino motherboard，click“My Computer”→ “Properties” → “Device Manager” → “Viewing Ports (COM and LTP)”， If you can see as the Figure 2.1.4

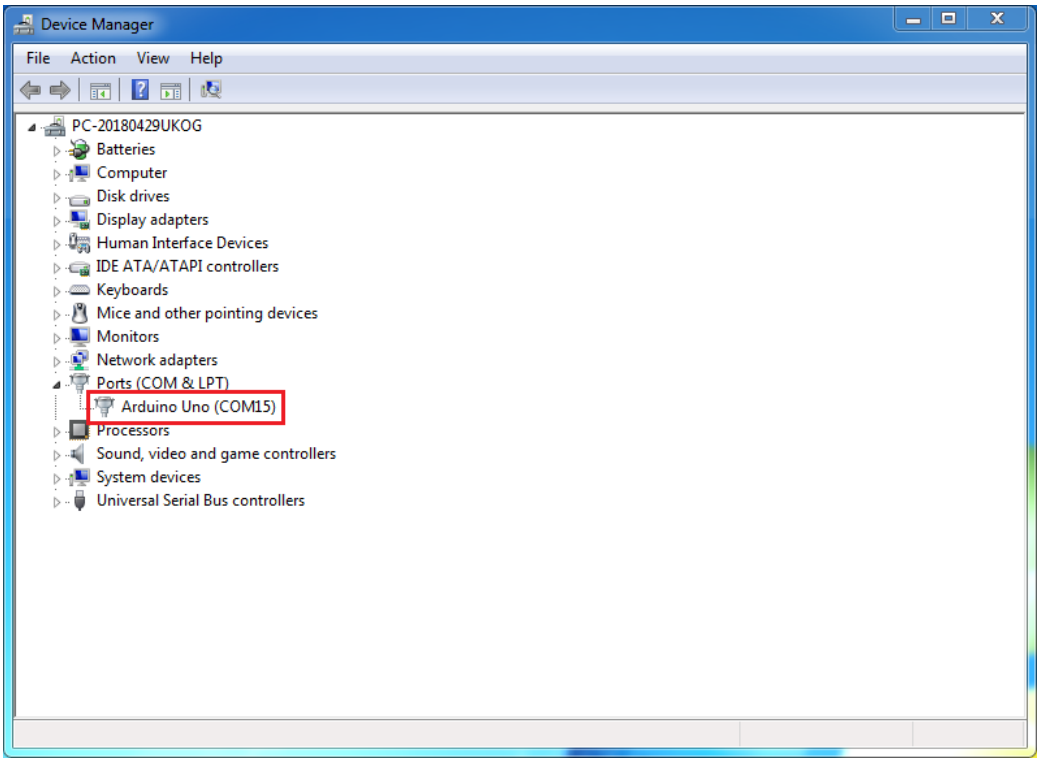


Figure 2.1.4 Driver installation success interface

that indicates the driver has been installed successfully， At this time we open the IDE, select the corresponding development board model and port in the toolbar to use normally. If you see Figure 2.1.5, it means that the computer does not recognize the development board and you need to install the driver yourself.

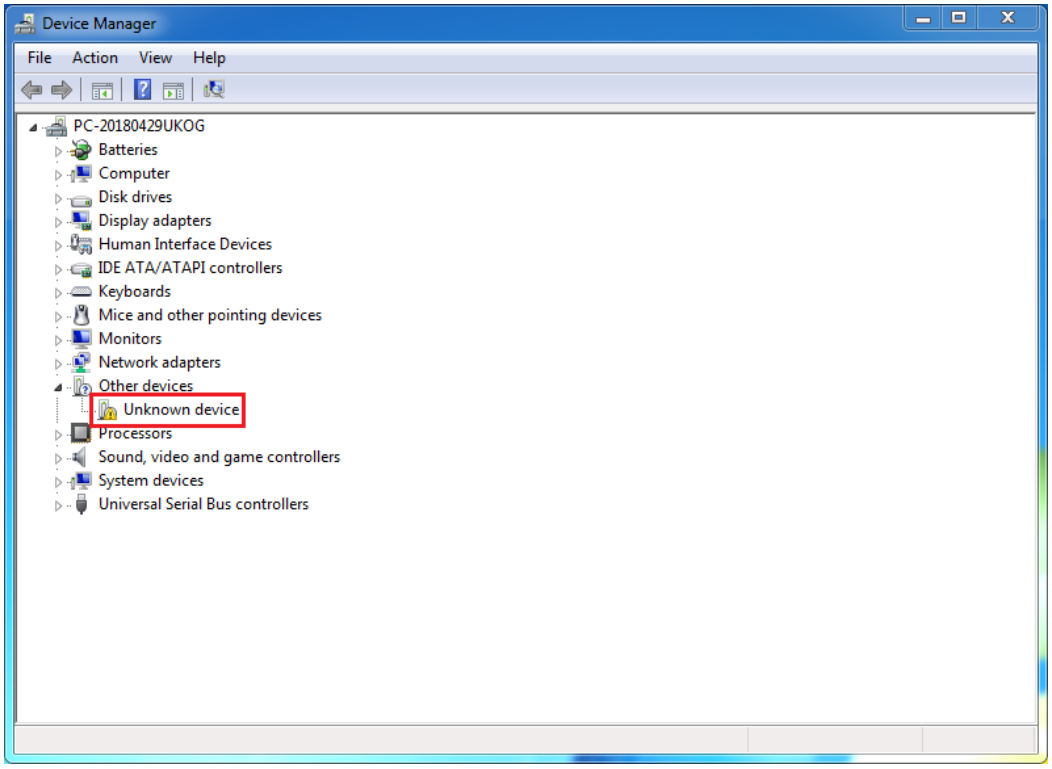


Figure 2.1.5 Driver is not successfully installed interface

Notice:

1）If you connect the controller board to the computer, the computer does not respond. Right-click "My Computer" and select Open Device Manager then find viewing port (com & lpt). If there is no com or lpt, or only an unknown device, there is a problem with the controller board or the USB cable.

2）Right-click "My Computer" and select Device Manager, find the viewing ports (COM and LPT). If there is a yellow Arduino UNO exclamation point, this means you need to install the driver yourself.

3）If you install the driver again and again, it eventually fails. Please uninstall the driver and re-install> install the driver automatically> restart the computer.

2.1.2 Install Driver

If your computer is a Windows 7 system

1）Right-click on "My Computer" and open the Device Manager, find viewing the ports (COM and LPT). At this point you will see a "USB Serial Port", right-click "USB Serial Port" and select the "Update Driver Software" option.

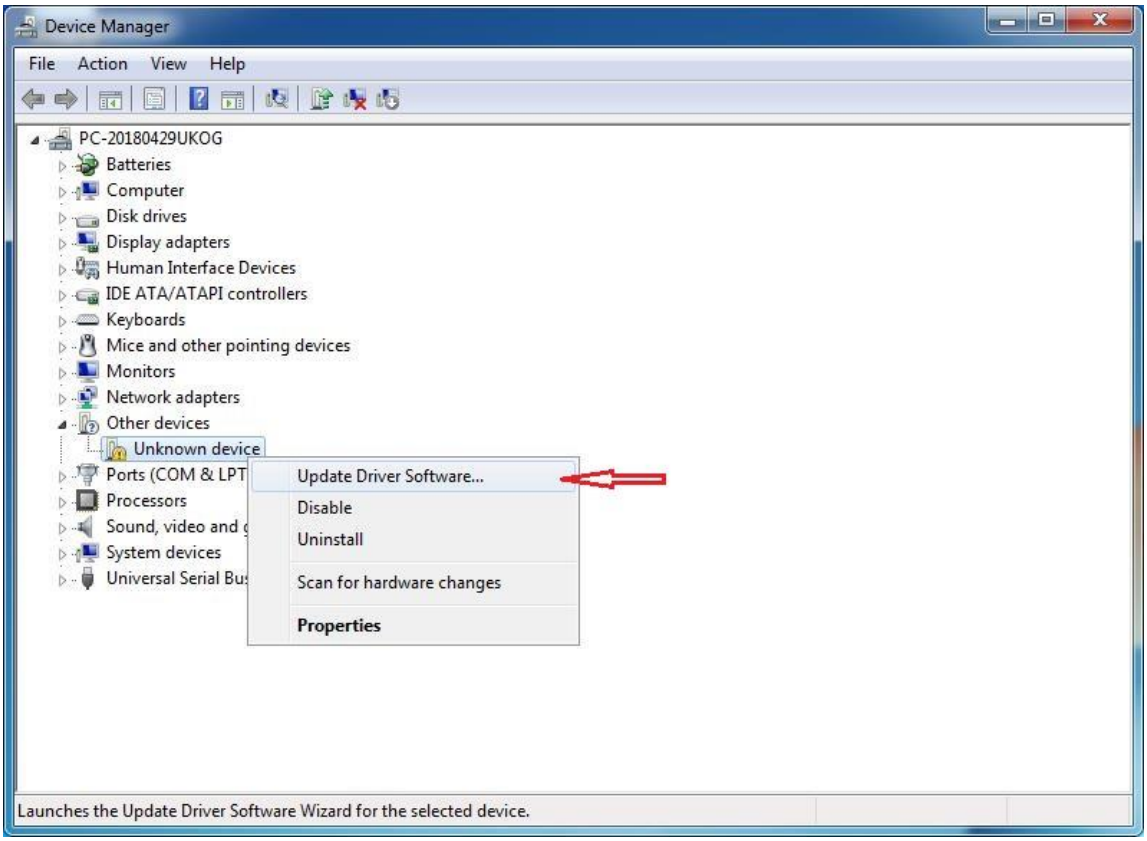


Figure 2.1.6 Updated Driver Interface

2）Next, select the "Browse my computer for driver software" option.

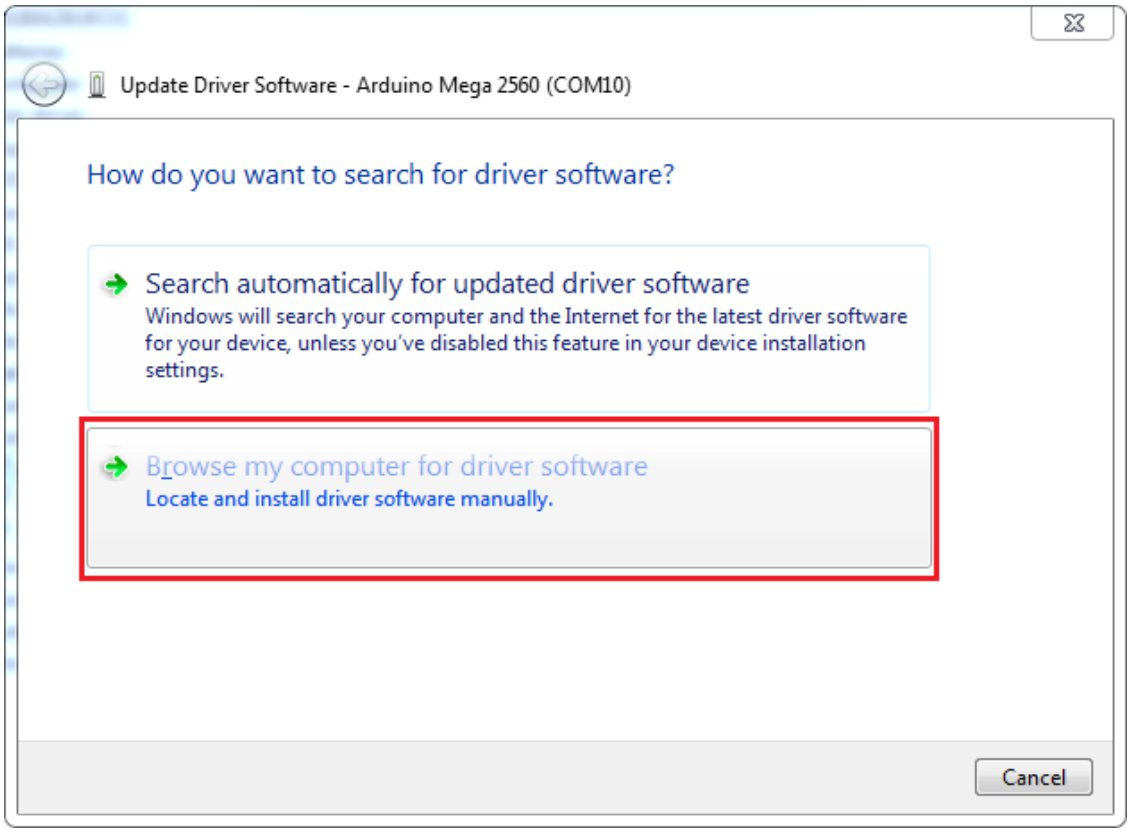


Figure 2.1.7 Driver Update Selection Screen

3）Finally select the driver file named "FTDI USB Drivers" located in the "Drivers" folder of the Arduino software download.

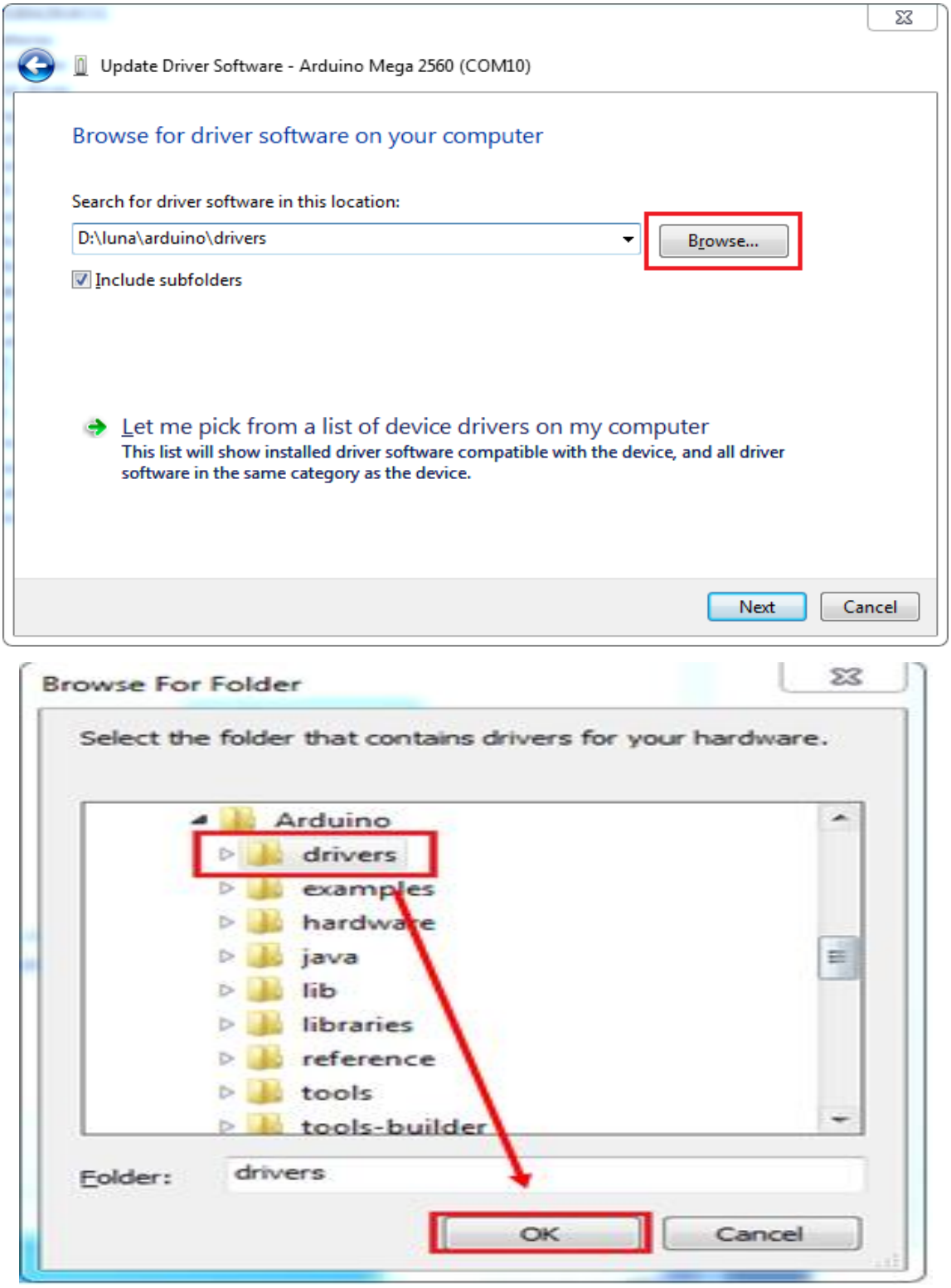


Figure 2.1.8 Driver file selection interface

4）If you have already installed, the following figure will automatically inform you that the driver was successful.

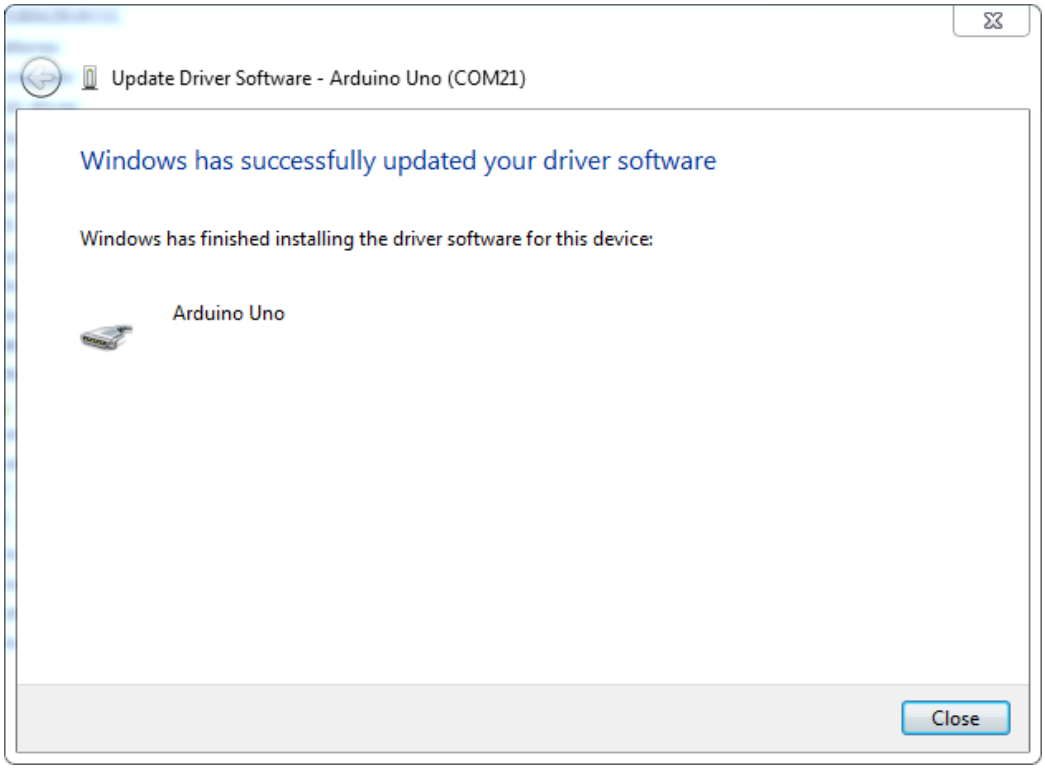


Figure 2.1.9 Driver Installation Successful Interface

At this time, we return to the "Device Manager" interface, the computer has successfully identified Arduino, as shown in the below Figure 2.1.10 .then open the Arduino compilation environment, you can open the Arduino trip.

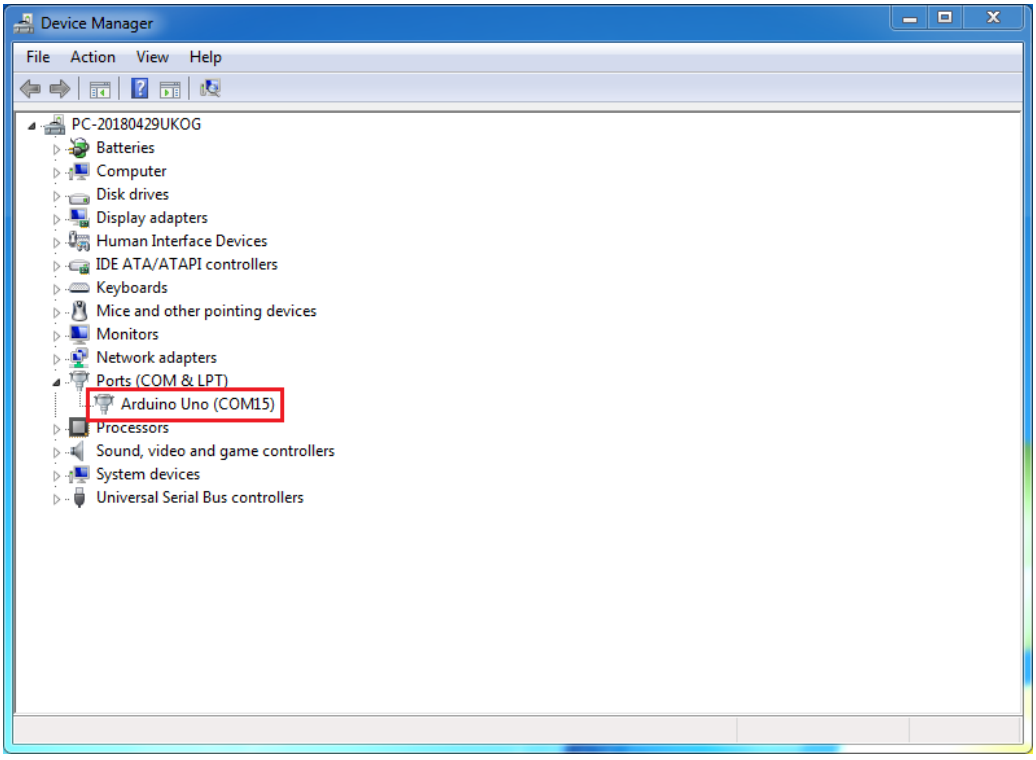


Figure 2.1.10 Driver Success Recognition Interface

Notice In Win10 system, some Arduino are connected to the computer (non-genuine chips are difficult to identify), the system will automatically download the corresponding driver, so you can not install the driver yourself, but in the Win7 system, you have to do it yourself.

In addition, we can see that the USB serial port is identified as COM15 in the above figure, but it may be different with different computer, you may be COM4, COM5, etc., but USB-SERIAL CH340, this must be the same. If you do not find the USB serial port, you may have installed it incorrectly or the system is incompatible.

2、If your computer is a Windows 8 system: Before installing the driver, you should save the files you are editing because there will be several shutdowns during the operation.

1）Press "Windows key" + "R"

2）Iput shutdown.exe / r / o / f / t 00

3）Click the "OK" button.

4）The system will reboot to the "Select an option" screen

5）Select "Troubleshooting" from the "Select an option" screen

6）Select "Advanced Options" from the "Troubleshoot" screen

7）Select “Windows startup settings screen” from “Advanced Options”

8）Click the "Restart" button

9）The system will reboot to the “Advanced Boot Options” screen

10）Select "Disable Driver Signature Enforcement"

11）Once the system is booted, you can install Arduino driver the same as Windows7

3、If your computer is a Windows XP system: The installation steps are basically the same as for Windows 7, please refer to the above Windows 7 installation steps.

2.1.3 IDE Interface Introduction

Nextly,we introduce the Arduino IDE interface, firstly enter the software directory. Then you can see the arduino.exe file and double-click to open the IDE. As shown in Figure 2.1.11.

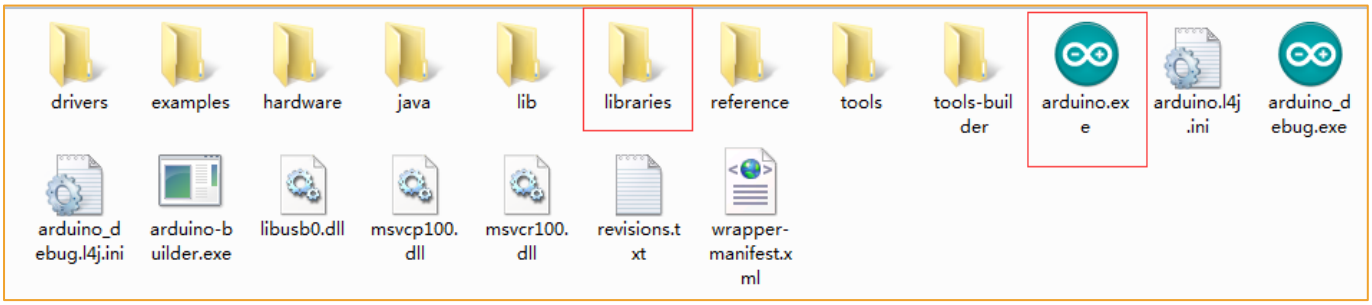


Figure 2.1.11 Software Catalog

1.The first thing you can see is the interface of the following figure. The functions of the toolbar buttons are "Compile" - "Upload" - "New Program" - "Open Program" - "Save Program" - "Serial Monitor" , as shown in Figure 2.1.12.

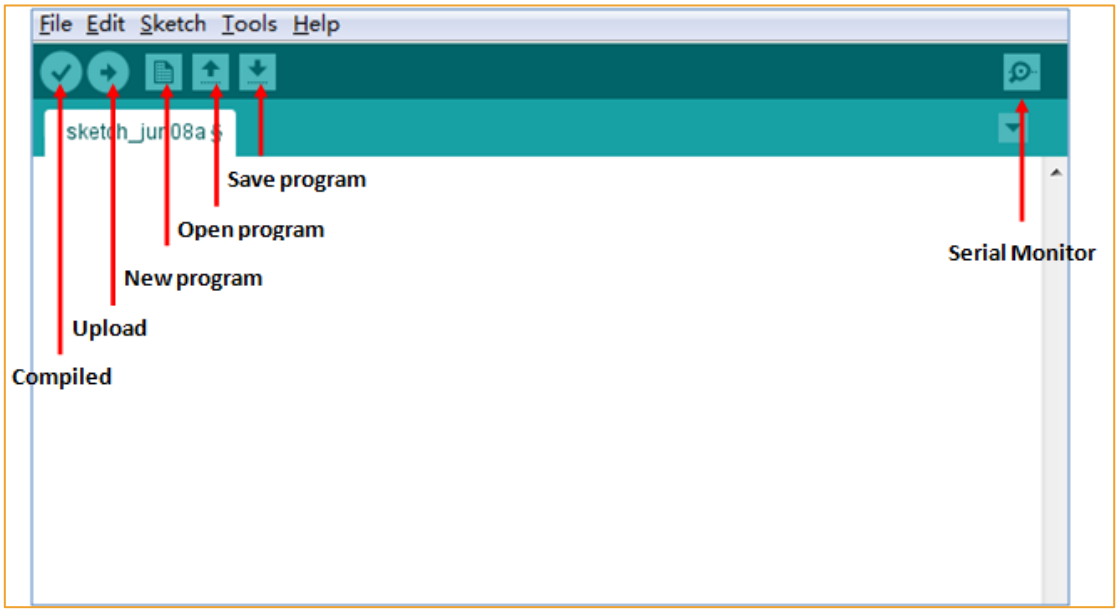


Figure 2.1.12 Arduino IDE Interface

2.There are 5 menus on the menu bar, but we mainly introduce File and Tools. Click File, the interface as shown in Figure 2.1.13 will be displayed, you can see the Examples and Preference options. The Examples are some of the Arduino's own programs, these are compiled without errors, the normal use of the program, a great help for beginners. The Preference option, It's mainly about the parameter settings, such as language, fonts and so on.

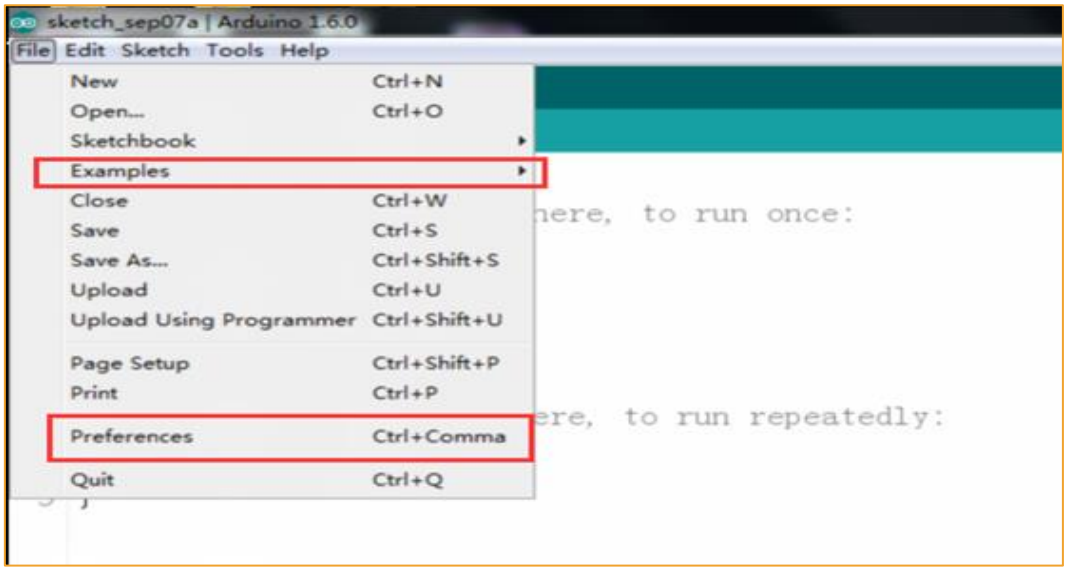


Figure 2.1.13 File Menu Bar Options

3.Click Tools, the interface shown in Figure 2.1.14 will pop up. Here we can see two options: Board and Port. In the board option, we can see the commonly used Arduino development board model, we only need to choose according to their own development board. In the Port option, the USB serial port is mainly selected, as shown in Figure 2.1.15. If you are not sure, you can check it in the "Device Manager" and select the corresponding COM port.

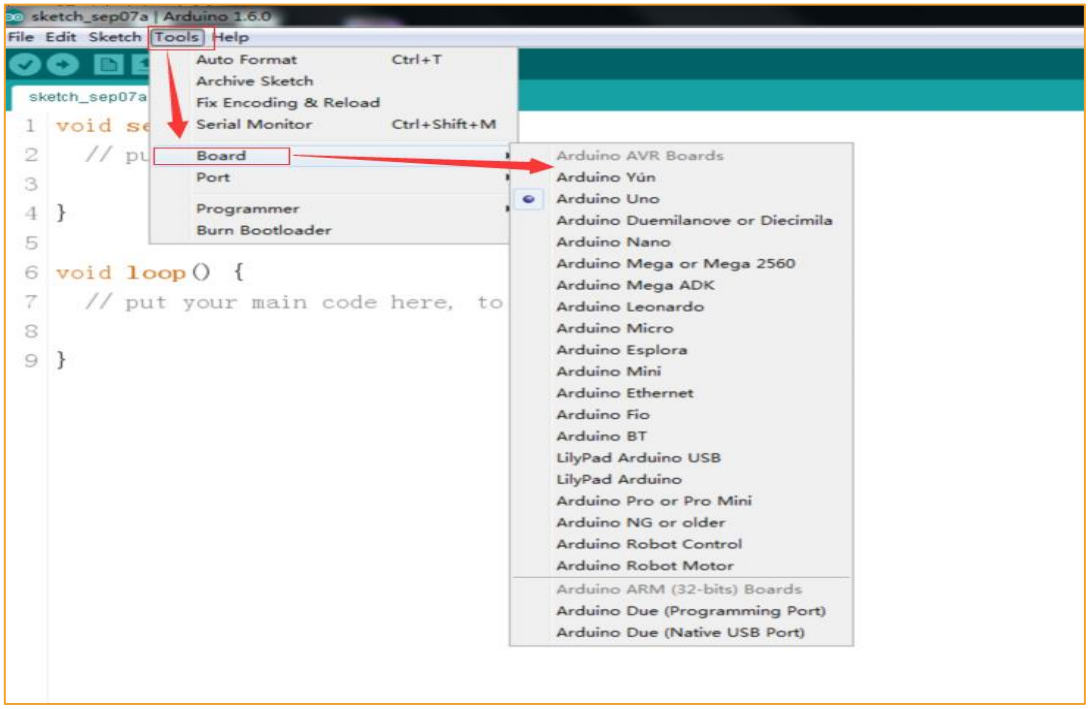
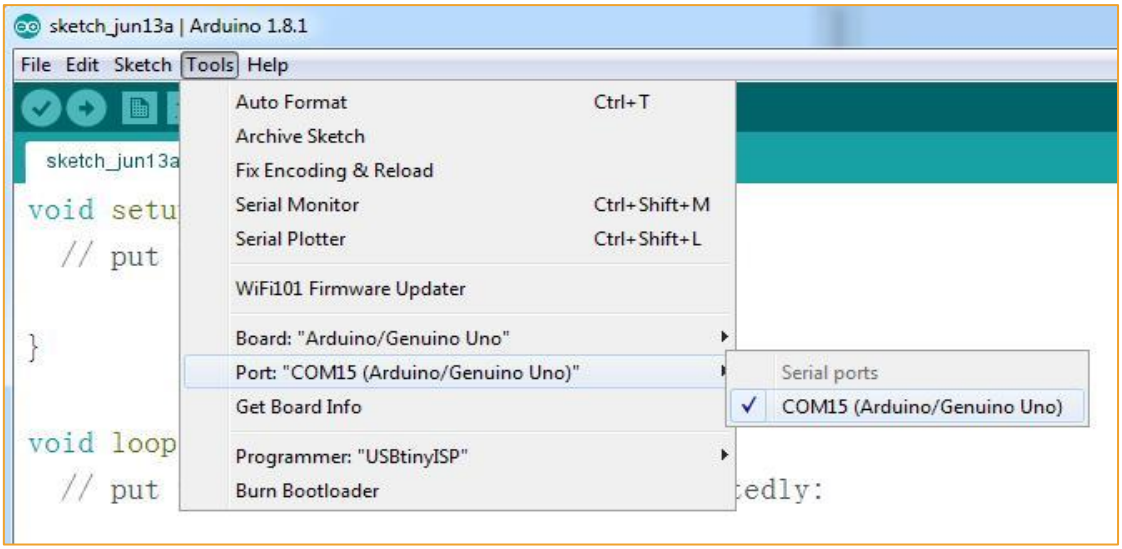


Figure 2.1.14 Tools interface



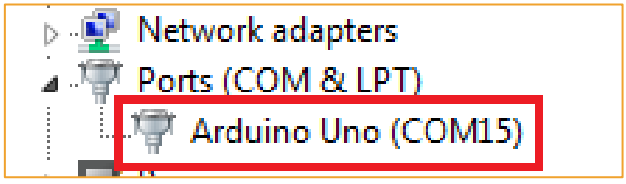


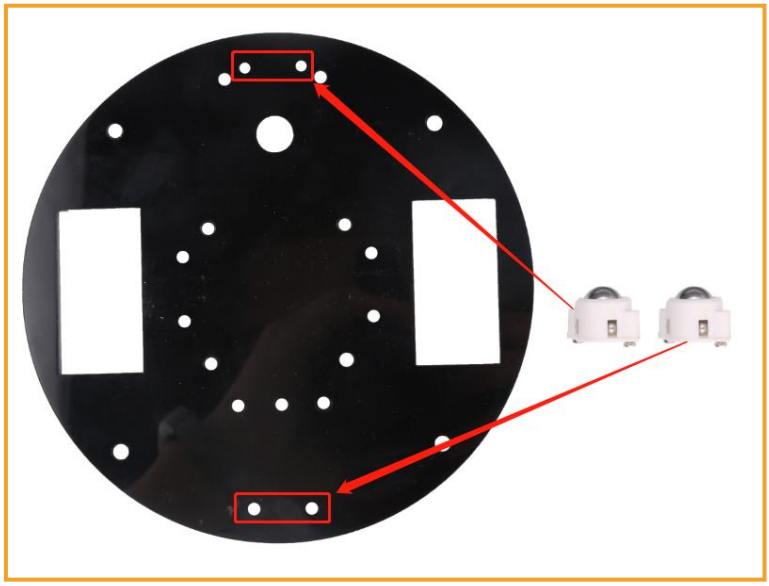
Figure 2.4.14 USB serial port choice

So far, we have basically completed all the work. The next step is actual experiments. Open any program in Examples. First compile the program. If it is compiled correctly, it can be directly downloaded to the development board and the corresponding device of the connection number. With wires, you can see the corresponding phenomenon.

3.1 Beetle Bot Assembly

3.1.1 Base board universal wheel installation

Firstly, we open the box, take out all the components and put it on the table lightly. (Note: There are many devices, be careful when installingto prevent some devices from being lost)



Note: The universal wheel is fixed by using its own two screws.

Figure 3.1.1.1 Schematic diagram of the universal wheel installation

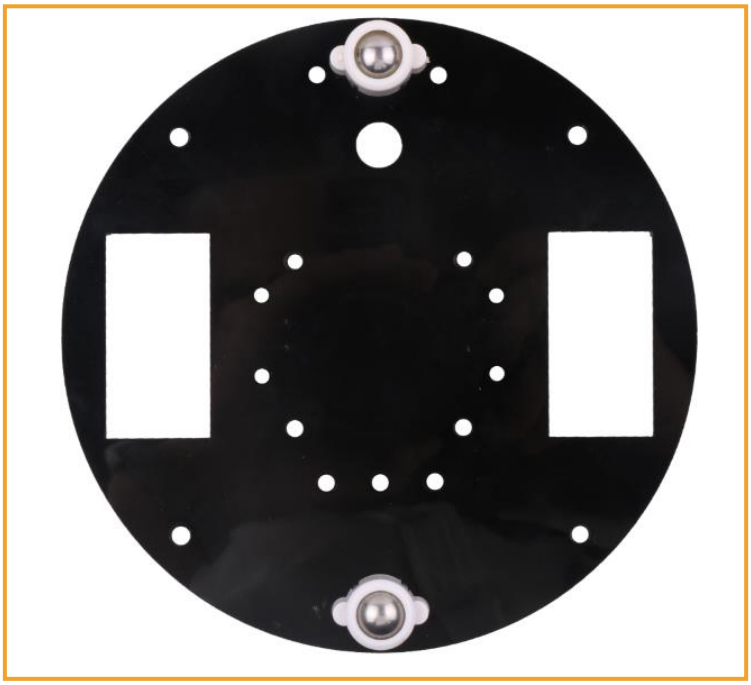
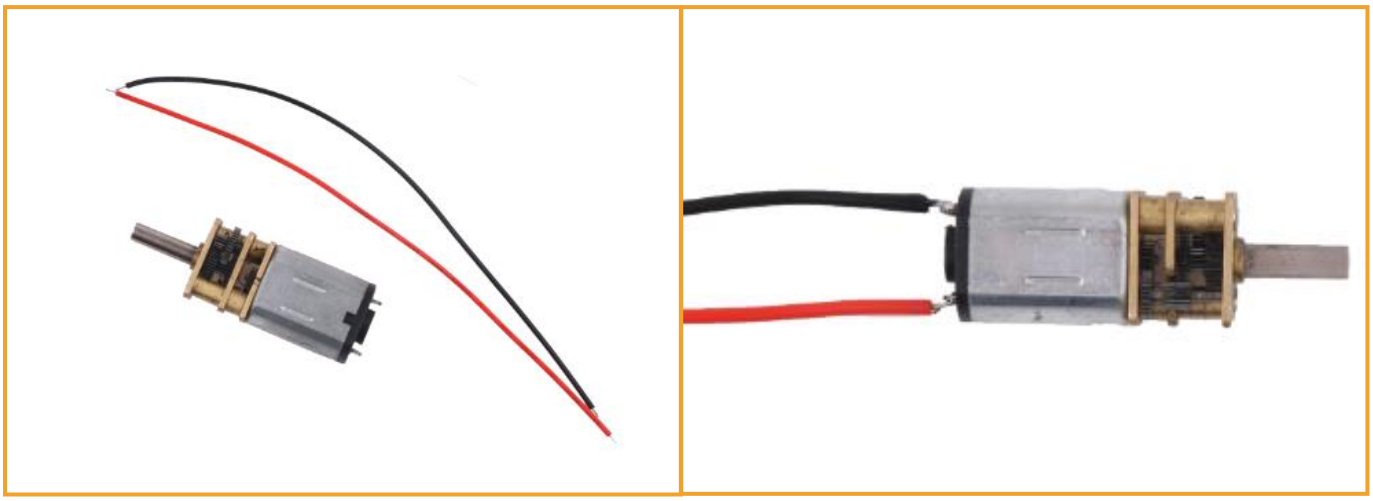


Figure 3.1.1.2 Effect diagram of the universal wheel installation

3.1.2 Install motor and wheel

Step 1: Weld the wire to the motor



Note: The motor pin is marked with positive and negative poles, positive welding red wire and negative welding black wire.

