Getting started

Thank you for choosing Freenove products!
First, read the document **About_Battery.pdf** in the unzipped folder.
If you did not download the zip file, please download it and unzip it via link below.
https://github.com/Freenove/Freenove_4WD_Car_Kit/archive/master.zip

Get Support and Offer Input

Freenove provides free and responsive product and technical support, including but not limited to:

- Product quality issues
- Product use and build issues
- Questions regarding the technology employed in our products for learning and education
- Your input and opinions are always welcome
- We also encourage your ideas and suggestions for new products and product improvements

For any of the above, you may send us an email to:

support@freenove.com

Safety and Precautions

Please follow the following safety precautions when using or storing this product:

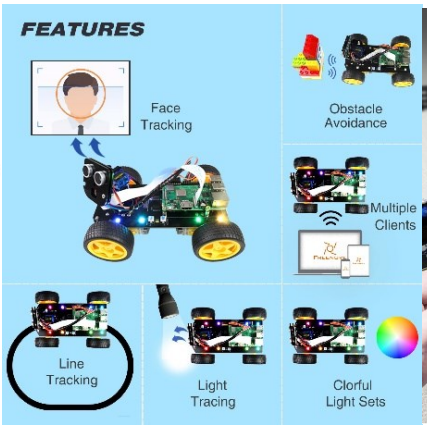
- Keep this product out of the reach of children under 6 years old.
- This product should be used only when there is adult supervision present as young children lack necessary judgment regarding safety and the consequences of product misuse.
- This product contains small parts and parts, which are sharp. This product contains electrically conductive parts. Use caution with electrically conductive parts near or around power supplies, batteries and powered (live) circuits.
- When the product is turned ON, activated or tested, some parts will move or rotate. To avoid injuries to hands and fingers, keep them away from any moving parts!
- It is possible that an improperly connected or shorted circuit may cause overheating. Should this happen, immediately disconnect the power supply or remove the batteries and do not touch anything until it cools down! When everything is safe and cool, review the product tutorial to identify the cause.
- Only operate the product in accordance with the instructions and guidelines of this tutorial, otherwise parts may be damaged or you could be injured.
- Store the product in a cool dry place and avoid exposing the product to direct sunlight.
- After use, always turn the power OFF and remove or unplug the batteries before storing.

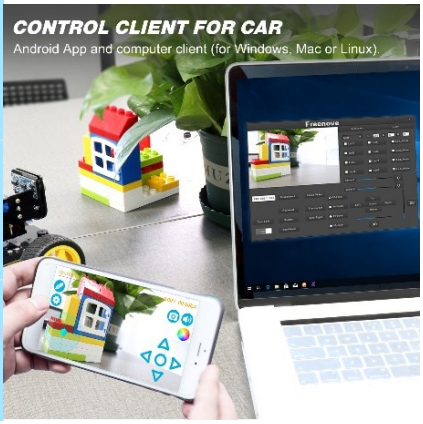
Car and Robot for Raspberry Pi

We also have cars and robot kit for Raspberry Pi. If you are interesting in them, please visit our website for details.

http://www.freenove.com/store.html

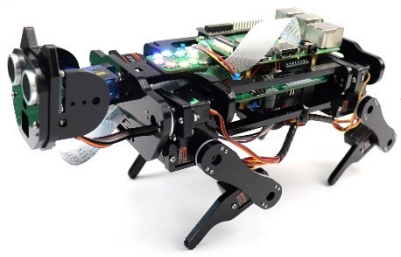
FNK0043 Freenove 4WD Smart Car Kit for Raspberry Pi

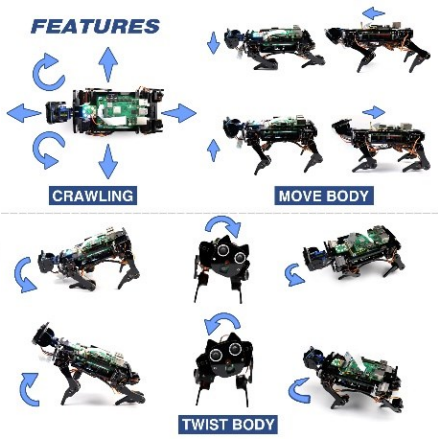


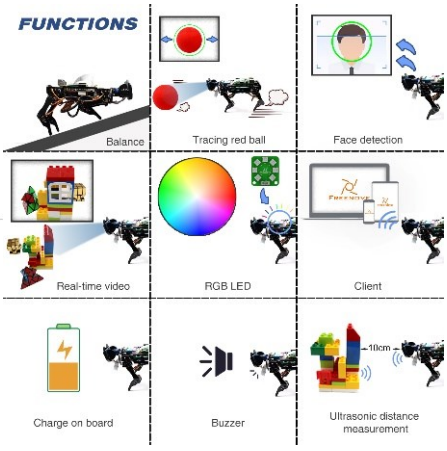


https://www.youtube.com/watch?v=4Zv0GZUQjZc

FNK0050 Freenove Robot Dog Kit for Raspberry Pi







https://www.youtube.com/watch?v=7BmlZ8_R9d4&t=35s

About Freenove

Freenove provides open source electronic products and services worldwide.

Freenove is committed to assist customers in their education of robotics, programming and electronic circuits so that they may transform their creative ideas into prototypes and new and innovative products. To this end, our services include but are not limited to:

- Educational and Entertaining Project Kits for Robots, Smart Cars and Drones
- Educational Kits to Learn Robotic Software Systems for Arduino, Raspberry Pi and micro: bit
- Electronic Component Assortments, Electronic Modules and Specialized Tools
- **Product Development and Customization Services**

You can find more about Freenove and get our latest news and updates through our website:

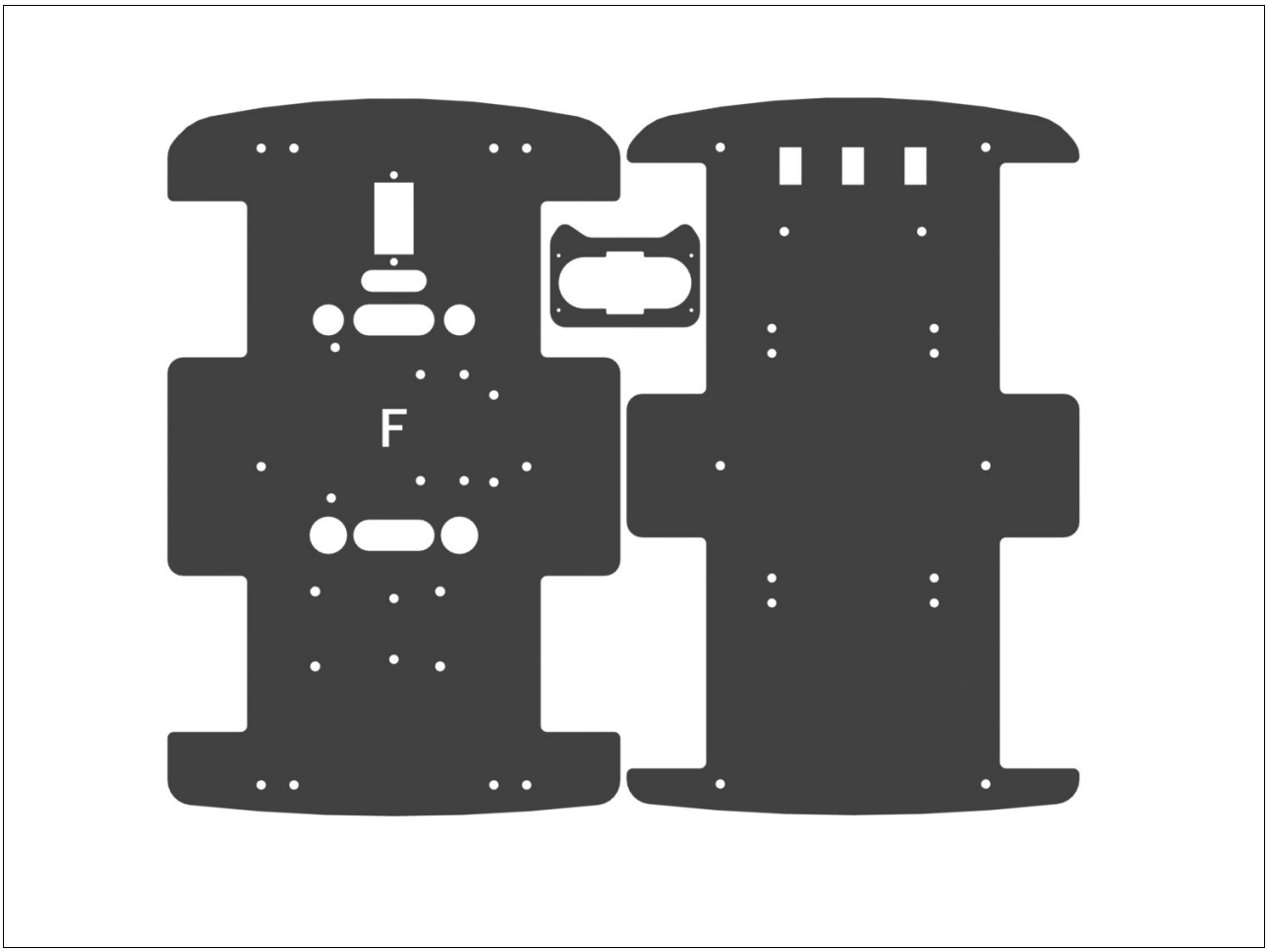
http://www.freenove.com

Copyright

Contents

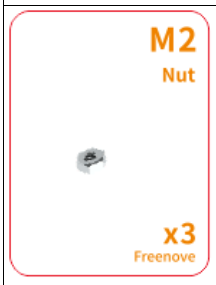
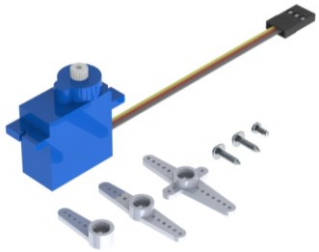
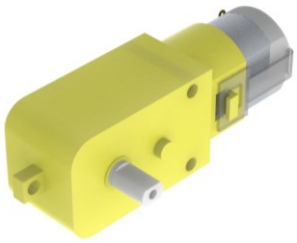
List

Acrylic Parts



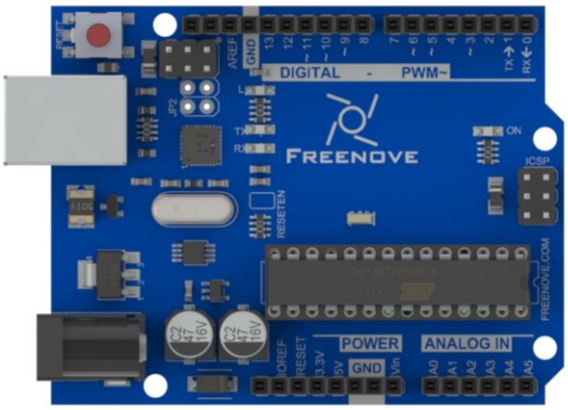
The surface of the acrylic parts is covered with a layer of protective film. You need to remove it first. Some holes in the acrylic parts may have residues. You also need to clean them before using.