Figure 3.1.2.1 Effect diagram of motor welding

Step 2: Install the tire

Note: Align the motor drive shaft flat section with the tire hole

Figure 3.1.2.2 Effect diagram of Tire installation

Step 3: Install the motor

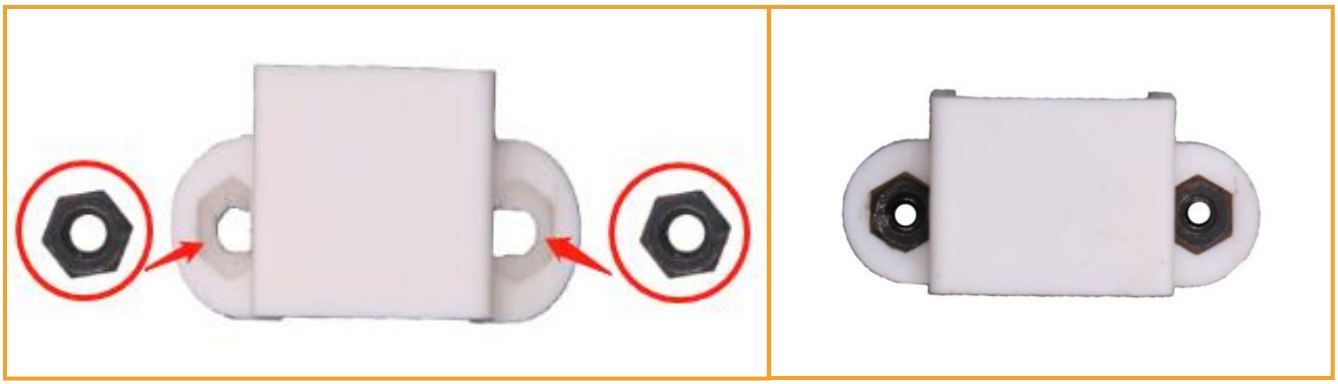
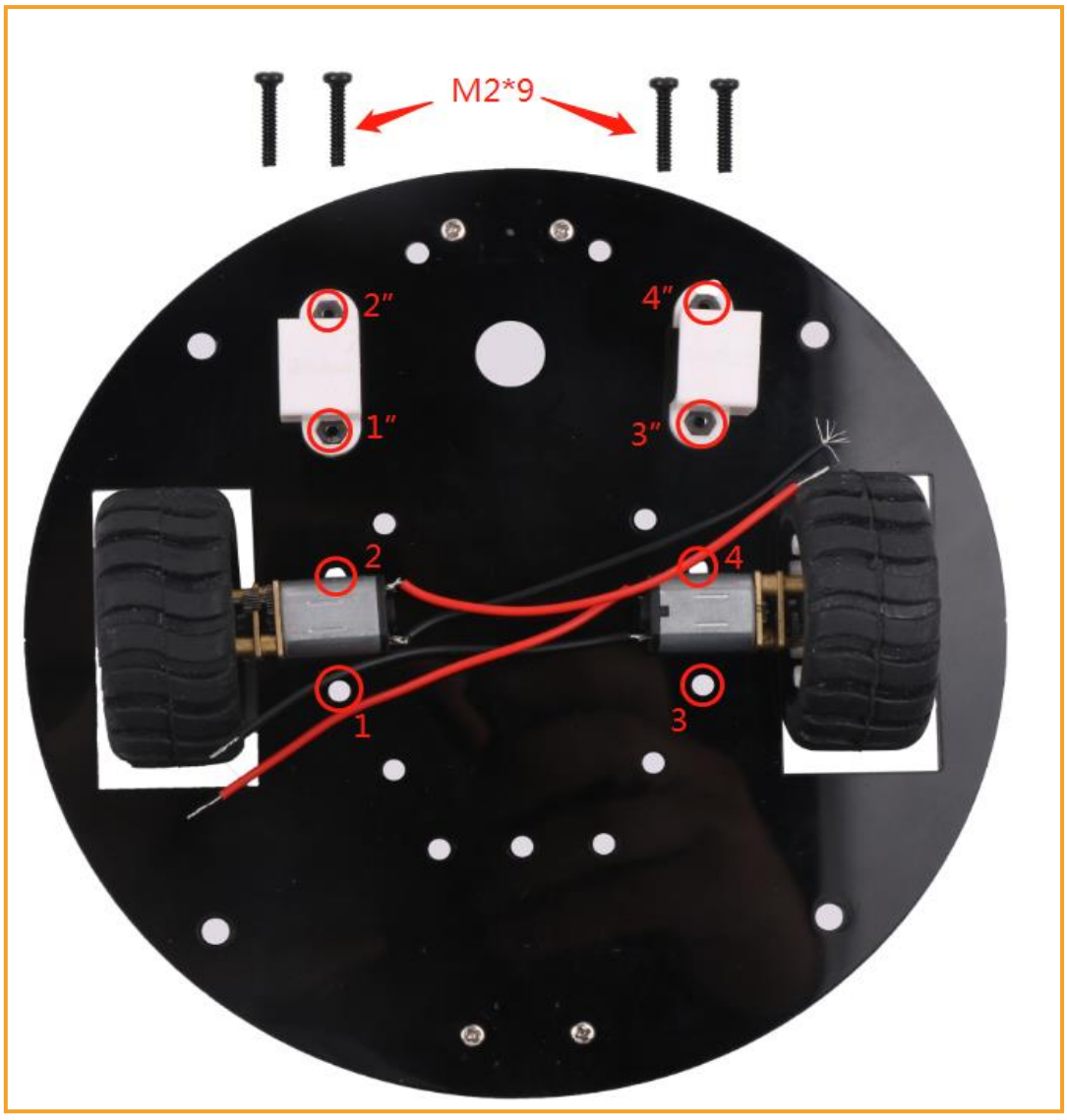


Figure 3.1.2.3 Effect diagram of motor mounting bracket installation



Note: Hole 1 corresponds to the nut 1", tightened from the bottom with an M2*9 Phillips screw, and so on.

Figure 3.1.2.4 Motor installation diagram

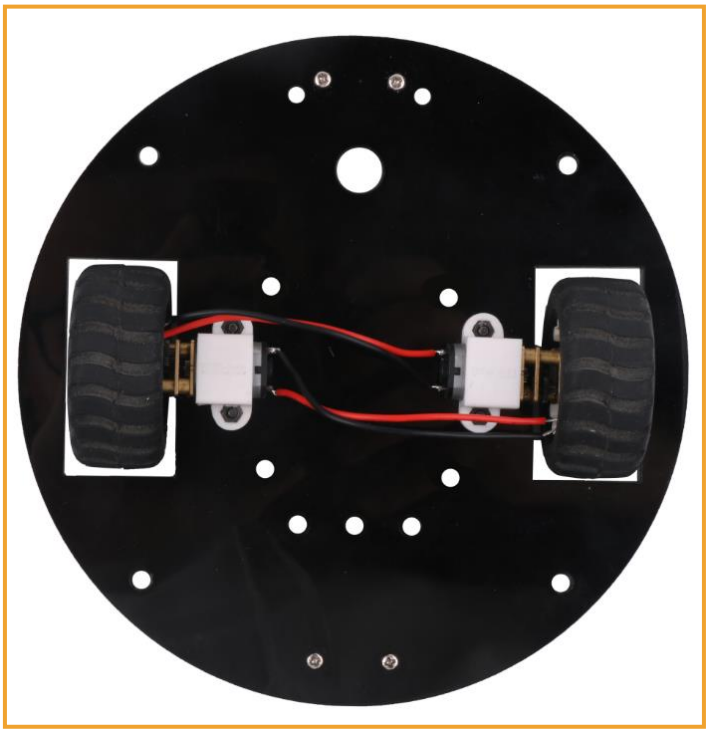


Figure 3.1.2.5 Motor installation effect diagram

3.1.3 Motor Driver Board Installation

Step 1: motor drive board pillar installation

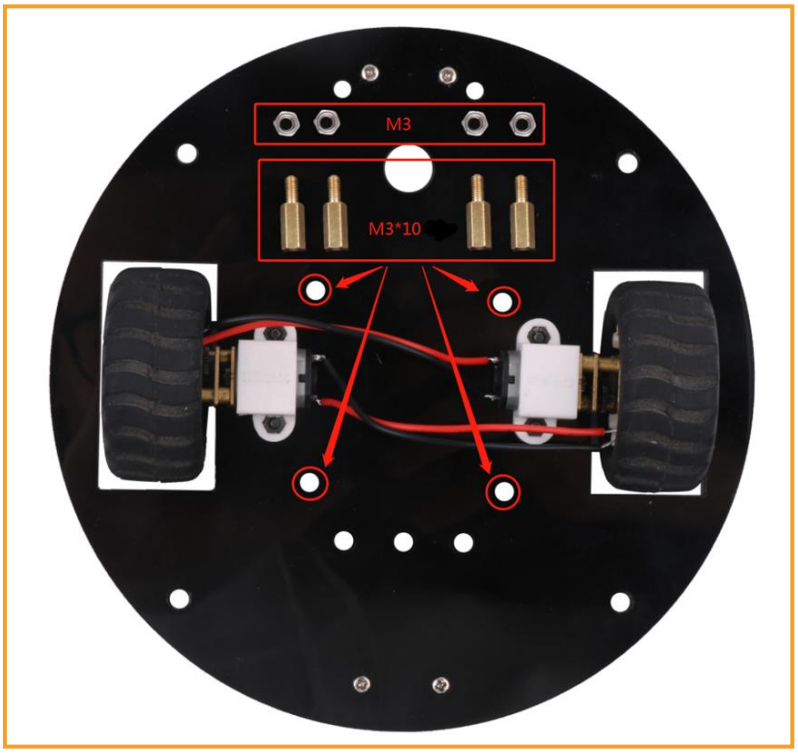


Figure 3.1.3.1 Schematic diagram of motor drive board pillar installation

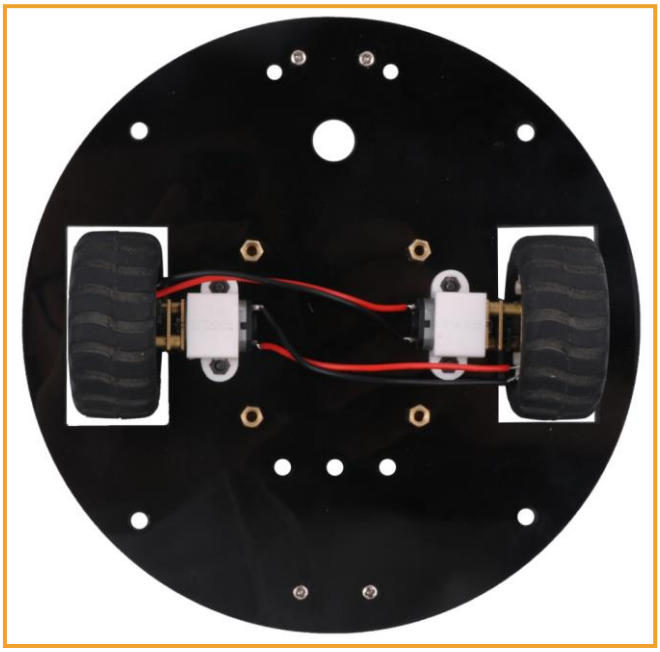
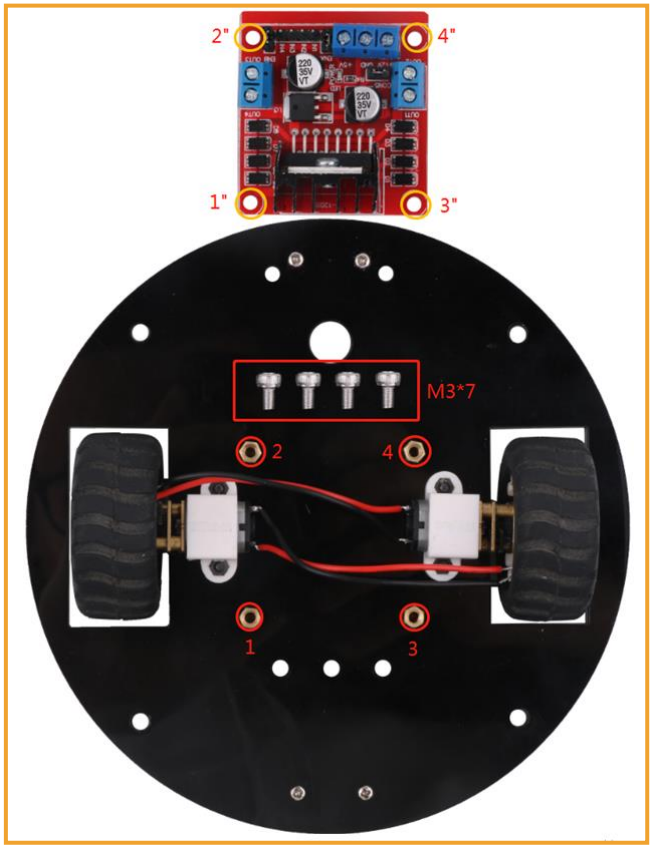


Figure 3.1.3.2 Effect diagram of motor drive board pillar installation

Step 2: motor drive board installation



Note: Hole 1 corresponds to 1″, hole 2 corresponds to 2″, **tightened** with M3*7 screw and so on

Figure 3.1.3.3 Schematic diagram of Motor Driver Board Installation

Figure 3.1.3.4 Effect diagram of Motor Driver Board Installation

Note: The left motor negative pole is connected to OUT1, the positive pole is connected to OUT2; the right motor negative pole is connected to OUT3, and the positive pole is connected to OUT4.

Figur 3.1.3.5 Motor and drive board connection diagram

3.1.4 Tracing module and copper column installation

Step 1: Tracing module installation

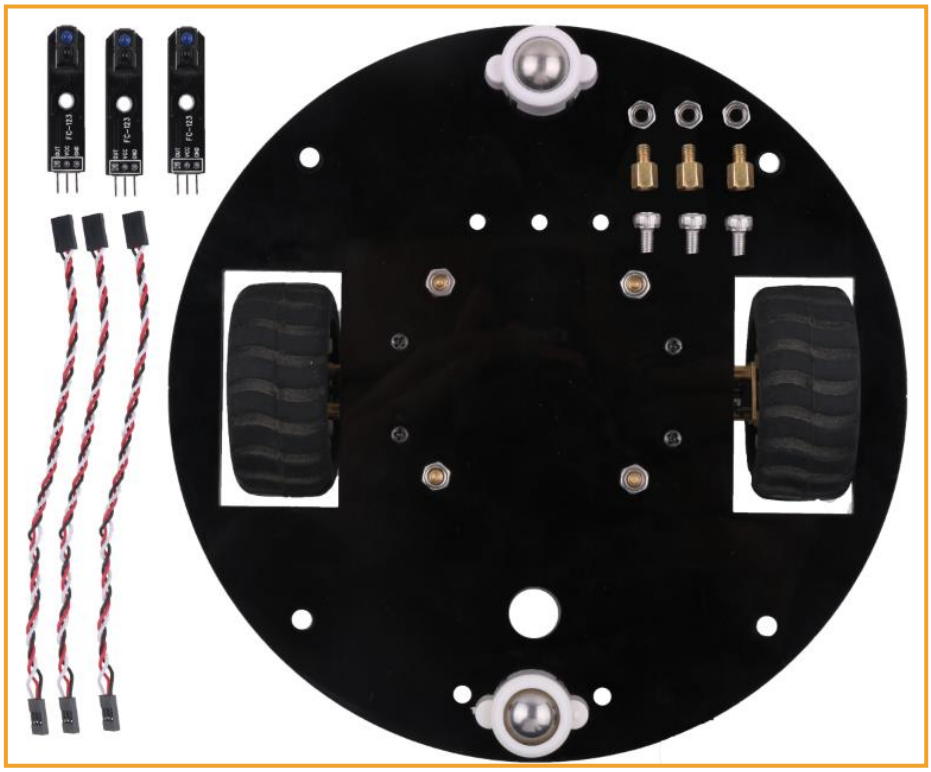


Figure 3.1.4.1 Tracing module installation checklist

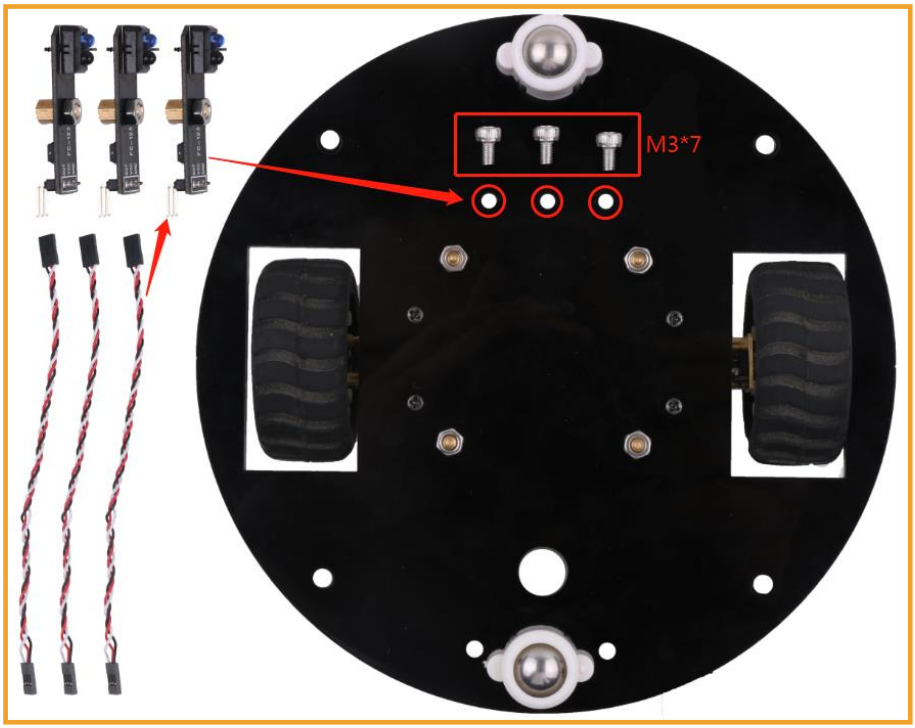


Figure 3.1.4.2 Schematic diagram of the tracing module installation

Figure 3.1.4.3 Effect diagram of the tracing module installation

Step 2: the upper acrylic plate pillar installation

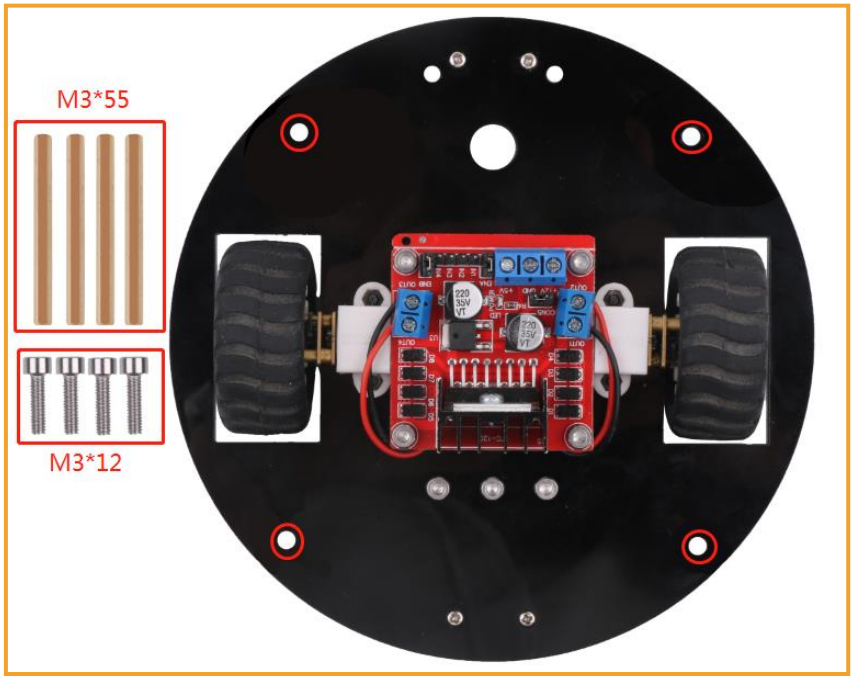
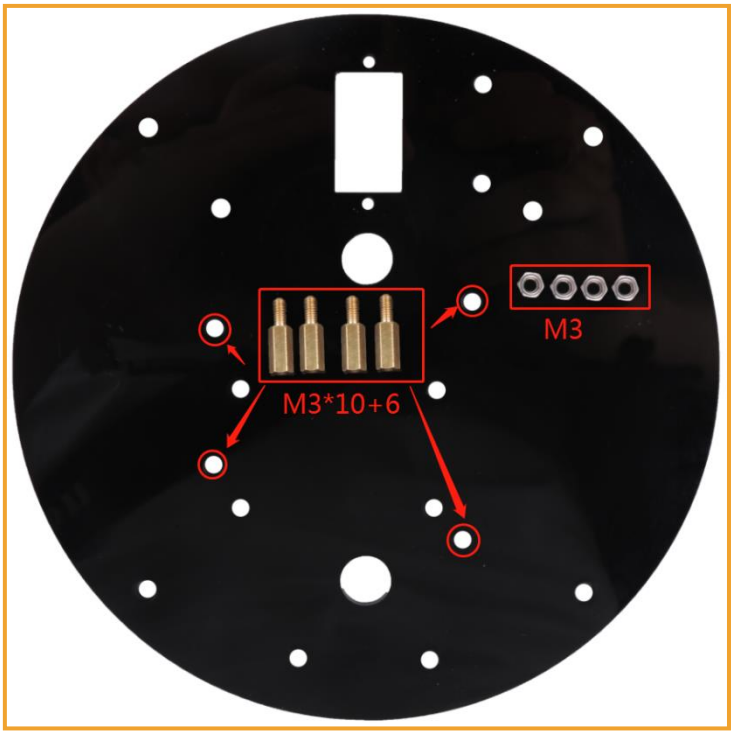


Figure 3.1.4.4 Schematic diagram of the installation of the upper acrylic plate pillar

Figure 3.1.4.5 Effect diagram of the installation of the upper acrylic plate pillar

3.1.5 Battery box and Keywish Uno R3 motherboard installation

Step 1: Keywish Uno R3 Motherboard pillar installation



Note: This side is the front of the acrylic plate

Figure 3.1.5.1 Schematic diagram of Keywish Uno R3 Motherboard pillar installation

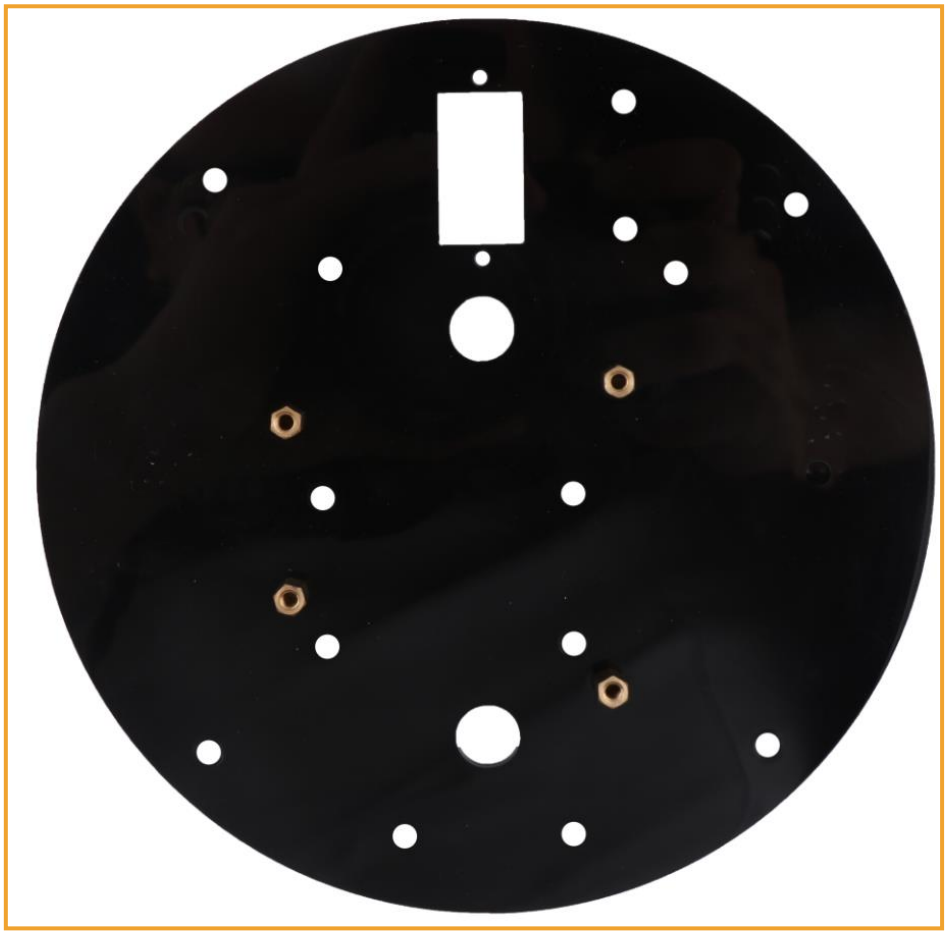


Figure 3.1.5.2 Effect diagram of Keywish Uno R3 Motherboard pillar installation

Step 2: Battery box installation

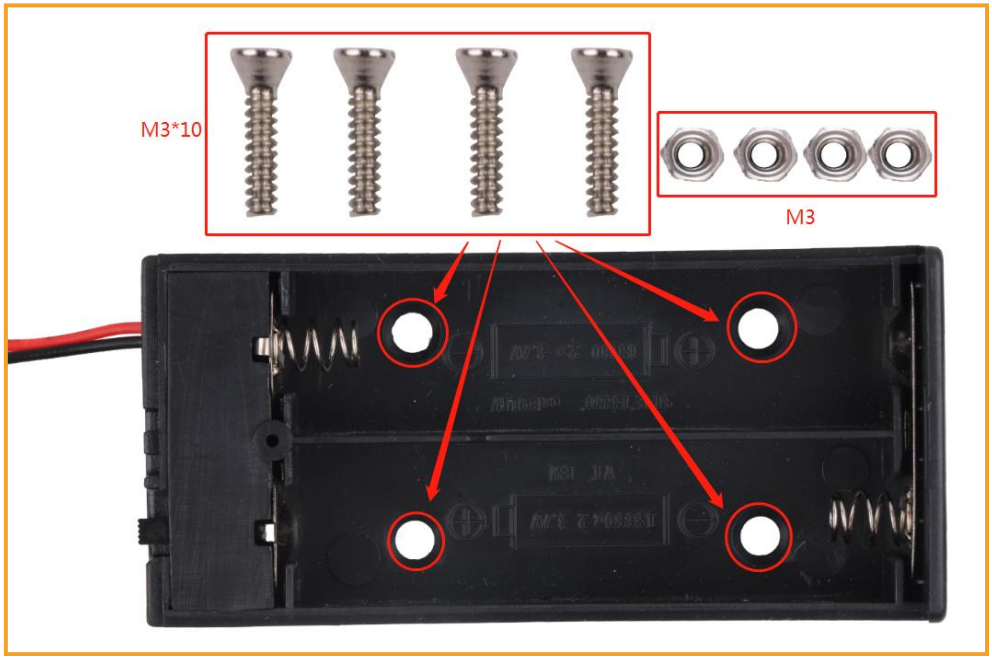


Figure 3.1.5.3 Schematic diagram of Battery box fixed

Figure 3.1.5.4 Effect diagram of Battery box fixed

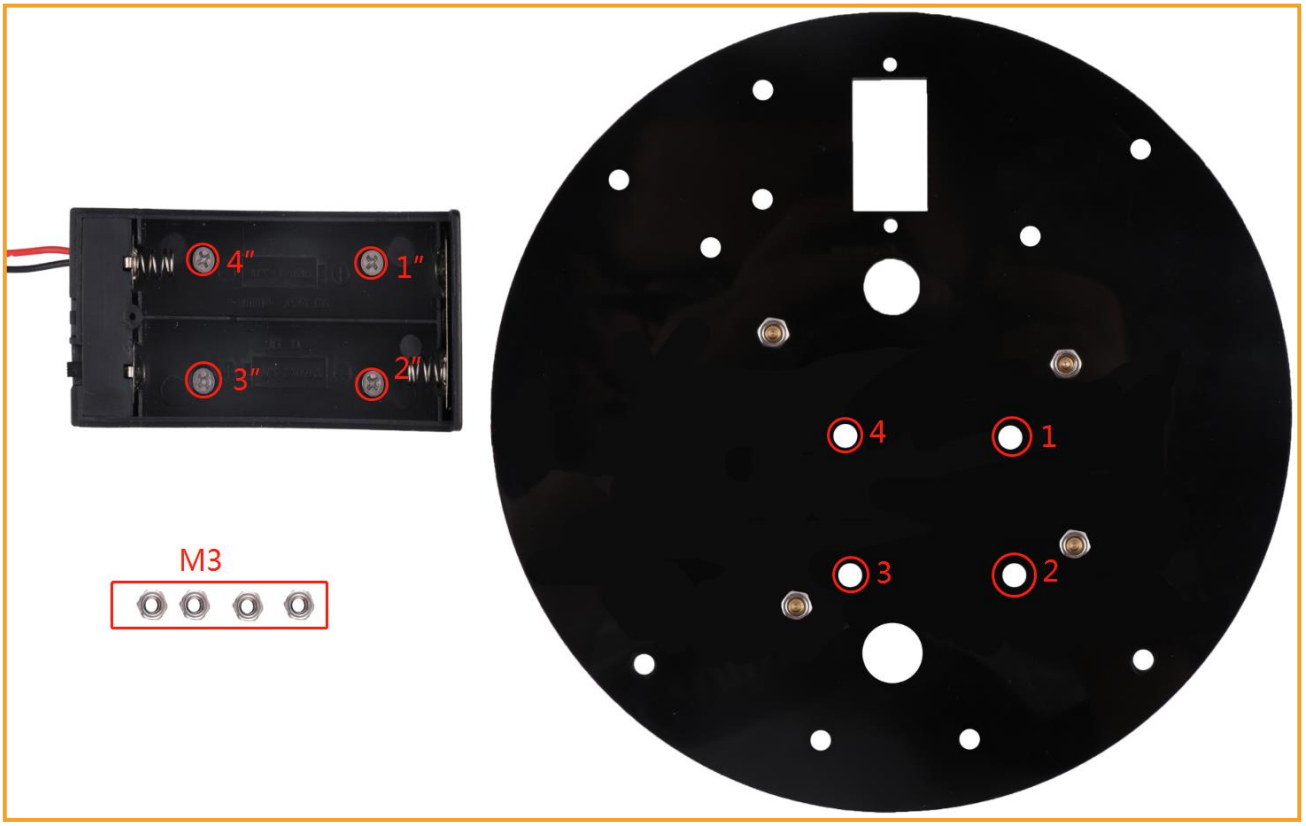
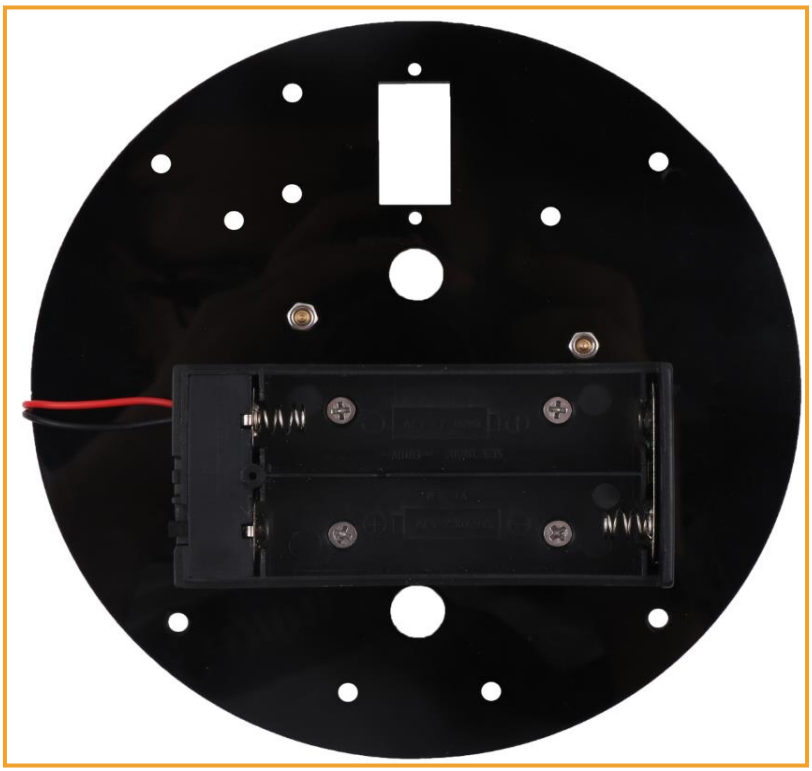


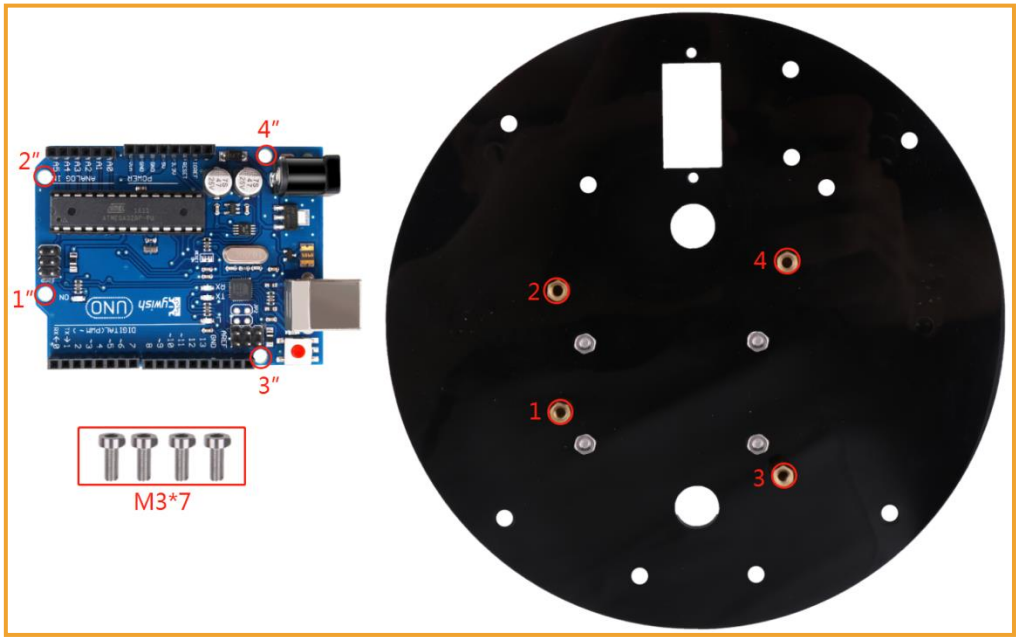
Figure 3.1.5.5 Schematic diagram of Battery box installation



Note：Put the batteries on the battery box which has been installed.

Figure 3.1.5.6 Effect diagram of Battery box installation

Step 3: Keywish Uno R3 motherboard installation



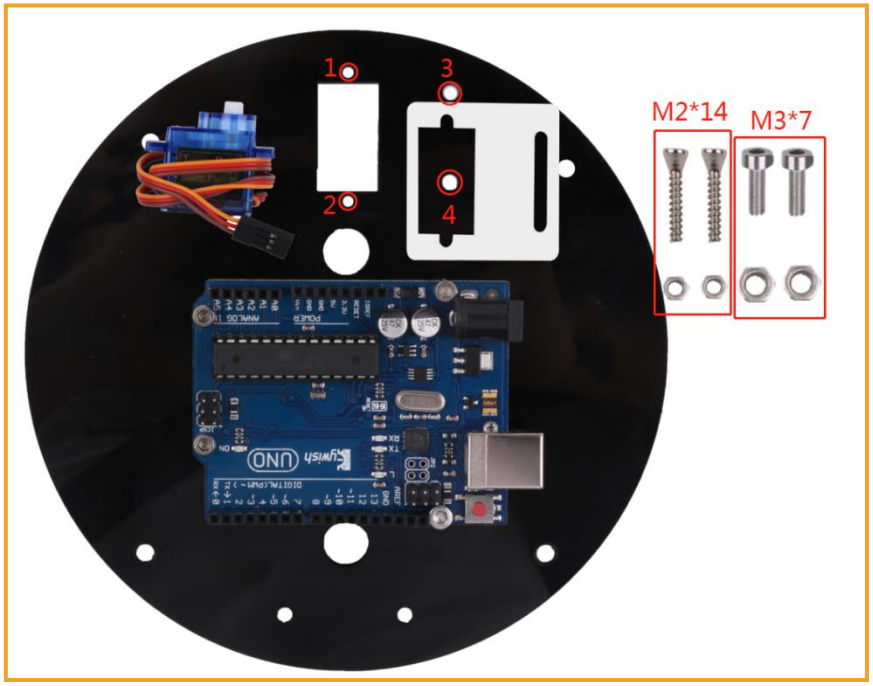
Note: 1 corresponds to 1" 2 corresponds to 2" 3 corresponds to 3" 4 corresponds to 4", fixed with M3*7 screw.

Figure 3.1.5.7 Schematic diagram of Keywish Uno R3 motherboard installation

Figure 3.1.5.8 Effect diagram of Keywish Uno R3 motherboard installation

3.1.6 Installation of the Servo and ultrasonic

Step 1: Servo Installaion



Note: Firstly, place the servo base on the corresponding position of the acrylic plate and then put the servo and fix it with screws. The holes 1 and 2 are fixed with M2*14 screws, and the 3 and 4 holes are fixed with M3*7 screws.

Figure 3.1.6.1 Schematic diagram of steering gear installation

Figure 3.1.6.2 Effect diagram of servo installation

Step 2: Installation of ultrasonic

Note: Firslty, use the rudder paddle to fix the ultrasonic bracket

Figure 3.1.6.3 Schematic diagram of steering gear installation

Note: When fixing, please put the rudder paddle in the middle of the ultrasonic bracket to fix it.

Figure 3.1.6.4 Effect diagram of the rudder propeller fixed ultrasonic bracket

After using the rudder propeller to fix the ultrasonic bracket, you can add the ultrasonic bracket to the servo, but don't worry. In order to reduce the angle adjustment of the steering gear, we first adjust the servo to 90 degrees, and the following procedure (Lesson\ModuleDemo\ Servo\ServoCorrect\ ServoCorrect.ino") is

```
#include <Servo.h>
Servo myservo;// create servo object to control a servo
int pos = 90; // variable to store the servo position
void setup() {
   myservo.attach(13);// attaches the servo on pin 13 to the servo object
}

void loop() {
   myservo.write(pos);// tell servo to go to position in variable 'pos'
   delay(15);// waits 15ms for the servo to reach the position
}
```

copied into the compilation environment, then connect servo signal line (orange) is to the Arduino No. 13 IO port and fixed by screws, as shown in Figure 3.1.6.5.

Figure 3.1.6.5 Effect diagram of ultrasonic installation

3.1.7 Infrared obstacle avoidance module installation

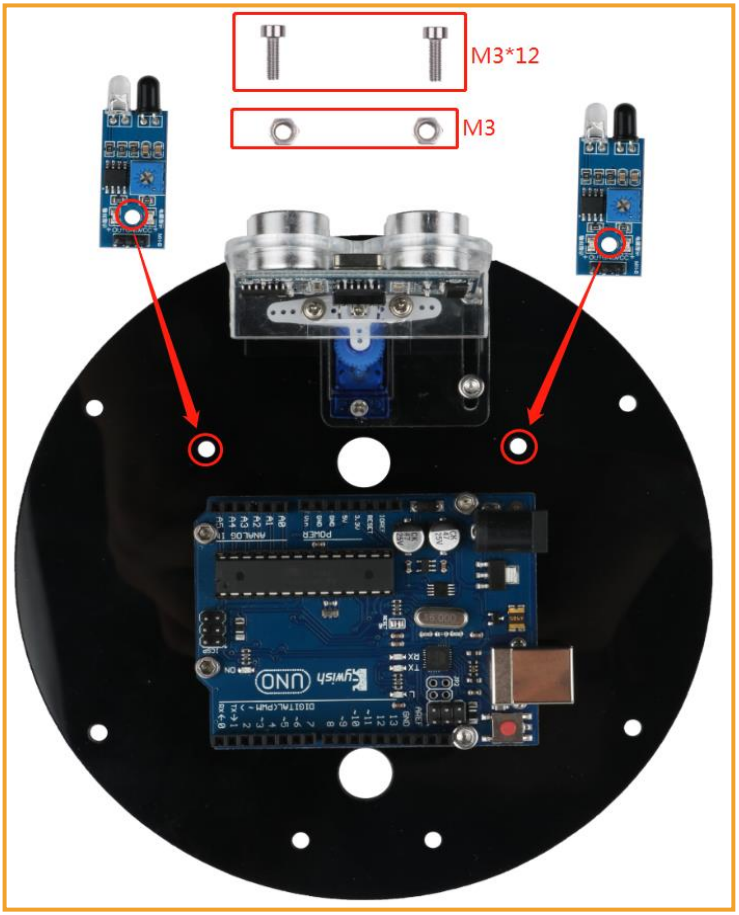


Figure 3.1.7.1 Schematic diagram of infrared obstacle avoidance module installation

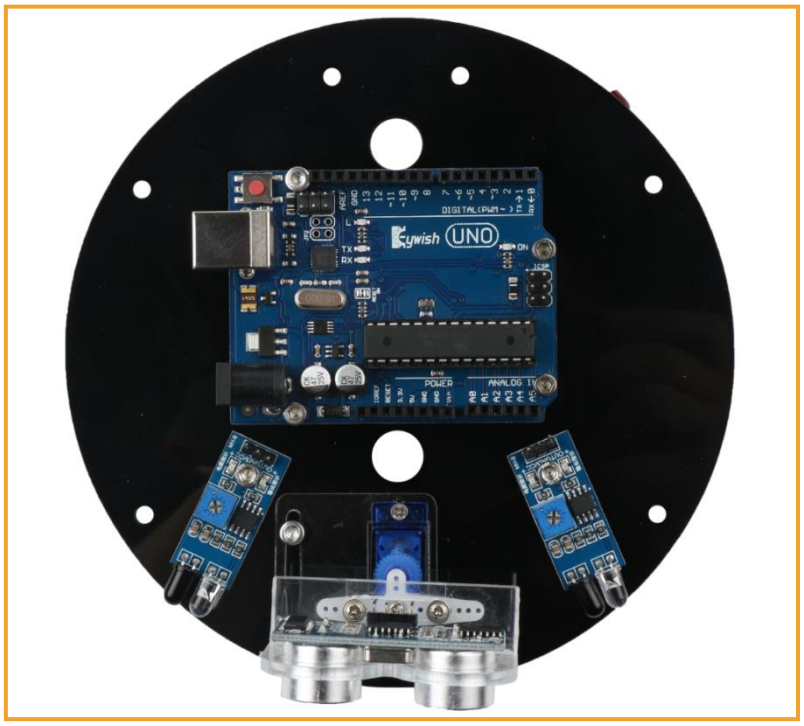


Figure 3.1.7.2 Effect diagram of infrared obstacle avoidance module installation

3.1.8 Voltage display module installation

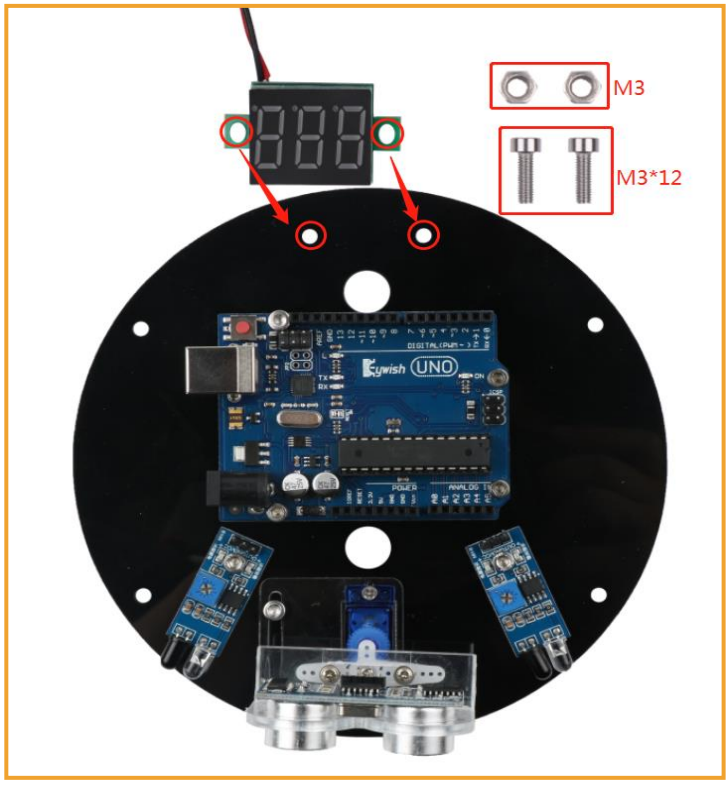


Figure 3.1.8.1 Schematic diagram of voltage display module installation

Figure 3.1.8.2 Effect diagram of voltage display module installation

3.1.9 Infrared remote control receiver installation

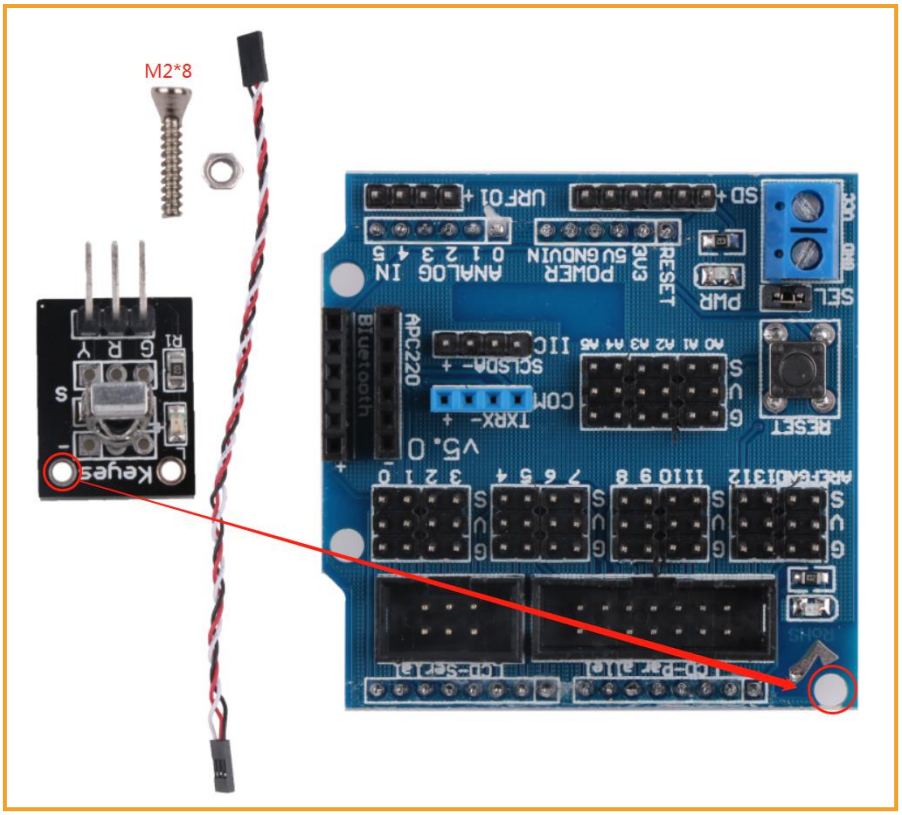


Figure 3.1.9.1 Schematic diagram of infrared remote control receiver installation

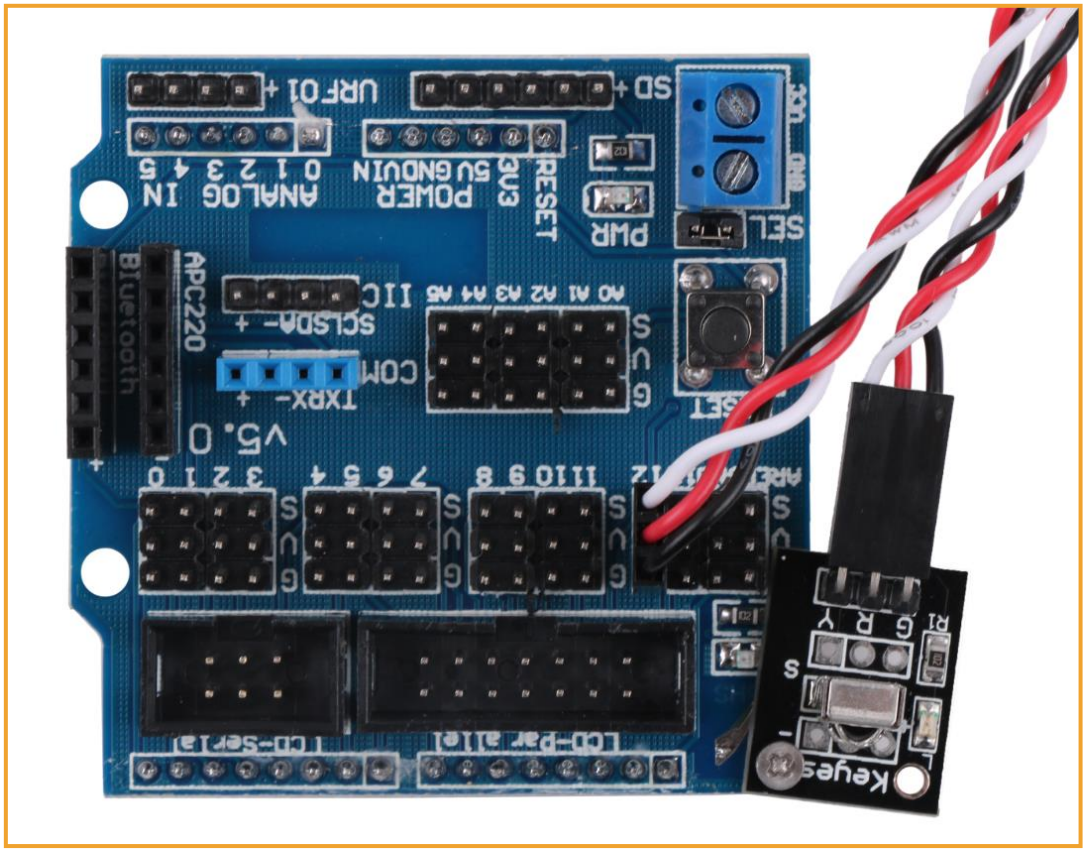


Figure 3.1.9.2 Effect diagram of infrared remote control receiver installation

3.1.9 Welding power cord

Connect the power cable: Firstly, find the matching two power cables (the same as the wires used by the motor, one red and one black) and connect the two cables to the DC power head. The DC power connector is shown in Figure3.1.10.1. The rubber ring marked as "1" can be removed to open the shell and then welding the wires to the +12V and GND, as shown in Fig.3.1.36.

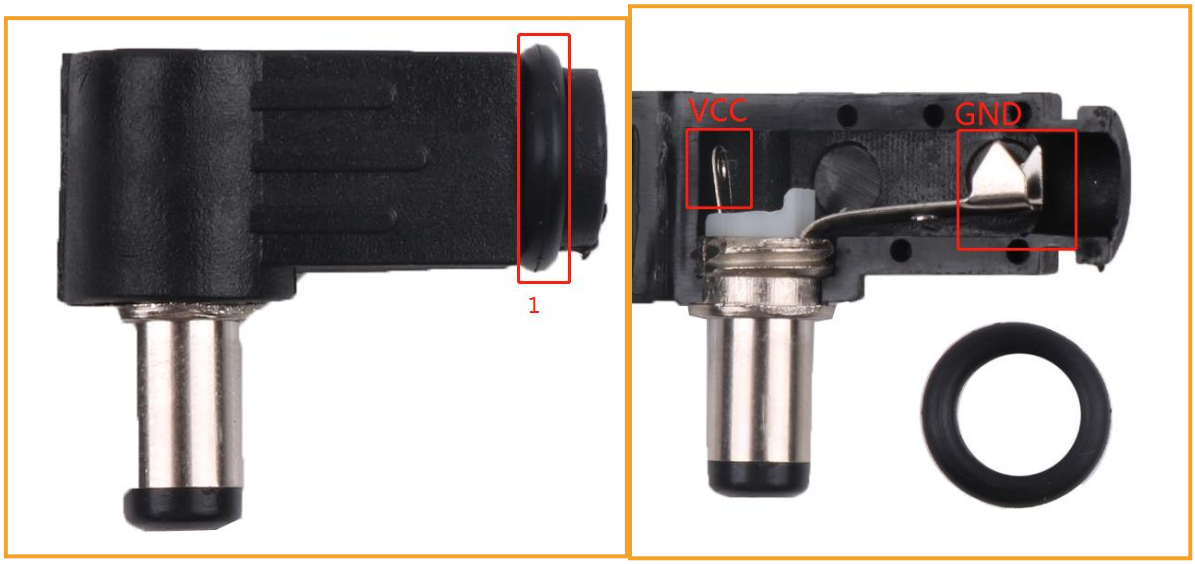


Figure 3.1.9.1 Power DC Head

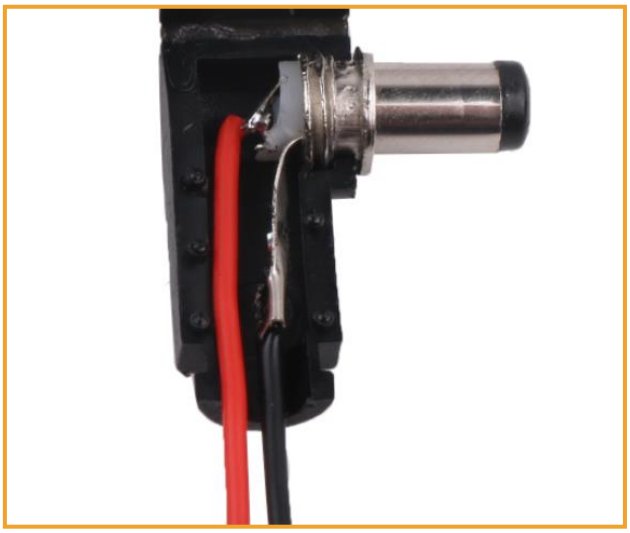
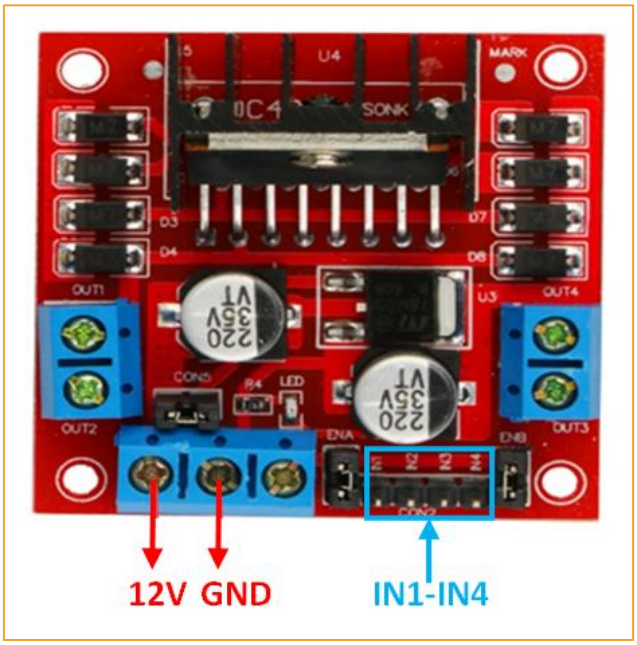


Figure 3.1.9.2 Schematic diagram of wire welding

3.1.10 Whole Assembly

For the whole assembly, first insert the “4Pin wire” into the “IN1—IN4” on the motor drive, and thread the other end of the tracing module from the bottom of the trolley to the top, as shown in Figure 3.1.11.1.

Then screw the battery box and the voltage display module and the DC head wire together (as shown in Figure 3.1.11.2), connect the positive pole (red line) to +12V, and the negative pole (black line) to GND.



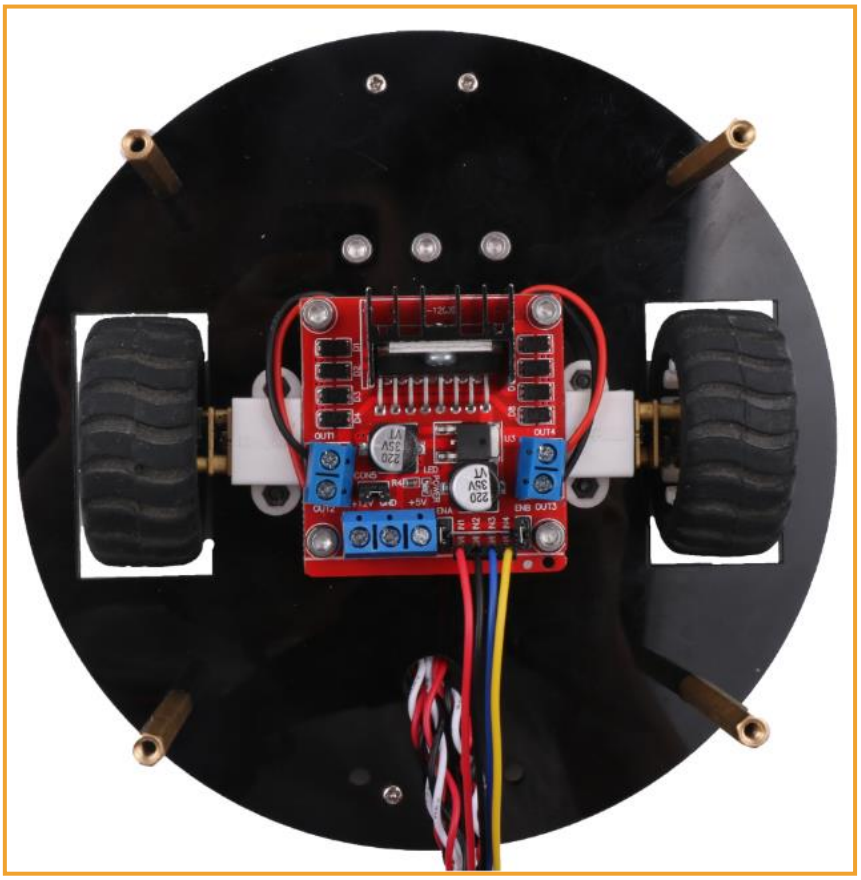


Figure 3.1.10.1 Effect Diagram of Wires Arrangement

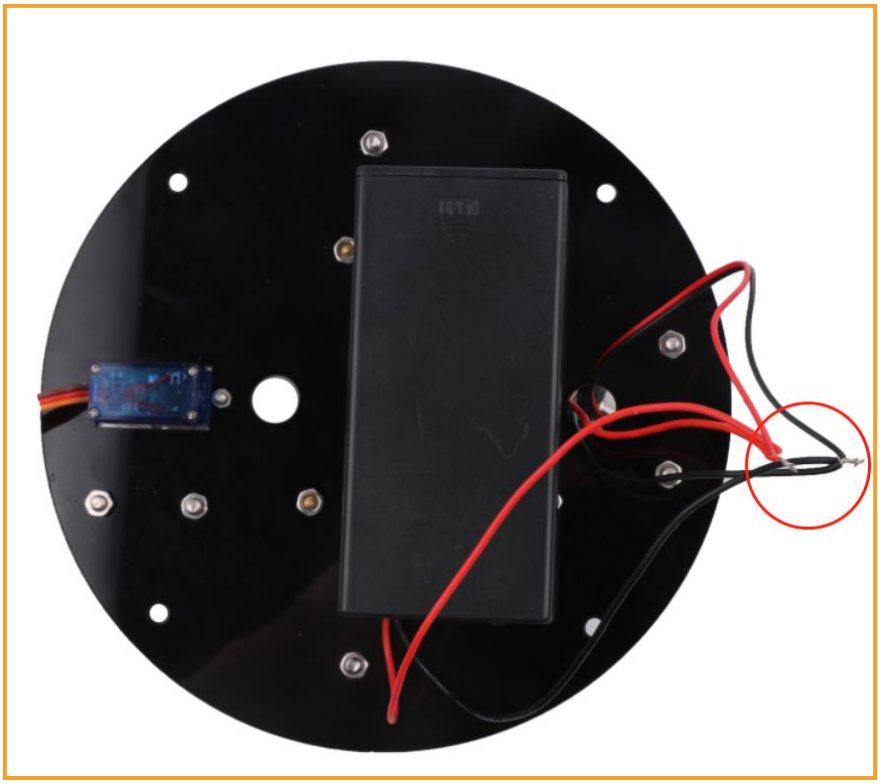
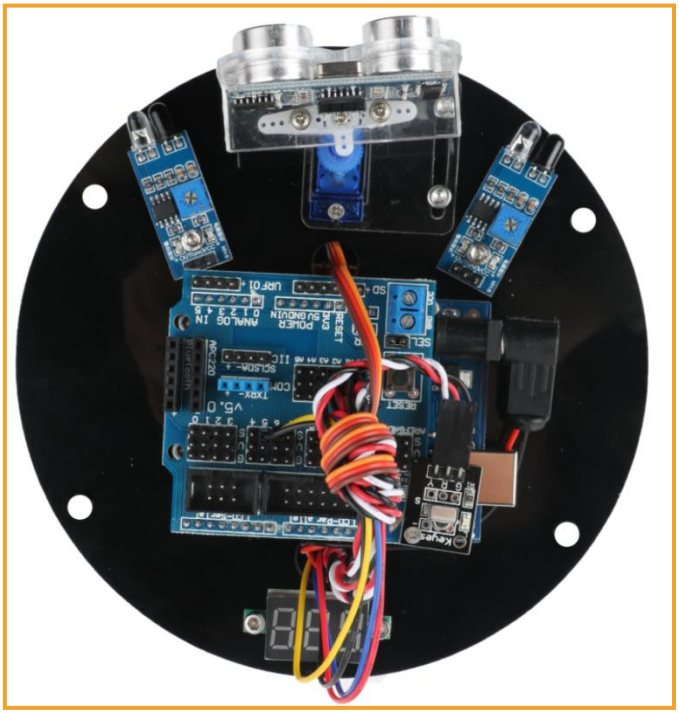


Figure 3.1.10.2 Battery connection diagram



Note:When docking, put the lines of the tracing module and the motor drive module all on the upper acrylic plate, and install the expansion board.

Figure 3.1.10.3 Upper acrylic plate docked with the lower acrylic plate

3.1.11 Expansion board wiring diagram

Through the previous steps, we have completed the installation of the main structure of the car. Now we wire the car to connect the cable. The connection method is as shown in the following figure. For the specific experiment of a certain module, we will introduce it in detail through the following chapters.

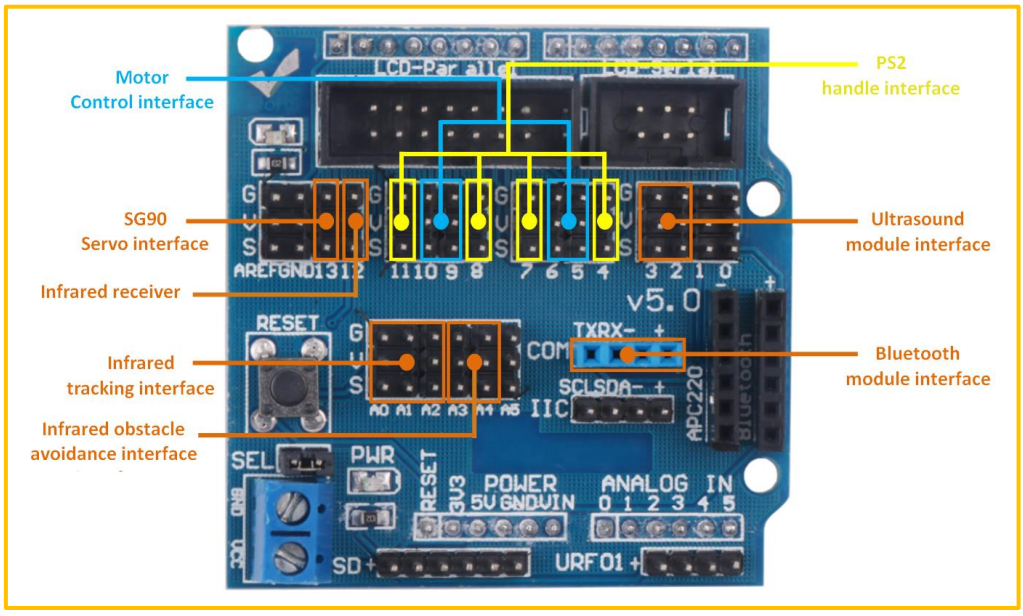
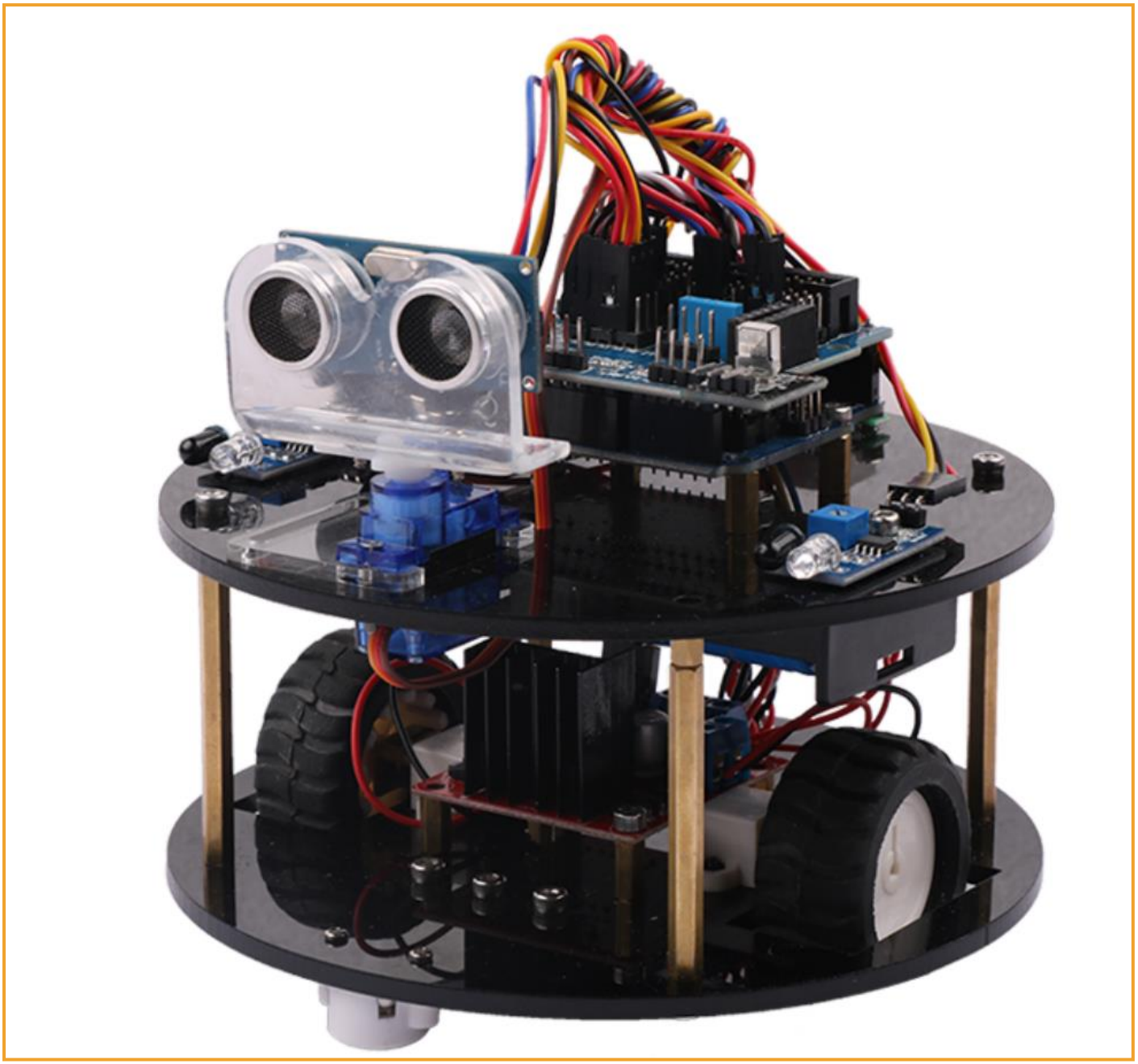


Figure 3.1.11.1 Expansion board connection diagram

The full installation of the car is as shown below



So far, the basic assembly of car has been completed. We believe you have some basic knowledge of your car’s structure, function and some modules through a short period of time, then you can achieve the corresponding functions only by downloading the program to the development board, each function has a corresponding program in CD, so please enjoy playing. However, if you can read the program and write your own program, there will be more fun, now let's go to the software section!

3.2 Beetle Bot Module experiment

3.2.1 Walking Principle of the Car

In the "Beetle-Bot" car, we choose the L298N as the motor driver chip for it is a high voltage and current full-bridge driver chip, the chip uses 15 pins package. It is a special motor driven integrated circuit (two H bridges) with high voltage and current full-bridge driver. And it contains 4 channel logic drive circuit, basically belongs to a kind of two-phase and four-phase special motor drive which contains two H bridges of high voltage large current. The output current is 2A, the maximum current is 4A, the maximum working voltage is 50V, which can drive the load under 46V and 2A, such as high power DC motor, stepper motor, solenoid valve and so on. The chip with two enable control terminals uses the standard logic level to control signals, allows or prohibits the device to work when the input signal is not interfered, it has a logic power input terminal which can enable the internal logic circuit to work under low voltage, and feedback the variation to the control circuit. Especially, the input can be connected directly with the MCU and easily controlled. When the DC motor is driven, the stepper motor can be directly controlled, and it can be turned forward and reversely, which only needs to change the logic level of the input. The pin arrangement is shown in Fig.3.2.1. The pin 1 and 15 can separately connect to the current sampling resistor and form the current sensing signal.

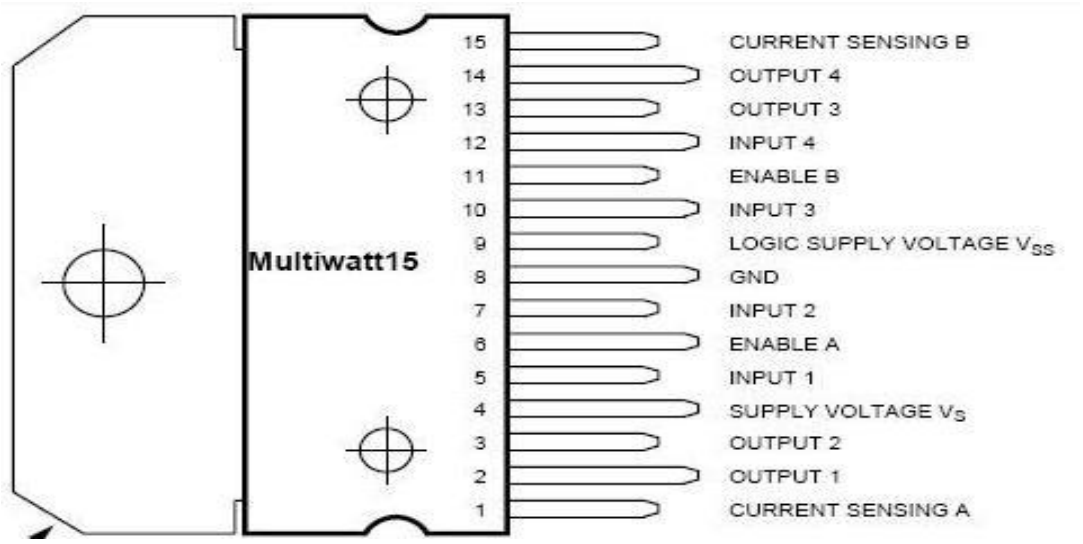


Figure .3.2.1.1 Arrangement of Chip Pins

L298N can drive 2 motors which are connected between OUTl, OUT2 and OUT3, OUT4. 5, 7, 10 and 12 pin are connected to input control level for controlling the positive and negative rotation of the motor, ENA, ENB are connected to control enable terminal for controlling the running and shutdown of the motor. Its characteristics:

- Signal indicator
- The speed is adjustable

- The strong anti-interference ability with photoelectric isolation
- Overvoltage and overcurrent protection
- Controlling of two motors separately
- Controlling the stepper motor
- The speed control with PWM pulse width
- Positive and negative rotation

ENA	IN1	IN2	Motor status
H	H	L	Forward
H	L	H	Reversal
H	IN2	IN1	Quick stop
L	X	X	Stop

Figure .3.2.1.2 Logic Function Chart

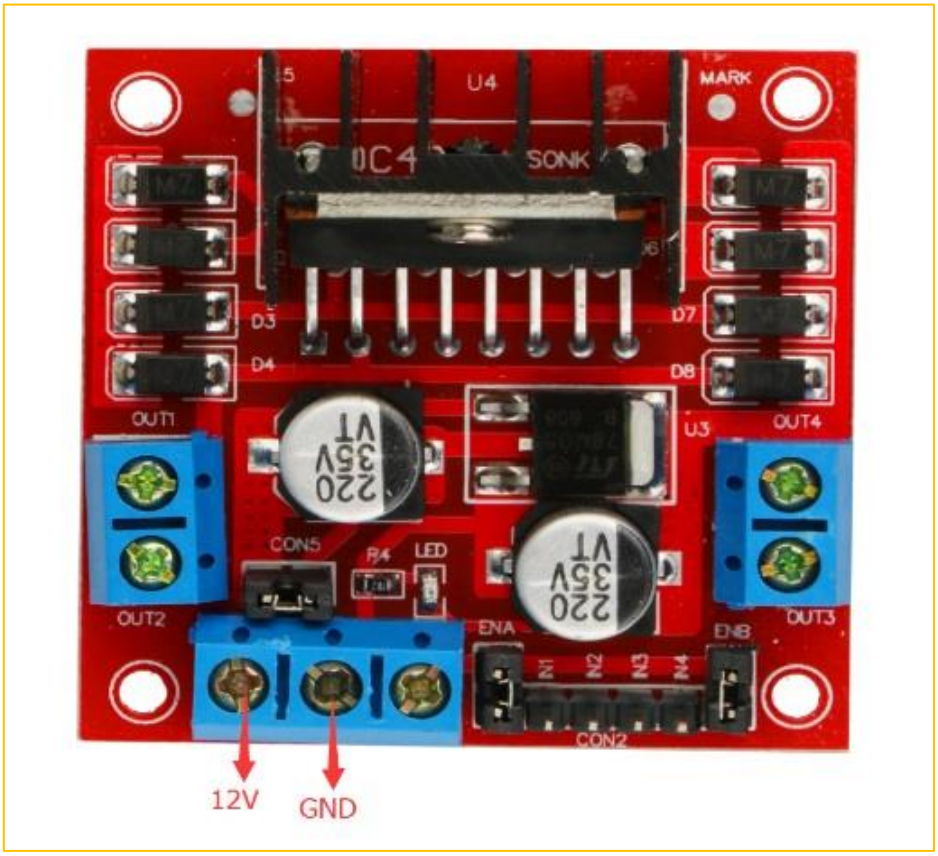


Fig.3.2.1.3 Module Physical Map

Detailed L298N chip data please refer to "Beetle-Bot\Document\L298N_datasheet.pdf"

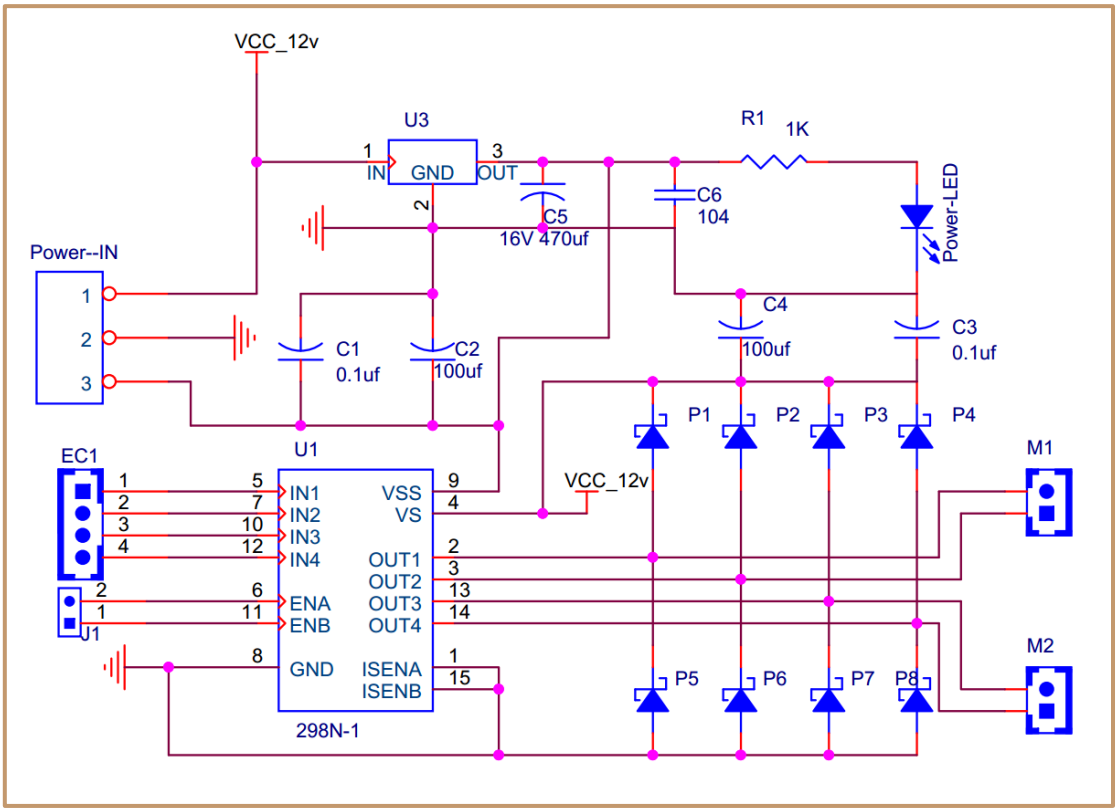


Figure .3.2.1.4 Schematic Diagram of Motor Drive

Four DC motors with high power L298N drive enable "Beetle-Bot" to run faster than conventional two-wheel car, the acceleration time is shorter and the structure is more stable. However, in the actual application, we need to adjust the speed of the car because of environmental or other factors, yet this does not affect the forward, backward, stop, flexible steering of the car, so we use PWM to control the speed of the motor(Note: PWM is a way to simulate the simulation output via square waves with different duty cycles.), Arduino PWM port outputs a series of square waves with fixed frequency, the power and current of the motor can be amplified after receiving the signal, thereby changing the motor's speed. The speed coordination of two motors on the right and left wheels can achieve the forward, backward, turning and other functions of the car. Figure 2.4.5 shows the sequence diagram of PWM duty cycles.