Mechanical Parts

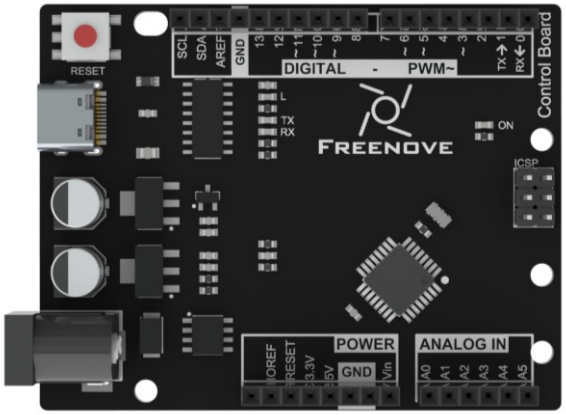
<table>
<tr><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td> 
</td><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Servo x1
  </td><td>Motor x4
  </td></tr>
<tr><td colspan="2">Driver wheel x 4
  </td><td>Motor bracket package x 4
Screw M3*8 , Nut M3, Screw M3*30
  </td></tr>
</table>

Electronic Parts

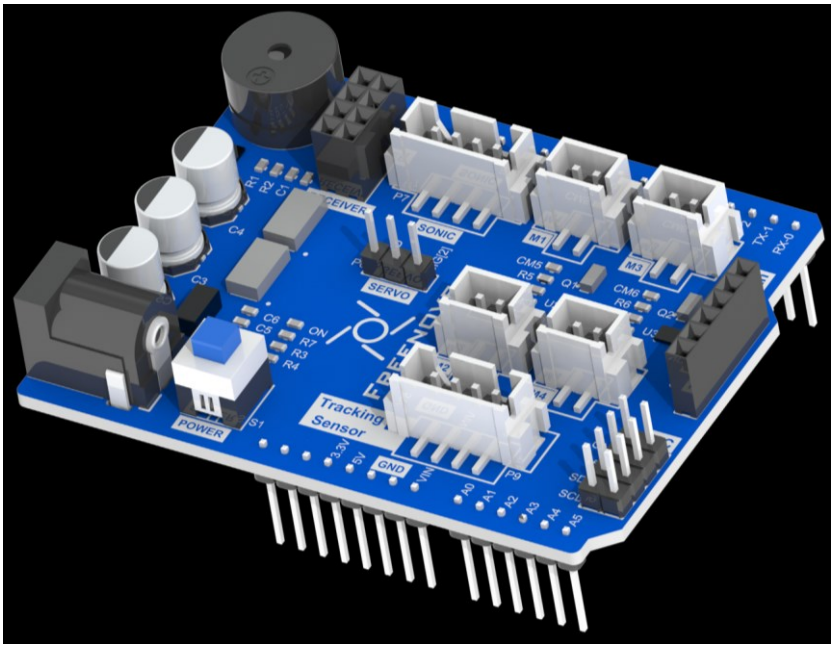
Freenove control board (Your kit will randomly contain **one control board**)



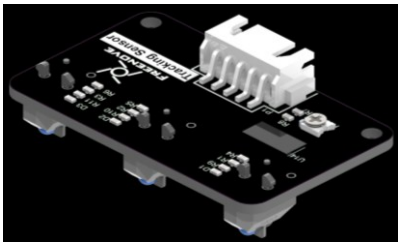
Or



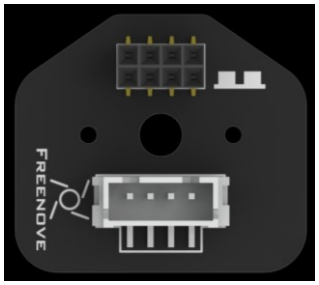
Freenove 4WD extension board

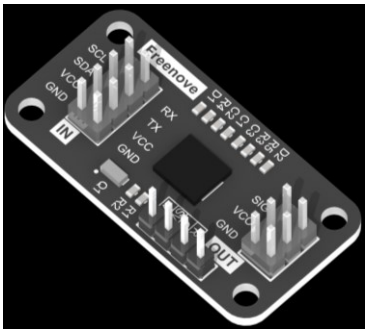
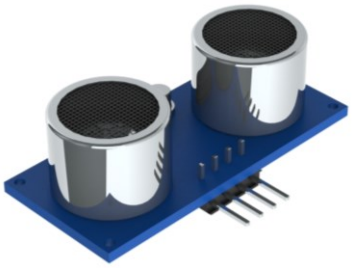
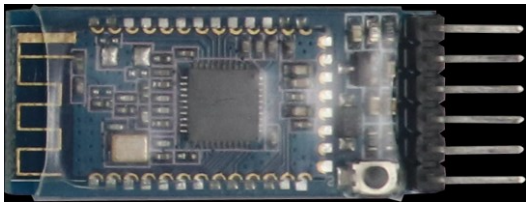
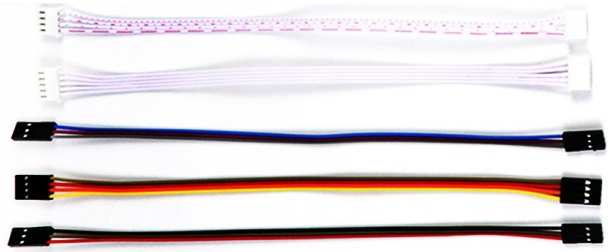
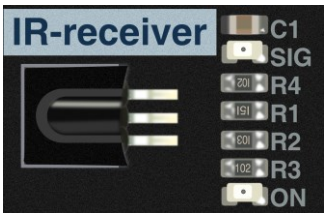
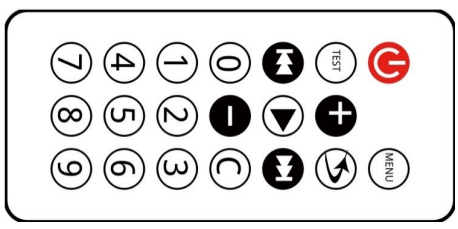


Line-tracking infrared sensor



Ultrasonic module connector



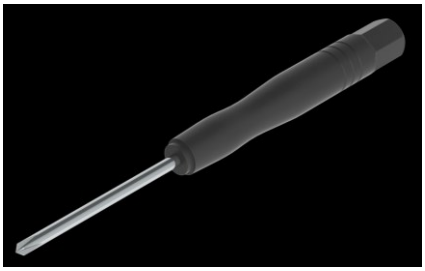
WS2812B_LED_controller 	WS2812B_LED 
Ultrasonic module 	Bluetooth 
Wires 	USB Cable 
IRreceiver 	Tape 
IR remote 	Battery holder 

RF remote kit package (**only contained in the Version with RF remote control**)

The remote kit also has two versions, **black** or **blue**. You will receive one of them randomly.

Tools

Cross Screwdriver x1



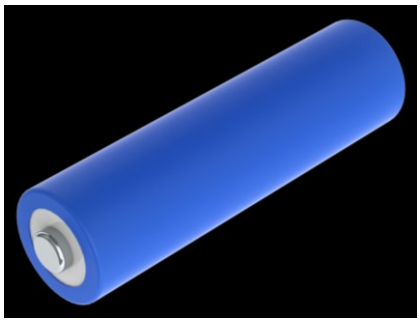
Socket x1 (**only one is included in the kit**)

Needed but NOT Included

18650 3.7V **rechargeable** lithium battery x2

It is easier to find proper battery on eBay than Amazon.

Please refer to **About_Battery.pdf** in the unzipped folder.



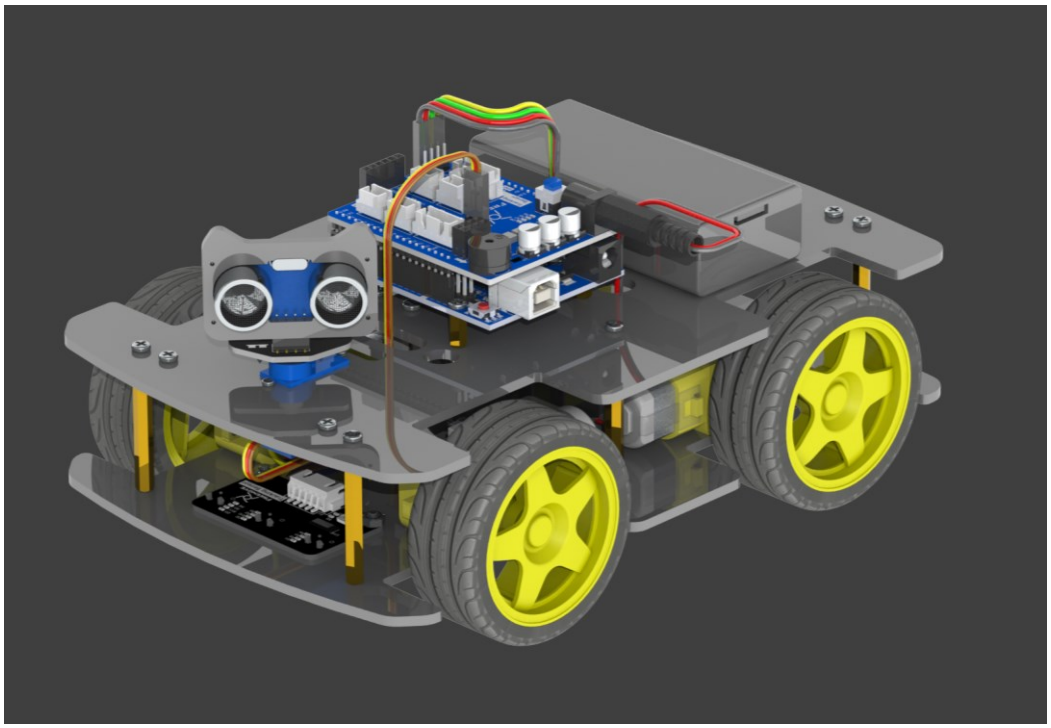
Preface

This 4WD Smart car is based on Arduino.

This tutorial will start from controlling electronic components such as LED, motor, servo, to building multifunctional car.
Please follow the tutorial step by step patiently.

If you have any concerns, please feel free to contact us. Our support team will provide quick and free customer service.

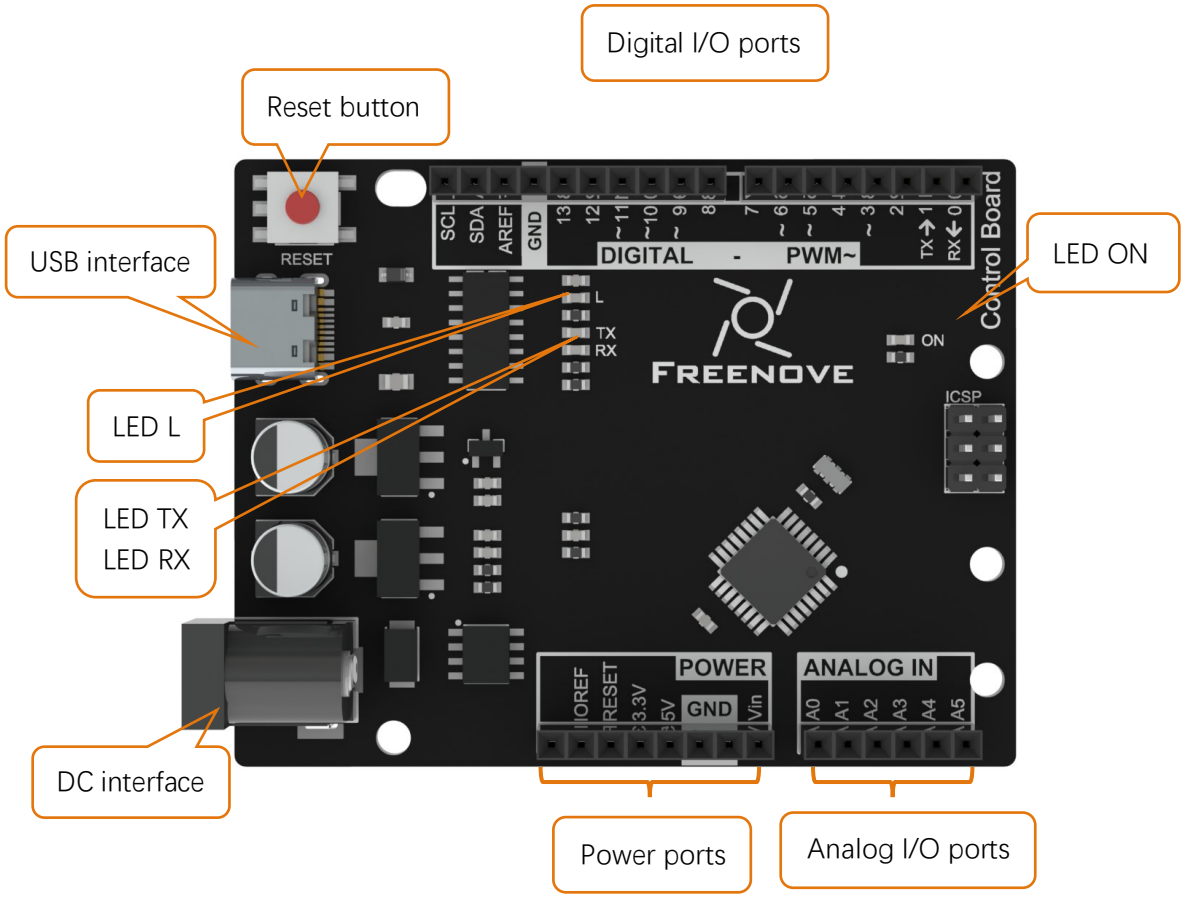
support@freenove.com



Control Board and Software

Control Board

Diagram of Freenove Control board is shown below:

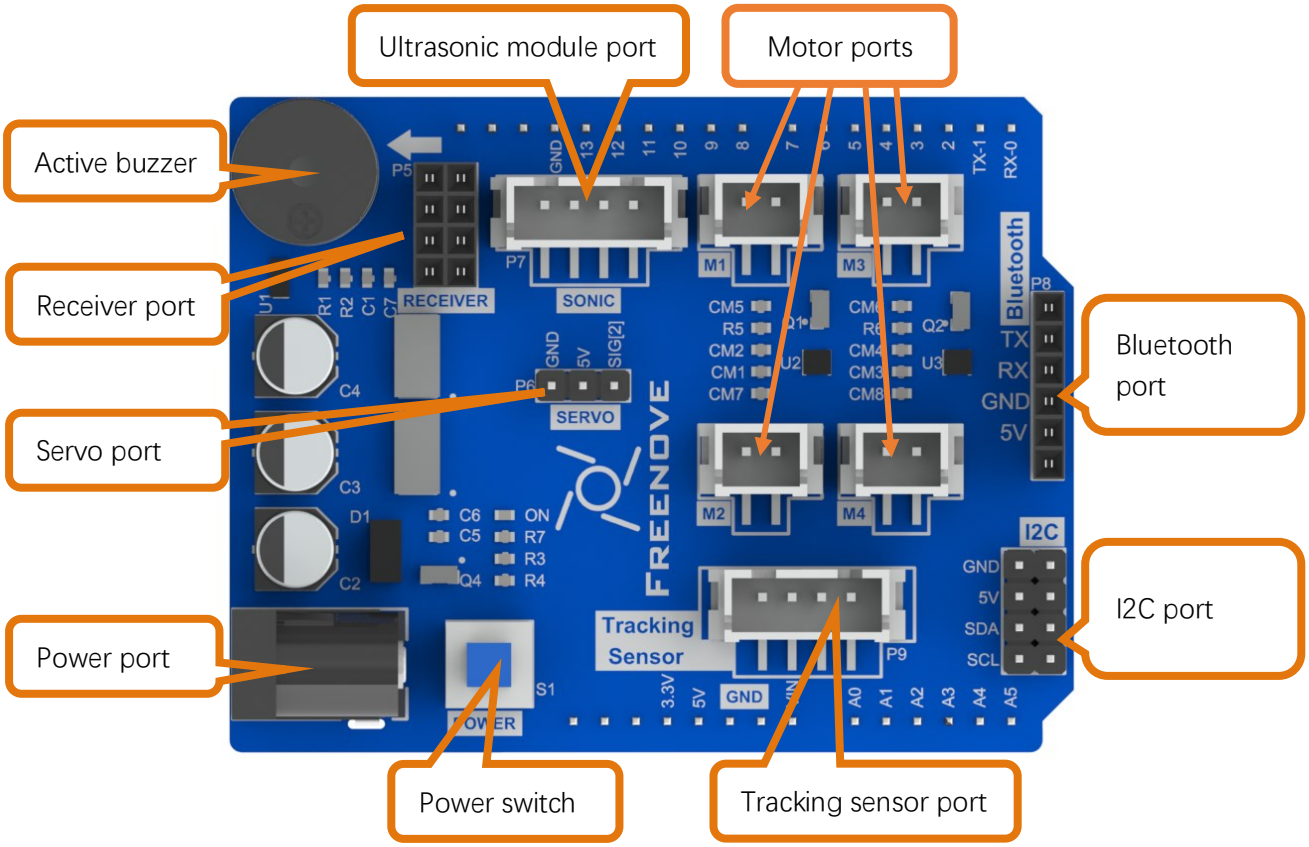


- Digital I/O ports is used to connect to other components or modules, to receive an input signal, or to send a control signal. Usually, we name it by adding a "D" in front of the number, such as D13.
- Only digital I/O ports with "~" can output PWM, Pin 3, 5, 6, 9, 10, 11.
- USB interface is used to provide power, upload code or communicate with PC.
- LED L is connected to digital I/O port 13 (D13).
- LED TX, RX is used to indicate the state of the serial communication.
- DC interface is connected with DC power to provide power for the board.
- Power ports can provide power for electronic components and modules.
- Analog I/O ports can be used to measure analog signals.
- LED ON is used to indicate the power state.

If you want to learn more about how to use Freenove control board to create interesting projects, please visit http://www.freenove.com/store.html for Arduino kits designed for beginners.

Extension Board

Freenove 4WD Extension board is below:



Receiver port can connect to RF module or IR module.

Relationship between extension board and the control board is as below.

Pins of Arduino	Ports of extension	Pin multiplexing	Description
0	UART-Bluetooth		Bluetooth
1			
2	Servo		
3	Direction-Right		Direction of right motor
4	Direction-Left		Direction of left motor
5	Motor-Right		Speed of right motor -PWM
6	Motor-Left		Speed of right motor -PWM
7	Trig		Ultrasonic module
8	Echo		
9	SPI-NRF24L01	IR-remote	CE/IR-remote
10			CS
11	SPI-NRF24L01		MOSI
12			MISO
13			SCK
A0	Battery voltage	Buzzer	The ADC uses an internal 5V reference and the acquisition voltage is the battery voltage *1/4. Output to the Buzzer through the comparator. When higher than the 4.5V, output HIGH.
A1	Line-tracking sensor		Left
A2			Middle
A3			Right
A4	I2C		SDA
A5			SCL

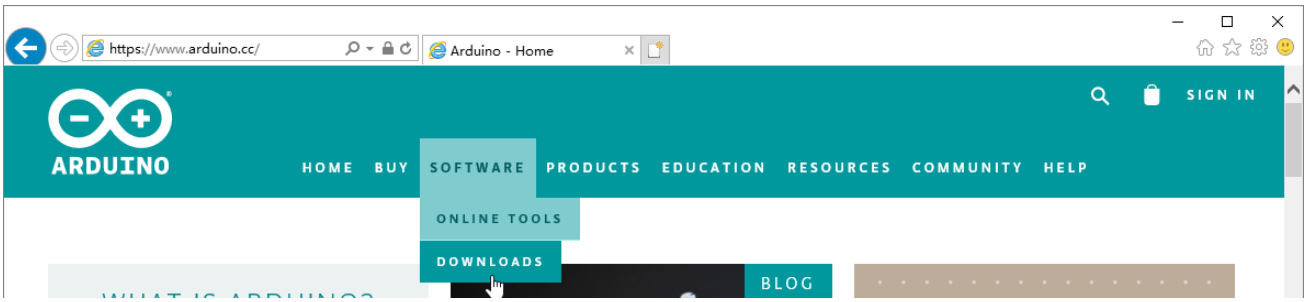
Chapter 0 Preparation, Assembly and Play

If you have any concerns, please feel free to contact us via support@freenove.com

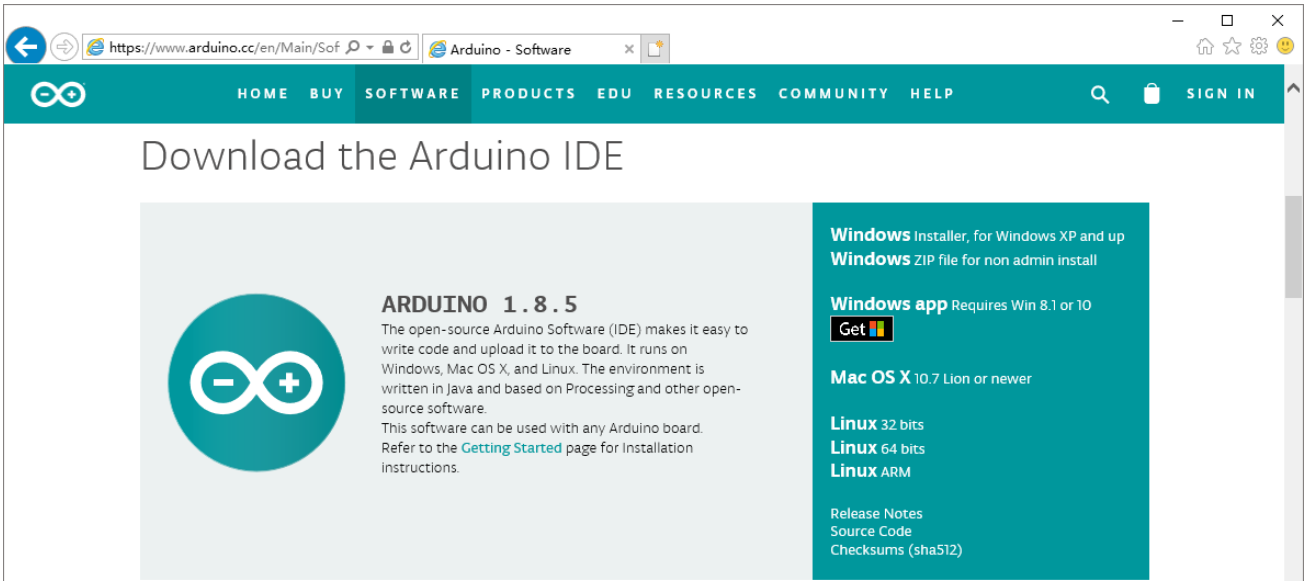
0.1 Preparation

Step 1 Download and Install IDE.

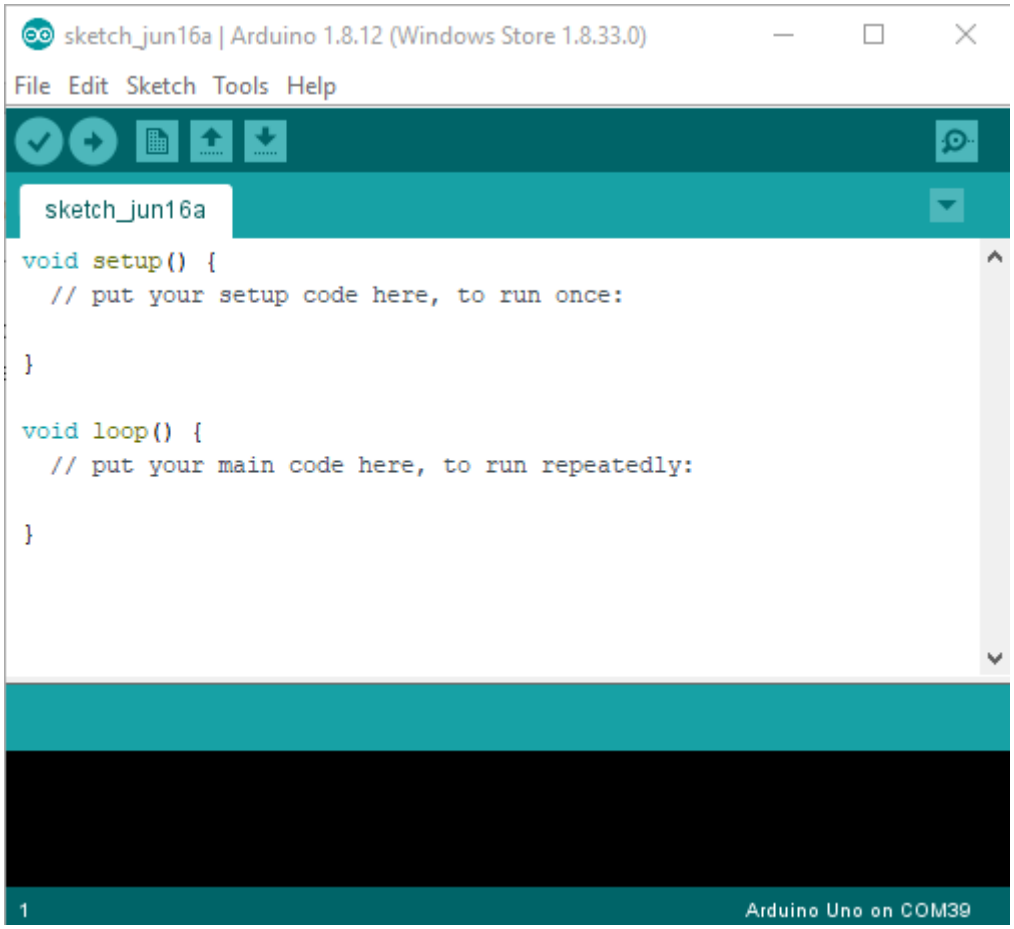
Please visit https://www.arduino.cc, then click "SOFTWARE" – "DOWNLOADS" to enter the downloads page.



Find "Download the Arduino IDE". Microsoft Windows users please click the "Windows Installer".



After the download completes, run the installer and complete the installation.
Open the Arduino Software, the interface of Arduino Software is as follows:



Programs written with Arduino Software (IDE) are called **sketches**. These sketches are written in the text editor and are saved with the file extension **.ino**. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and it also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom right corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

Verify, Checks your code for errors compiling it.

Upload, Compiles your code and uploads it to the configured board.

New, Creates a new sketch.

Open, Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.

Save, Saves your sketch.

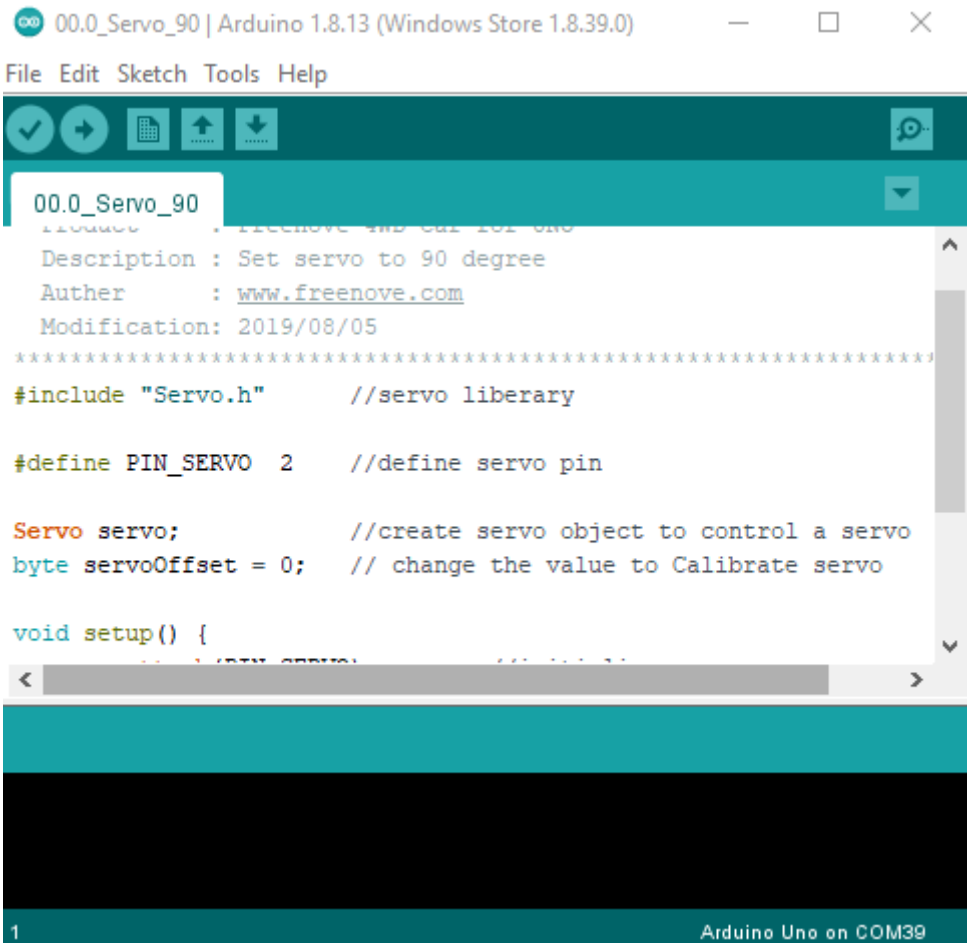
Serial Monitor, Opens the serial monitor.

Additional commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

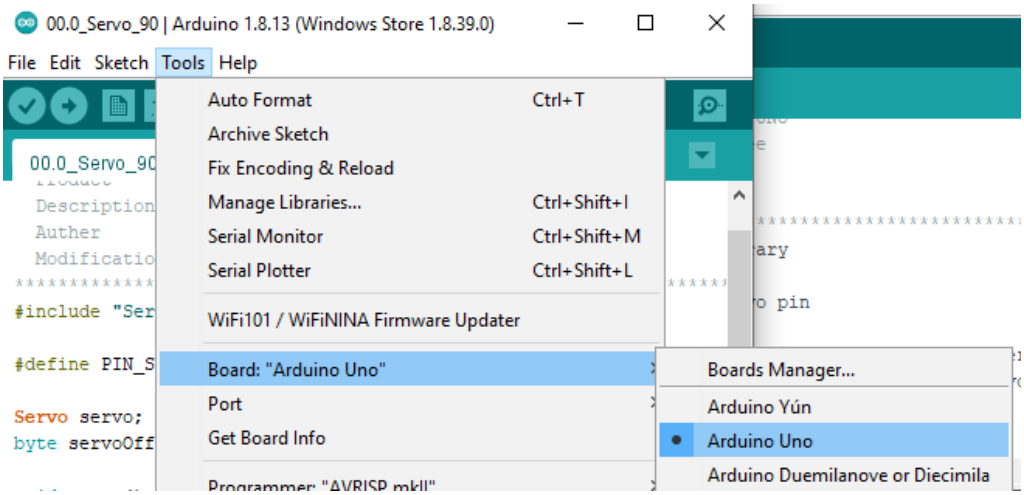
Step 2 Upload Servo Code (Necessary)

Upload code in **Freenove_4WD_Car_Kit\Sketches\00.0_Servo_90** to Freenove control board of the car.
This code will be used in assembly step. Do not skip this step

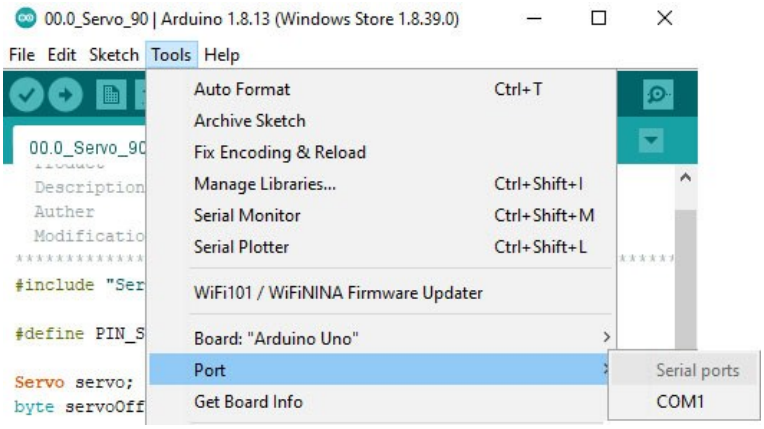
Open the example sketch "**00.0_Servo_90** " with Arduino IDE.



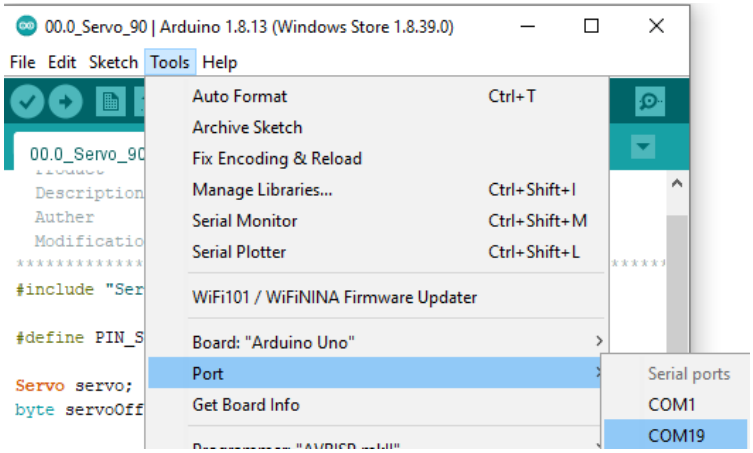
Select board "**Arduino/Genuino Uno**".



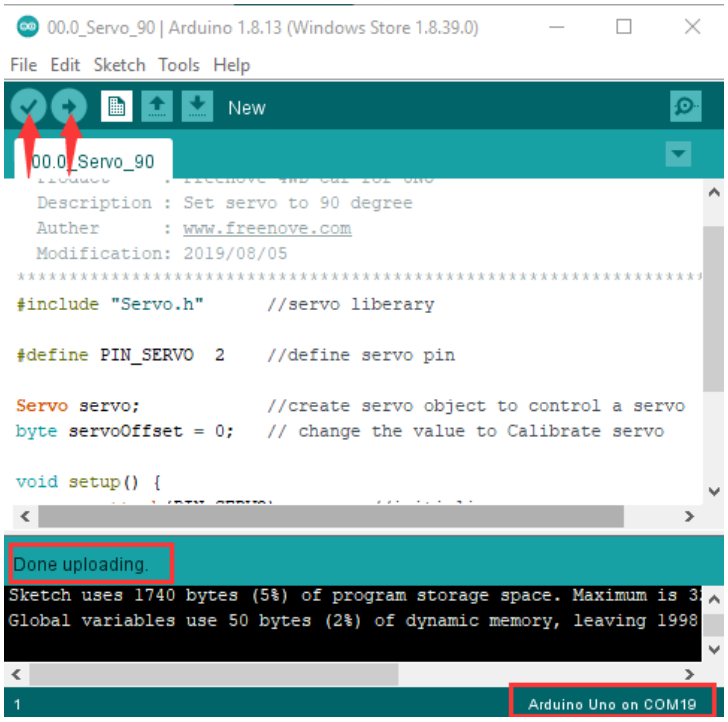
Check Port before connecting the control board to computer.



Connect the control board to computer via USB cable. And then check Port again. You will find a COM added. That is the port of control board. Here it is COM19. In your computer, it would be another number.



Click "Verify" button and “Upload” button. “Done uploading” indicates that the code is uploaded successfully.

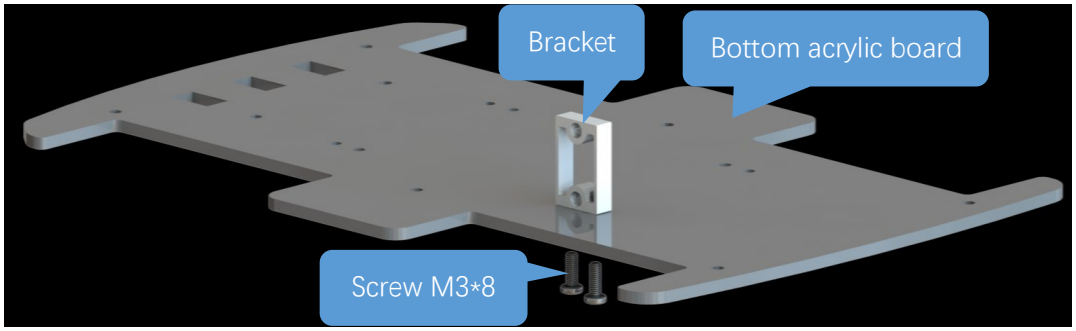

00.0_Servo_90 | Arduino 1.8.13 (Windows Store 1.8.39.0)
File Edit Sketch Tools Help
New
00.0_Servo_90
Description : Set servo to 90 degree
Auther : www.freenove.com
Modification: 2019/08/05
#include "Servo.h" //servo liberary
#define PIN_SERVO 2 //define servo pin
Servo servo; //create servo object to control a servo
byte servoOffset = 0; // change the value to Calibrate servo
void setup() {
Done uploading.
Sketch uses 1740 bytes (5%) of program storage space. Maximum is 3
Global variables use 50 bytes (2%) of dynamic memory, leaving 1998
1
Arduino Uno on COM19

0.2 Assembly

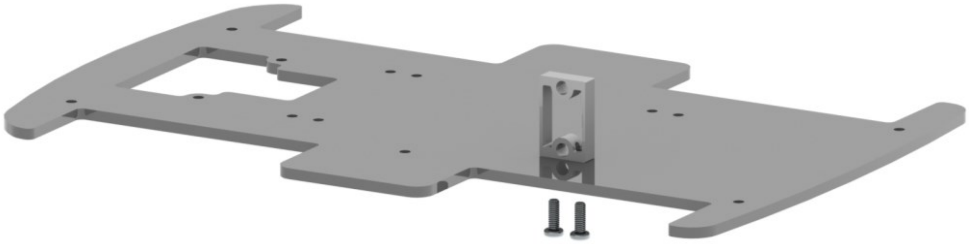
If you have any concerns, please feel free to contact us via support@freenove.com

In this chapter, we will learn how to assemble this car. This will spend some time. Please be patient.