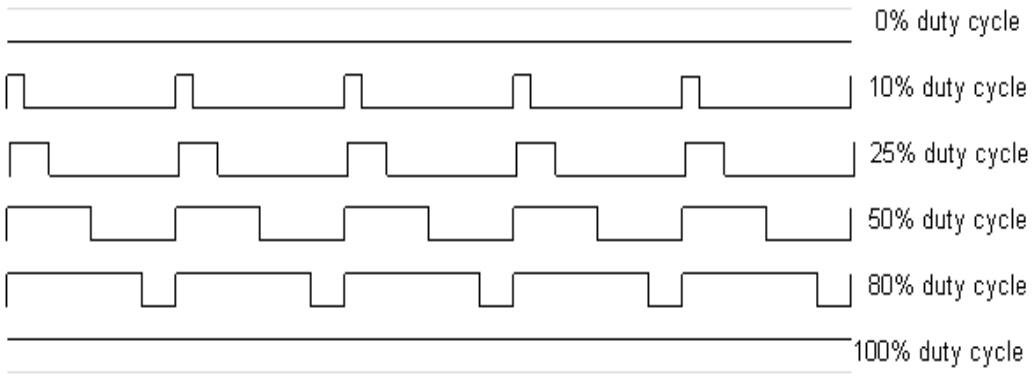


Figure .3.2.1.5 Sequence Diagram of PWM Duty Cycles

In Arduino, analog voltage can't be output, only 0 or 5V digital voltage value, we can use high resolution counter and the duty cycles of the square wave modulation method to encode a specific level of analog signal. The PWM signal is still digital, because at any given time, the full amplitude of DC power supply is either 5V (ON) or 0V (OFF). The voltage or current source is added to the analog load with a ON or OFF repetitive pulse sequence. When the DC power supply is added to the load, the power supply is on, otherwise the power supply is off. As long as the bandwidth is enough, any analog value can use PWM to encode. The output voltage value is calculated by the on and off time. Output voltage = (turn-on time / pulse time) * maximum voltage. Fig.2.4.6 shows the corresponding voltage to the pulse change.

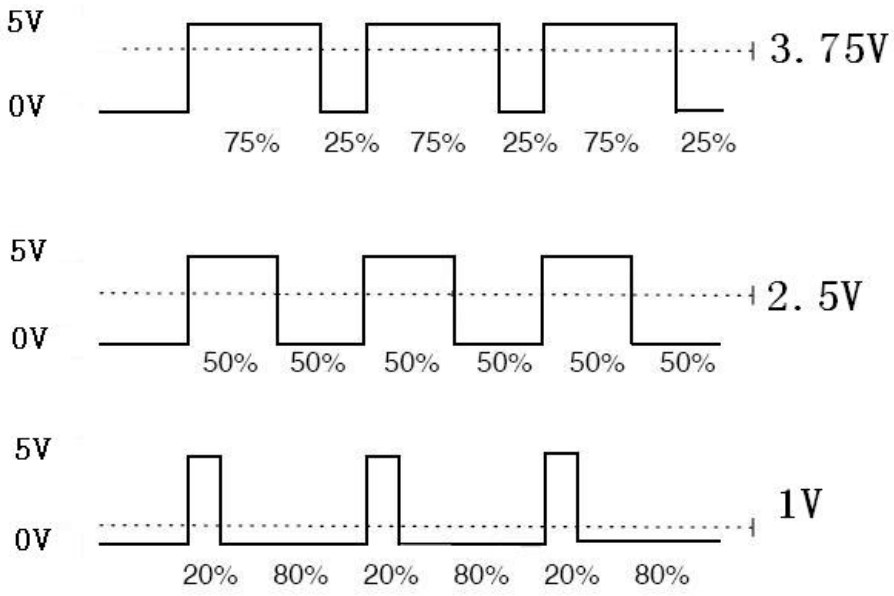
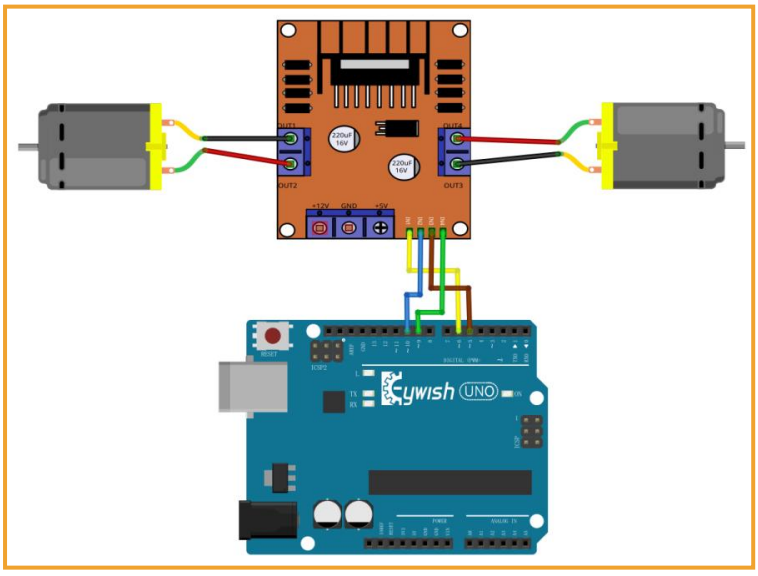


Figure 3.2.1.6Relation between Pulse and Voltage

In the "Beetle-Bot" car experiment, we use Arduino UNO R3 as the main control board. By referring to the chip data, we will know that Arduino UNO has 6 PWM pins, namely digital interfaces 3, 5, 6, 9, 10, 11, and we select 5, 6, 9, 10 as the motor control IO, the connection is shown in Fig3.1.2.7



L298N	Arduino Uno
IN1	D6
IN2	D10
IN3	D5
IN4	D9

Figure 3.1.2.7Arduino and L298N driver board connection diagram and connection table

The L298N and Arduino expansion board wiring is as follows:

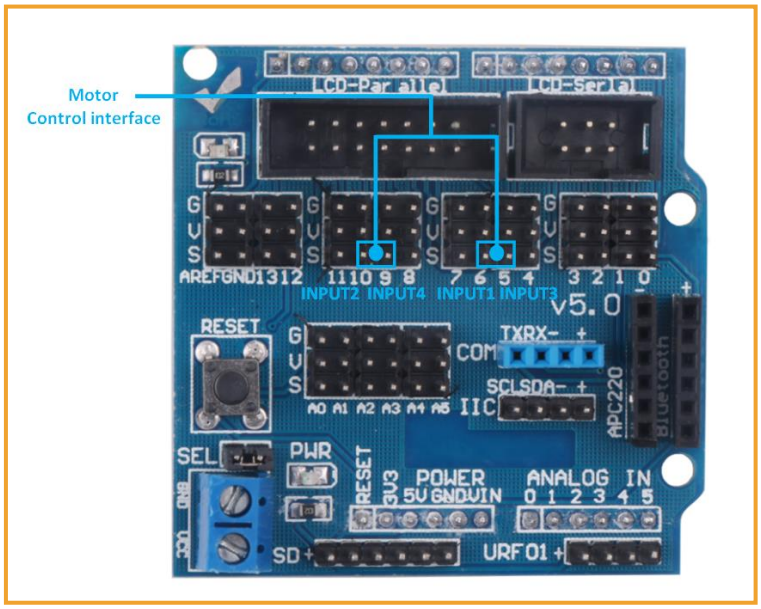


Figure.3.1.2.8 L298N and Arduino Expansion Board Connection Diagram

After the connection, we do not know whether the motor can work normally or not, so we need to do a simple test by copying the following code (You can also open the program in the CD directly.) into the IDE development environment and downloading to the development board. And turning on the power (power connection is introduced the tenth and eleventh steps in 3.1.2) to observe the wheels rotation, if "going forward 5s----stopping 1s---- going back 5s---- stopping 1s----turning left 3s----stop 1s----turning right 3s" are normal, the connection is correct, otherwise the polarities of the motor may be reversed, then you need to adjust slightly.

Program flow chart is as follows:

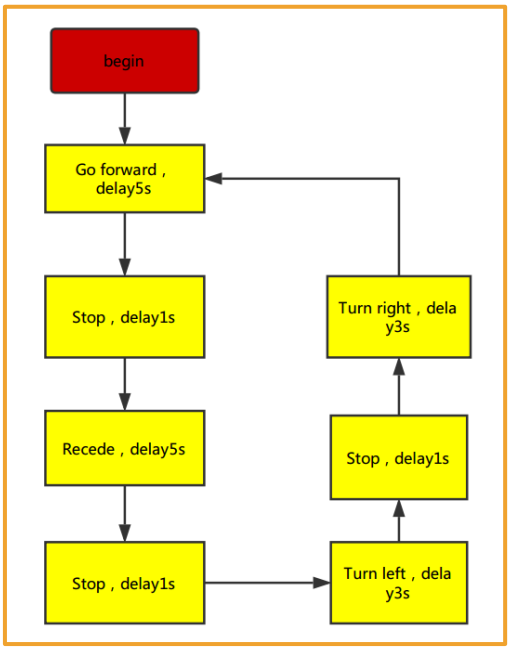


Figure 3.1.2.9 Motor Test Flow Chart

Note: This test and IO selection are only for reference, you can choose other IO ports or use other wiring methods according to your own ideas.

Test demo code path:“Lesson\ModuleDemo\Motor Test\MotorTest\MotorTest.ino”

```
int INPUT3_PIN = 5; //PWMA
int INPUT4_PIN = 9; //PWMA
int INPUT1_PIN = 6; //PWMB
int INPUT2_PIN = 10; //PWMB
void setup() {
  Serial.begin(9600);
  pinMode(INPUT3_PIN, OUTPUT);
  digitalWrite(INPUT3_PIN, LOW); // When not sending PWM, we want it low
  pinMode(INPUT4_PIN, OUTPUT);
  digitalWrite(INPUT4_PIN, LOW); // When not sending PWM, we want it low
  pinMode(INPUT1_PIN, OUTPUT);
  digitalWrite(INPUT1_PIN, LOW); // When not sending PWM, we want it low
  pinMode(INPUT2_PIN, OUTPUT);
  digitalWrite(INPUT2_PIN, LOW); // When not sending PWM, we want it low
}

void loop() {
  analogWrite(INPUT2_PIN, 0);
  analogWrite(INPUT1_PIN, 200);//the speed value of motorA is 200
  analogWrite(INPUT3_PIN, 0);
  analogWrite(INPUT4_PIN, 200);//the speed value of motorB is 200
  delay(5000);
  //******** ******************************//forward
  analogWrite(INPUT2_PIN, 0);
  analogWrite(INPUT1_PIN, 0); //the speed value of motorA is 0
  analogWrite(INPUT3_PIN, 0);
  analogWrite(INPUT4_PIN, 0); //the speed value of motorB is 0
  delay(1000);//******************************************//stop
  analogWrite(INPUT2_PIN, 200);//the speed value of motorA is 200
  analogWrite(INPUT1_PIN, 0);
  analogWrite(INPUT3_PIN, 200);//the speed value of motorB is 200
  analogWrite(INPUT4_PIN, 0);
  delay(5000);//*********************************************//back
  analogWrite(INPUT4_PIN, 0);
  analogWrite(INPUT3_PIN, 0); //the speed value of motorA is 0
  analogWrite(INPUT2_PIN, 0);
  analogWrite(INPUT1_PIN, 0); //the speed value of motorB is 0
  delay(1000);
  //******* ************************************//stop
```

```
  analogWrite(INPUT2_PIN, 0);
  analogWrite(INPUT1_PIN, 0);
  analogWrite(INPUT3_PIN, 0);
  analogWrite(INPUT4_PIN, 150);//the speed value of motorA is 150
  delay(3000);
  //****************************************//left
  analogWrite(INPUT2_PIN, 0);
  analogWrite(INPUT1_PIN, 0); //the speed value of motorA is 0
  analogWrite(INPUT3_PIN, 0);
  analogWrite(INPUT4_PIN, 0); //the speed value of motorB is 0
  delay(1000); //****************************************//stop
  analogWrite(INPUT2_PIN, 0);
  analogWrite(INPUT1_PIN, 150);//the speed value of motorB is 150
  analogWrite(INPUT3_PIN, 0);
  analogWrite(INPUT4_PIN, 0);
  delay(3000);//*** ************************************//right
}
```

Bynow, the car can move normally, next we will add several common sensor modules.

3.2.2 Infrared Obstacle Avoidance

3.2.2.1 Introduction of Infrared Obstacle Avoidance Sensor

Infrared obstacle avoidance module is a pair of infrared transmitting and receiving tubes, the former launches a certain frequency infrared, the receiving tube will receive the reflected infrared when the infrared detects the obstacles. After the signal is processed by the comparator circuit, the green LED lights, and the signal output port outputs digital signal at the same time (a low level signal). The detection distance can be adjusted through the potentiometer knob, the effective distance range is 2-30cm, the working voltage is 3.3V-5V. The sensor uses infrared, so the anti-interference ability is very strong, the measurement accuracy is very high when the distance is moderate. In addition, the module can be assembled easily and used conveniently, it can be widely used in robot obstacle avoidance, car obstacle avoidance and the black&white line tracing and many other occasions.

3.2.2.2 Working Principle

1, The module output port OUT can be directly connected with the IO port of the microcontroller, and directly drive a 5V relay; the connection mode is: VCC-VCC; GND-GND; OUT-IO (A3 and A4), as shown in Fig.3.2.9 and Chart 3.2.1.

2, The module uses the 3-5V DC power as power supply. When the power is on, the indicator will light.

3, The diameter of installation hole is 3mm, you can use the same size screws (screws in the kit).

Pin wiring definition (only for reference, you can define according to your own ideas):

arduino Uno	Infrared Obstacle Avoidance Module
VCC	VCC
GND	GND
A3	The left module
A4	The right module

Chart 3.2.2.1 Pin Wiring Definition

3.2.2.3 Module Parameters

The working principle of infrared obstacle avoidance sensor is very simple, that is the reflection property of objects. In a certain range, if there is no obstacle, the infrared ray emitted will gradually weaken because of the farther distance of transmission, and finally disappear. If there are obstacles, the infrared will be reflected to the receiving head. As soon as the sensor detects the signal, it can confirm that there are obstacles in front of the circuit board, the green indicator will light, the OUT port continuously outputs low level signal to MCU at the same time, the MCU conducts a series of analysis to ensure that the two wheels of car works properly and avoids the obstacle beautifully. The schematic diagram of the sensor is shown in Fig.3.2.8. Infrared detector can be divided into active and passive according to its working mode.

Active infrared detector is equipped with infrared light source, it can detect the location of the object through covering the light source, reflection, refraction and other optical means.

Passive infrared detector has no light source, and it can measure the position, temperature, or infrared imaging of the detected object by receiving the characteristic spectral radiation of the detected object.

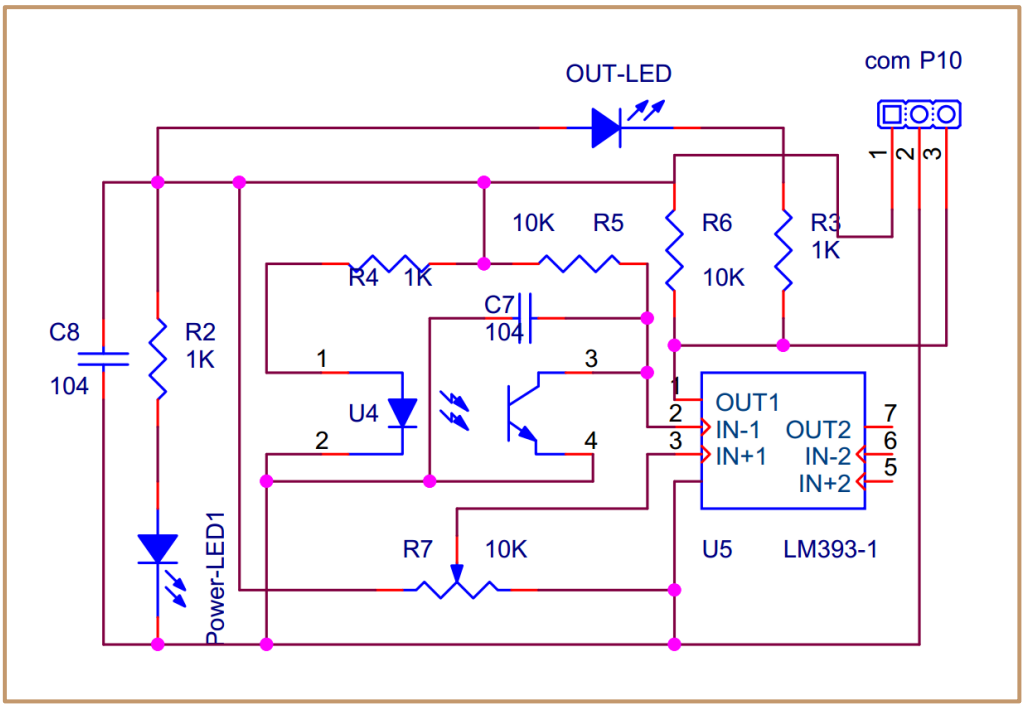


Figure .3.2.10 Infrared obstacle avoidance schematic diagram

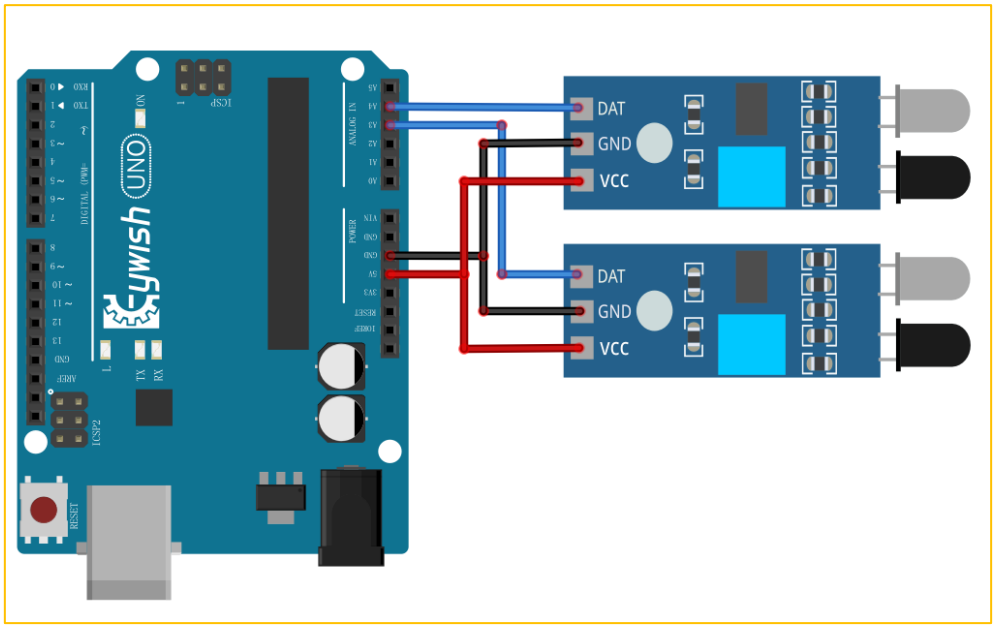


Figure 3.2.2.3 Connection of Arduino and Sensor

Note: This module can adjust the detection distance by the potentiometer, the detection distance is 2-30cm, if it is found that the distance detection is not very sensitive, you can use the potentiometer to achieve the desired results (rotating the potentiometer clockwise will increase the detection distance; counterclockwise will decrease), it is shown in Fig.3.2.2.4.。

Manual adjustment is shown in the following diagram:

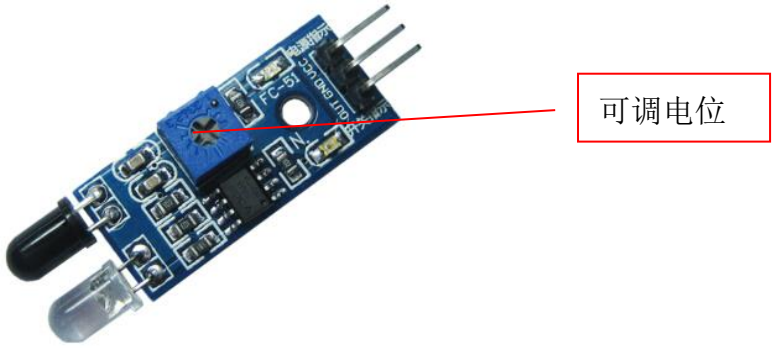


Figure .3.2.2.4 Diagram of Distance Detection Adjustment

3.2.2.4 Wire connection

As the figure shown below, the upper column is the "GND" interface, the middle column is the "VCC" interface, and the lower column is the "OUT" interface, where "A3" corresponds to the "OUT" of the left infrared obstacle avoidance, and "A4" corresponds to the right infrared obstacle avoidance. "OUT".

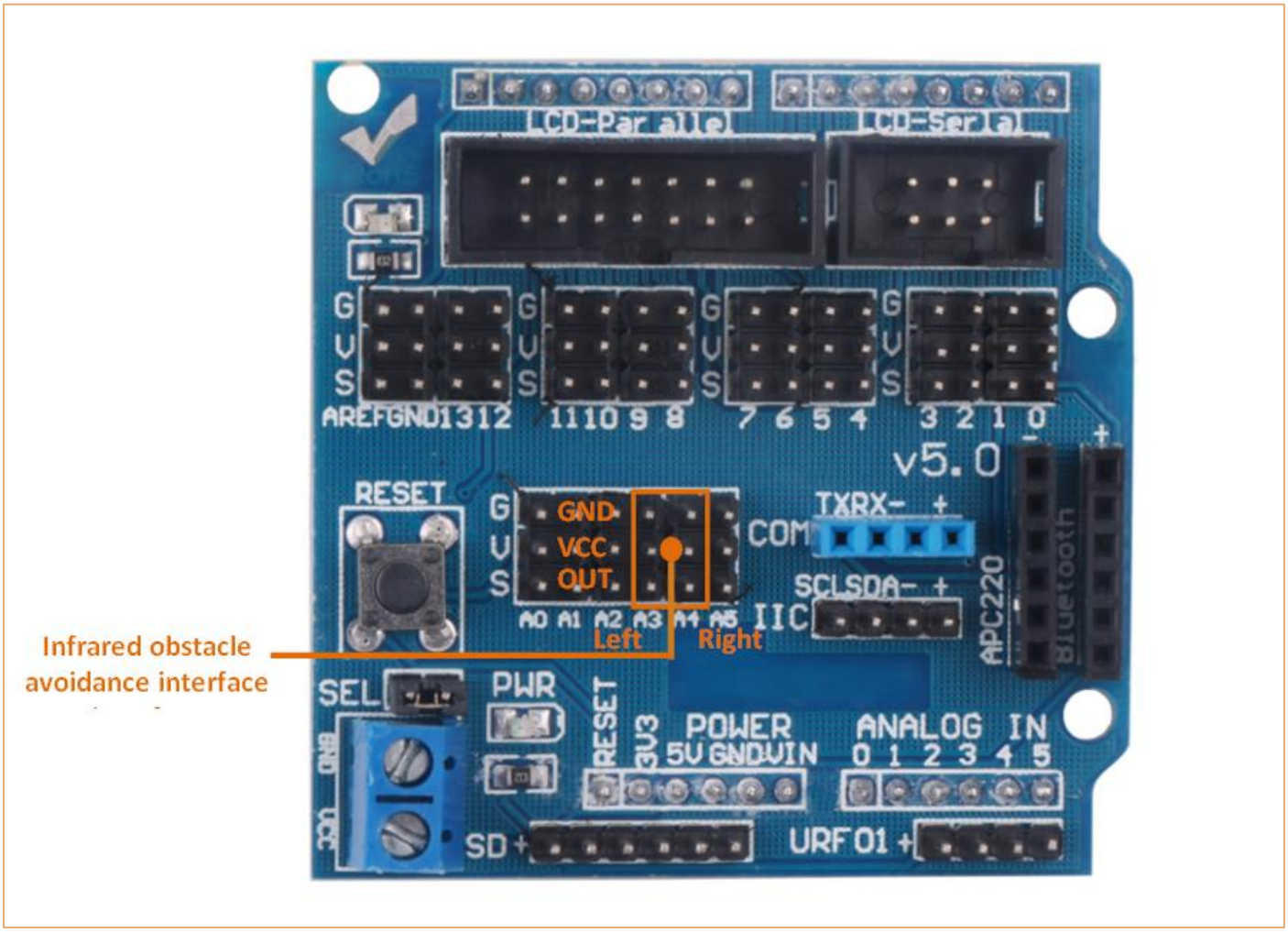


Figure 3.2.2.5 Wire connection diagram

3.2.2.5 Experimental Procedures

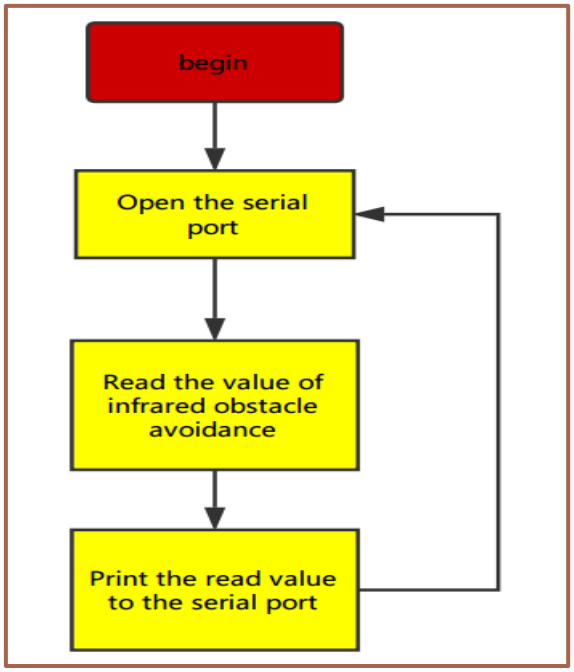
1, Fixing the two sensors on the car and connecting them to Arduino with wires.(Already done)

2, Testing the sensitivity of module, namely opening the switch on the battery box and the indicator will light, placing obstacles the 10cm away from the infrared tubes, adjusting the potentiometer until the output indicator lights up.

3, Module test. Copying the following code to the IDE compiler environment (you can also open the program directly in the CD), downloading it to the development board, opening the serial port monitor (baud rate is 9600) and observing the changes of data when there is an obstacle (Figure 3.2.14) and no obstacle (Figure 3.2.15).

Note: Here we connect the infrared obstacle avoidance signal output port to the analog port on Arduino (A0-A5), so the serial port prints out analog value, you can connect it to digital port (2-13), and the serial port will only print out "0" and "1".

Program flow chart is as follows:



Test demo path:：Lesson\ModuleDemo\InfraredAvoidanceTest\ InfraredAvoidanceTest.ino

```
const int leftPin = A3;
const int rightPin = A4;
int dl;
int dr;
void setup() {
  Serial.begin(9600);
  pinMode(leftPin, INPUT);
  pinMode(rightPin, INPUT);
  delay(1000);
}
void loop() {
  dl = analogRead(leftPin);
  dr = analogRead(rightPin);
  Serial.print("left:");
  Serial.print(dl);
  Serial.print("   ");
  Serial.print("right:");
  Serial.println(dr);
}
```

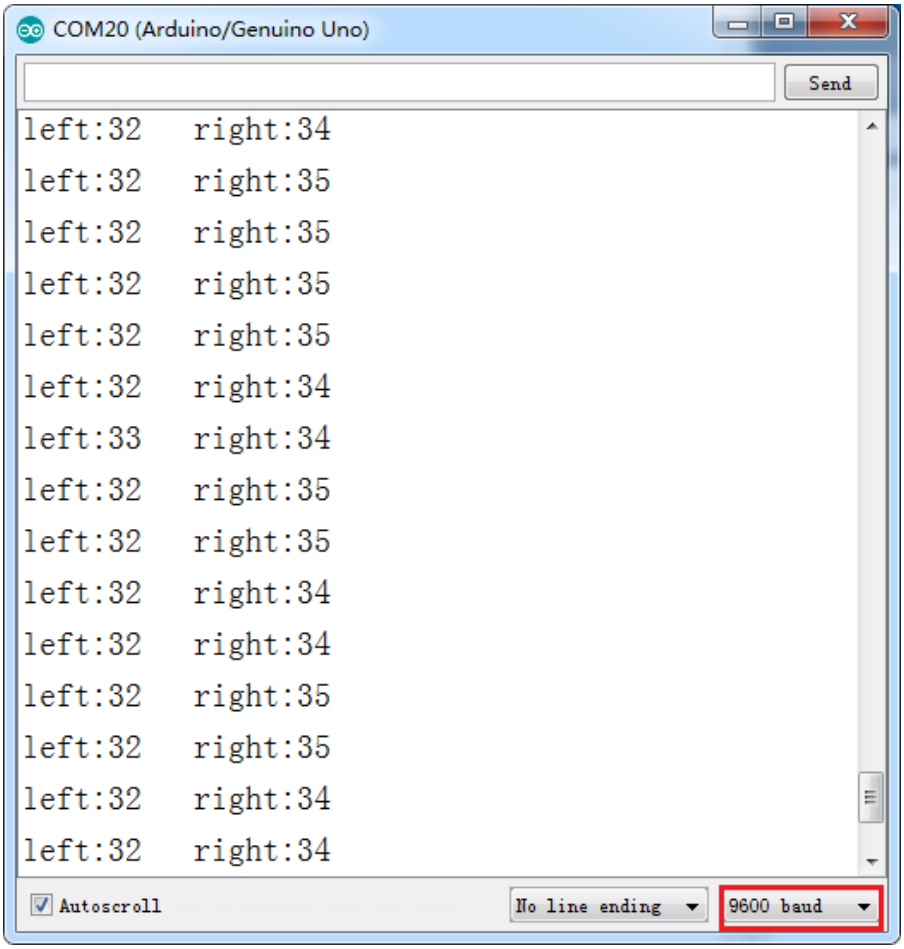


Figure .3.2.2.6 Diagram of Data with Obstacles

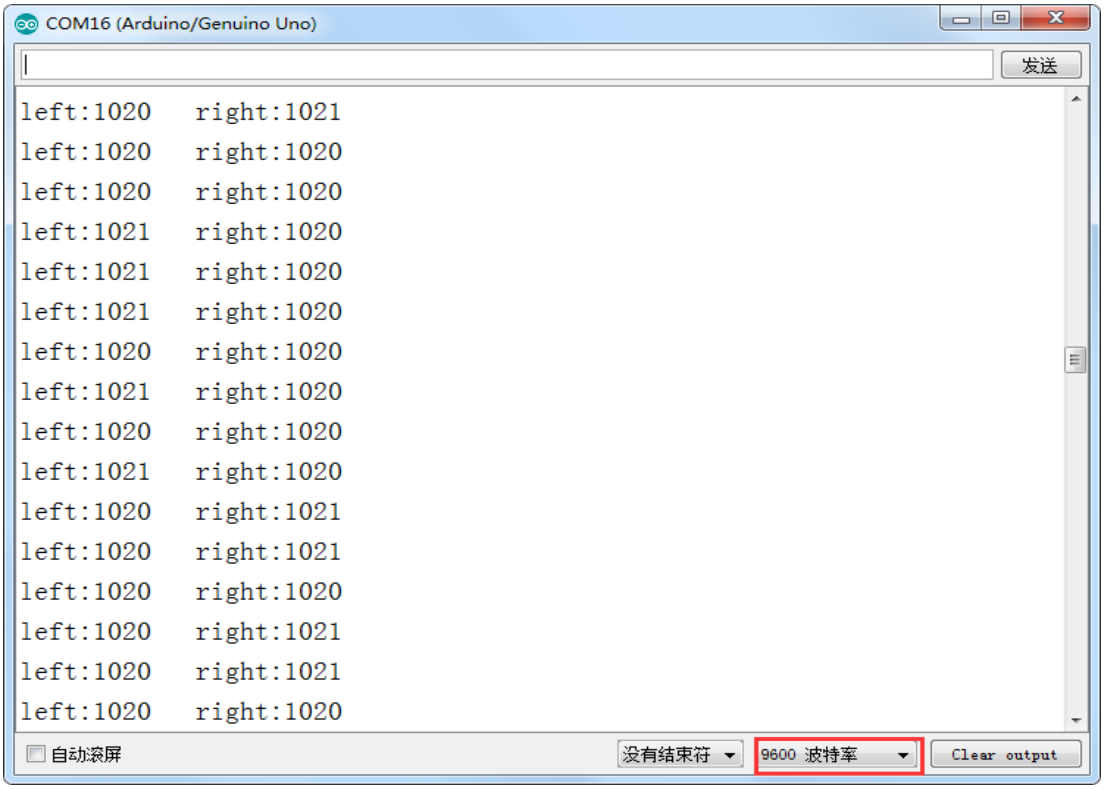
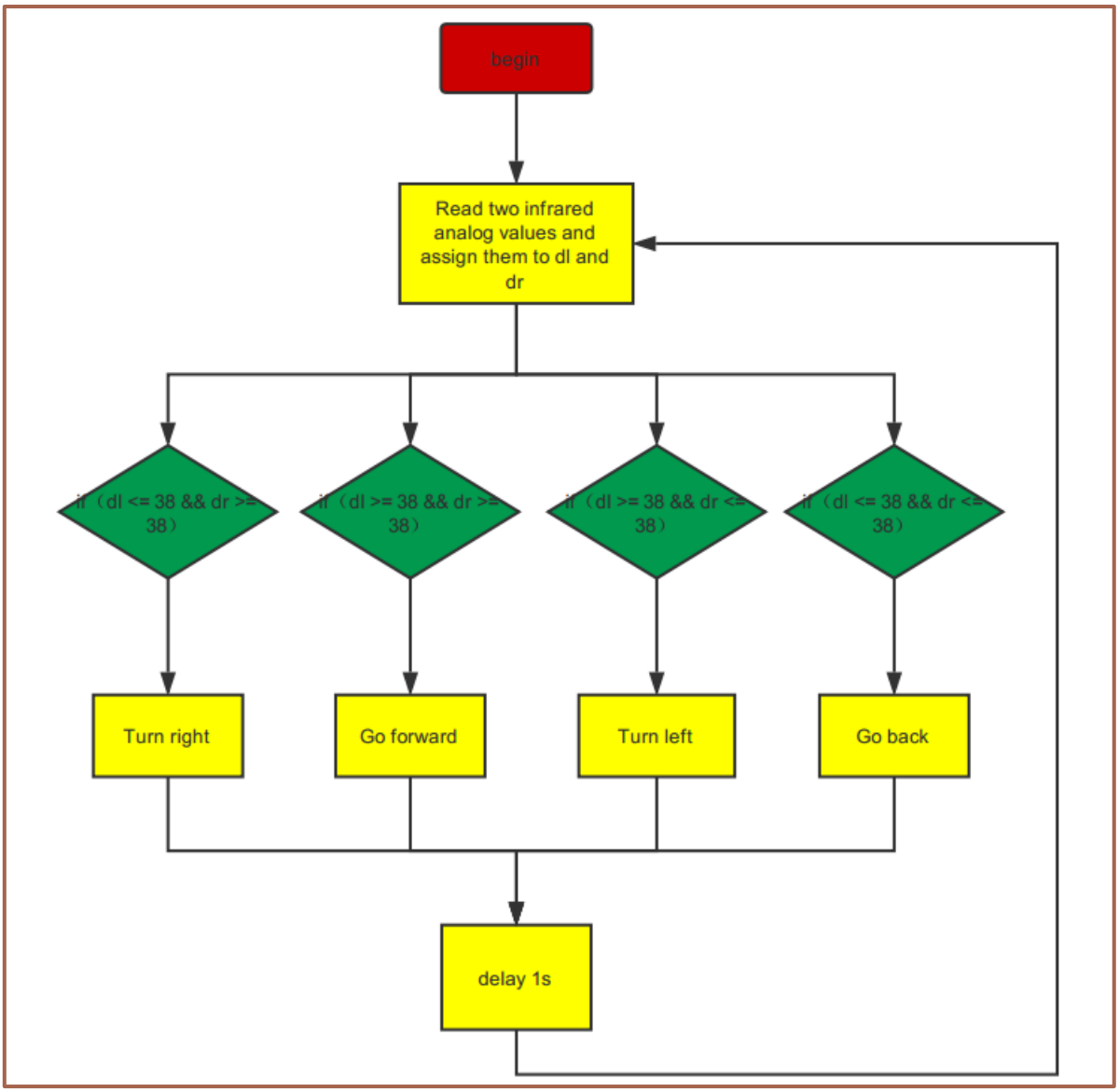


Figure .3.2.2.7 Diagram of Data without Obstacles

3.2.2.5 Software Design

3.2.2.5.1 Program flow chart



3.2.2.5.2 Program code

In the above steps, we have tested the car's driving and obstacle avoidance module respectively, they have achieved the desired results, here the "infrared obstacle avoidance" actually has been explained in this section, but we have not put the programs of two parts together, so we now integrate the program of the two parts and complete this great "infrared obstacle avoidance" project. Firstly, let's read the complete program:

Demo code path：Lesson\Advanced experiment\Beetle_InfraredAvoidance\Beetle_InfraredAvoidance\Beetle_InfraredAvoidance.ino

```
int INPUT3_PIN = 5; //PWMA
int INPUT4_PIN = 9; //DIRA**************************************Right
int INPUT1_PIN = 6; //PWMB
```

```
int INPUT2_PIN = 10 ; //DIRB**************************************Left
/*Define 4 motor control terminals, connected to IN1-IN4 on the motor RightValueive board。
*/
const int leftPin = A3;
const int rightPin  = A4; // Define the two signal receiving ends of the sensor
float LeftValue;
float RightValue;// Define two margins to store the values read by both sensors

void setup() {
  Serial.begin(9600); // Set the serial port baud rate to 9600,
  pinMode(leftPin, INPUT);
  pinMode(rightPin , INPUT);// Set the working mode of two sensor pins, namely "input"
  digitalWrite(INPUT3_PIN, LOW); // When not sending PWM, we want it low
  pinMode(INPUT4_PIN, OUTPUT);
  digitalWrite(INPUT4_PIN, LOW); // When not sending PWM, we want it low
  pinMode(INPUT1_PIN, OUTPUT);
  digitalWrite(INPUT1_PIN, LOW); // When not sending PWM, we want it low
  pinMode(INPUT2_PIN, OUTPUT);
  digitalWrite(INPUT2_PIN, LOW); // When not sending PWM, we want it low
  delay(1000);
}

void loop() {
  LeftValue = analogRead(leftPin);
  RightValue = analogRead(rightPin ); // Read the values collected by both sensors and assign
them to the defined variables。
  if (LeftValue >= 38 && RightValue <= 38) {/*If the value collected by the left sensor is
greater than or equal to 38 and the RightValue value is less than or equal to 38, the following
program in {} is executed (LeftValue> = 38, there is no obstacle on the left, RightValue<=
38 shows that there is an obstacle on the RightValue, so at this time the car is turning
to the side without obstacles (ie, turning to the left). From Figure 3.2.11, we know that
the simulated value will RightValueop below about 35 in the event of an obstacle , But in
order to reduce the error, we set the threshold at 38 to prevent the car from judging the
error because of the error. We can also customize other values. If we use the digital port
to receive the value of the sensor, we only return "0" and "1" ", But the same way to judge.
The reason why I did not use digital IO, because we use the digital IO port in other places.*/
    analogWrite (INPUT4_PIN, 180);
    analogWrite(INPUT3_PIN, 0);
    analogWrite (INPUT2_PIN, 0);
    analogWrite(INPUT1_PIN, 100); /*Set a PWM value, the maximum PWM is 255, but the speed
of the car should not be too fast when walking, otherwise it can not hit the obstacle in
time when the obstacles are suddenly encountered.*/
```

```
    Serial.print(LeftValue);
    Serial.print("   ");
    Serial.print(RightValue);
    Serial.print("   ");
    Serial.println("Turning left"); /*Through the "Serial Monitor" print the current status
of the car and the value collected by the sensor*/
    delay(300);
    analogWrite(INPUT4_PIN, 0);
    analogWrite(INPUT3_PIN, 0);
    analogWrite(INPUT2_PIN, 0);
    analogWrite(INPUT1_PIN, 0);
    delay(1000);//**********************************//Turning left
  }

  if (LeftValue <= 38 && RightValue <= 38) {/*If the value collected by the left sensor is
less than or equal to 38 and the RightValue value is less than or equal to 38, the following
program in {} is executed (LeftValue <= 38, indicating that there is an obstacle on the left
and RightValue<= 38 shows that there is an obstacle on the RightValue, so at this time the
car is rotated 180 degrees backwards. In the experiment, the car can just turn around 180
degrees after 500ms of rotation. Because the DC motor can not precisely control the angle
like the steering gear, An approximate value, of course, different motor speed is not the
same, the time used is not the same, so everyone in the experiment can be based on the
circumstances may be.)*/
    analogWrite(INPUT4_PIN, 255);
    analogWrite(INPUT3_PIN, 0);
    analogWrite(INPUT2_PIN, 255);
    analogWrite(INPUT1_PIN, 0);
    Serial.print(LeftValue);
    Serial.print("   ");
    Serial.print(RightValue);
    Serial.print("   ");
    Serial.println("Turning around");/*Through the "Serial Monitor" print the current status
of the car and the value collected by the sensor.*/
    delay(500);
    analogWrite (INPUT4_PIN, 0);
    analogWrite (INPUT3_PIN, 0);
    analogWrite (INPUT2_PIN, 0);
    analogWrite (INPUT1_PIN, 0); /*Rotate 180 degrees and stop*/
    delay(1000);//**********************************//Turning around
  }
  if (LeftValue <= 38 && RightValue >= 38) /*If the left sensor is less than or equal to
38 and the RightValue value is greater than or equal to 38, the following program in {} is
```

```
executed (LeftValue <= 38, indicating that there is an obstacle on the left, RightValue>
= 38 shows that there is no obstacle on the left, so at this moment the car is turning to
the side without obstacle, that is, turning to the RightValue)*/
  {
    analogWrite(INPUT4_PIN, 100);//the speed value of motorA is val
    analogWrite(INPUT3_PIN, 0);
    analogWrite(INPUT2_PIN, 0);
    analogWrite(INPUT1_PIN, 180);//the speed value of motorA is val
    Serial.print(LeftValue);
    Serial.print("   ");
    Serial.print(RightValue);
    Serial.print("   ");
    Serial.println("Turning RightValue");
    delay(300);
    analogWrite(INPUT4_PIN, 0);
    analogWrite(INPUT3_PIN, 0);
    analogWrite(INPUT2_PIN, 0);
    analogWrite(INPUT1_PIN, 0);/*Car must stop after each rotation, if you do not stop there
will be the phenomenon of rotating around. */
    delay(1000);//*********************************Turning RightValue
  }
  if (LeftValue >= 38 && RightValue >= 38) {/* Judge two values collected by the sensor.
If the value collected by the left sensor is greater than or equal to 38 and the RightValue
value is greater than or equal to 38, execute the following program in {} (LeftValue > =
38, indicating that there is no obstacle on the left and RightValue > = 38 that there is
no obstacle on the left, so the car at this time straight */
    int value = 200; /*When the straight line has a PWM value of 200, if the value is too
large, the speed of the car will be very fast, which may lead to the car can not hit the
obstacle in time when it encounters the obstacle. */
    analogWrite (INPUT4_PIN, value);
    analogWrite(INPUT3_PIN, 0);//the speed value of motorA is value
    analogWrite (INPUT2_PIN, 0);
    analogWrite(INPUT1_PIN, value);//the speed value of motorB is value
    Serial.print(LeftValue);
    Serial.print("   ");
    Serial.print(RightValue);
    Serial.print("   ");
    Serial.println("go");//*********************************/ / forward
  }
}
```

In the above program, we made comments on some part of the program in order to make you to learn and understand the program easily, the program is relatively simple, you can write your own programs to give the car more skills. Of course, if you want to use it directly, we have the corresponding source program in the CD.