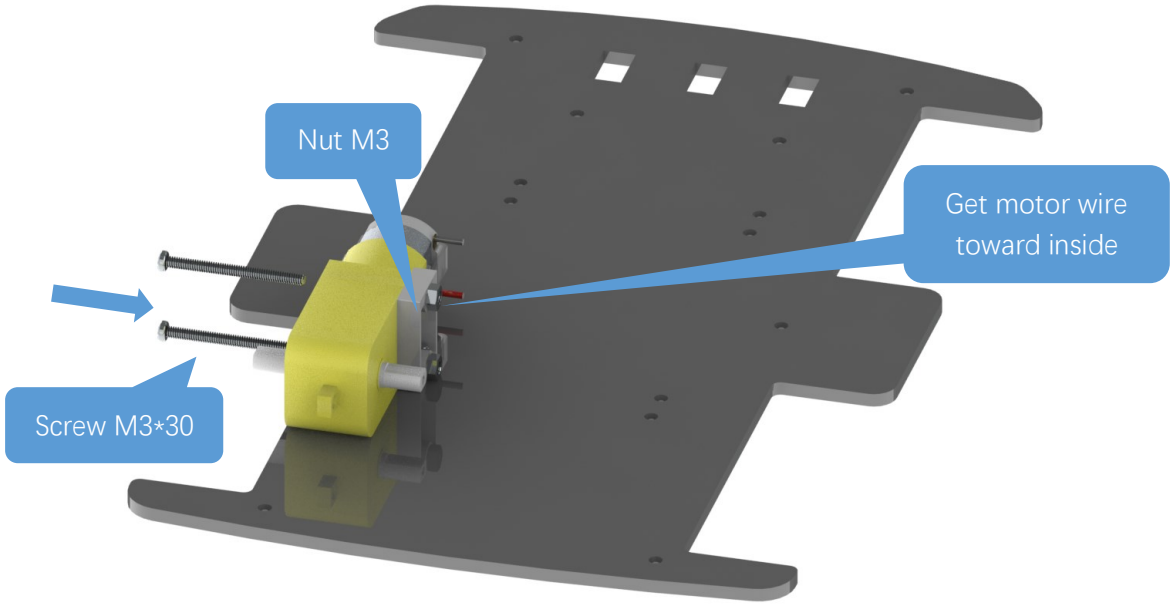
Install Motor Bracket on bottom acrylic board, with screw M3*8 **in the same bag**.



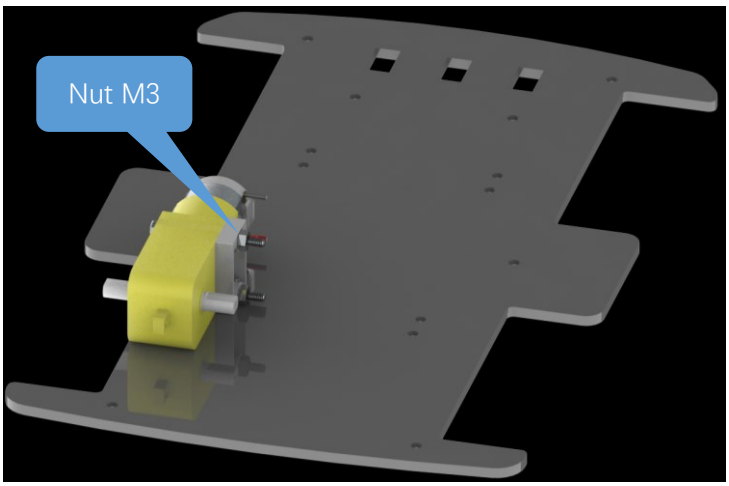
In your kit, the bottom acrylic board may look like below:



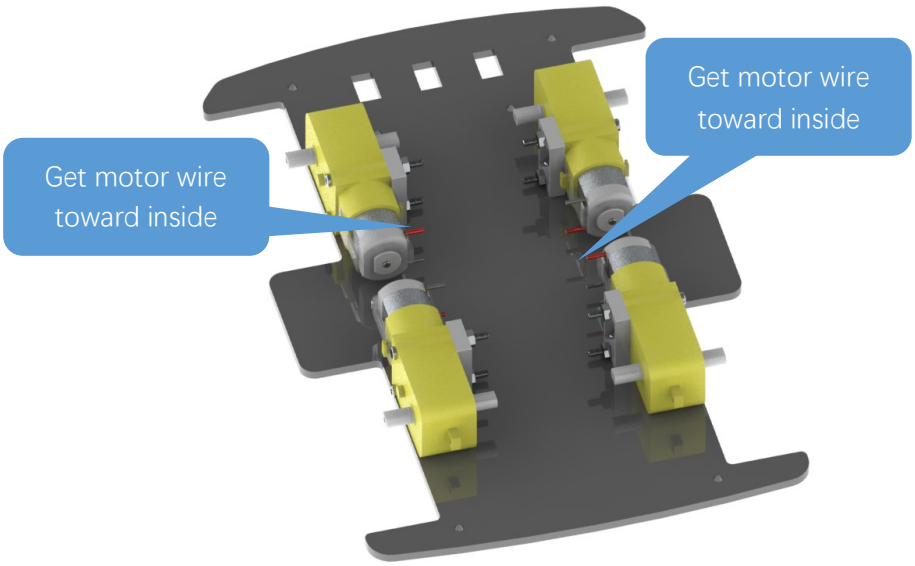
Install Motor Bracket on bottom acrylic board, with screw M3*30 and Nut M3 **in the same bag**.



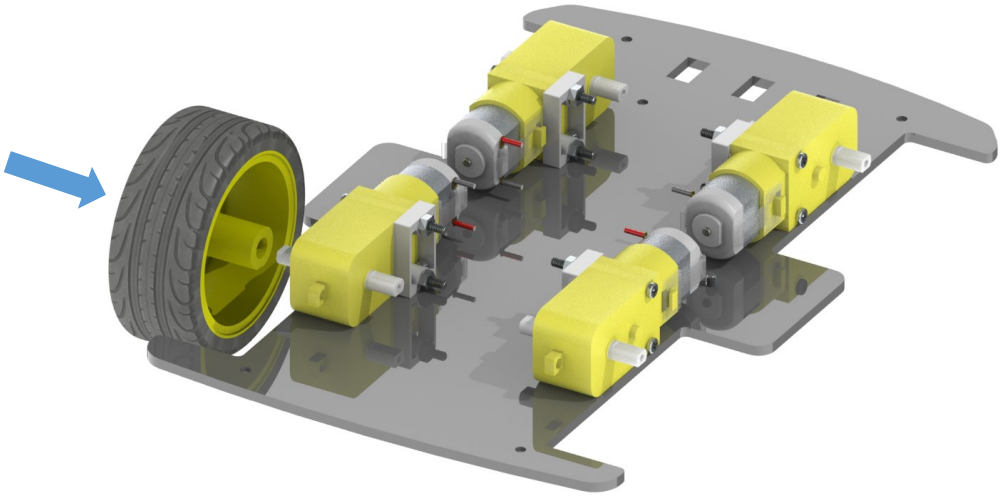
And then one motor will be installed successfully.



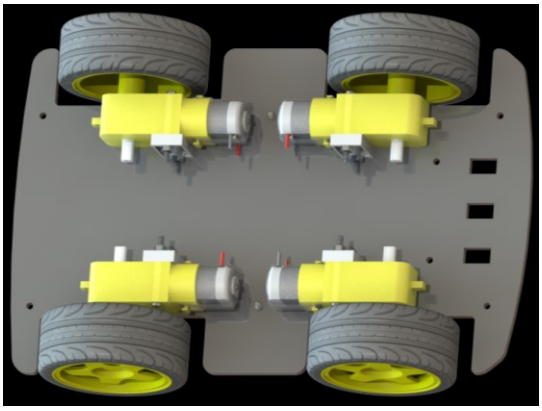
Install the rest 3 sets of motors with the same method.



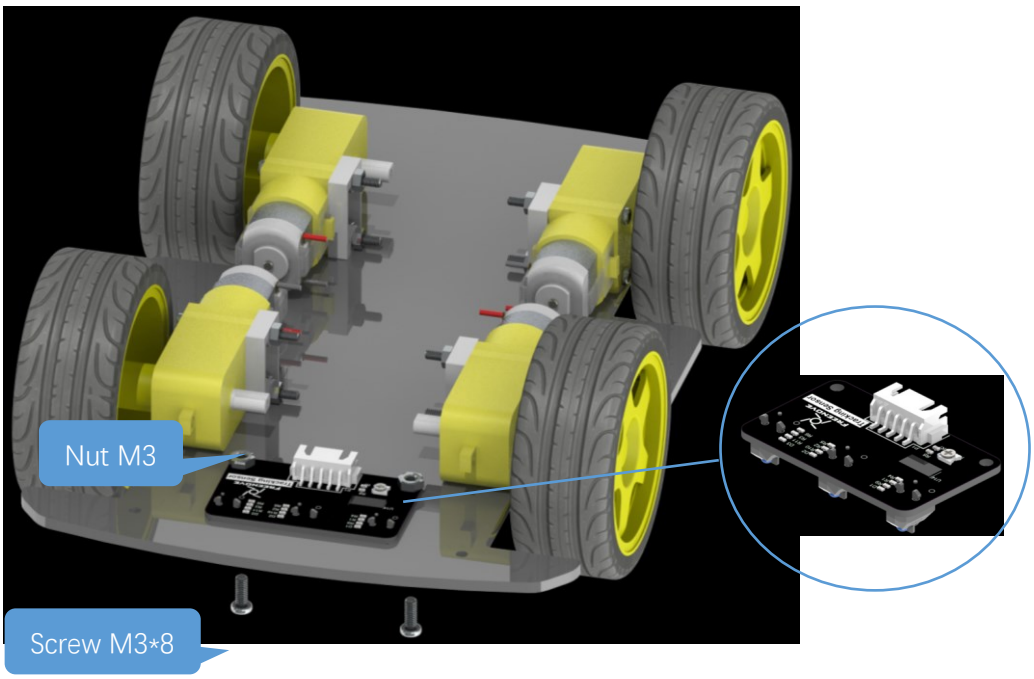
Install wheel. Note, the hole is not round.



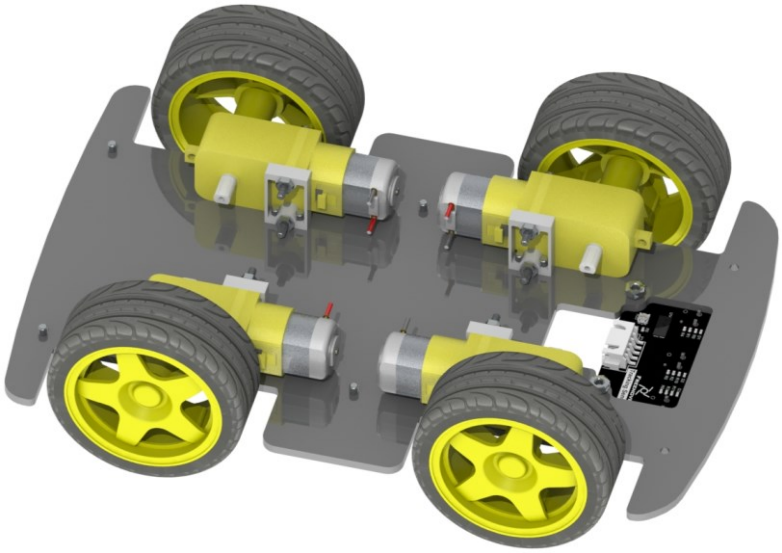
Install the rest 3 wheels.