3.2.3 Infrared Tracing

3.2.3.1 Introduction of Infrared Tracing Sensor

After infrared obstacle avoidance, let us learn the infrared tracing, their nature of work is the same, using basically the same module, just in different ways, to achieve different functions. In this section when we study, we have to pay attention to the color of the line we trace (the black line or white line), if your floor is black, you should trace the white line (pasting white line on the floor); if it is white, then you should trace the black line (pasting black line on the floor); you just need to make a distinct difference between the track and the ground environment.

Black line tracking refers to the car drives along the black line on the white floor, it can know where to drive according to the received reflected light due to the different light reflection coefficient on the black and white floor.

White line tracking refers to the car drives along the white line on the black floor, it can know where to drive according to the received reflected light due to the different light reflection coefficient on the white and black floor.

In the "Beetle-Bot" car, we use the TCRT5000 sensor as tracing module, TCRT5000 infrared reflection sensor is a photoelectric sensor which consists of an infrared emitting diode and an NPN infrared photoelectric transistor. The detectable reflective distance is 1mm-25mm, the sensor is specially equipped with M3 fixed installation holes, so it is easy to adjust the direction, it also has the 74HC14 Schmidt trigger inverter which ensure the clean signal, the good wave shape and the strong driving ability. It can be applied to robot obstacle avoidance, robot tracing (detecting black line in white background and detecting white line in black background), which is the necessary sensor for tracing line robot and other occasions. The PCB size is 3.5cm*1cm, and the physical map is shown in Fig.3.2.3.1.。

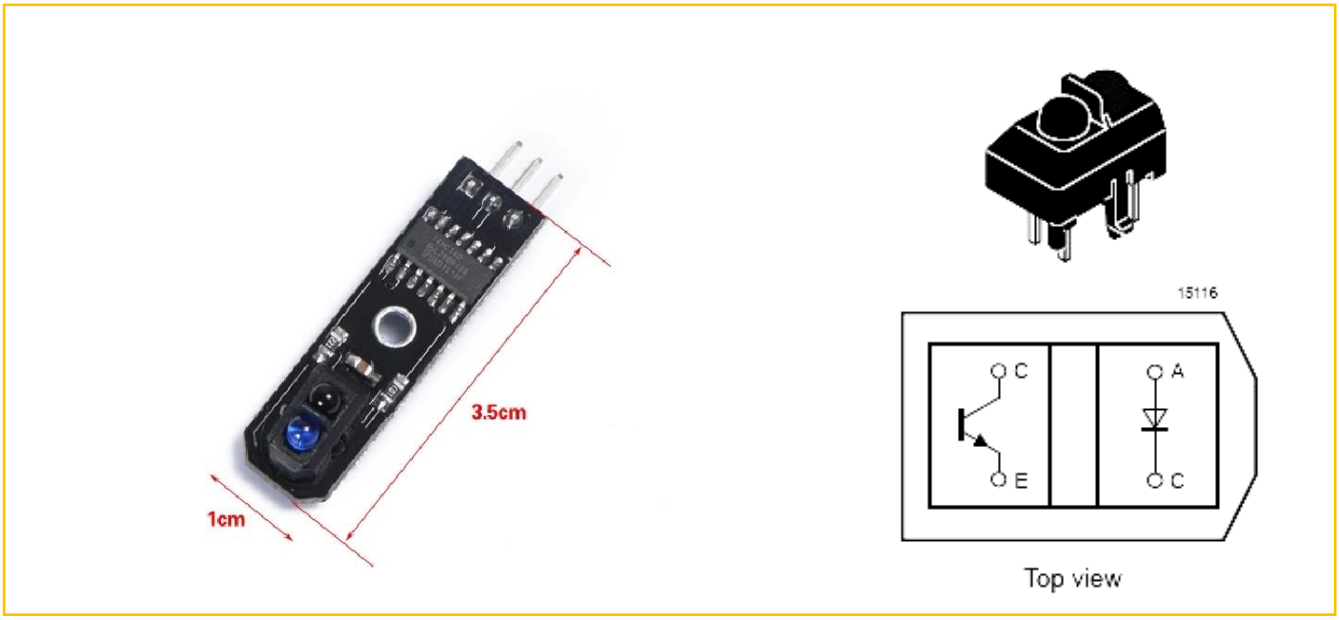


Figure .3.2.3.1 Physical Map of the Module

3.2.3.2 Working Principle

In the above, we talked about two patterns of tracing-the white line and the black line. In fact, either the black line or the white line, we usually adopt the infrared detection method.

Infrared detection method means that different objects with different colors have the different infrared reflection characteristics. The car launches the infrared to the ground continually during driving process, the infrared receiving tube will be in a shutdown state and the output of the module is low level when the emitted infrared is not reflected or the reflected infrared is not strong enough, and indicating diode will be off; when the diffuse reflection occurred on a white floor, the intense reflected infrared will be received by the receiving tube on the car, the photosensitive triode will be saturated, the output end of the module is high level and the indicating diode will light.

As is shown in the schematic diagram 3.2.3.2 (U1 is comparator, such as LM358, LM324, LM393, LM339 and a series of comparators, we use the 74HC14D comparator in TCRT5000), A and C are connected to the light emitting diode, C and E to the receiving diode, as shown in Fig.3.2.3.1 In the "Beetle-Bot" car, we use three modules, two in the left and right sides, one in the middle. Its installation is shown in Fig.3.2.3.3 ,the tracing sensors are in a straight line.

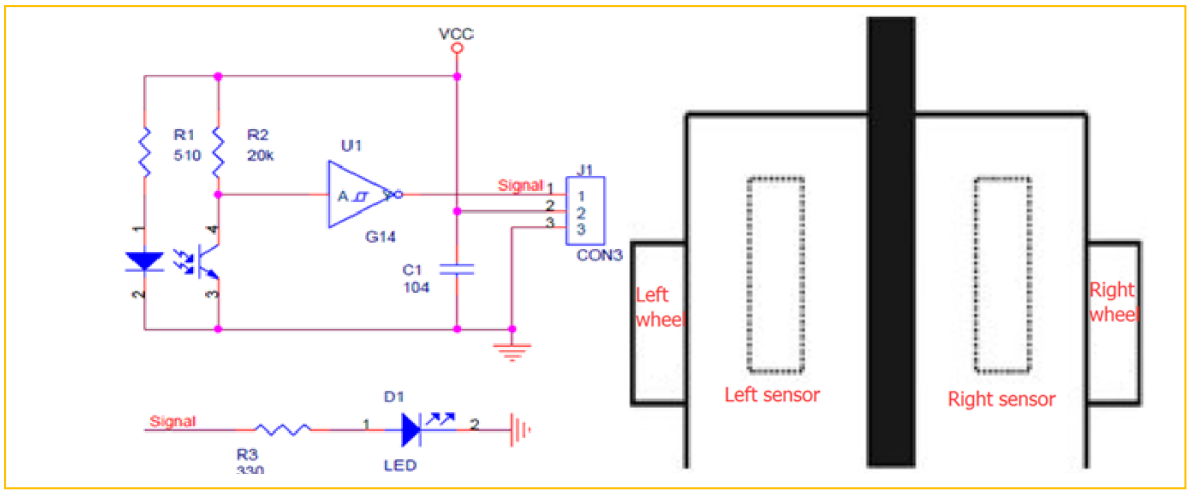


Figure 3.2.3.2 Schematic Diagram of Tracing Module Figure 3.2.3.3 Diagram of Tracing Module Installation

The X1 and Y1 are the first direction control sensors, and the width of the two sensors on the same side of the black line must not be greater than the width of the black line. When the car is moving forward, the driving track is always between the two first level sensors X1 and Y1 (the black track as shown in Fig. 3.2.18), when the car deviates from the black line:

If the left X1 detects the black line which can't be detected by the right Y1 and intermediate sensors, the the car has shifted to the right, then the car will turn left slightly, and keeping intermediate sensor always detecting the black line; if the right Y1 detects the black line which can't be detected by the left X1 and intermediate sensors, the the car has shifted to the left, then the car will turn right slightly, and keeping intermediate sensor always detecting the black line; if the car turns back on the track driving along the black line, X1 and Y1 can all detect the white line, and send high level to the microcontroller.

3.2.3.3 Module Parameters

- Using TCRT5000 infrared reflection sensor
- The detection distance: 1mm~25mm, the focal distance is 2.5mm
- The comparator output signal waveform is clean, good-shape and it has more than 15mA strong drive ability.
- The working voltage: 3.3V-5V
- Using wide voltage comparator 74HC14D, digital output (0 and 1)
- Easy-to-install fixed bolt holes

Detailed parameters please refer to "Beetle-bot\Document\ TCRT5000.pdf"

Note:

between positive and negative poles. Connecting the VCC to 3.3V or 5V, the OUT output port to the microcontroller IO port directly. The I/O port on Arduino should be set for input mode / receiving mode,

otherwise it can’t be used. As for other MCU, such as ARM or more advanced control boards, if the I/O ports need to be used as the input and output mode, they have to be set to the input mode / receiving mode. The 51 series microcontrollers can be used directly, there is no need to set the input and output mode.

3.2.3.5 Wire connection

As shown in the below figure, the upper column is the "GND" interface, the middle column is the "VCC" interface, and the lower column is the "OUT" interface, where "A0" corresponds to the "OUT" of the left trace, and "A1" corresponds to the "OUT" of the intermediate trace. "A2" corresponds to the "OUT" of the right trarce.

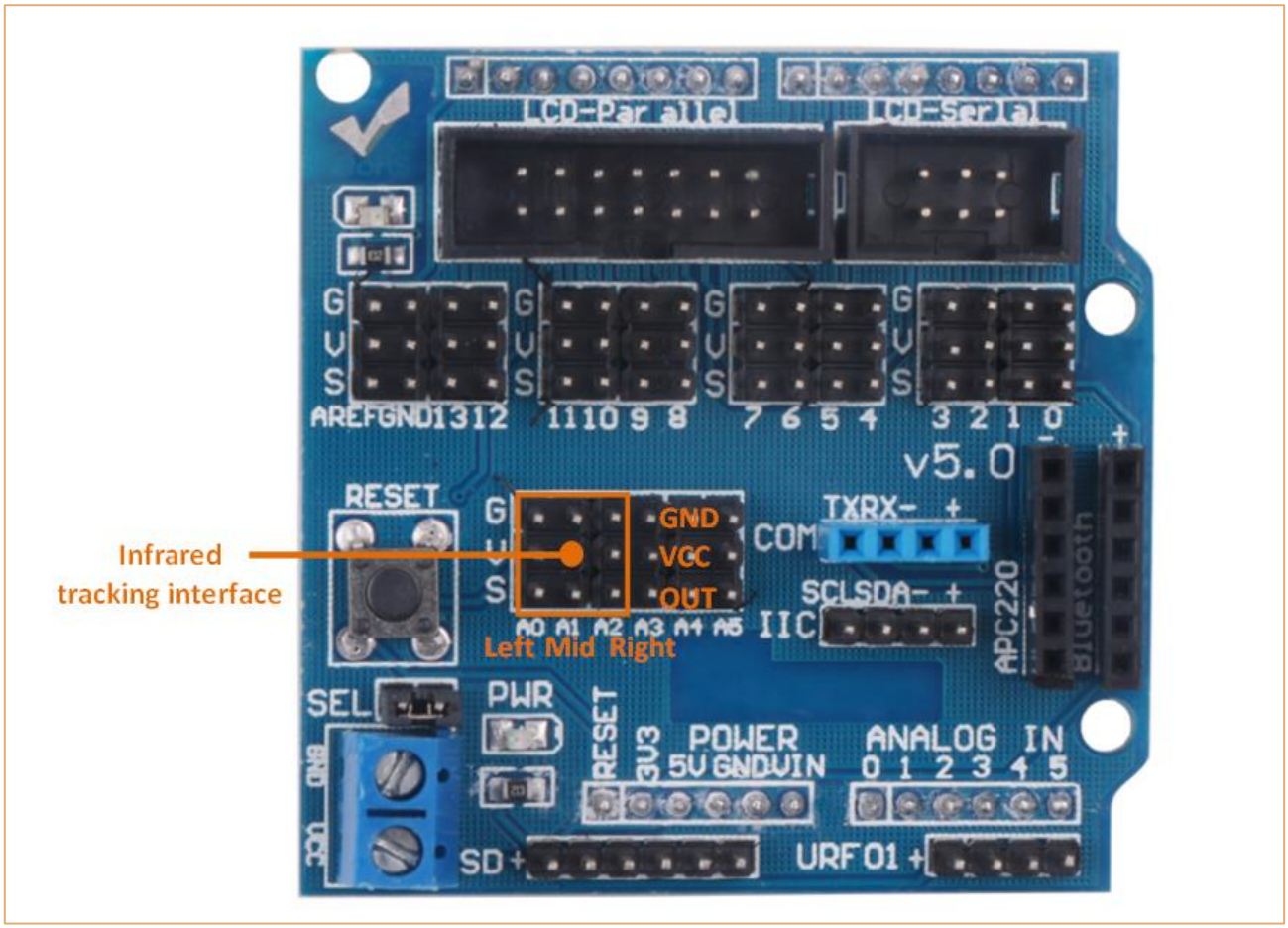


Figure .3.2.3.4Wire connection diagram

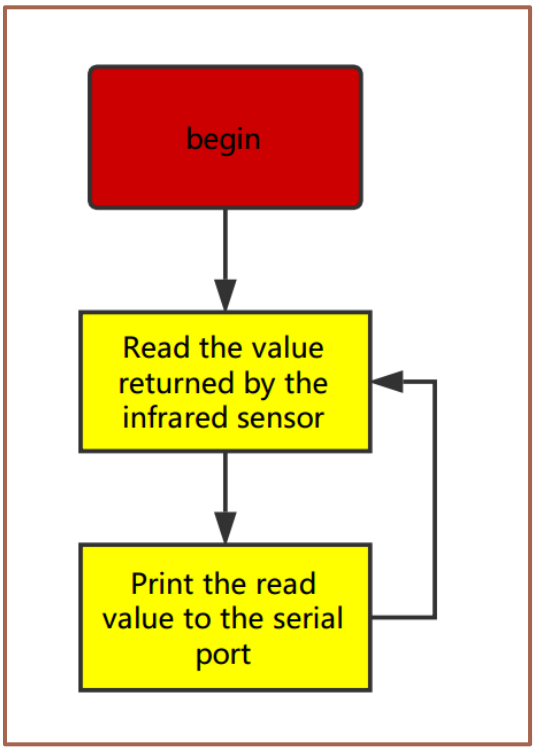
3.2.3.6 Experimental Procedures

1, Fixing the sensor on the car (the assembly is completed) and connecting it to the Arduino as shown in Fig.3.2.19.

2, Making the track.If your floor is white, then you could stick a black tape to form a loop, otherwise stick a white tape, the shape of track is based on your own desires, the best width of the tape is 13-18mm. In this manual, we use the black track, as shown in Fig.3.2.20.

3, Module test. Copying the following codes to the IDE compiler environment (you can also open the program in the CD directly) and downloading to the development board, opening the serial port monitor (baud rate is 9600) to observe the changes of data when there is the white line (Fig. 3.2.21) and is not the white line (Figure 3.2.22).

Program flow chart is as follows:



Note: Here we connect the signal output port of infrared obstacle avoidance to the analog port on Arduino (A0-A5), so the serial port monitor prints analog values, you can connect it to the digital port (2-13), then the serial port monitor will print out only "0" and "1".

Test demo code path : "Lesson\ModuleDemo\InfraredTracingTest\InfraredTracingTest.ino"

```
void setup() {
    Serial.begin(9600);
}
void loop() {
    int left, centre, right;
    left = analogRead(A0);
    centre = analogRead(A1);
    right = analogRead(A2);
    Serial.print("right:");
    Serial.print(right);
    Serial.print("   ");
    Serial.print("centre:");
    Serial.print(centre);
    Serial.print("   ");
    Serial.print("left:");
    Serial.print(left);
    Serial.println("   ");
}
```

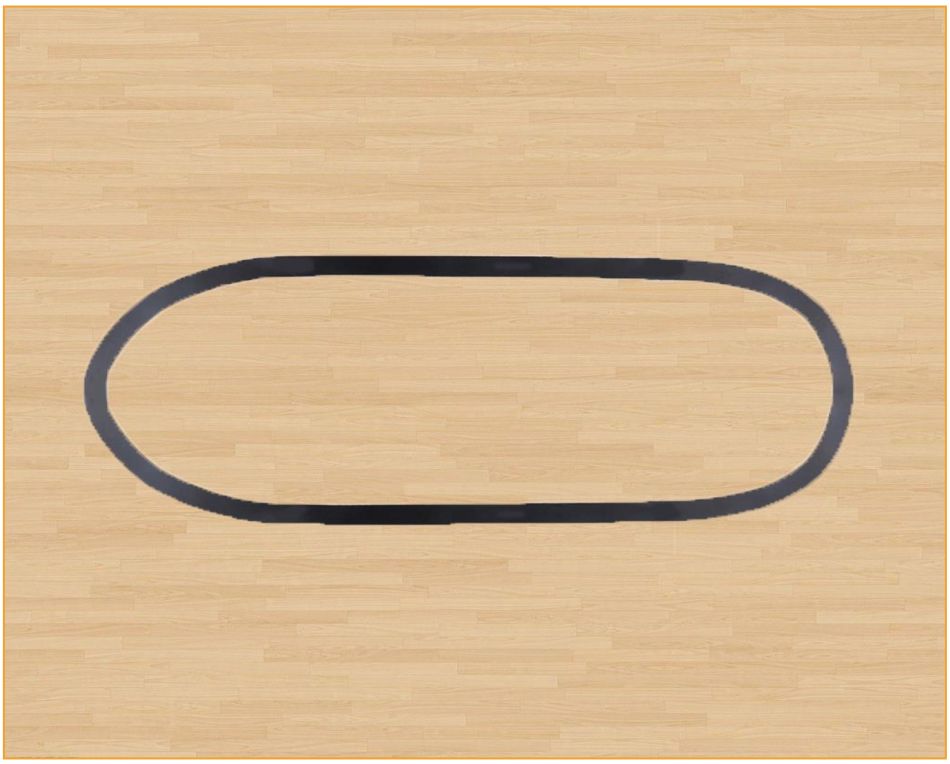


Figure 3.2.3.5 Example of the Black Track

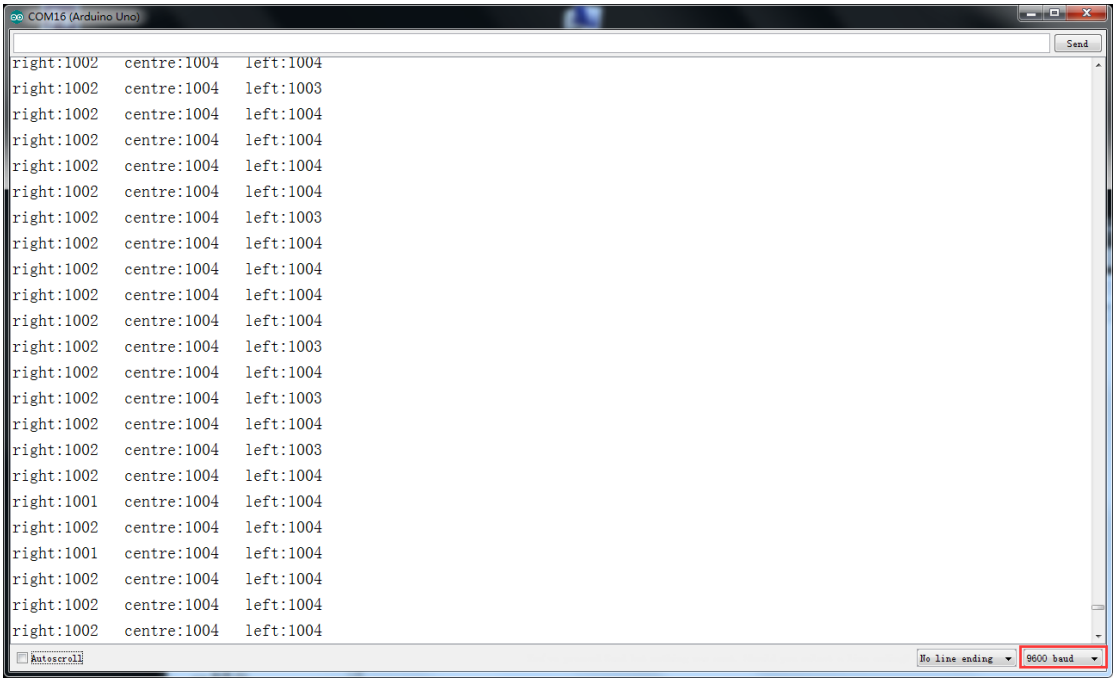


Fig 3.2.3.6 The Data When the Sensor Does Not Detect the Black Line

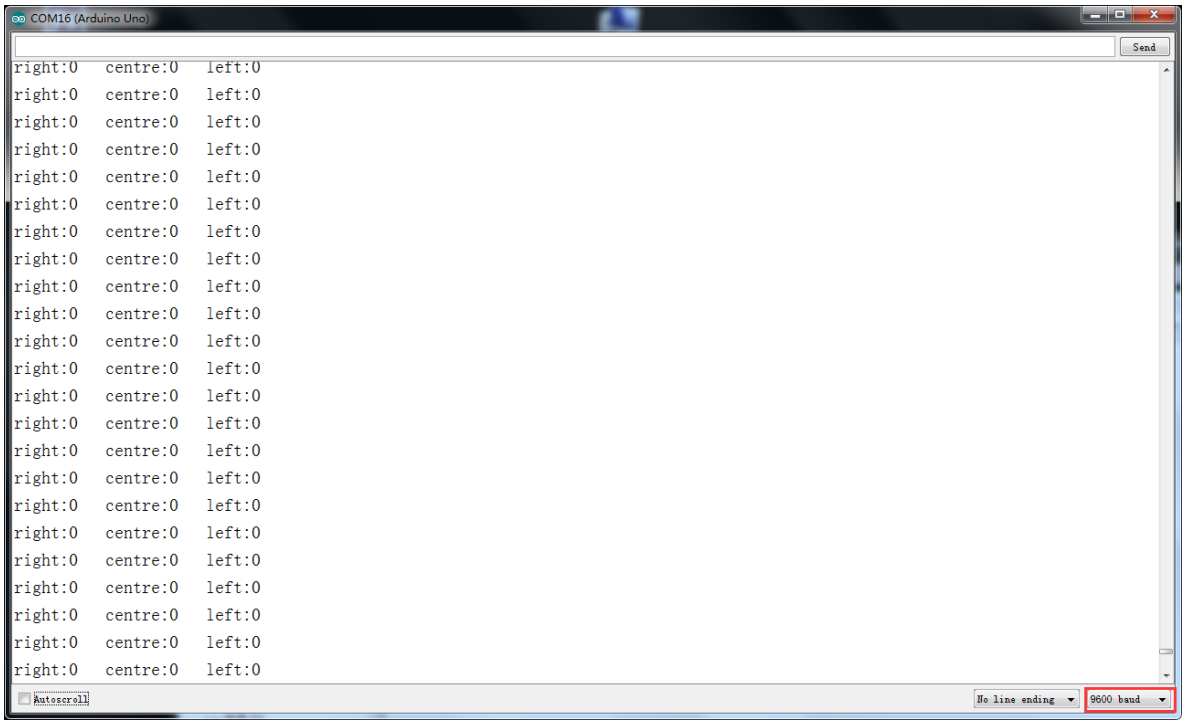


Fig 3.2.3.7 The Data When the Sensor Does Not Detect the Black Line

From Fig.3.2.3.6 and Fig.3.2.3.7 we can see that the output is high level when the sensor does not detect the black line, low level when detects the black line. We use the analog port to collect the sensor's signal, so the printed value is analog that, the high level is reaching 1024, the low level is 0. After we master the working principle of the sensor, our tracing car in this section comes to an end, then let us open a car journey.

3.2.3.5 Software Design

3.2.3.5.1. Program flow chart

When the car enters the tracing mode, it keeps scanning the I/O port of the MCU connected to the sensors, once detecting the changes in signal at the I/O port, the corresponding procedure will be implemented, the corresponding signal will be sent to the motor so as to correct the status of the car.

When the car enters the tracing mode, it keeps scanning the I/O port of the MCU

connected to the sensors, once detecting the changes in signal at the I/O port, the corresponding procedure will be implemented. If the left sensor detects the black line (the left half of the car walked across the black line, the car body is trended right), the car should turn the left; If the right sensor detects the black line (the right half of the car walked across the black line, the car body is trended left), the car should turn the right. After the direction adjustment, the car walks forward, and continues to detect the black line repeatedly. The tracing flow chart is shown in Figure 3.2.3.8.

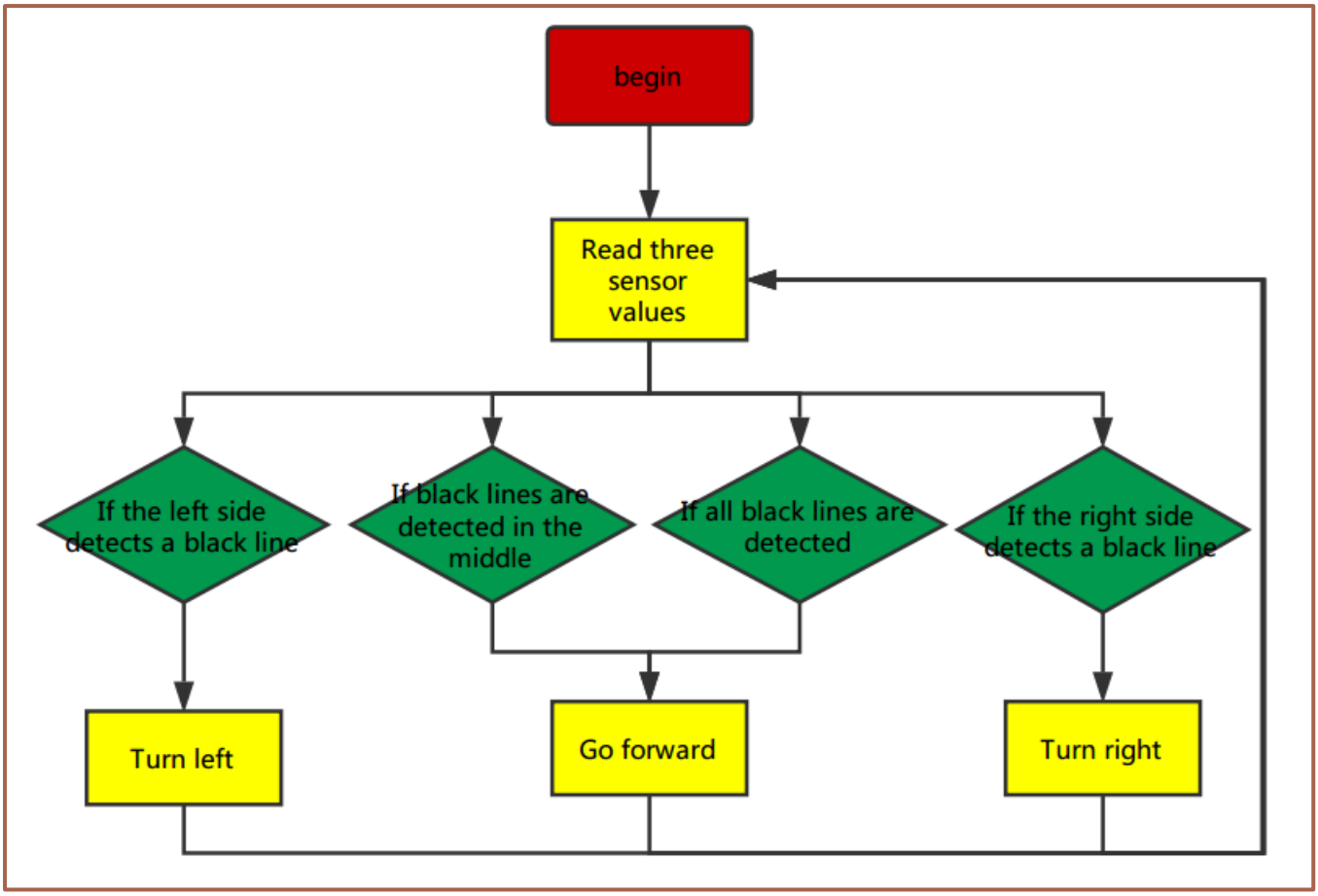


Figure 3.2.3.8 the Tracing Flow Chart

3.2.3.5.2. Program Code

Demo code path：Lesson\Advanced experiment\Beetle_InfraredTracing\Beetle_InfraredTracing.ino

```
int INPUT3_PIN = 5; //PWMA
int INPUT4_PIN = 9; //DIRA**************************************left
int INPUT1_PIN = 6; //PWMB
int INPUT2_PIN = 10; //DIRB**************************************right
/*Define 4 motor control terminals, connected to IN1-IN4 on the motor drive board.*/
void setup() {
  Serial.begin(9600); /*Set the baud rate to 9600 */
}
void loop()
{
  int left, centre, right; /*Define 3 sensors */
  left = analogRead(A0);
  centre = analogRead(A1);
  right = analogRead(A2); /*Read the value collected by 3 sensors */
  Serial.println(centre);
  if ((right >= 975) && (centre <= 8) && (left >= 975)) {
    /* Judge the collected value, if right > the sensor captures a value that is low and reads 0 after analog IO. However, to reduce the error, we set the threshold In 8, to prevent the error caused by the car to determine the wrong, we can
     customize the other values, if the use of digital port to receive the value of the sensor returns only "0" and "1", but to determine the same way. The reason why I did not use digital IO, because we use the digital IO port in other places. */
    int value = 150; /*Set a PWM value, the maximum value of PWM is 255, but the speed should not be too fast when tracing the car, otherwise the car will shake more in the tracing process.*/
    analogWrite (INPUT4_PIN, value);
    analogWrite(INPUT3_PIN, 0); //the speed value of motorA is value
    analogWrite (INPUT2_PIN, 0);
    analogWrite(INPUT1_PIN, value); //the speed value of motorB is value
  }
  else if ((right <= 8) && (centre >= 975) && (left >= 975)) { /* The value collected to judge, if the center > = 975 and left > = 975 are greater than 975, indicating that the middle and left sensors did not detect the black line, right <= 8 shows the right sensor detects a black line, then the car Has left to the left, or the black line has been turning to the right, so the car should turn to the right. */
    analogWrite (INPUT3_PIN, 0);
    analogWrite(INPUT4_PIN, 0); //the speed value of motorA is value
    analogWrite (INPUT2_PIN, 0);
    analogWrite(INPUT1_PIN, 180); //the speed value of motorB is value
  }
  else if ((right >= 975) && (centre >= 975) && (left <= 8)) { /* Judge the collected value,
```

```
if center > = 975 and right > = 975 are greater than 975, indicating that the middle and right sensors did not detect the black line, left <= 8 shows that the left sensor detects the black line, then the car Has been to the right deviation, or the black line has turned to the left, so the car should turn left at this time. */
    int value = 130;
    analogWrite (INPUT4_PIN, value);
    analogWrite(INPUT3_PIN, 0); //the speed value of motorA is value
    analogWrite (INPUT1_PIN, 0);
    analogWrite(INPUT2_PIN, 0); //the speed value of motorB is value
  }
  if ((right <= 8) && (centre <= 8) && (left <= 8)) {/* The value collected to judge, if the center <= 8, left <= 8 and right <= 8 are greater than 8, indicating 3 sensors have detected a black line, then the car has reached the "ten" intersection, because We have only 3 sensors, no way to make more sophisticated judgments, so only let the car choose to go straight. */
    int value = 130;
    analogWrite (INPUT4_PIN, value);
    analogWrite(INPUT3_PIN, 0); //the speed value of motorA is value
    analogWrite (INPUT2_PIN, 0);
    analogWrite(INPUT1_PIN, value); //the speed value of motorB is value
  }
}
```

3.2.4 Ultrasonic Obstacle Avoidance

In this product, we will integrate the ultrasonic module and steering engine together and make the two parts working at the same time, which greatly increases the effectiveness of the data and the flexibility of the car, the main working flow: When the power is on, steering engine will automatically rotates to 90 degrees, the MCU will read data from the reflected ultrasonic. If the data is greater than the security value, the car will continue to drive forward, otherwise the car will stop, then the steering engine will rotate 90 degrees to the right. After that, the MCU reads data from the reflected ultrasonic again, the steering engine rotates 180 degrees to the left, then reading data again, the steering engine rotate 90 degrees, the MCU will contrast the two detected data, if the left data is greater than the right data, the car will turn left, otherwise turn right, if the two data are both less than the safety value, the car will turn around.

3.2.4.1 Suite Introduction

1.The steering gear

The steering gear is also called servo motor which is originally used in ships, since it can control the angle continuously through the program, so it has been widely used in intelligent steering robot to achieve all kinds of joint movement, the characteristics steering gear are small volume, large torque, high stability, simple external mechanical design. Either in hardware or software design, the design of steering engine is an important part of car controlling, the steering gear is mainly composed of the following parts in general, steering wheels, gear group, position feedback potentiometer, DC motor and control circuit (shown in Fig 3.2.4.1). Fig 3.2.4.2 shows the most commonly used 9G steering gear now.

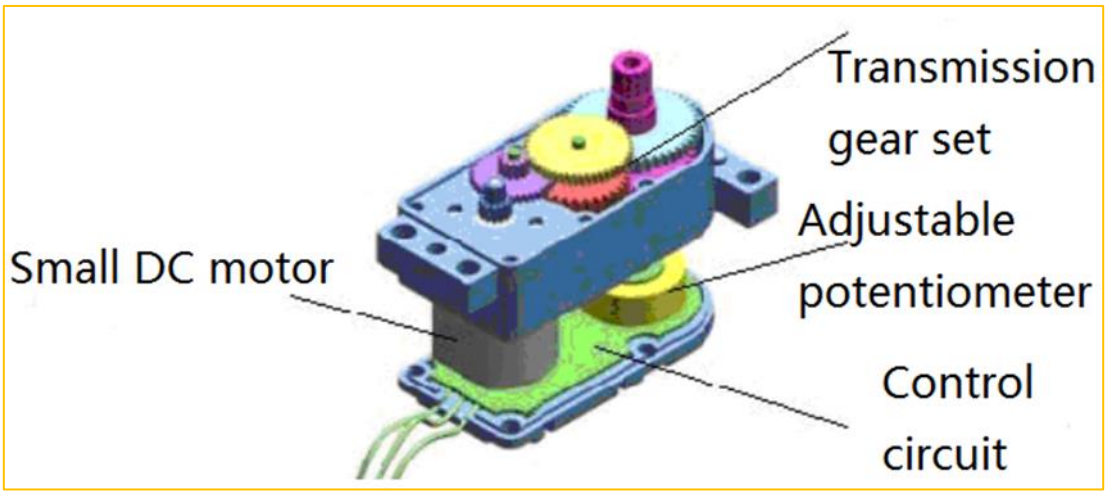


Figure 3.2.4.1 Diagram of Steering Gear

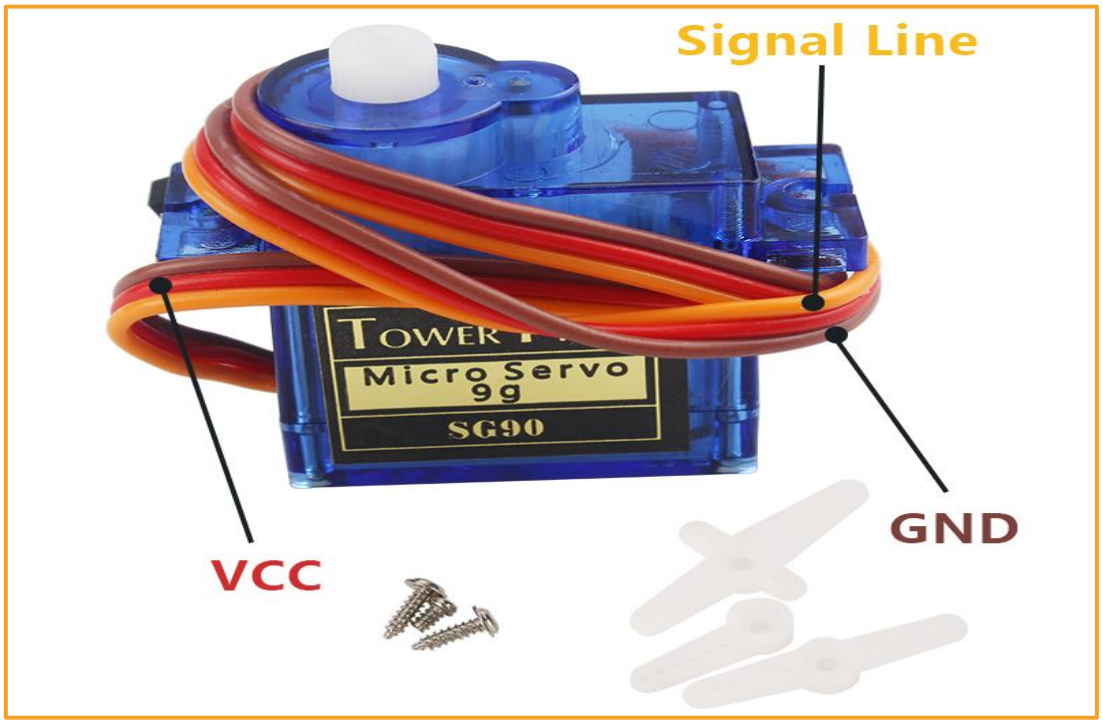


Figure 3.2.4.2 Physical Map of Steering Gear

2、The ultrasonic

An ultrasonic sensor is a device that transforms other forms of energies into ultrasonic energy with desired frequency or transforms the ultrasonic energy into other forms of energy with the same frequency. The ultrasonic sensors are commonly classified into two categories, the acoustic type and the hydrodynamic type. Acoustic type mainly has: 1, piezoelectric sensor; 2, magneto strictive sensor sensor; 3, electrostatic sensor. The hydrodynamic type includes the gas whistle and the liquid whistle. At present most of the ultrasonic sensors are working using piezoelectric sensors. Distance measurement with the ultrasonic is also a hot spot.

In the "Beetle-Bot" car, we use HC-SR04 ultrasonic module which has the 2cm-400cm non-contact distance sensing function, the measurement accuracy can achieve to 3mm; the temperature sensor can correct the measured results using the GPIO communication mode, the module has a stable and reliable watchdog. The module includes an ultrasonic transmitter, receiver and control circuit, which can measure distance and steer like in some projects. The smart car can detect obstacles in front of itself, so that the smart car can change direction in time, avoid obstacles. A common ultrasonic sensor is shown in Fig.3.2.4.3.

Figure 3.2.4.3 Physical Map of Ultrasonic Module

3.2.4.2 Suite Parameters

1. Steering gear

The steering gear has three input wires as shown in Fig.3.2.25, the red is power wire, while the brown is the ground, which guarantee the basic energy supply for the steering gear. The power supply has two kinds of specifications (one is 4.8V, the other is 6.0V)which are corresponding to different torque standards, the 6.0V torque is higher than the 4.8V torque; and the another one is the signal control wire, which is generally white in Futaba, orange in JR. Noticing that some of the SANWA's power wires are on the edge rather than the middle which need to be identified, so you need to remember that the red is power wire, the brown is ground wire.

2. Ultrasonic wave

1, Working voltage: 4.5V~5.5V. In particular, voltage above 5.5V is not allowed definitely

2, Power consumption current: the minimum is 1mA, the maximum is 20mA

3, Resonant frequency: 40KHz;

4, Detection range: 4 mm to 4 meters. Error: 4%. In particular, the nearest distance is 4mm, the longest distance is 4 meters, and the data outputs continuously without setting anything. 5, Temperature measurement range: 0℃to +100℃; precision: 1℃

6, Illumination measurement range: bright and dark;

7, Data output mode: icc and uart (57600bps), users can choose any of them; UART mode uses 7 bytes as a group, and the 3 data stared with 0x55 are the distance values; the 2 data started with 0x66 are the

temperature value; the 2 data started with 0x77 are the illumination values. 0x55\0x66\0x77 are the data headers in order to distinguish the 3 data;

8, Supporting the following 2 detection methods: 1, continuous detection; 2, controlled intermittent detection;

9, Distance data format: using mm as the smallest data unit, double byte 16 hexadecimal transmission;

10, Temperature data format: using Celsius degree as the smallest unit, single byte hexadecimal transmission;

11, Light data format: single byte 16 hexadecimal transmission; the value is big when it is dark, small when it is bright;

12, Working temperature: 0℃~+100℃

13, storage temperature: -40 to +120 degrees Celsius

14, Size: 48mm*39mm*22mm (H)

15, The size of fixing holes: 3*Φ3mm; Gap:10mm

3.2.4.3 Working Principle

1. Steering gear

The control signal enters the signal modulation chip by the receiver channel, gets the DC bias voltage. The steering gear has a reference circuit which generates a reference signal with a period of 20ms and a width of 1.5ms. Comparing the obtained DC bias voltage with the voltage of the potentiometer and obtaining the output voltage difference. Finally, the positive and negative output voltage difference in the motor driver chip decide the positive and negative rotation of motor. When the speed of motor is certain, the cascade reducer gear will drive potentiometer to rotate so that the voltage difference is reducing to 0, the motor will stop rotating.

When the control circuit receives the control signal, the motor will rotate and drive a series of gear sets, the signal will move to the output steering wheel when the motor decelerates. The the output shaft of steering gear is connected with the position feedback potentiometer, the potentiometer will output a voltage signal to the control circuit board to feedback when the steering gear rotates, then the control circuit board decides the rotation direction and speed of the motor according to the position, so as to achieve the goal. The working process is as follows: control signal→control circuit board→motor rotation→gear sets deceleration→steering wheel rotation→position feedback potentiometer→control circuit board feedback.

The control signal is 20MS pulse width modulation (PWM), in which the pulse width varies linearly from 0.5-2.5MS, the corresponding steering wheel position varies from 0-180 degrees, which means the output

shaft will maintain certain corresponding degrees if providing the steering gear with certain pulse width. No matter how the external torque changes, it only changes position until a signal with different is provided as shown in Fig.3.2.4.4. The steering gear has an internal reference circuit which can produce reference signal with 20MS period and 1.5MS width, there is a comparator which can detect the magnitude and direction of the external signal and the reference signal, thereby produce the motor rotation signal.

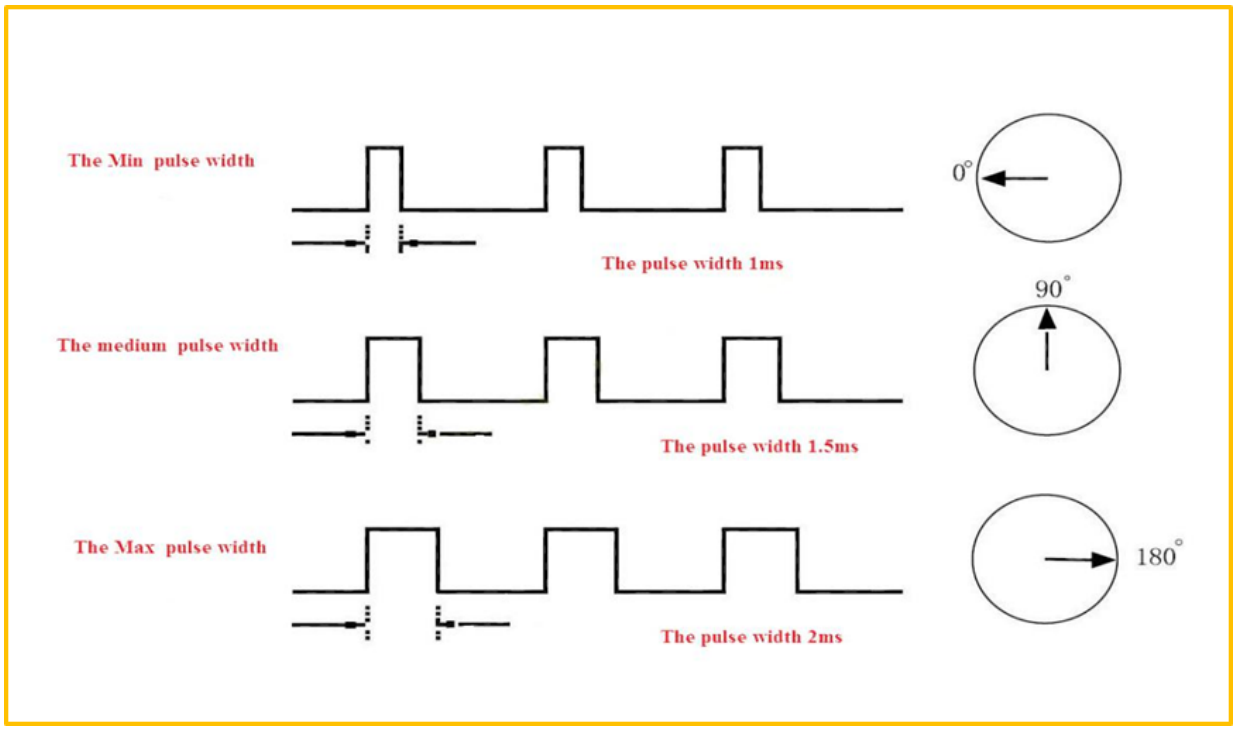


Figure 3.2.4.4 Relationship between the Motor Output Angle and Input Pulse

2、The ultrasonic

The most commonly used method of ultrasonic distance measurement is echo detection method, the ultrasonic transmitter launches ultrasonic toward a direction and starting the time counter at the same time, the ultrasonic will reflect back immediately when encountering a blocking obstacle and stopping the counter immediately as soon as the reflected ultrasonic is received by the receiver. The working sequence diagram is shown in Fig 3.2.4.5. The velocity of the ultrasonic in the air is 340m/s, we can calculate the distance between the transmitting position and the blocking obstacle according to the time t recorded by the time counters, that is: s=340*t/2.

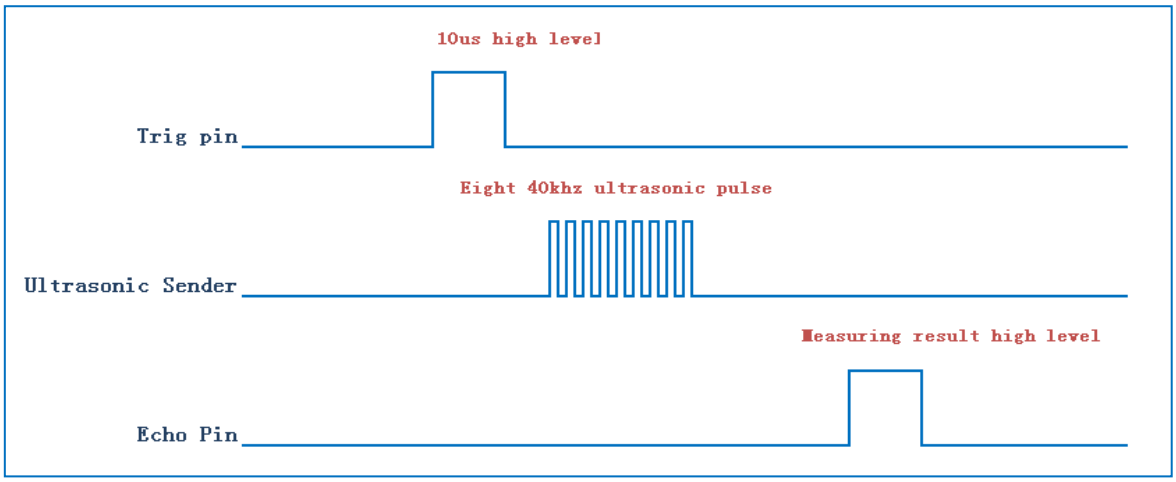


Figure 3.2.4.5 the Ultrasonic Working Sequence

Let us analyze the working sequence, first the trigger signal starts the HC-RS04 distance measurement module, which means the MCU sends an at least 10us high level to trigger the HC-RS04,the signal sent inside of the module is responded automatically by the module, so we do not have to manage it, the output signal is what we need to pay attention to. The output high level of the signal is the transmitting and receiving time interval of the ultrasonic, which can be recorded with the time counter, and don't forget to divided it with 2.

The ultrasonic is a sound wave which will be influenced by temperature. If the temperature changes little, it can be approximately considered that the ultrasonic velocity is almost unchanged in the transmission process. If the required accuracy of measurement is very high, the measurement results should be to corrected with the temperature compensation. Once the velocity is determined, the distance can be obtained. This is the basic principle of ultrasonic distance measurement module which is shown in Fig3.2.4.6：

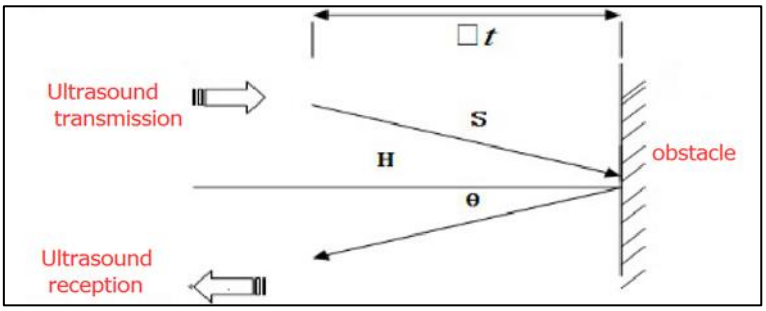


Figure 3.2.4.6 the Principle of Ultrasonic Distance Measurement Module

The ultrasonic is mainly divided into two parts, one is the transmitting circuit and the other is the receiving circuit, as shown in Fig3.2.4.7. The transmitting circuit is mainly composed of by the inverter 74LS04 and ultrasonic transducer T40, the first 40kHz square wave from the Arduino port is transmitted

through the reverser to the one electrode on the ultrasonic transducer, the second wave is transmitted to another electrode on ultrasonic transducer, this will enhance the ultrasonic emission intensity. The output end adopts two parallel inverters in order to improve the driving ability. the resistance R1 and R2 on the one hand can improve the drive ability of the 74LS04 outputting high level, on the other hand, it can increase the damping effect of the ultrasonic transducer and shorten the free oscillation time.

The receiving circuit is composed of the ultrasonic sensor, two-stage amplifier circuit and a PLL circuit. The reflected signal received by the ultrasonic sensor is very weak, which can be and amplified by the two-stage amplifier. PLL circuit will send the interrupt request to the microcontroller when receiving the signal with required frequency. The center frequency of internal VCO in the PLL LM567 is , the locking bandwidth is associated with C3. Because the transmitted ultrasonic frequency is 40kHz, the center frequency of the PLL is 40kHz, which only respond to the frequency of the signal, so that the interference of other frequency signals can be avoided.

The ultrasonic sensor will send the received the signal to the two-stage amplifier, the amplified signal will be sent into the PLL for demodulation, if the frequency is 40kHz, then the 8 pins will send low level interrupt request signal to the microcontroller P3.3, the Arduino will stop the time counter when detecting low level.

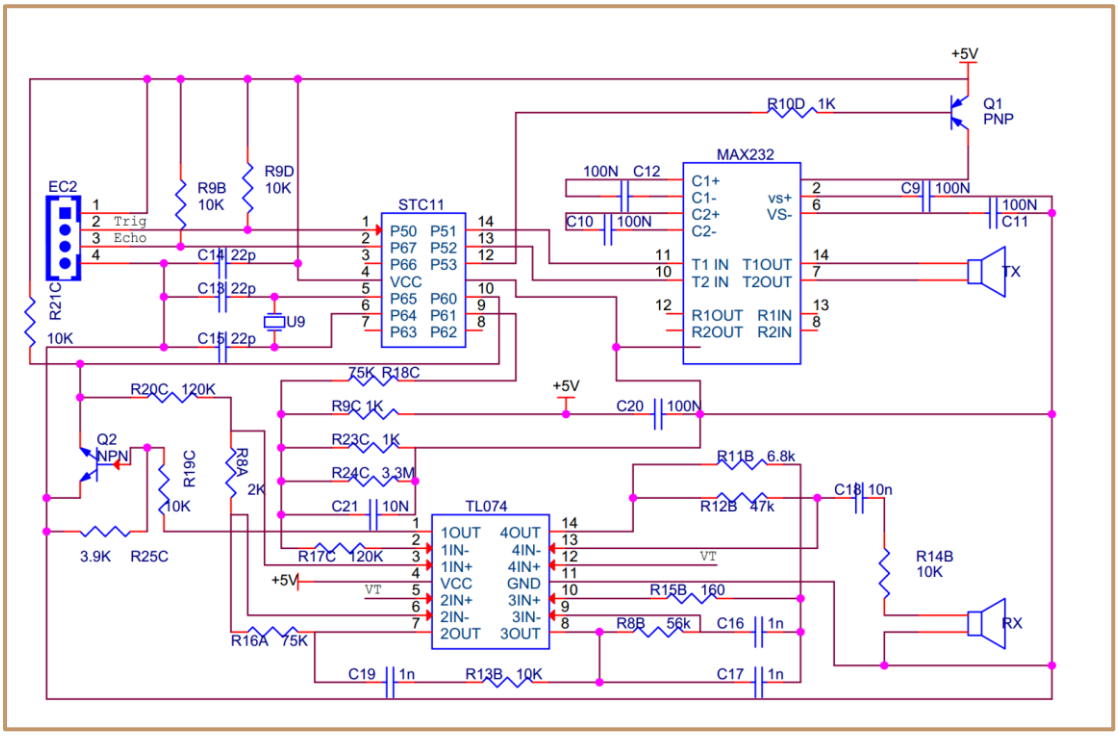


Figure 3.2.4.7 Schematic Diagram of Ultrasonic Transmitting and Receiving

3.2.4.4 Experimental Procedures

Connecting the steering gear and ultrasonic module to the Arduino motherboard as shown in Fig 3.2.4.8(you can choose other IO ports according to your own ideas).

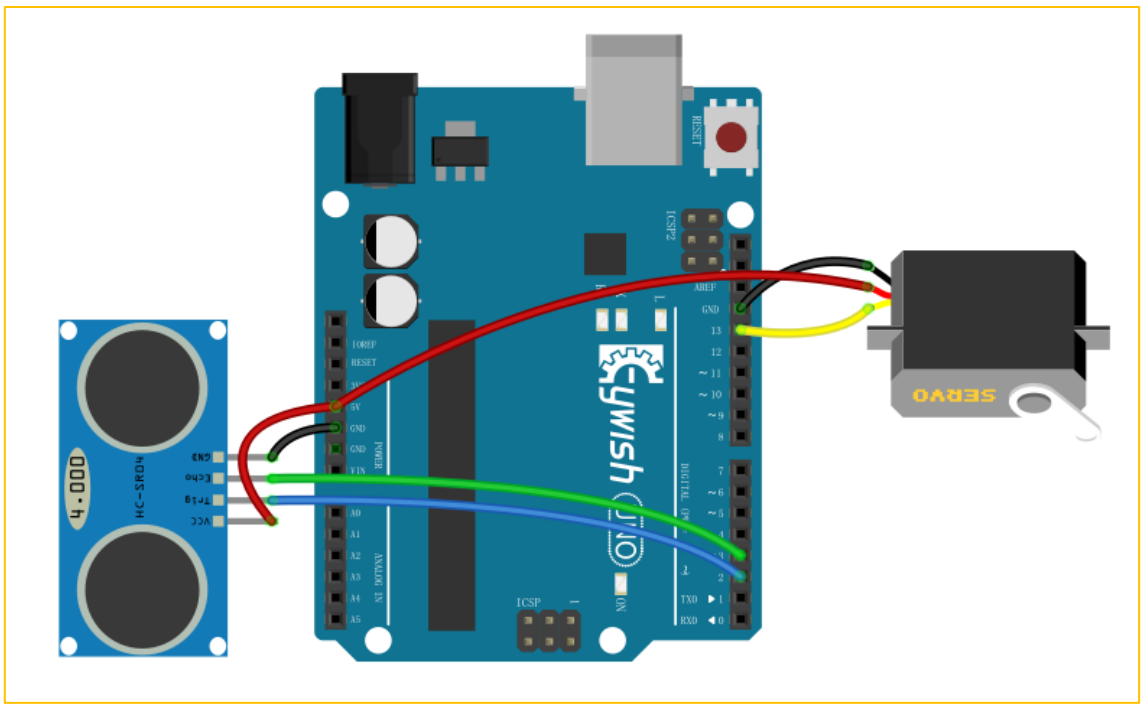


Figure 3.2.4.8 Wiring of the Steering Gear and Ultrasonic Module

3.2.4.5 Wire connection

As shown in the below figure, the servo's "S" is connected to pin 13, the ultrasonic "Trlg" is connected to pin 2, the "Echo" is connected to pin 3, the "VCC" pin is connected to VCC, and the "GND" pin is connected to GND.

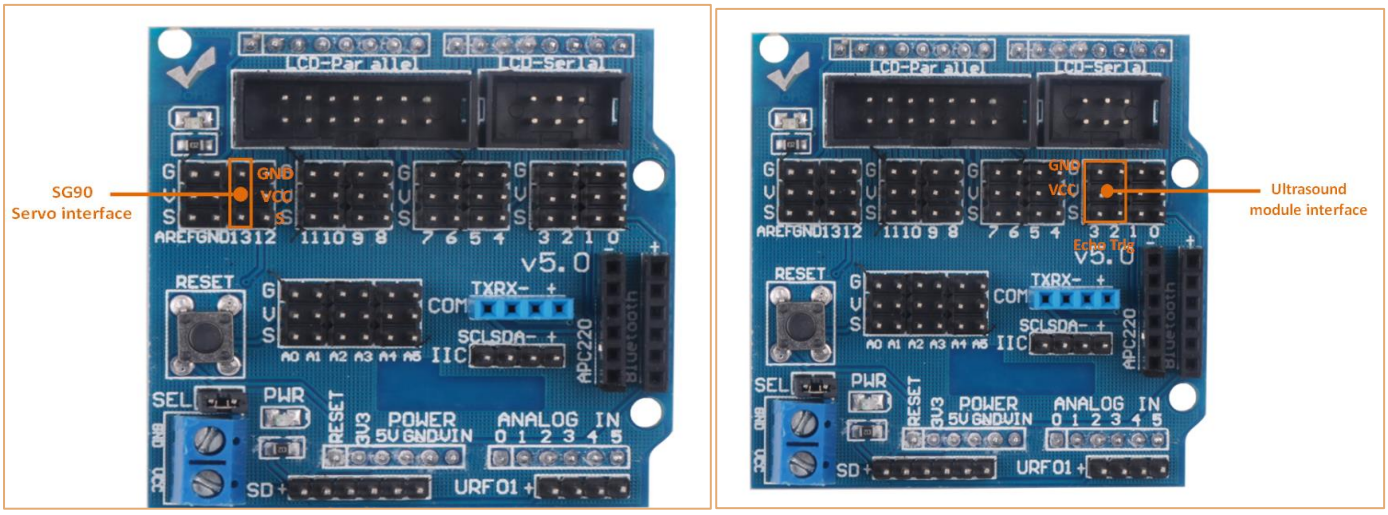
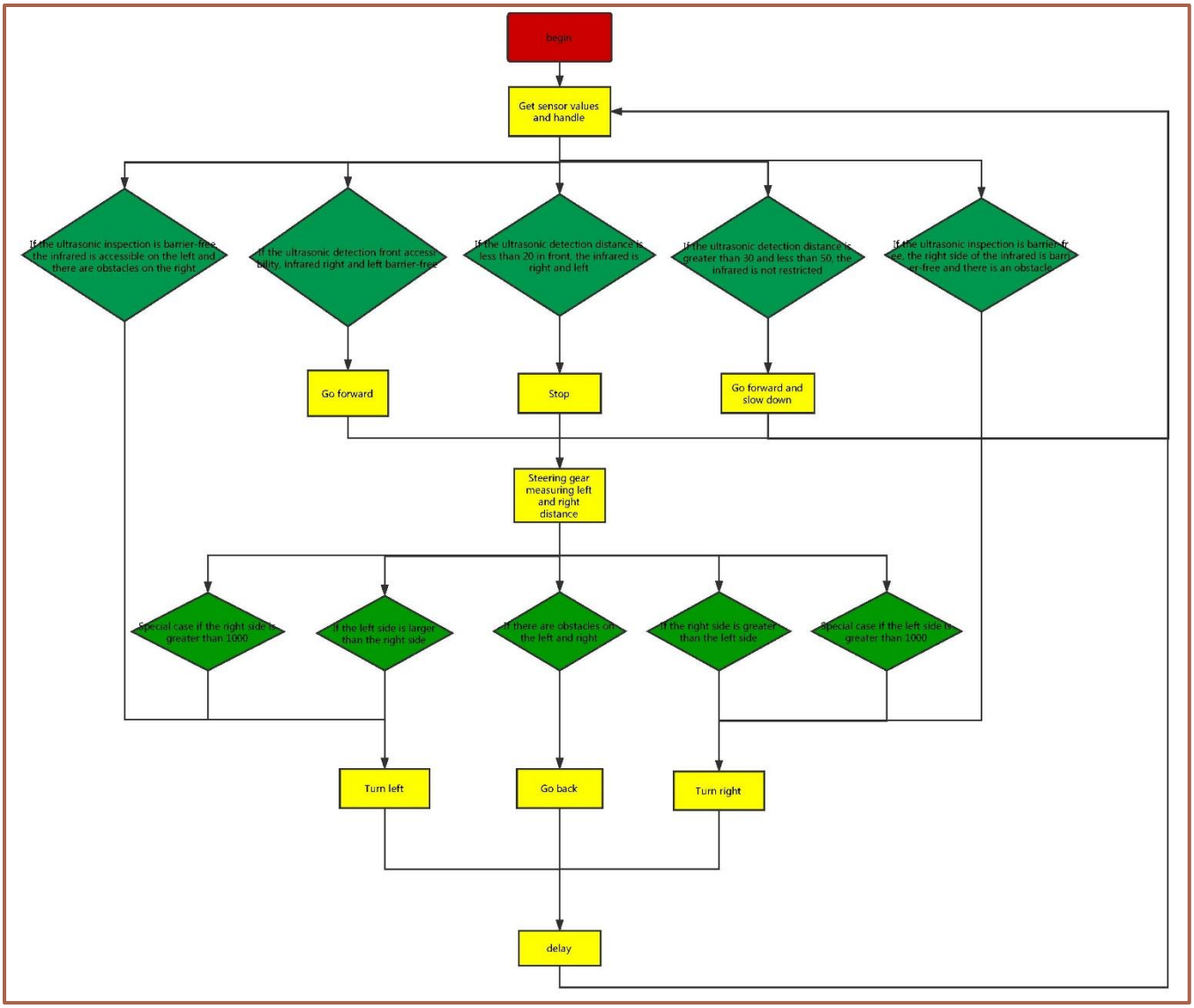


Figure 3.2.4.9 Wiring diagram of steering gear, ultrasonic wave and expansion board

3.2.4.6 Software design

3.2.4.6.1 Program flow chart



3.2.4.6.2 Program Code

Demo code path：Lesson\Advanced experiment\Beetle_Ultrasound\Beetle_Ultrasound.ino

```
#include "Ultrasonic.h"
#define INPUT2_PIN 10 // PWMB
#define INPUT1_PIN 6  // DIRB  ---  right
#define INPUT4_PIN 9  // PWMA
#define INPUT3_PIN 5  // DIRA  ---  left
#define SERVO_PIN 13
#define ECHO_PIN 3
#define TRIG_PIN 2
#define INFRARED_AVOIDANCE_LEFT_PIN A3
```

```
#define INFRARED_AVOIDANCE_RIGHT_PIN A4
#define IA_THRESHOLD 40
#define UL_LIMIT_MIN 50
#define UL_LIMIT_MID 40
#define UL_LIMIT_MAX 2000
Ultrasonic Ultrasonic(TRIG_PIN, ECHO_PIN, SERVO_PIN);/*Define ultrasonic and servo pins*/

void setup()
{
  Serial.begin(9600);
  pinMode(INFRARED_AVOIDANCE_LEFT_PIN, INPUT);
  pinMode(INFRARED_AVOIDANCE_RIGHT_PIN, INPUT);
  pinMode(INPUT1_PIN, OUTPUT);
  pinMode(INPUT2_PIN, OUTPUT);
  pinMode(INPUT3_PIN, OUTPUT);
  pinMode(INPUT4_PIN, OUTPUT);
  Ultrasonic.SetServoBaseDegree(90);/*Adjust the initial angle of the steering gear
according to the steering gear error*/
  Ultrasonic.SetServoDegree(90);/*Set the servo angle*/
}

void loop()
{
  uint16_t RightValue, LeftValue;
  uint16_t UlFrontDistance, UlLeftDistance, UlRightDistance;
  RightValue = analogRead(INFRARED_AVOIDANCE_RIGHT_PIN);/*The infrared obstacle avoidance
module collects the left data*/
  LeftValue = analogRead(INFRARED_AVOIDANCE_LEFT_PIN);/*The infrared obstacle avoidance
module collects the right data*/
  UlFrontDistance = Ultrasonic.GetUltrasonicFrontDistance();/*The ultrasonic module
collects the front data*/
  if (((RightValue > IA_THRESHOLD) && (LeftValue > IA_THRESHOLD)) && ((UlFrontDistance >
UL_LIMIT_MID) && (UlFrontDistance < UL_LIMIT_MAX)))
  /*According to the data collected by the ultrasonic module and the infrared obstacle
avoidance module,
    it is judged whether there is an obstacle in front, and if there is no obstacle, go
straight.*/
  {
    analogWrite(INPUT2_PIN, 0);
    analogWrite(INPUT1_PIN, 200);
    analogWrite(INPUT3_PIN, 0);
    analogWrite(INPUT4_PIN, 200);
```

```
  }
  else if ((RightValue > IA_THRESHOLD) && (LeftValue < IA_THRESHOLD))
  /*The data collected by the infrared obstacle avoidance module determines whether there is an obstacle on the right side. If not, turn right.*/
  {
    analogWrite(INPUT2_PIN, 0);
    analogWrite(INPUT1_PIN, 150);
    analogWrite(INPUT3_PIN, 0);
    analogWrite(INPUT4_PIN, 0);
    delay(200);
  }
  else if ((RightValue < IA_THRESHOLD) && (LeftValue > IA_THRESHOLD))
  /*The data collected by the infrared obstacle avoidance module determines whether there is an obstacle on the left side. If not, turn left.*/
  {
    analogWrite(INPUT2_PIN, 0);
    analogWrite (INPUT1_PIN, 0);
    analogWrite(INPUT3_PIN, 0);
    analogWrite(INPUT4_PIN, 150);
    delay(200);
  }
  else if ((RightValue < IA_THRESHOLD) && (LeftValue < IA_THRESHOLD))
  /*The data collected by the infrared obstacle avoidance module determines whether there are obstacles on the left and right sides, if any, rotates 180 degrees.*/
  {
    analogWrite(INPUT2_PIN, 0);
    analogWrite(INPUT1_PIN, 150);
    analogWrite(INPUT3_PIN, 150);
    analogWrite(INPUT4_PIN, 0);
    delay(150);
  }
  else if (((RightValue > IA_THRESHOLD) && (LeftValue > IA_THRESHOLD)) && ((UlFrontDistance < UL_LIMIT_MID) || (UlFrontDistance > UL_LIMIT_MAX)))
  /*According to the data collected by the ultrasonic module and the infrared obstacle avoidance module, it is determined whether there is an obstacle in front. For example,
   * the infrared obstacle avoidance module determines that there is no obstacle in front, and the ultrasonic module determines that the right obstacle is an obstacle,
   * first stops the car, and uses the ultrasonic module to perform left and right.*/
  {
    analogWrite(INPUT2_PIN, 0);
    analogWrite(INPUT1_PIN, 0);
    analogWrite(INPUT3_PIN, 0);
```

```
    analogWrite(INPUT4_PIN, 0);
    Ultrasonic.SetServoDegree(0);/*Servo rotation to 0 degrees*/
    UlRightDistance = Ultrasonic.GetUltrasonicRightDistance();/*The ultrasonic module collects the right side*/
    Ultrasonic.SetServoDegree(180);/*Servo rotation to 180 degrees*/
    UlLeftDistance = Ultrasonic.GetUltrasonicLeftDistance();/*The ultrasonic module collects the left side*/
    if ((UlRightDistance > UL_LIMIT_MIN) && (UlRightDistance < UL_LIMIT_MAX) && (UlRightDistance > UlLeftDistance))
    /*According to the ultrasonic module to collect the data on the left and right sides to determine whether there is an obstacle on the right side.*/
    {
      analogWrite(INPUT2_PIN, 0);
      analogWrite(INPUT1_PIN, 150);
      analogWrite(INPUT3_PIN, 0);
      analogWrite(INPUT4_PIN, 0);
      delay(200);
    }
    else if ((UlLeftDistance > UL_LIMIT_MIN) && (UlLeftDistance < UL_LIMIT_MAX) && (UlLeftDistance > UlRightDistance))
   /*According to the ultrasonic module to collect the data on the left and right sides to determine whether there is an obstacle on the left side.*/
    {
      analogWrite(INPUT2_PIN, 0);
      analogWrite (INPUT1_PIN, 0);
      analogWrite(INPUT3_PIN, 0);
      analogWrite(INPUT4_PIN, 150);
      delay(200);
    }
    else if ((UlRightDistance < UL_LIMIT_MIN) && (UlLeftDistance < UL_LIMIT_MIN) )
    /*According to the ultrasonic module to collect the data on the left and right sides to determine whether there are obstacles on the left and right sides*/
    {
      analogWrite(INPUT2_PIN, 0);
      analogWrite(INPUT1_PIN, 150);
      analogWrite(INPUT3_PIN, 150);
      analogWrite(INPUT4_PIN, 0);
      delay(300);
    }
  }
}
```

In front the sections, we focus on the "automatic driving", and they are obstacle avoidance experiments. We didn't seem to have relationship with the car, it is lack of fun. Now in the next few sections, we will develop the car from other aspects to make sure that we are able to control the car personally, then we will start from the "infrared remote control", followed by "2.4G handle remote control" and the last is "mobile phone Bluetooth control".

3.2.5 Infrared Remote Control

3.2.5.1 Suite Introduction

Infrared remote control is widely used in every field which is known to everyone, since it can control other electrical appliances, naturally it can control the Beetle-Bot car. Let us take a look at the infrared remote control first:

Infrared wireless remote control kit consists of Mini ultra-thin infrared remote controller and 38KHz infrared receiver module, the remote controller has 17 function keys, the transmission distance is up to 8 meters which is very suitable for controlling equipment indoor. The infrared receiving module can receive 38KHz modulation remote control signal. Through the Arduino programming, the decoding operation of Infrared wireless remote control signal can be realized so as to produce all kinds of remote control robot and interactive works. The suite is shown in Fig.3.2.5.1.

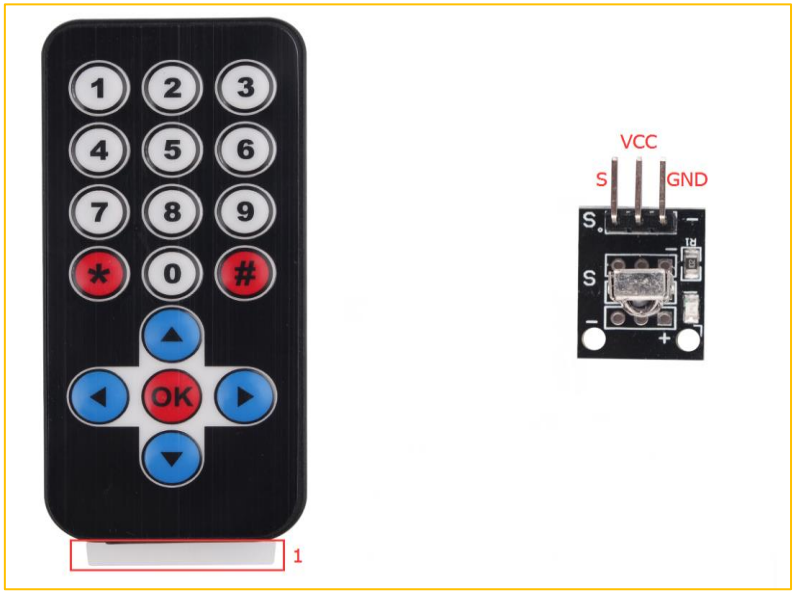


Figure 3.2.5.1 Physical Map of Remote Control Suite

Infrared remote control system is mainly divided into modulation, transmitting and receiving parts. The transmitting part is mainly composed of keyboard, remote control specific integrated circuit, exciter and infrared light emitting diode. The integrated circuit is the core part of the launch system which consists internal oscillation circuit, timing circuit, scanning signal generator, key input encoder, instruction decoder, user code converter, digital modulation circuit and buffer amplifier and so on. It can produce a key scanning pulse signal, translate the key code, then obtain remote control commands of the keys by telecommand encoder (remote control encoding pulse). Through pulse amplitude modulation of the 38KHZ carrier signal, the infrared diode can transmit infrared remote control signal.