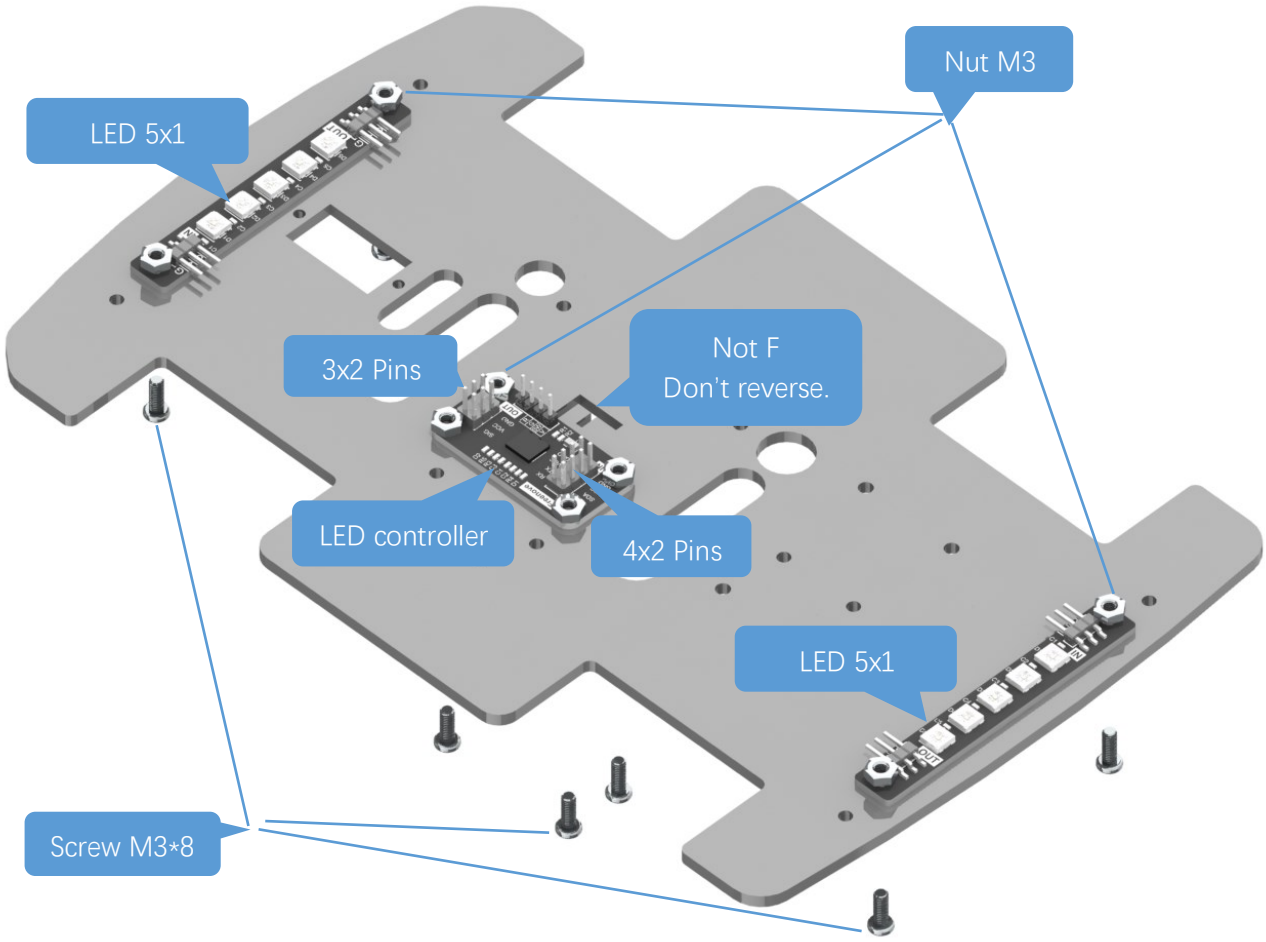
Install line tracking module with Screw M3*8 and Nut M3.



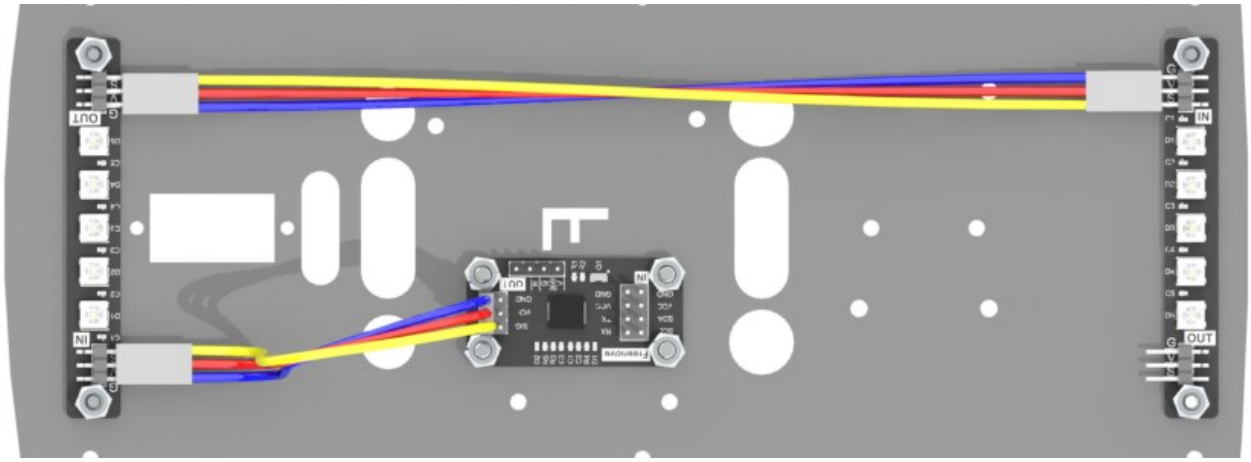
If you have other kind of bottom acrylic board, please install the tracking module like below.



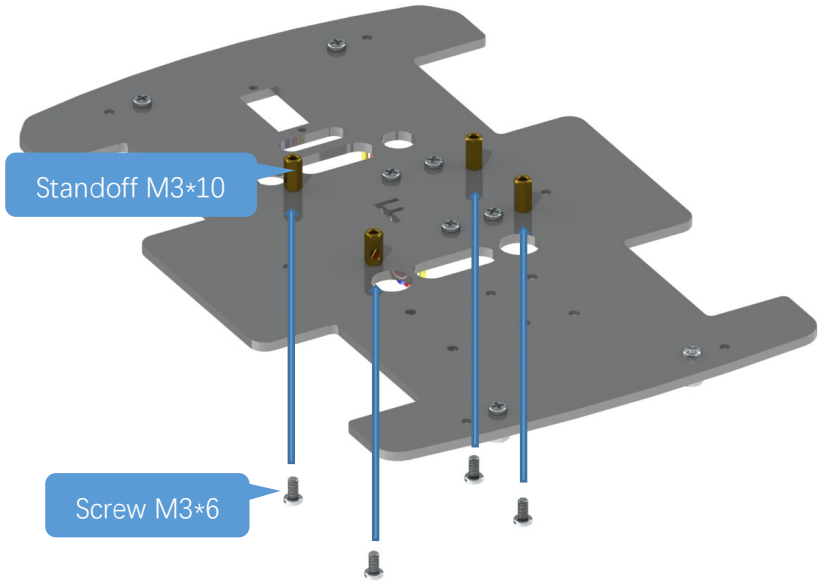
Install LED 5x1 and LED controller to top acrylic board.



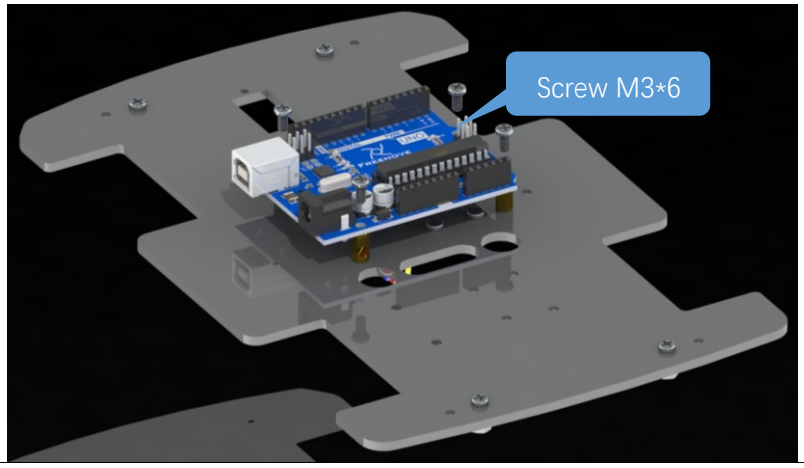
Connect wiring of LED controller and LED 5X1. The wire need to be twisted 180°.



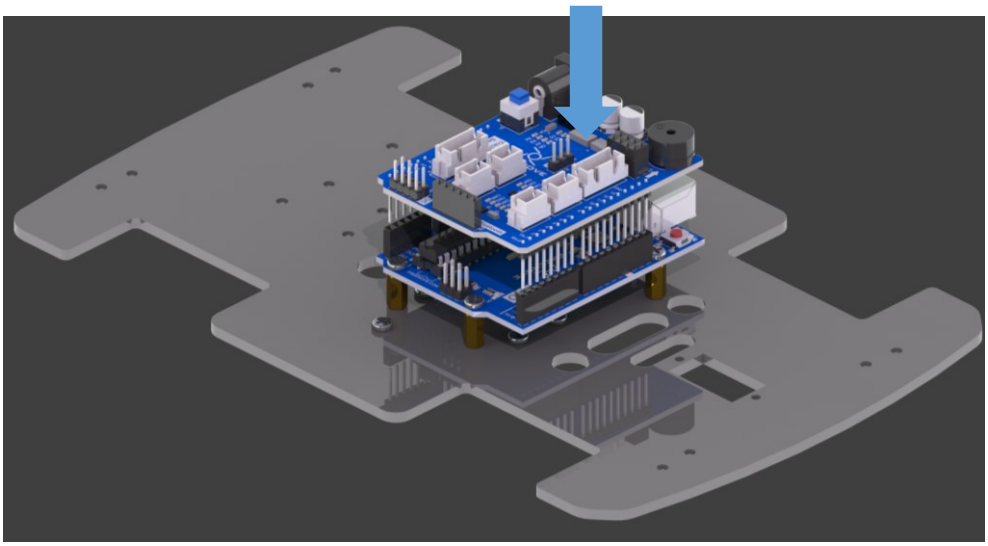
Install standoff on top acrylic board.



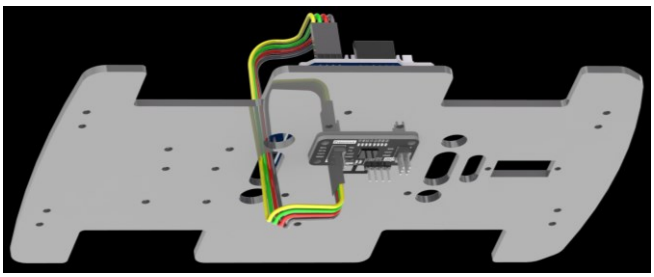
Install Freenove control board.

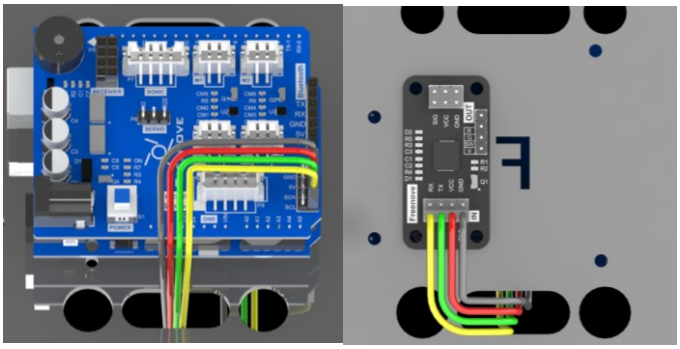


Plug extension into control board.