In the infrared receiver, photoelectric converter (usually a photodiode or photoelectric triode, here we use PIN photodiode) converts the received infrared light instruction signal into a corresponding electrical signal. The received signal is very weak and interference is particularly large, in order to achieve the accurate detection and signal conversion, in addition to the infrared photoelectric conversion device with high performance, choosing the reasonable circuit design with good performance is also required. The most common photoelectric conversion device is a photodiode. When the photosensitive surface of the PN junction is irradiated by light, the semiconductor material of PN junction absorbs light energy and converts the light energy into electric energy. When the reverse voltage is added to the photodiode, the reverse current in the diode will change with the change of the incident light intensity. The larger the radiation intensity is, the larger the reverse current will be, the reverse current of the photoelectric varies with the incident light pulse.

In the Beetle-Bot car, the integrated infrared receiving head has three pins, including the power supply pin, grounding and signal output pin. The circuit is shown in Fig.3.2.5.1. Ceramic capacitors is a decoupling capacitor which can filter the output signal interference. The 1 end is the output of the demodulation signal which is directly connected to the number 2 port on the Arduino. When the infrared coded signal is transmitted, it will be processed by the infrared joint, then outputs the square wave signal, and directly supplied to the Arduino, and the corresponding operation is carried out to control the motor.

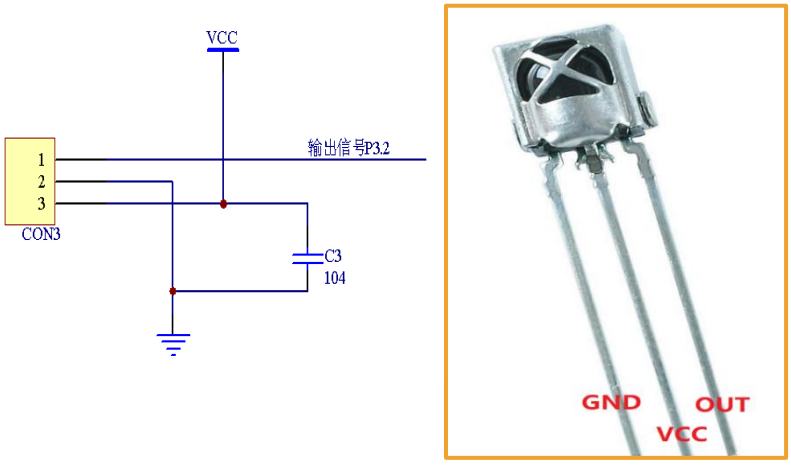


Figure .3.2.5.1 Circuit Diagram and Physical Map of Infrared Receiving Head

3.2.5.2 Working Principle

The remote control system in general composed of the remote controller (transmitter), and receiver, when you press any button on the remote control, it will generate the corresponding encoding pulse and output various control pulse signals based on the infrared, infrared monitor diode sends the signal to the the amplifier and the pulse amplitude limiter, the limiter controls the pulse amplitude at a certain level, regardless of the distance of infrared transmitter and receiver. AC signal enters the band-pass filter which can pass the 30KHZ to the load wave 60KHZ and enters the comparator through the demodulation circuit. The comparator outputs high or low level and restores the output signal waveform. The system procedure diagram is shown in Fig.3.2.5.2.

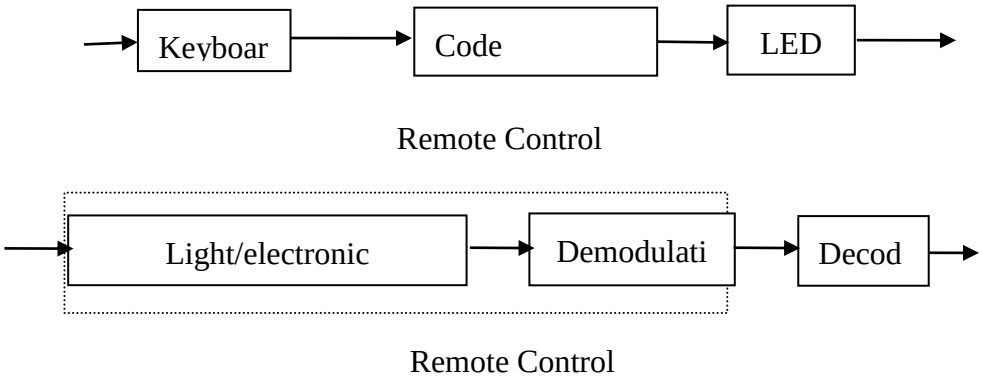


Fig 3.2.5.2 Remote Control System

3.2.5.3 Experimental Procedures

1, Installing the infrared receiving head on the development board (if it has been installed in the the eighth step in "3.1.2", please ignore. The complete installation is shown in Fig.3.1.5.3.

2, Referring to Fig.3.2.5.3 and connecting the infrared receiving module to the Arduino board (you can choose other IO ports according to your own ideas)

Note: Do not reverse power line, otherwise the receiving head will burn up.

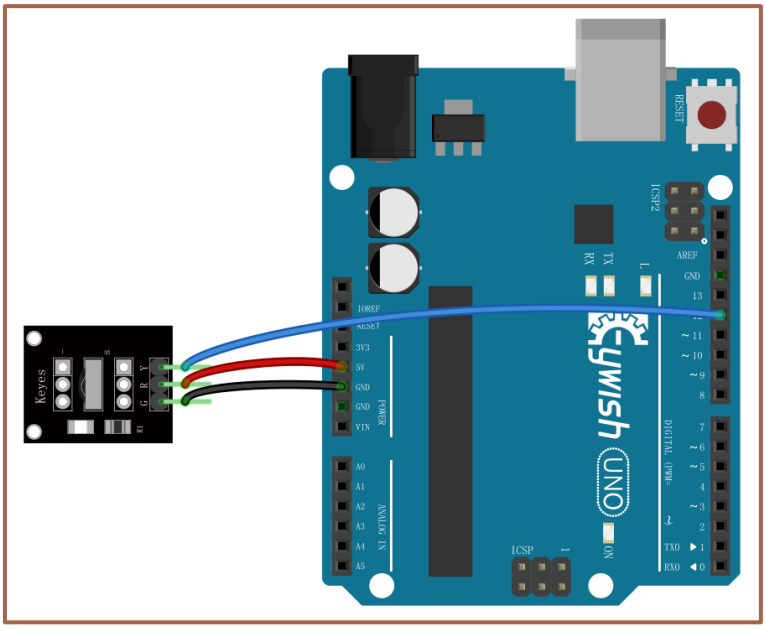
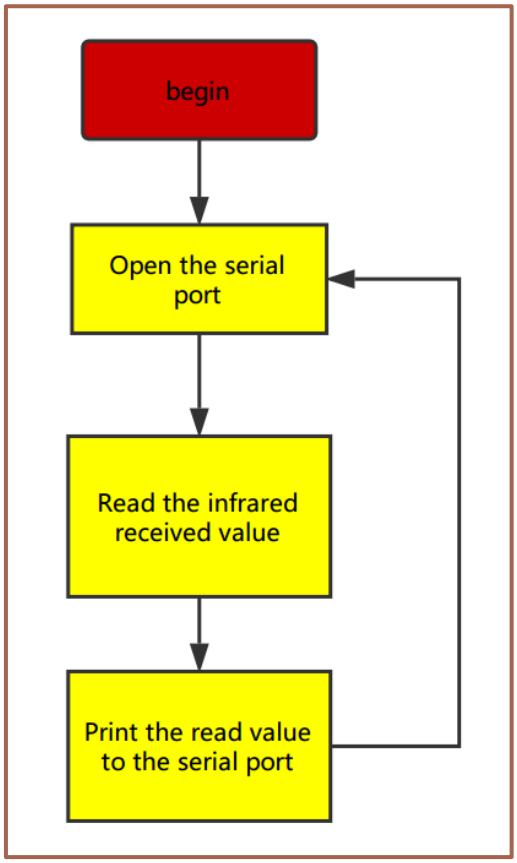


Figure .3.1.5.3 Infrared Module Connection Diagram

3, Copying the following program to IDE (you can also directly open the matching program in the CD), and downloading to the development board, pulled out the transparent plastic sheet marked as "1" in the Fig.3.2.33. Then opening the serial port monitor, observing and recording the values on it while pressing keys on the remote control towards the receiving head as shown in Fig.3.2.38.

Program flow chart is as follows:



Code Path: Lesson\ModuleDemo\IrTest\ IrTest.ino

```
#include "IRremote.h"
IRremote ir(12);
unsigned char keycode;
char str[128];
void setup() {
    Serial.begin(9600);
    ir.begin();
}
void loop()
{
    if (keycode = ir.getCode()) {
        String key_name = ir.getKeyMap(keycode);
        sprintf(str, "Get ir code: 0x%x key name: %s \n", keycode, (char
*)key_name.c_str());
        Serial.println(str);
    } else {
       //   Serial.println("no key");
    }
    delay(110);
}
```

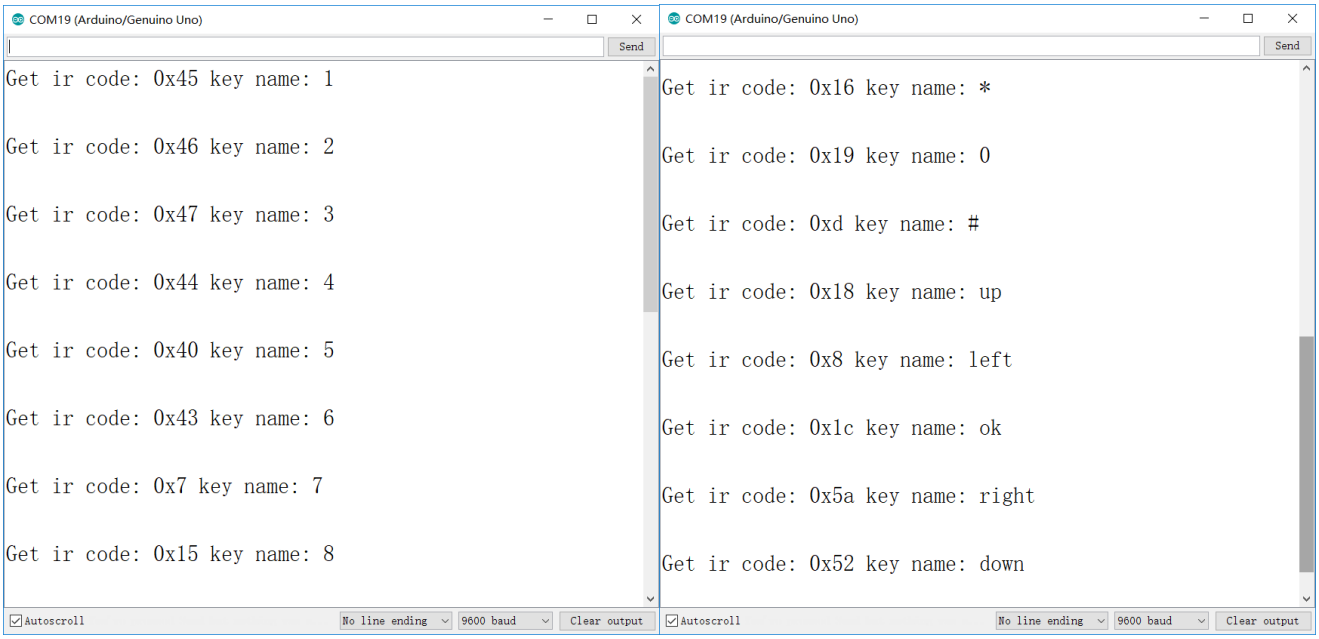


Figure 3.2.5.4 Remote Encoding Query

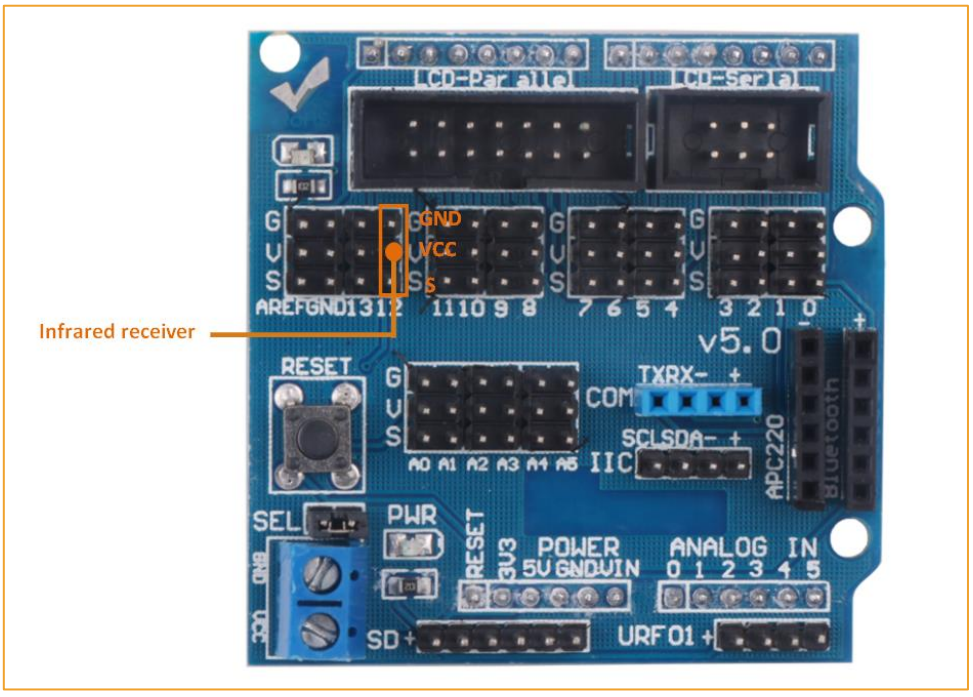
In Figure 3.2.5.4, we can see two values of Ir Code "0x45" and keyname "1", where "0x45" is the code of a key of the remote controller, and "1" is the name of the key function of the remote controller.

The key mapping table is as follows:

```
ST_KEY_MAP irkeymap[KEY_MAX] = {
    {"1", 0x45},
    {"2", 0x46},
    {"3", 0x47},
    {"4", 0x44},
    {"5", 0x40},
    {"6", 0x43},
    {"7", 0x07},
    {"8", 0x15},
    {"9", 0x09},
    {"0", 0x19},
    {"*", 0x16},
    {"#", 0x0D},
    {"up", 0x18},
    {"down", 0x52},
    {"ok", 0x1C},
    {"left", 0x08},
    {"right", 0x5A}
};
```

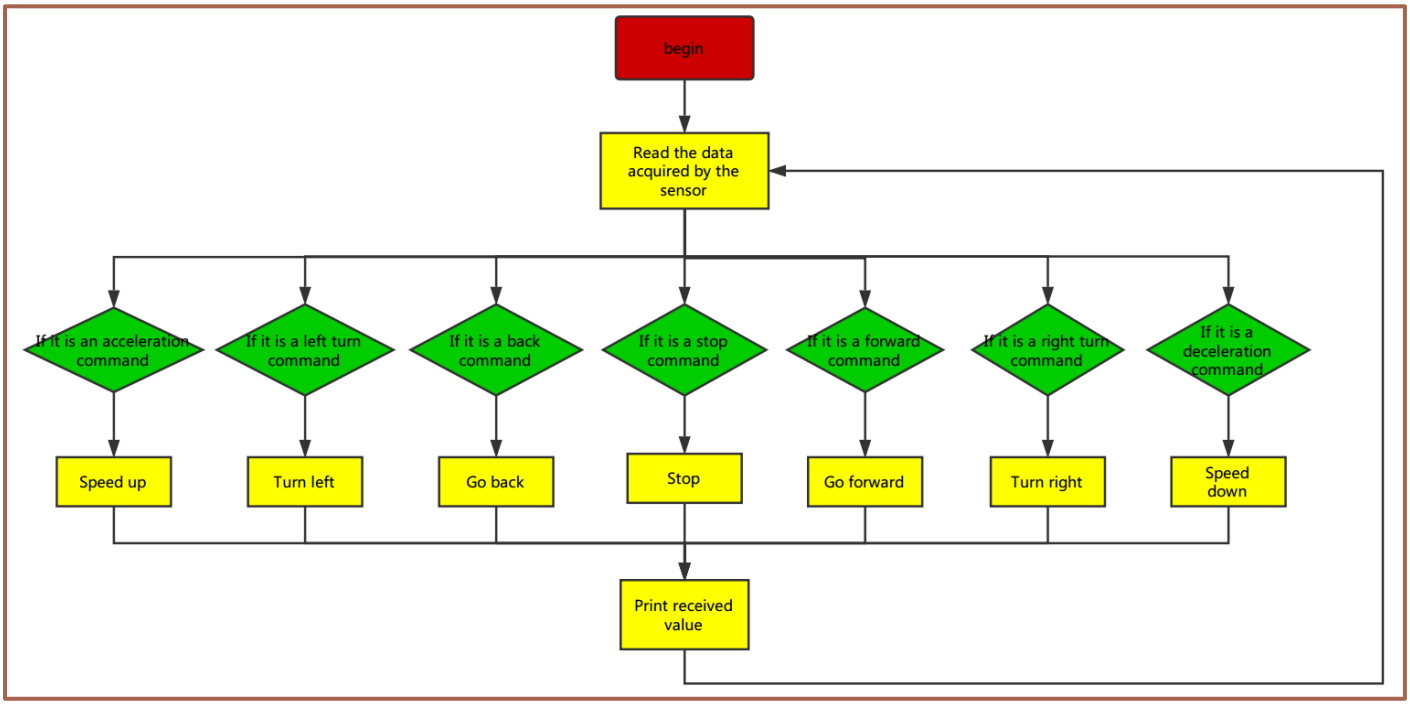
3.2.5.4 Wire connection

As shown in the below figure, the infrared receiver's “s” pin is connected to “12” pin, the “-” pin is connected to “GND”, and the middle pin is connected to “VCC”.



3.2.5.5 Software Design

3.2.5.5.1 Program flow chart



3.2.5.5.2 Program code

Code Path : Lesson\Advanced experiment\ Beetle_IR\ Beetle_IR.ino

```
#include "IRremote.h"/*In this section, we use infrared remote control, so we need to call the corresponding library file, as for what is in the library file, we will not study, and interested friends can drive research. We have put this library files on the CD-ROM, we need to copy this folder to the Arduino IDE installation path "libraries" folder, otherwise the program can not compile. */
int INPUT3_PIN = 5;//PWMA
int INPUT4_PIN = 9;//DIRA**************************************left
int INPUT1_PIN = 6;//PWMB
int INPUT2_PIN = 10;//DIRB**************************************right
int RECV_PIN = 12;
long expedite1 = 22;
long expedite2 = 13;
byte advance = 24;
byte back = 82;
byte stop = 28;
byte left = 8;
byte right = 90;
IRremote *mIrRecv;
```

```
void setup() {
  Serial.begin(9600);
  pinMode(INPUT1_PIN, OUTPUT);
  pinMode(INPUT3_PIN, OUTPUT);
  pinMode(INPUT4_PIN, OUTPUT);
  pinMode(INPUT2_PIN, OUTPUT);
  mIrRecv = new IRremote(RECV_PIN);
  mIrRecv->begin();// Initialize the infrared receiver
}
void loop() {
  byte irKeyCode;
  if (irKeyCode = mIrRecv->getCode()) {/* Read the valueue received by the infrared */
    if (irKeyCode == advance) { /* Judgment on the received valueue, if this valueue is advence,
execute the following {} command, here is the forward instruction. */
      int value = 150;
      analogWrite (INPUT4_PIN, value);
      analogWrite(INPUT3_PIN, 0);//the speed valueue of motorA is value
      analogWrite (INPUT2_PIN, 0);
      analogWrite(INPUT1_PIN, value);//the speed valueue of motorB is value
      delay(500);  // Receive the next valueue
    }
    if (irKeyCode == expedite1) {/*Judgment on the valueue received, if this valueue is
expedite1,execute the command in{}below, here is the acceleration 1 command.*/
      int value = 200;
      analogWrite (INPUT4_PIN, 0);
      analogWrite(INPUT3_PIN, value);//the speed valueue of motorA is value
      analogWrite (INPUT2_PIN, 0);
      analogWrite(INPUT1_PIN, value);//the speed valueue of motorB is value
      delay(500);
    }
    if (irKeyCode == expedite2) {/*Judgment on the received valueue, if the valueue is
expedite2,execute the command {} below,here for the acceleration 2 command. */
      int value = 255;
      analogWrite(INPUT4_PIN, 0);
      analogWrite(INPUT3_PIN, value);//the speed valueue of motorA is value
      analogWrite(INPUT2_PIN, 0);
      analogWrite(INPUT1_PIN, value);//the speed valueue of motorB is value
      delay(500);
    }
    if (irKeyCode == stop) { /* To judge the valueue received, if this valueue is stop, execute
the command in the following {}, here is the stop instruction. */
      int value = 0;
```

```
      analogWrite(INPUT4_PIN, 0);
      analogWrite(INPUT3_PIN, value);//the speed valueue of motorA is value
      analogWrite(INPUT2_PIN, 0);
      analogWrite(INPUT1_PIN, value);//the speed valueue of motorB is value
      delay(500);
    }
    if (irKeyCode == left) {/* Judgment on the received valueue, if the valueue is left,
execute the command in the following {}, here is the instruction to the left. */
      int value = 150;
      analogWrite(INPUT4_PIN, value);
      analogWrite(INPUT3_PIN, 0);//the speed valueue of motorA is value
      analogWrite(INPUT1_PIN, 0);
      analogWrite(INPUT2_PIN, 0);//the speed valueue of motorB is value
      delay(500);/* Rotate 500ms to the left and stop, otherwise the car will always be spinning
around. */
      analogWrite(INPUT4_PIN, 0);
      analogWrite(INPUT3_PIN, 0);//the speed valueue of motorA is 0
      analogWrite(INPUT2_PIN , 0);
      analogWrite(INPUT1_PIN, 0);//the speed valueue of motorB is 0
    }
    if (irKeyCode == right) {/* Judgment on the received valueue, if the valueue is right,
execute the command in the following {}, here is the command to the right. */
      int value = 150;
      analogWrite(INPUT3_PIN, 0);
      analogWrite(INPUT4_PIN, 0);//the speed valueue of motorA is value
      analogWrite(INPUT2_PIN, 0);
      analogWrite(INPUT1_PIN, value);//the speed valueue of motorA is value
      delay(500);
      analogWrite(INPUT4_PIN, 0);
      analogWrite(INPUT3_PIN, 0);//the speed valueue of motorA is 0
      analogWrite(INPUT2_PIN, 0);
      analogWrite(INPUT1_PIN, 0);//the speed valueue of motorB is 0
    }
    if (irKeyCode == back) {/* Judgment on the received valueue, if the valueue is back,
execute the command {} below, here for the back instruction. */
      int value = 150;
      analogWrite(INPUT3_PIN, value);
      analogWrite(INPUT4_PIN, 0);//the speed valueue of motorA is value
      analogWrite(INPUT1_PIN, 0);
      analogWrite(INPUT2_PIN, value);//the speed valueue of motorA is value
      delay(500);
    }
```

```
    Serial.println(irKeyCode, HEX);// Output the receive code in hexadecimal
    Serial.println();// Add a blank line for easy viewing of the output
  }
}
```

The above is the infrared remote reference program, you can open it in the CD and download to the development board, the instructions of remote control in shown in Fig.3.2.5.5.

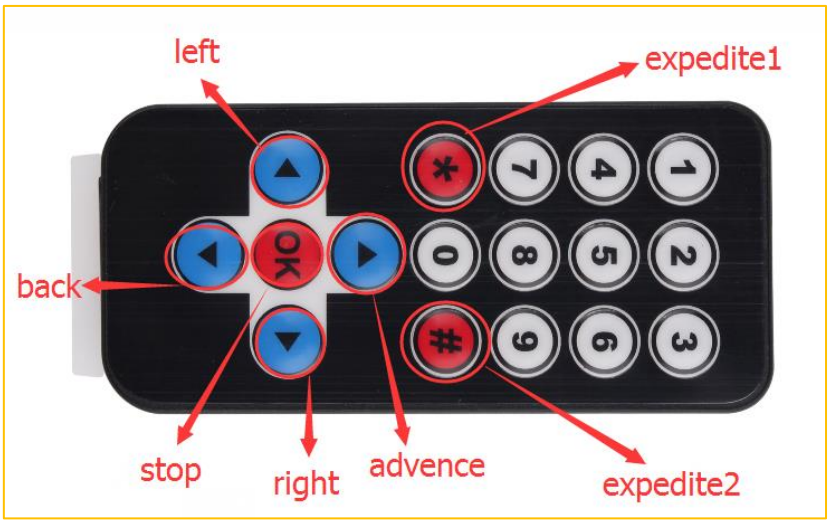


Fig 3.2.5.5 Infrared Remote Control Instructions

3.2.6 Mobile Phone Bluetooth Control

3.2.6.1 Suite Introduction

Beetle-Bot supports Bluetooth mobile phone Bluetooth remote control function is JDY-16 BLE (Bluetooth Low Energy).JDY-16 transparent transmission module is based on the Bluetooth 4.2 protocol standard, operating frequency range is 2.4GHZ range, modulation mode is GFSK, maximum transmission power is 0db, maximum transmission distance is 80 meters, adopts original imported chip design, and supports users to modify device by AT command. Names, service UUIDs, transmit power, pairing passwords, and other instructions are convenient and quick to use. For module information, see Beetle-bot\Document\JDY-16-V1.2(English manual).pdf

The physical diagram of module is shown in Fig 3.2.6.1

Fig 3.2.6.1 JDY-16 Module

1) JDY-16 function introduction

1：BLE high speed transparent transmission supports 8K Bytes rate communication.

2：Send and receive data without byte limit, support 115200 baud rate continuously send and receive data.

3：Support 3 modes of work (see the description of AT+STARTEN instruction function).

4：Support (serial port, IO, APP) sleep wake up.

5：Support WeChat Airsync, WeChat applet and APP communication.

6：Support 4 channel IO port control.

7：Support high precision RTC clock.

8：Support PWM function (can be controlled by UART, IIC, APP, etc.).

9：Support UART and IIC communication mode, default to UART communication.

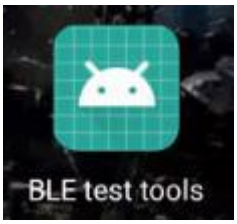
10: iBeacon mode (support WeChat shake protocol and Apple iBeacon protocol).

11: Host transparent transmission mode (transmission of data between application modules, host and slave communication).

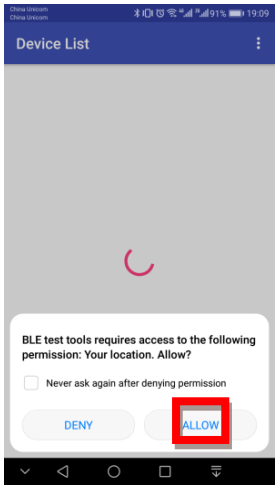
3.2.6.2 Use app to communicate with the serial port

1)Install “Beetle-bot\JDY-16\BLETestToos.apk”, then turn on Bluetooth.

2) Open app, you will be asked to allow location permission at first, please allow.

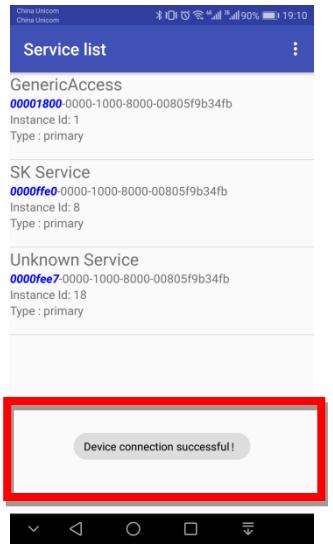
APP: 

3) If Bluetooth information is not displayed, please pull down to refresh， if it still don’t work,please close and reopen it.

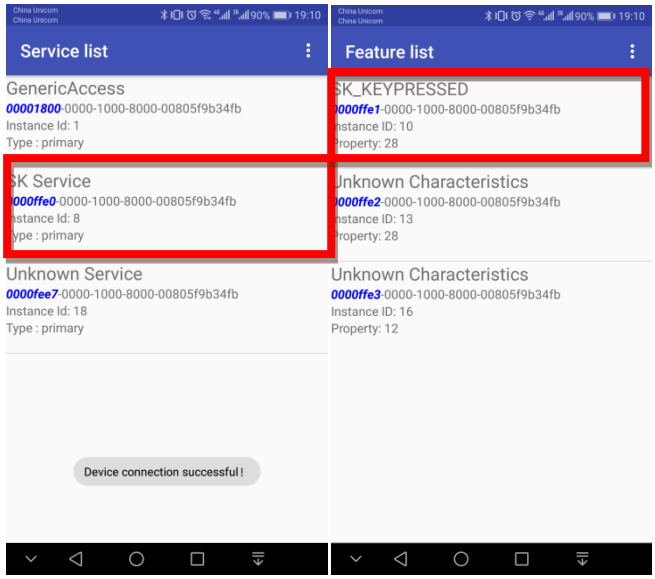


4) Select the JDY-16 Bluetooth model.

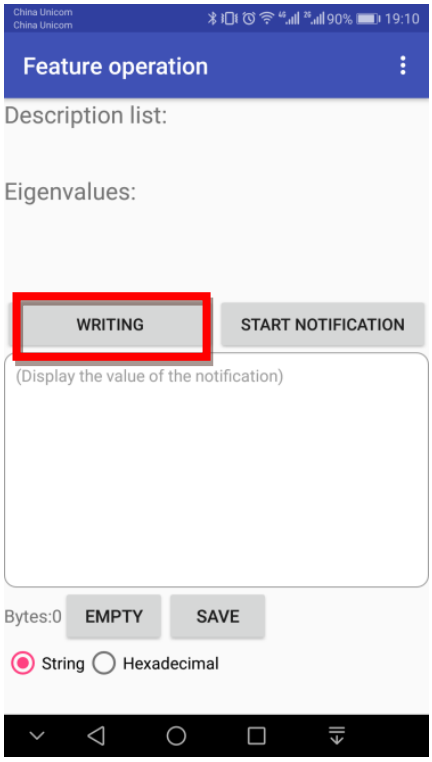
5) There will pop up a window which indicate connection success and jump to the operation interface:



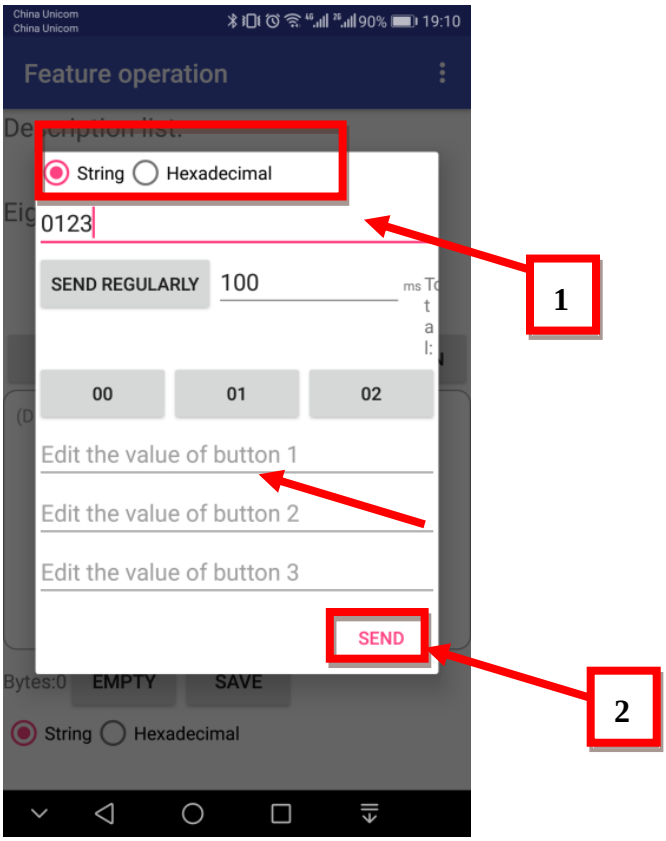
6) Click the second option “SK Service”,then click the first option “SK KEYPRESSED”:



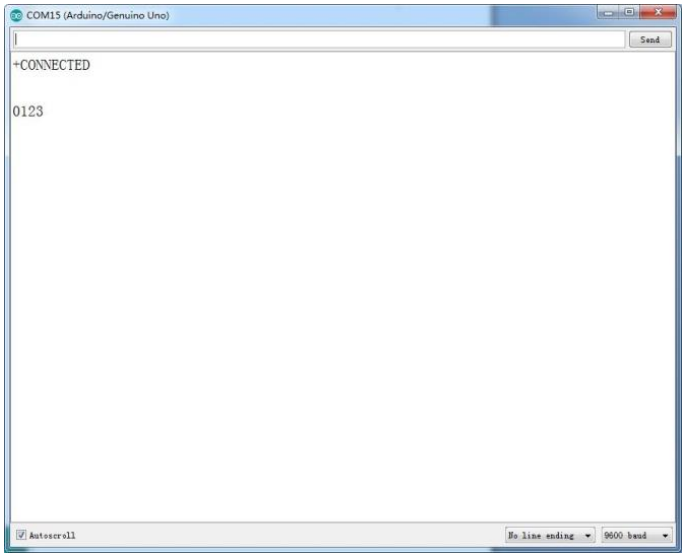
7) In the end you will jump to the user interface,click “WRITING” to send.



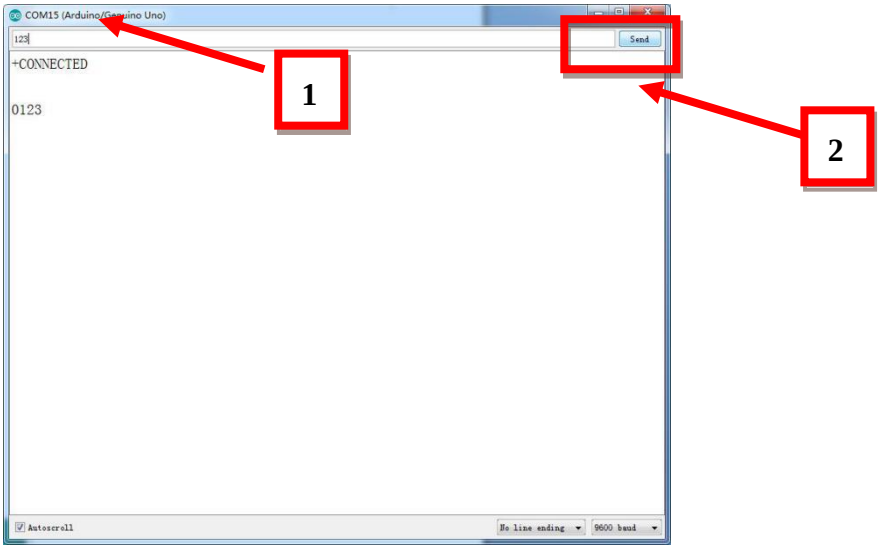
8) Write data on the pink line, click “send” to send.



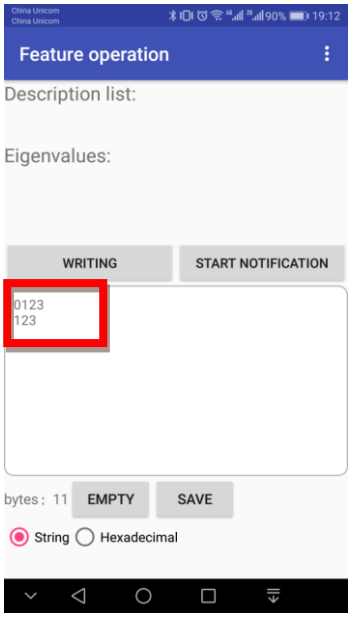
Serial port display as follow:



9) You can also send date through Arduino serial port to app.



10) The result as below:



During above test, both the PC and Android can send and receive data normally, indicating that the Bluetooth module communicates normally and achieves the expect effect. Next, it can be used as a bridge between "Aurora" and the APP to control "Aurora" to realize the desired features.

3.2.6.3 Bluetooth protocol

Using Bluetooth to control the car means we use the Android app to control the Bluetooth sending instructions to the Arduino serial port, so as to control the motor's forwarding, reversing, speed and so on. Since the wireless communication is involved, one of the essential issues is the communication problem between the two terminals. But there is no common "language" between them, so designing communication protocols are required to ensure a perfect interaction between Android and Arduino. The main process is: the Android identification terminal packs the detected commands into the corresponding data packets, and then sends them to the Bluetooth module (JDY-16). When JDY-16 receives the packets, it will transmit them to Arduino through the serial port, then Arduino to analyze the data packets and execute the corresponding actions. The data format sent by the upper computer end (Android) is as follows, which mainly contains 8 fields, we use a structure to represent it.

Protocol Header	Data Length	Device Type	Device Address	Function Code	Control Data	Check Sum	Protocol End Code

In the 8 fields above, we use a structural body to represent.

```
typedef struct
{
    unsigned char start_code;     // 8bit 0xAA
    unsigned char len;
    unsigned char type;
    unsigned char addr;
    unsigned short int function;   // 16 bit
    unsigned char *data;          // n bit
    unsigned short int sum;        // check sum
    unsigned char end_code;        // 8bit 0x55
}ST_protocol;
```

“Protocol Header” means the beginning of the packet, such as the uniform designation of 0xAA.
Data length” means except the valid data length of the start and end codes of the data.
“Device type” means the type of device equipment
“Device address” means the address that is set for control

“Function code” means the type of equipment functions that need to be controlled, the function types we currently support is as follows:

```
typedef enum
{
    E_BATTERY = 1,
    E_LED = 2,
    E_BUZZER = 3,
    E_INFO = 4,
    E_ROBOT_CONTROL = 5,
    E_ROBOT_CONTROL_SPEED = 6,
    E_TEMPERATURE = 7,
    E_IR_TRACKING = 8,
    E_ULTRASONIC = 9,
    E_VERSION = 10,
    E_UPGRADE = 11,
}E_CONTOROL_FUNC ;
```

The data means the specific control value of a car, such as speed, angle.

“Checksum” is the result of different or calculated data bits of the control instruction.

“Protocol end code ” is the end part of the data bag when receiving this data ,it means that the data pack has been sent, and is ready for receiving the next data pack, here we specified it as 0x55.

For example, a complete packet can be such as "AA 070101065000 5F55", in which:

"07" is Transmission Data Length 7 bytes.

"06" is the "device type", such as motor, LED, buzzer and so on. The 06 here refers to the transmission speed, and the 05 refers to the transmission direction.

"50 (or 0050)" is the controlling data, 0x50 in hexadecimal is 80 when converted to binary, which means the speed value is 80. If the data is 05, it means the controlling direction, that is 80 degrees (forward).

"005F" is the check sum, that is, 0x01+0x01+0x06+0x50=0x5F.

"55" is the tail of the protocol, indicating the end of data transmission.

Fig3.2.6.2 the Interface of Android APP

In the above Figure 3.2.6.2:

The "A, B" sections are the acceleration and deceleration buttons.

The "C" section includes the dashboard and the digital display area, and the two parts displaying synchronously. They represent the current speed.

The "D" section is a gravity remote sensing switch which can be switched to the gravity remote sensing mode.

The "E" section represents the Bluetooth name that is currently connected.

The "F" section indicates Bluetooth connection state. If the Bluetooth is not connected, the "disconnected" is displayed here.

The "G" section is a manual rocker, and sliding it allows the car to rotate.

The "I" section is a data return area, such as the current state, speed of the car, etc. The "H" section is the data packet, for example, the data is "AA 01 01 06 23 00 2B 55". At this time, the speed is 35 (23 is 16 hexadecimal data, which means 35 when converted to 10 hexadecimal).