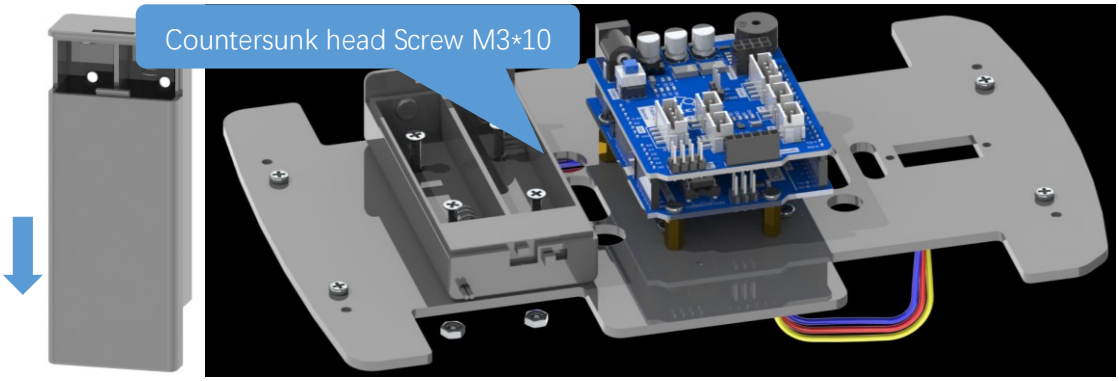


Connect LED controller with extension board with jumper wire F-F 4P. Wires of LED are hidden.
GND-GND, 5V-VCC, SCAL-RX, SDA-TX.

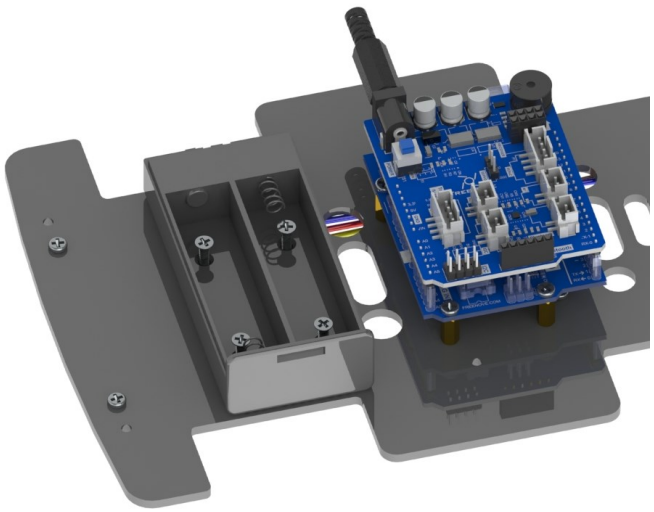




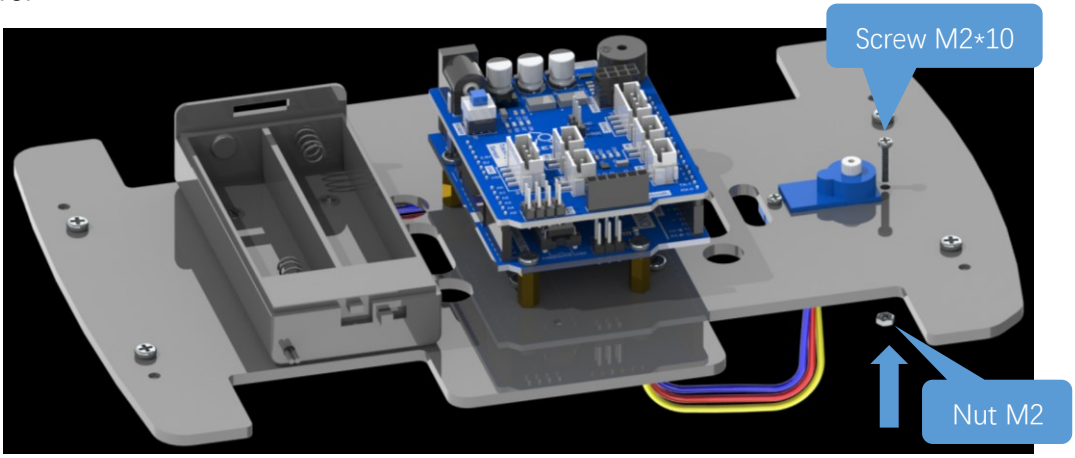
Install battery holder.



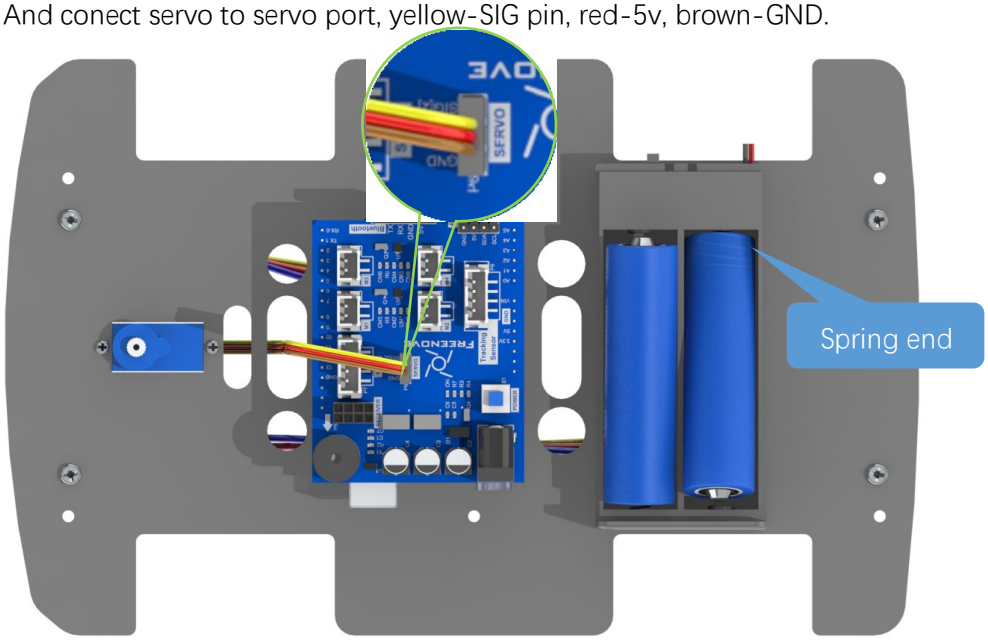
Some battery holders have short cable. Please rotate the battery holder for 180° to install.



Install servo.

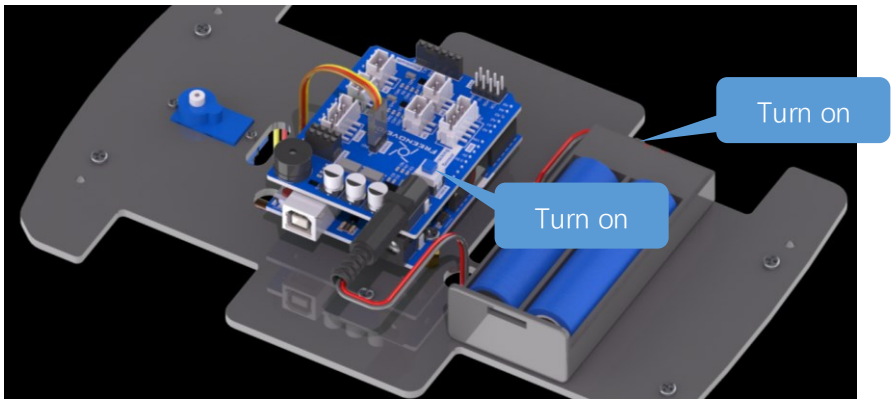


Install battery. And conect servo to servo port, yellow-SIG pin, red-5v, brown-GND.



18650 3.7V **rechargeable** lithium battery x2 **It is easier to find proper battery on eBay than Amazon.**

Connect power supply to extension board (the top board, not control board board).

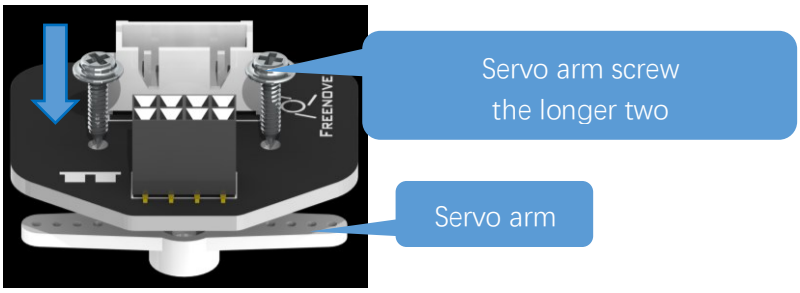


If you have uploaded code before, just turn on the switch of extension board and battery holder.

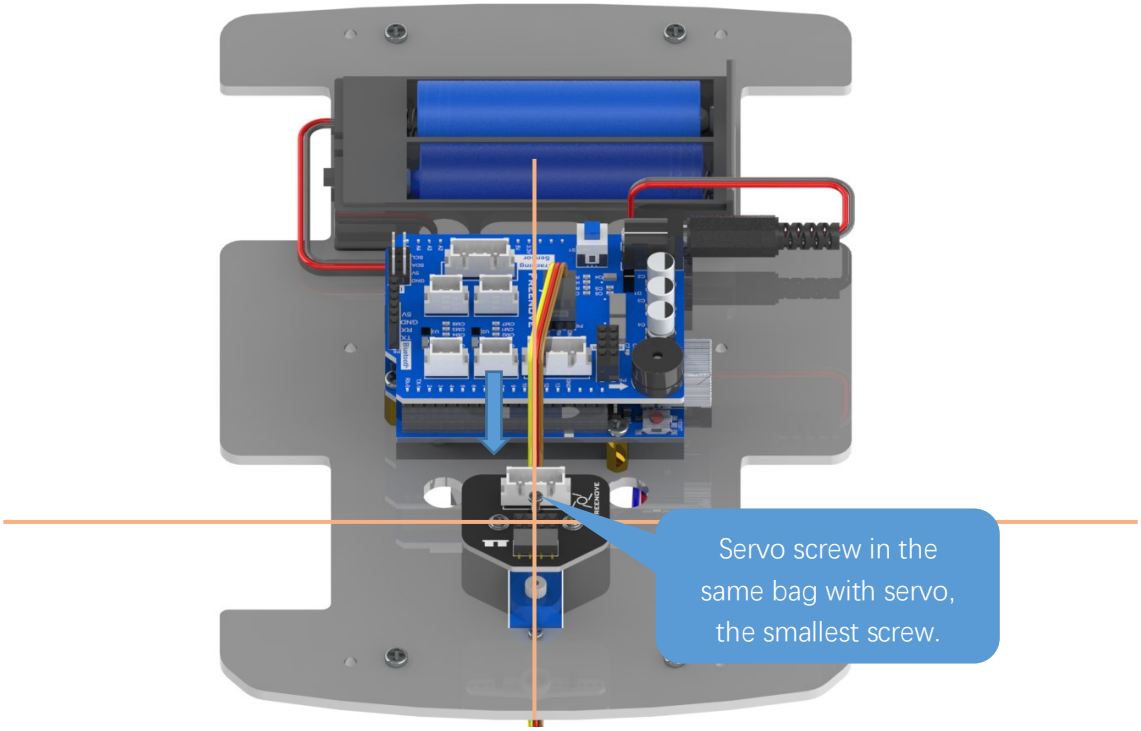
If not, you need upload the code.

Some battery holders have short cable. Please rotate the battery holder for 180° to install.

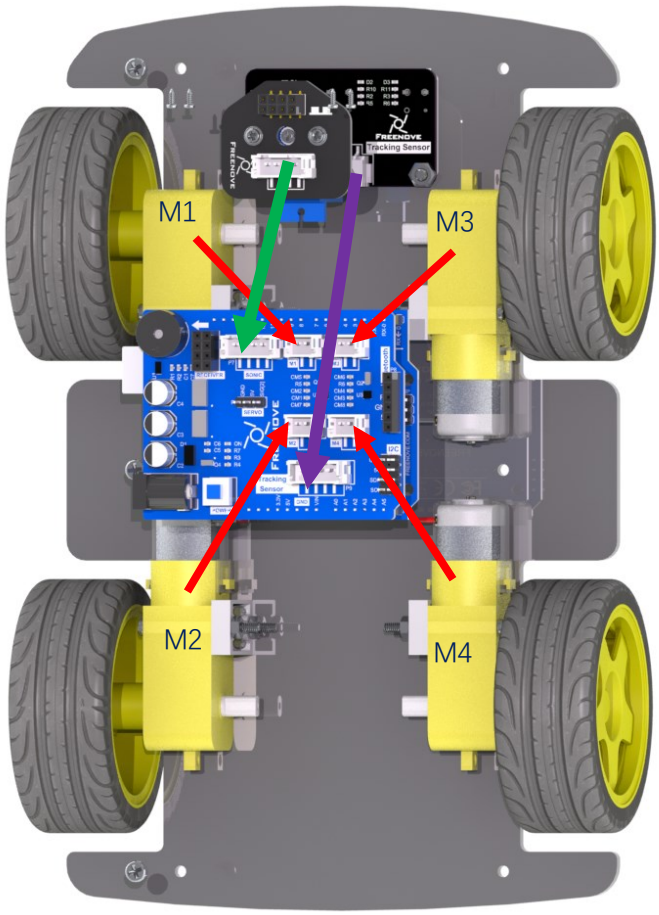
Install servo arm with sonic module connector. The screw and arm are in the same bag with servo.



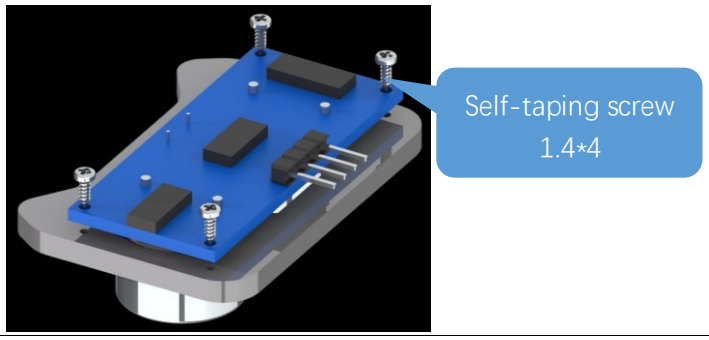
Connect servo arm to servo. Make sure servo arm is installed at 90 degrees.



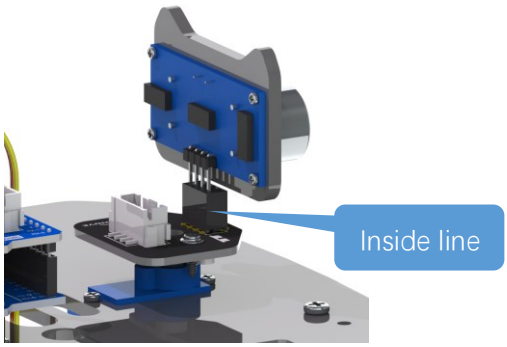
Connect different components to their corresponding ports as below.



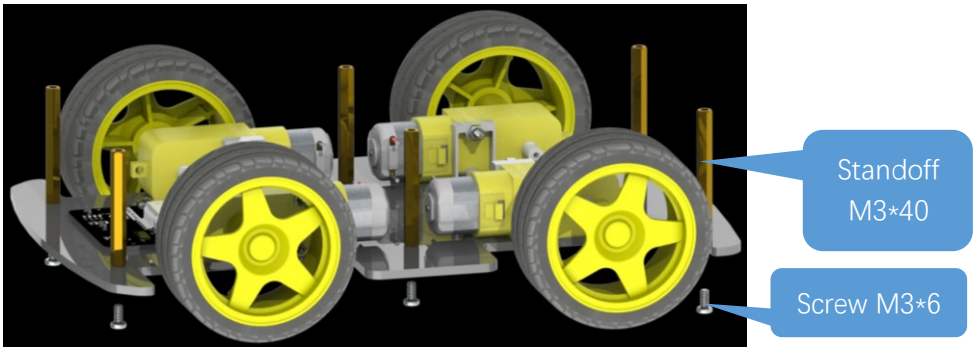
Install ultrasonic module to acrylic board.



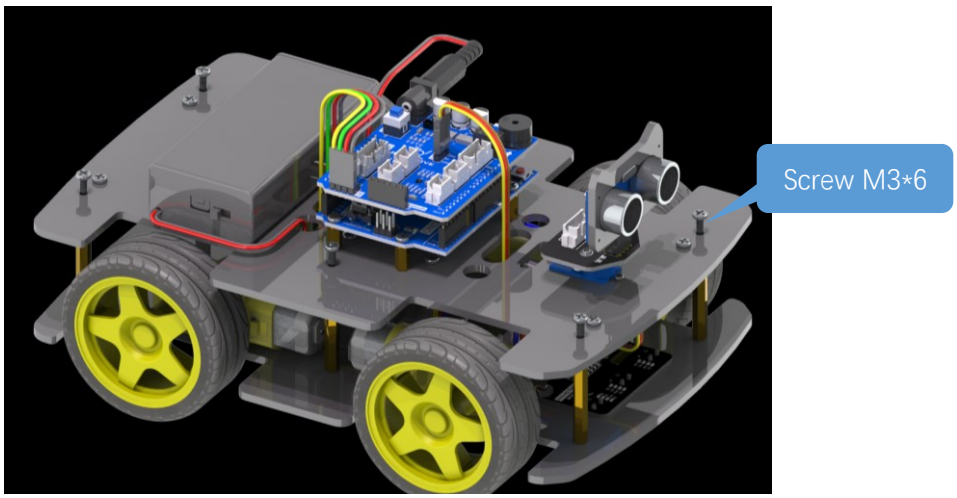
Plug ultrasonic module.



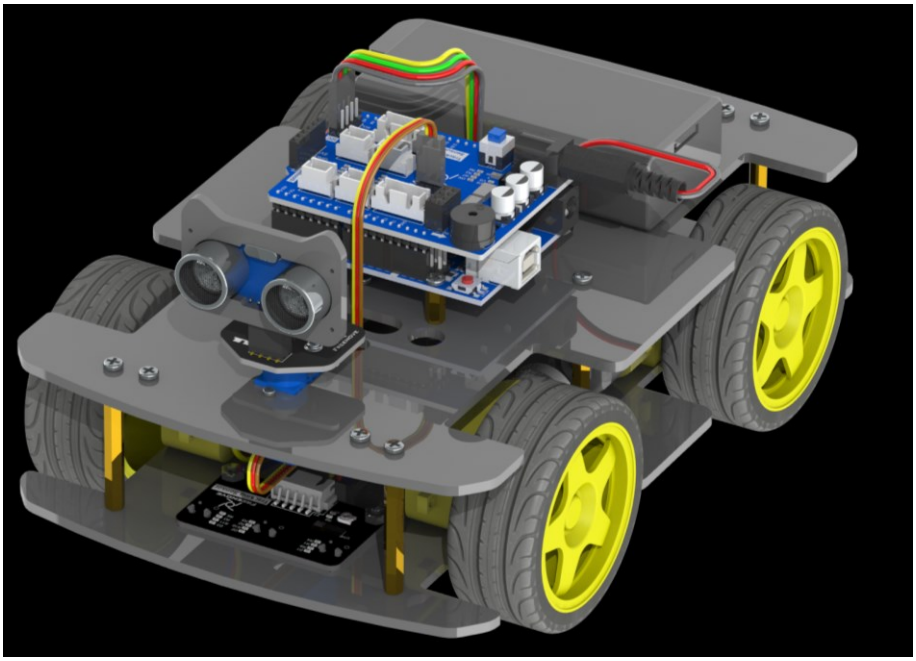
Install standoff M3*40 on bottom acrylic board.



Install top acrylic board to standoff.

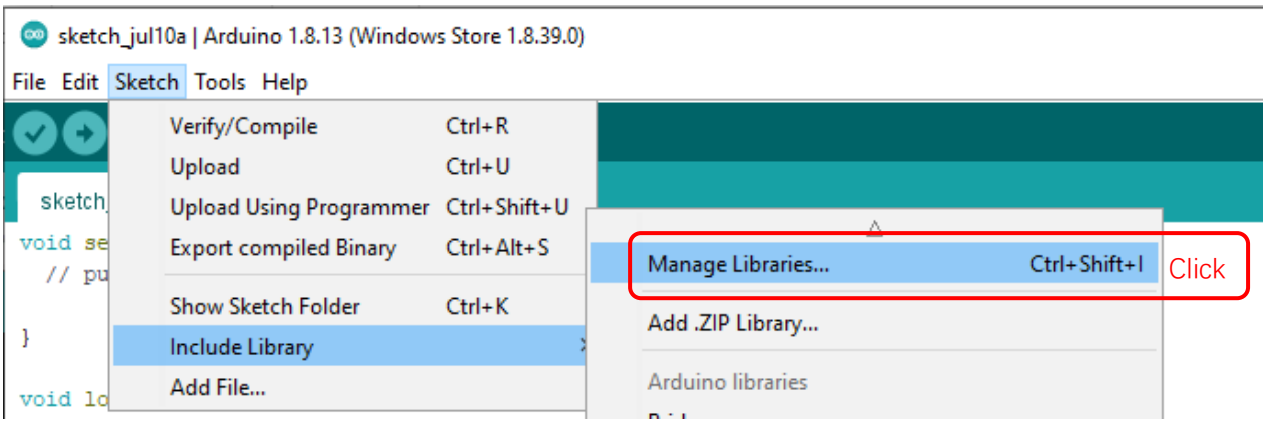


Now the whole assembly is completed.

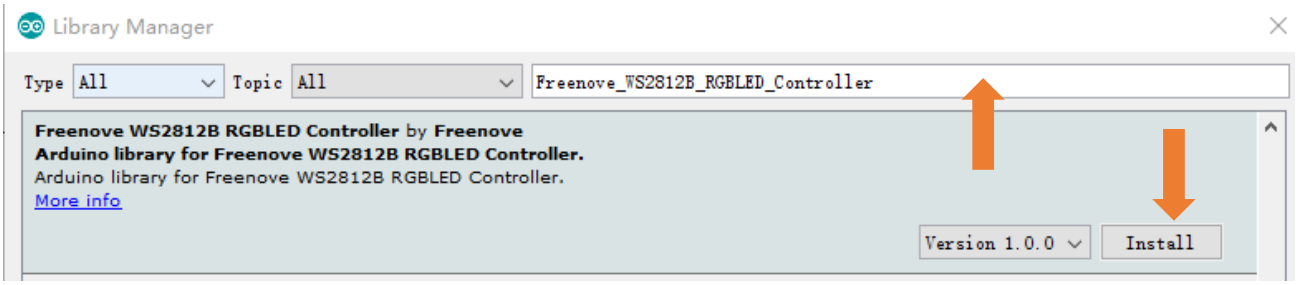


0.3 How to Play

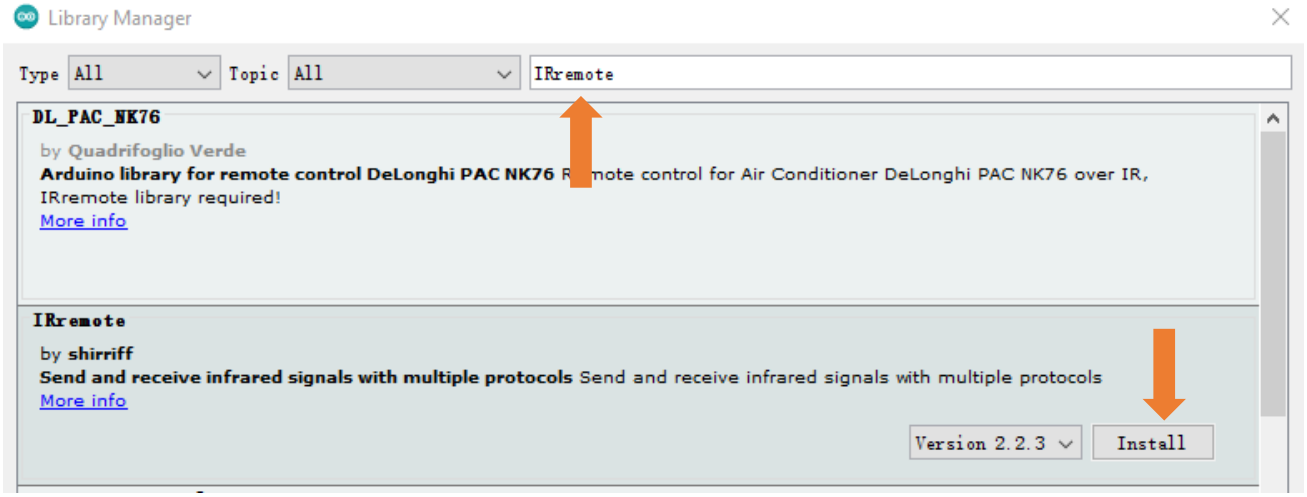
Step 1 Install libraries