If the transmitted data is "AA 01 01 05 00 5B 00 62 55", it means that the car is moving forward (05 is the direction control instruction, and the 005B means 91 when converted to binary number. By the Figure.3.2.48 we can know that 91 degree means the car is moving forward).

3.2.6.4 Experimental Procedures

Connect the Bluetooth module of the wire to "1" marked in Fig.3.2.51 Connection mode: JDY-16 VCC port on Bluetooth module is connected to Arduino 3.6V~6V DC power anode, GND port is connected to the cathode of the DC power, the RXD port is connected to the TXD port on Arduino extended board, TXD port is connected to RXD port on the board, as shown in Fig.3.2.52.

Note: Since Arduino UNO has only one serial port, the Bluetooth must be disconnected from the serial port when downloading the program, otherwise the download will fail.

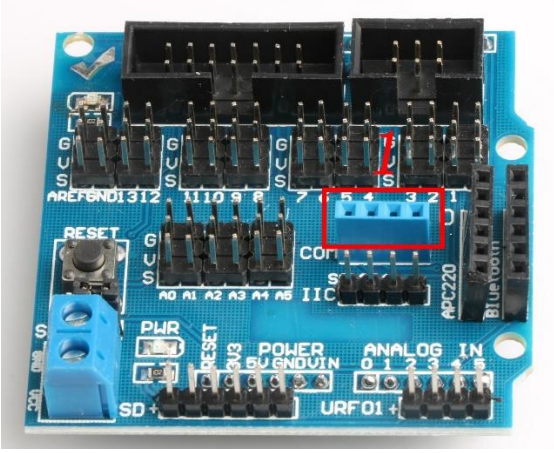


Figure 3.2.6.3 Wiring Positions of Bluetooth

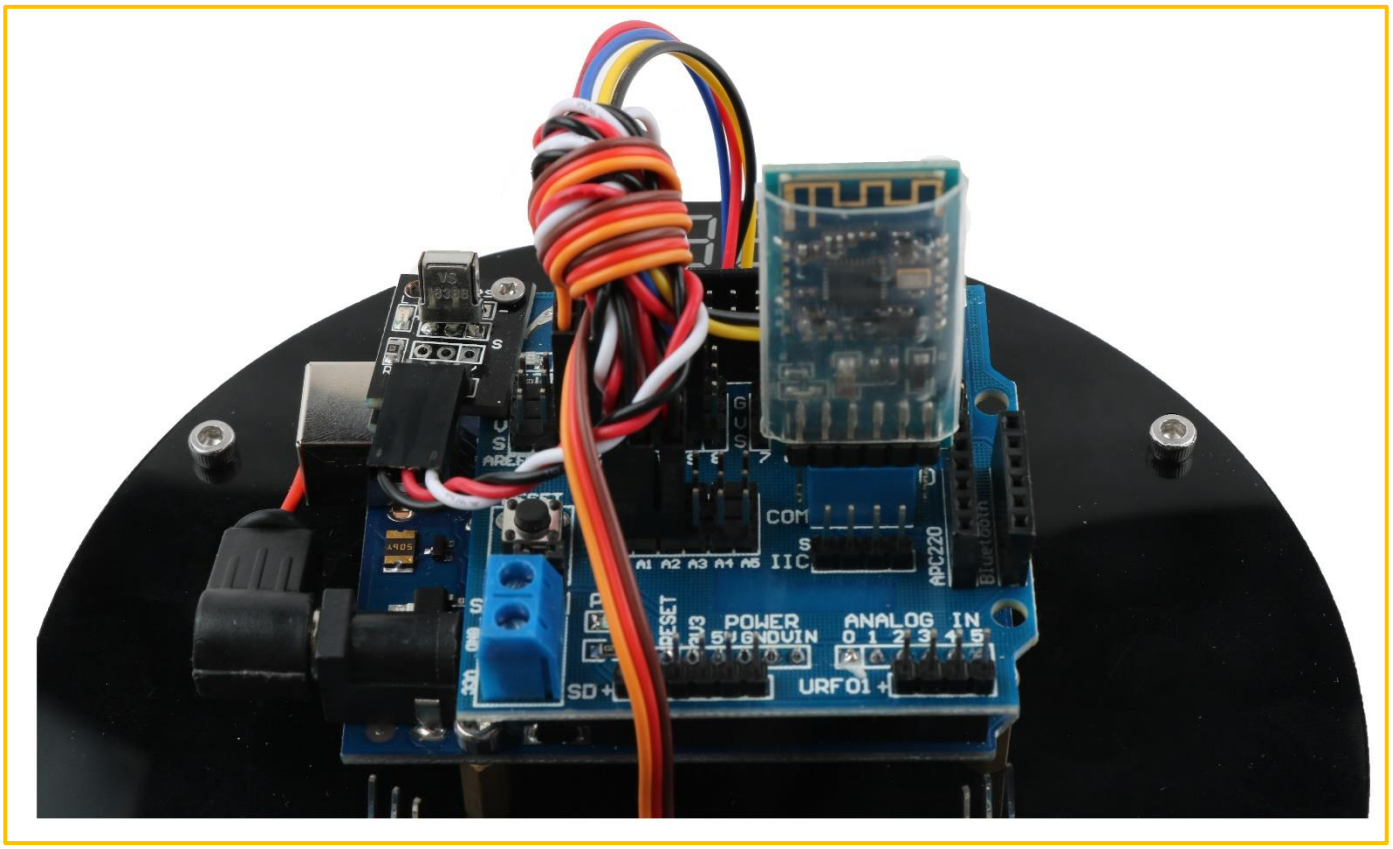


Figure 3.2.6.4 Installation of Bluetooth Module

2, Open the mobile phone Bluetooth and the APP (There is a software installation package on the CD, latter we will launch the IOS version.) You will see that the flashing of the Bluetooth module indicator slows down after successful connection. If you have downloaded the program to the development board before, you can use the phone to control the car directly, as shown in Fig.3.2.6.4.

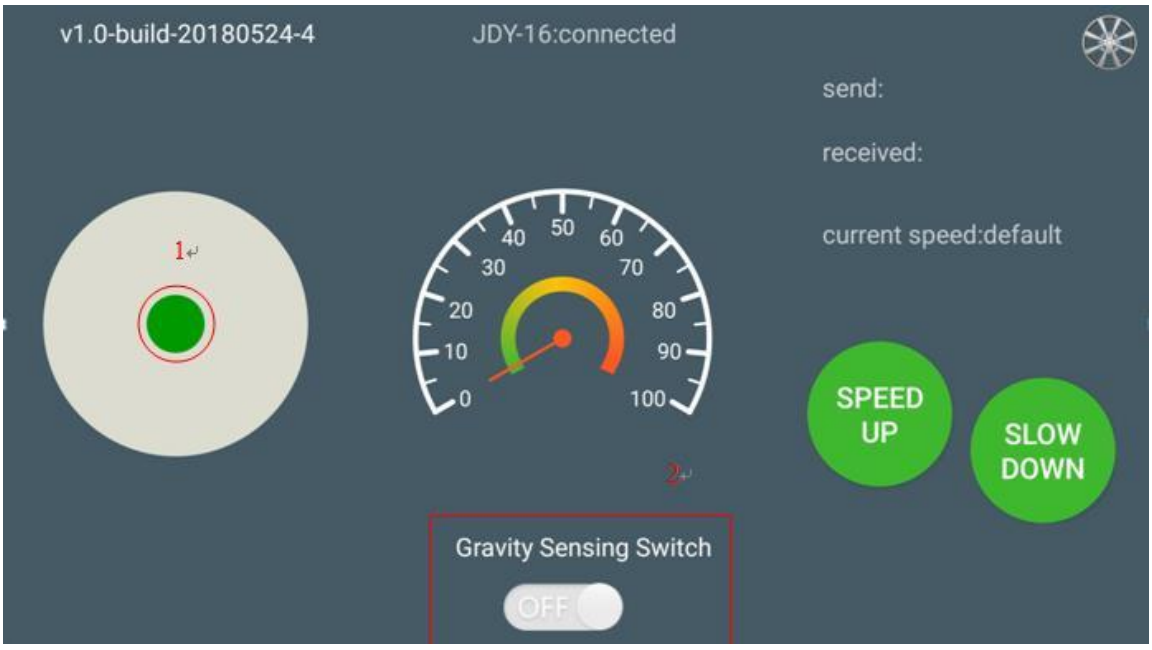
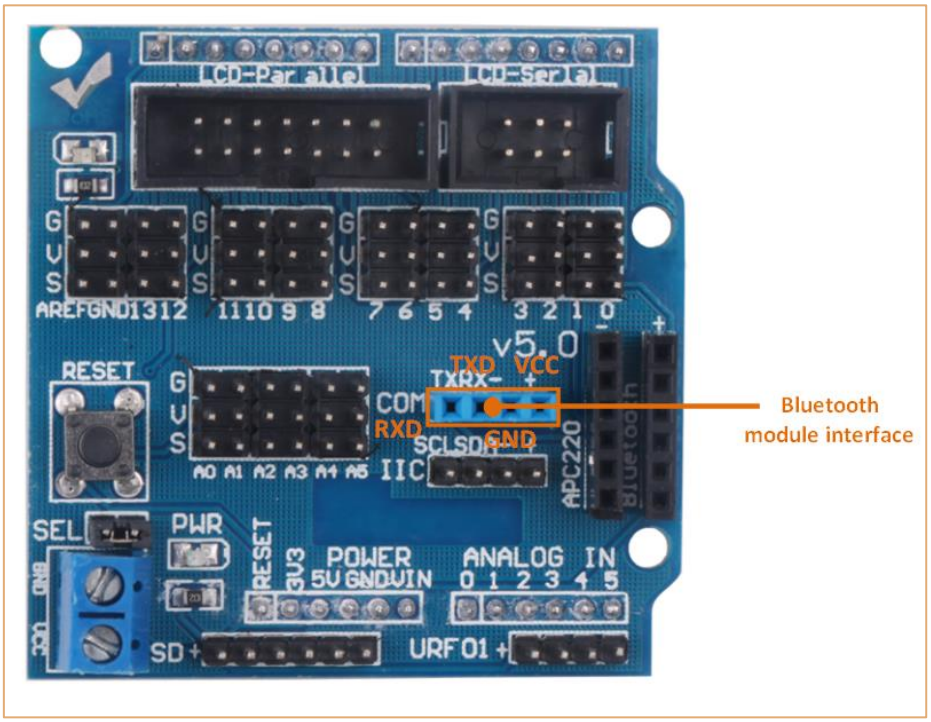


Figure .3.2.6.5. Diagram of APP Control

In Fig.3.2.6.5, we can see the logo "1" and "2". When the Bluetooth connection is successful, sliding green dot marked as "1" in any direction, the car will move towards the corresponding direction. Switching on the gravity sensor marked in "2", the APP is switched to the gravity induction mode, and you can control the movement direction of the car by shaking the mobile phone.

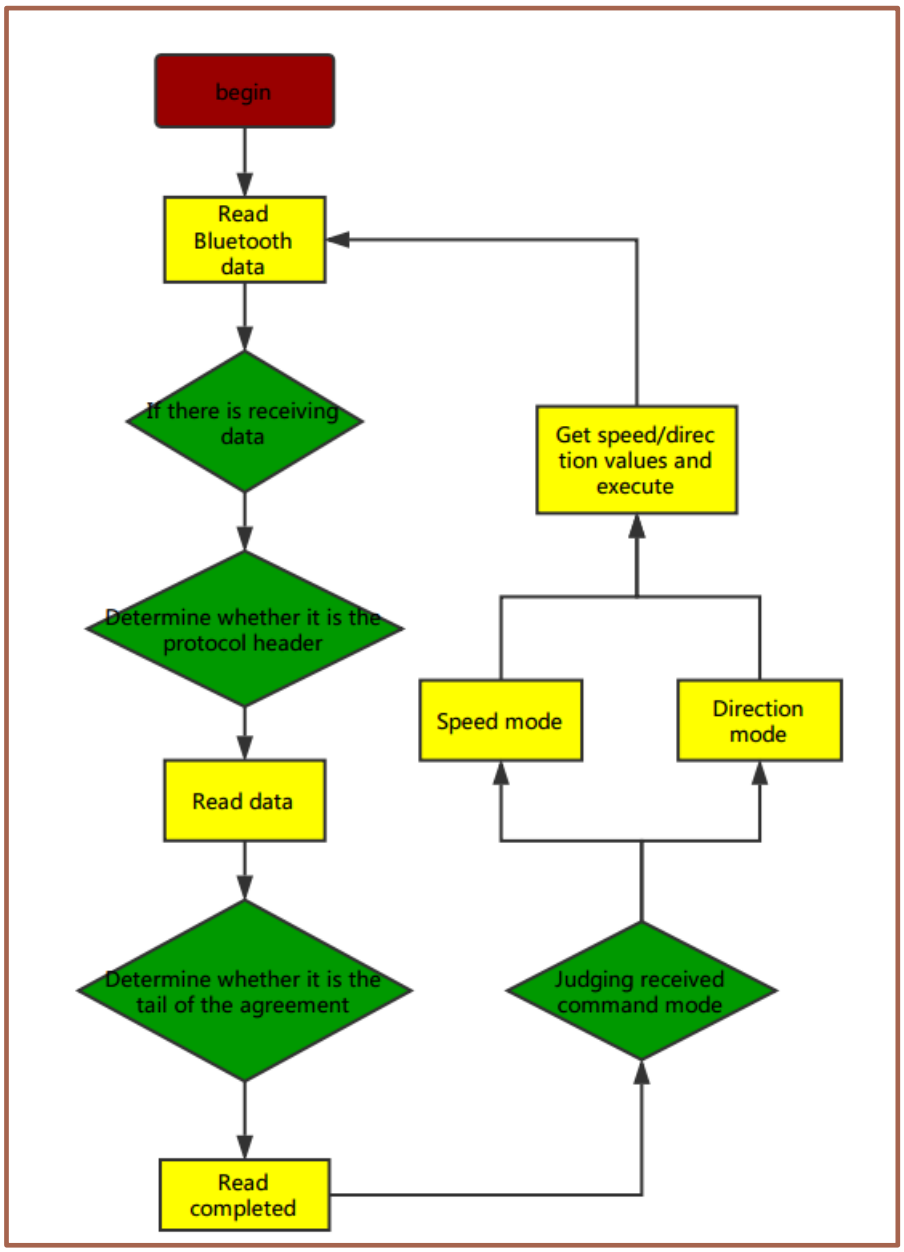
3.2.6.5 Wire connection

Bluetooth wire connection location as shown below:



3.2.6.5 Software Design

3.2.6.5.1 Program flow char



3.2.6.5.2 Program Code

Code path：Lesson\Advanced experiment\Beetle_Bluetooth\Beetle_Bluetooth\Beetle_Bluetooth.ino

```
#include "Beetlebot.h"
#include "ProtocolParser.h"
#include "KeyMap.h"
#include "debug.h"
#define INPUT2_PIN 10 // PWMB
#define INPUT1_PIN 6  // DIRB  ---  right
#define INPUT4_PIN 9  // PWMA
```

```
#define INPUT3_PIN 5  // DIRA  ---  left

ProtocolParser *mProtocol = new ProtocolParser();
Beetlebot beetle(mProtocol, INPUT2_PIN, INPUT1_PIN, INPUT3_PIN, INPUT4_PIN);
void setup()
{
    Serial.begin(9600);
    beetle.init();
    beetle.SetControlMode(E_BLUETOOTH_CONTROL);
}
void HandleBluetoothRemote()
{
    if (mProtocol->ParserPackage())
    {
        switch(mProtocol->GetRobotControlFun())
        {
            case E_INFO:
                break;
            case E_ROBOT_CONTROL_DIRECTION:
                beetle.Drive(mProtocol->GetRobotDegree());
                break;
            case E_ROBOT_CONTROL_SPEED:
                beetle.SetSpeed(mProtocol->GetRobotSpeed());
                break ;
            case E_VERSION:
                break;
        }
    }
}
void loop()
{
    mProtocol->RecevData();
    switch(beetle.GetControlMode())
    {
        case E_BLUETOOTH_CONTROL:
            HandleBluetoothRemote();
            DEBUG_LOG(DEBUG_LEVEL_INFO, "E_BLUETOOTH_CONTROL \n");
            break;
        default:
            break;
    }
}
```

3.2.7 PS2 Handle (Optional)

3.2.7.1 Suite Introduction

PS2 handle is SONY game remote control handle, SONY series game host is very popular in the world. Someone has come up with the idea of the PS2 handle, cracked the communication protocol, so that the handle can be connected to other devices for remote control, such as remote control of our familiar four wheeled vehicles and robots. Its outstanding features are cost-effective, rich buttons and easy to extend to other applications, Fig.3.2.7.1 shows a commonly used PS2 wireless handle.

Figure .3.2.7.1 PS2 Wireless Handle

The PS2 handle is composed of the handle and the receiver, the handle uses two AAA batteries as power supply. The controller and receiver use the same power supply whose voltage range is 3~5V, overvoltage, reverse connection will cause the receiver to burn out. There is a power switch on the handle, ON /OFF, when you switch it to ON, the light on handle will not stop flashing until the receiver is searched. In a certain period of time, if the receiver can't be find, the handle will enter the standby mode and the light will burn out, you can press "START" button to wake up the handle.

The receiver is powered by the Arduino as shown in Fig.3.2.7.2, in the absence of pairing, the green light flashes. When the handle switch is opened, the handle and receiver will pair automatically, and the light will be always on, and the handle is matched successfully. Button "MODE" (The above logo may be "ANALOG" in different handles, but it will not affect the usefulness), you can choose "red light mode" or "green light mode".

Some users see that the handle and receiver cannot match properly! Most of the problems are the incorrect wiring of the receiver and the program problems.

Solution: The receiver should only be connected with power supply (power line must be connected correctly), not any data lines and clock lines. The handle light will be always on when pairing is successful. Then checking whether the wiring and program transplantation are correct again.。

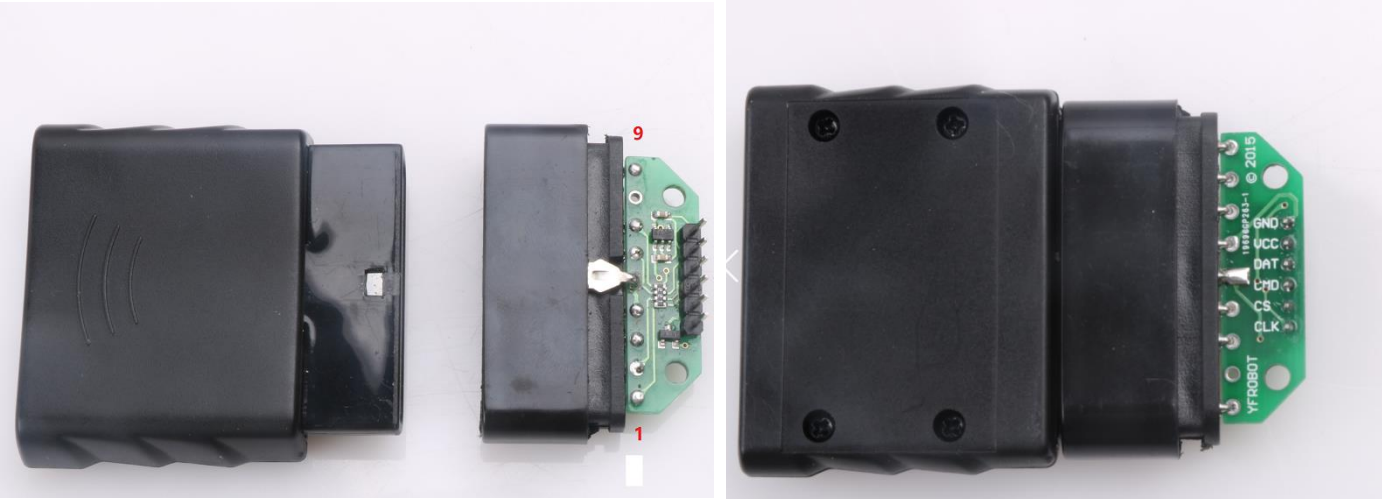


Figure .3.2.7.2 Remote Control Receiver Modul

There are 9 interfaces at the end of the receiving head, each of which is shown in the following table:

1	2	3	4	5	6	7	8	9
DI/DAT	DO/CMD	NC	GND	VDD	CS/SEL	CLK	NC	ACK

Note: The appearance of the receiver will be different due to different batches, some with a power light, some without, but the use and definition of the pins are the same.

DI/DAT: The signal flows from the handle to the host. This signal is a serial 8-bit data which is transmitted synchronously to the falling edge of the clock. The read of the signal is completed in the process of clock changing from high to low;

DO/CMD: The signal flows from the host to the handle. The signal is similar to the DI, a serial 8-bit data, which is transmitted synchronously to the falling edge of the clock;

NC: Empty port; .

GND: Ground;

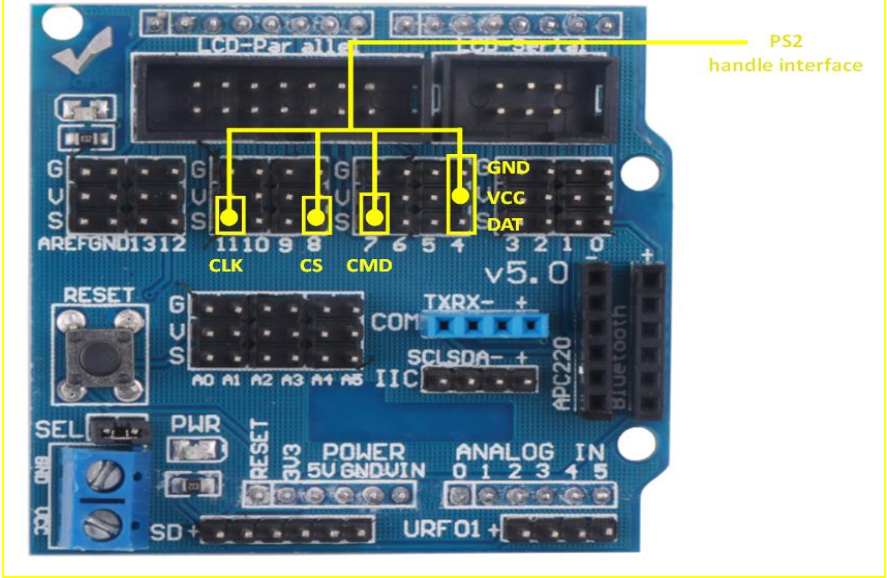
VDD: The 3~5V power supply;

CS/SEL: Providing trigger signals for handles, the level is low during communication;
CLK: The clock signal is sent by the host to maintain data synchronization;
NC: Empty port;
ACK: The response signal from the handle to the host. This signal changes to low in the last cycle of each 8-bit data sending, and the CS remains low. If the CS signal do not remain low, the PS host will try another device in about 60 microseconds. The ACK port is not used in programming.

3.2.7.2 Experimental Procedure

1、In order to wire simply, we use the PS2 receiver head adapter board and Arduino connection as follows:

As shown in the below figure, the six lines of the receiving head are connected to the following places respectively. “CLK” is connected to 11 pins, “CS” is connected to 8 pins, “CMD” is connected to pin 7, “DAT” is connected to 4 pins, and “VCC” is connected to 5V. , "GND" is grounded.



Ps2	Arduino Uno
DAT	4
CMD	7
CS	8
CLK	11

First remove the screw of the voltage display module, add two M3* 55 copper posts on the base plate (as shown in Figure 3.2.7.4), and tie the PS2 receiving head to the two copper posts with the cable tie. When the acrylic plate is docked with the lower acrylic plate, the hole of the voltage display module is exactly the same as the copper column, and the screw can be screwed directly into the copper column.

Figure 3. 2. 7. 3

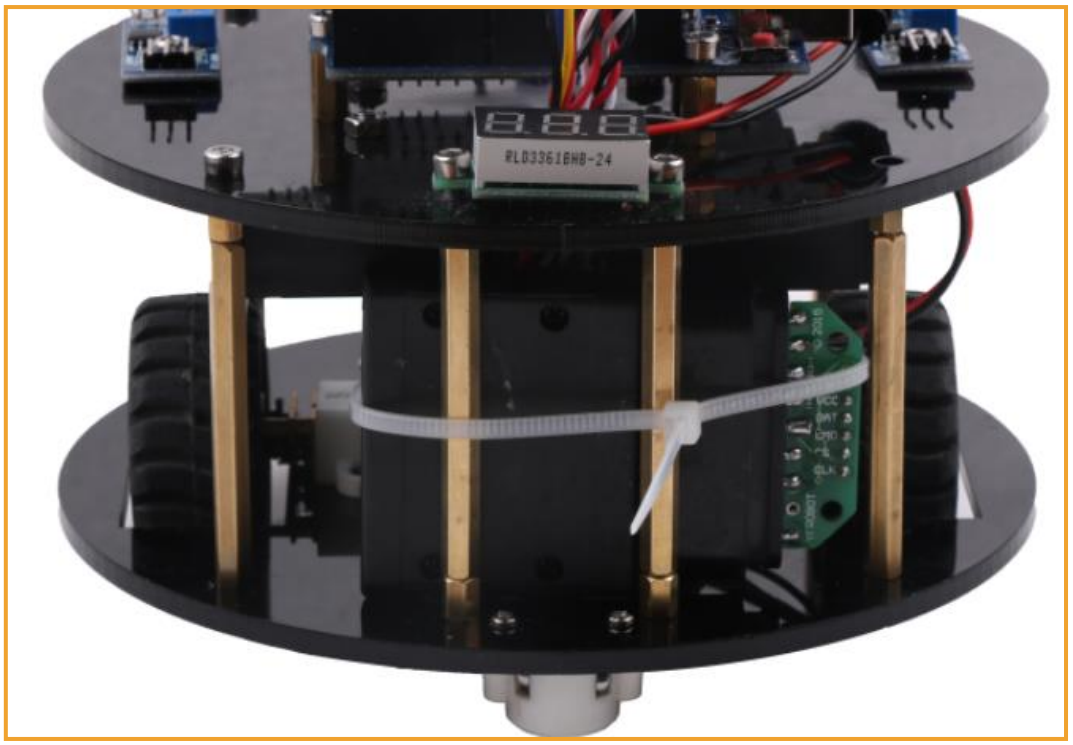


Figure 3.2.7.4 PS2 Receiver Fixed Overall Effect Diagram

Connect the wires to the Aruino expansion board as shown in Figure 2.3.58. Upon completion, as shown in Figure 3.2.7.5.

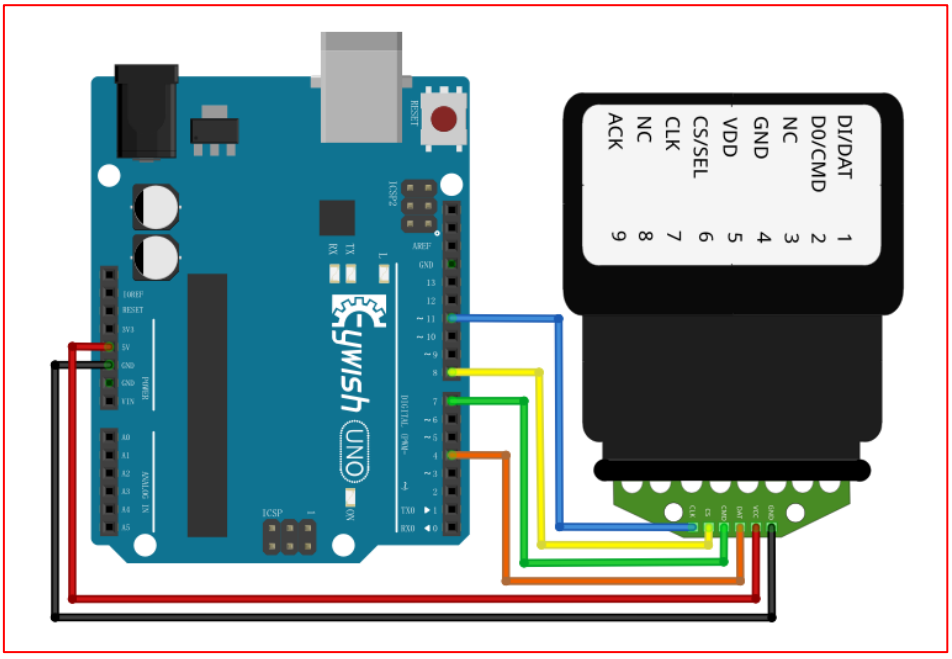


Figure 3.2.7.5 Connection of Arduino and Receiving Head Wires

4、Open “Lesson\ModuleDemo\PS2XTest\PS2XTest.ino”

Finally, download the program to the Arduino development board and turn on the PS2 remote control. If the receiver head is connected to the remote controller (or the pairing is successful), the indicator light on the receiver head is long, and the LED light is blinking. Finally, we open the "serial monitor", press any button on the remote control, you can see the corresponding data on the "serial monitor", as shown in Figure 3.2.7.5.

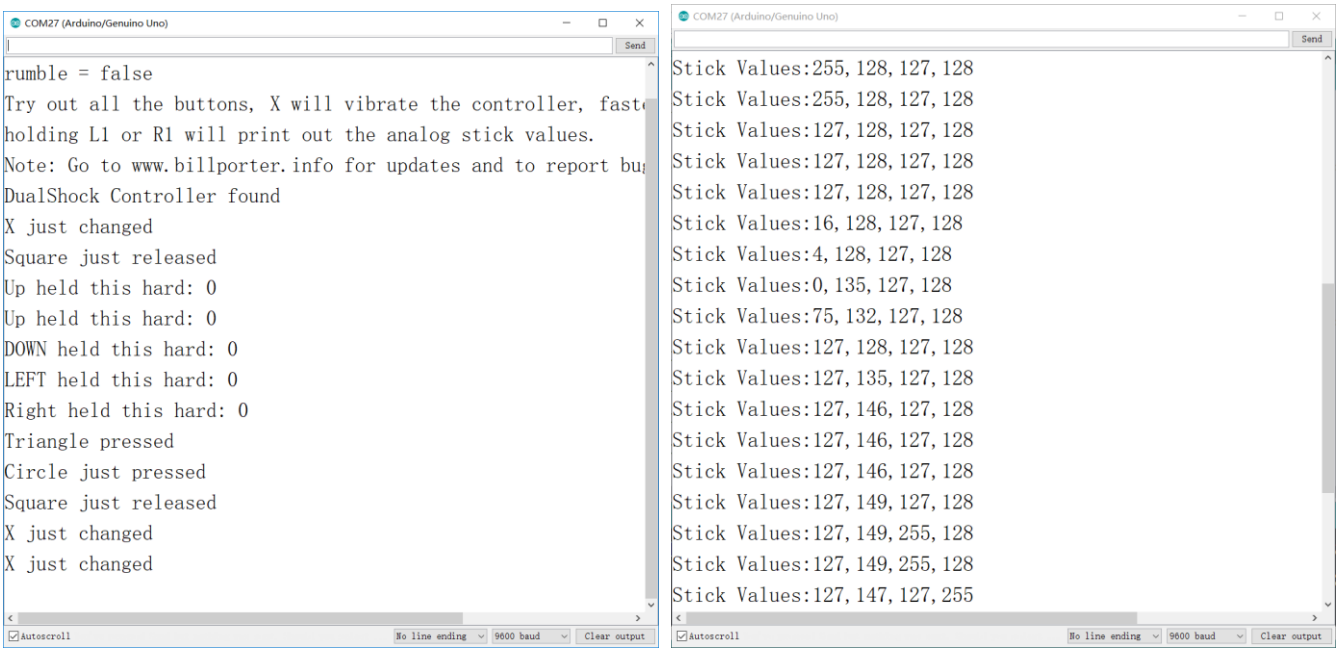


Figure 3.2.59 "serial monitor" data display

3.2.6.3 Software Design

```
#include "PS2X_lib.h"  //for v1.6

/******************************************************************
 * set pins connected to PS2 controller:
 *   - 1e column: original
 *   - 2e colmun: Stef?
 * replace pin numbers by the ones you use
 ******************************************************************/
#define PS2_DAT     4
#define PS2_CMD     7
#define PS2_SEL     8
#define PS2_CLK     11

/******************************************************************
 * select modes of PS2 controller:
 *   - pressures = analog reading of push-butttons
 *   - rumble    = motor rumbling
 * uncomment 1 of the lines for each mode selection
 ******************************************************************/
//#define pressures   true
#define pressures   false
//#define rumble      true
#define rumble      false

PS2X ps2x; // create PS2 Controller Class

//right now, the library does NOT support hot pluggable controllers, meaning
//you must always either restart your Arduino after you connect the controller,
//or call config_gamepad(pins) again after connecting the controller.

int error = 0;
byte type = 0;
byte vibrate = 0;

// Reset func
void (* resetFunc) (void) = 0;
```

```
void setup() {
  Serial.begin(9600);
  delay(500);  //added delay to give wireless ps2 module some time to startup, before configuring it
  //CHANGES for v1.6 HERE!!! **************PAY ATTENTION*************
  //setup pins and settings: GamePad(clock, command, attention, data, Pressures?, Rumble?) check for error
  error = ps2x.config_gamepad(PS2_CLK, PS2_CMD, PS2_SEL, PS2_DAT, pressures, rumble);

  if(error == 0){
    Serial.print("Found Controller, configured successful ");
    Serial.print("pressures = ");
    if (pressures)
      Serial.println("true ");
    else
      Serial.println("false");
    Serial.print("rumble = ");
    if (rumble)
      Serial.println("true)");
    else
      Serial.println("false");
    Serial.println("Try out all the buttons, X will vibrate the controller, faster as you press harder;");
    Serial.println("holding L1 or R1 will print out the analog stick values.");
    Serial.println("Note: Go to www.billporter.info for updates and to report bugs.");
  }
  else if(error == 1)
    Serial.println("No controller found, check wiring, see readme.txt to enable debug. visit www.billporter.info for troubleshooting tips");

  else if(error == 2)
    Serial.println("Controller found but not accepting commands. see readme.txt to enable debug. Visit www.billporter.info for troubleshooting tips");

  else if(error == 3)
    Serial.println("Controller refusing to enter Pressures mode, may not support it. ");

  type = ps2x.readType();
```

```
  switch(type) {
    case 0:
      Serial.println("Unknown Controller type found ");
      break;
    case 1:
      Serial.println("DualShock Controller found ");
      break;
    case 2:
      Serial.println("GuitarHero Controller found ");
      break;
    case 3:
      Serial.println("Wireless Sony DualShock Controller found ");
      break;
   }
}

void loop() {
  /* You must Read Gamepad to get new values and set vibration values
     ps2x.read_gamepad(small motor on/off, larger motor strenght from 0-255)
     if you don't enable the rumble, use ps2x.read_gamepad(); with no values
     You should call this at least once a second
   */
  if(error == 1){ //skip loop if no controller found
    resetFunc();
  }
  if(type == 2){ //Guitar Hero Controller
    ps2x.read_gamepad();          //read controller

    if(ps2x.ButtonPressed(GREEN_FRET))
      Serial.println("Green Fret Pressed");
    if(ps2x.ButtonPressed(RED_FRET))
      Serial.println("Red Fret Pressed");
    if(ps2x.ButtonPressed(YELLOW_FRET))
      Serial.println("Yellow Fret Pressed");
    if(ps2x.ButtonPressed(BLUE_FRET))
      Serial.println("Blue Fret Pressed");
    if(ps2x.ButtonPressed(ORANGE_FRET))
      Serial.println("Orange Fret Pressed");
    if(ps2x.ButtonPressed(STAR_POWER))
      Serial.println("Star Power Command");
```

```
    if(ps2x.Button(UP_STRUM))         //will be TRUE as long as button is pressed
      Serial.println("Up Strum");
    if(ps2x.Button(DOWN_STRUM))
      Serial.println("DOWN Strum");
    if(ps2x.Button(PSB_START))        //will be TRUE as long as button is pressed
      Serial.println("Start is being held");
    if(ps2x.Button(PSB_SELECT))
      Serial.println("Select is being held");
    if(ps2x.Button(ORANGE_FRET)) {     // print stick value IF TRUE
      Serial.print("Wammy Bar Position:");
      Serial.println(ps2x.Analog(WHAMMY_BAR), DEC);
    }
  }
  else { //DualShock Controller
    ps2x.read_gamepad(false, vibrate); //read controller and set large motor to spin at
'vibrate' speed
    if(ps2x.Button(PSB_START))        //will be TRUE as long as button is pressed
      Serial.println("Start is being held");
    if(ps2x.Button(PSB_SELECT))
      Serial.println("Select is being held");

    if(ps2x.Button(PSB_PAD_UP)) {      //will be TRUE as long as button is pressed
      Serial.print("Up held this hard: ");
      Serial.println(ps2x.Analog(PSAB_PAD_UP), DEC);
    }
    if(ps2x.Button(PSB_PAD_RIGHT)){
      Serial.print("Right held this hard: ");
      Serial.println(ps2x.Analog(PSAB_PAD_RIGHT), DEC);
    }
    if(ps2x.Button(PSB_PAD_LEFT)){
      Serial.print("LEFT held this hard: ");
      Serial.println(ps2x.Analog(PSAB_PAD_LEFT), DEC);
    }
    if(ps2x.Button(PSB_PAD_DOWN)){
      Serial.print("DOWN held this hard: ");
      Serial.println(ps2x.Analog(PSAB_PAD_DOWN), DEC);
    }
```

```
    vibrate = ps2x.Analog(PSAB_CROSS);  //this will set the large motor vibrate speed
based on how hard you press the blue (X) button
    if (ps2x.NewButtonState()) {       //will be TRUE if any button changes state (on to
off, or off to on)
      if(ps2x.Button(PSB_L3))
        Serial.println("L3 pressed");
      if(ps2x.Button(PSB_R3))
        Serial.println("R3 pressed");
      if(ps2x.Button(PSB_L2))
        Serial.println("L2 pressed");
      if(ps2x.Button(PSB_R2))
        Serial.println("R2 pressed");
      if(ps2x.Button(PSB_TRIANGLE))
        Serial.println("Triangle pressed");
    }

    if(ps2x.ButtonPressed(PSB_CIRCLE))              //will be TRUE if button was JUST
pressed
      Serial.println("Circle just pressed");
    if(ps2x.NewButtonState(PSB_CROSS))              //will be TRUE if button was JUST
pressed OR released
      Serial.println("X just changed");
    if(ps2x.ButtonReleased(PSB_SQUARE))             //will be TRUE if button was JUST
released
      Serial.println("Square just released");

    if(ps2x.Button(PSB_L1) || ps2x.Button(PSB_R1)) { //print stick values if either is
TRUE
      Serial.print("Stick Values:");
      Serial.print(ps2x.Analog(PSS_LY), DEC); //Left stick, Y axis. Other options: LX,
RY, RX
      Serial.print(",");
      Serial.print(ps2x.Analog(PSS_LX), DEC);
      Serial.print(",");
      Serial.print(ps2x.Analog(PSS_RY), DEC);
      Serial.print(",");
      Serial.println(ps2x.Analog(PSS_RX), DEC);
    }
  }
  delay(50);
}
```

In the above program, we instructed to read the test button. In this experiment we want to implement the PS2 remote control car function. We firstly define all the button functions as follows:

图3.2.34　PS2手柄功能按键示意图

Mark UP: move forward

Mark DOWN: move backward

Mark LEFT: turn left

Mark RIGHT: right

Mark A: speed up

Mark B: Left rotation

Mark C: slow down

Mark D: Right rotation

Ps2 Control program in Lesson \Comprehensive Experiment\Beetle_PS2\Beetle_PS2\ Beetle_PS2.ino

The program flow chart is as follows: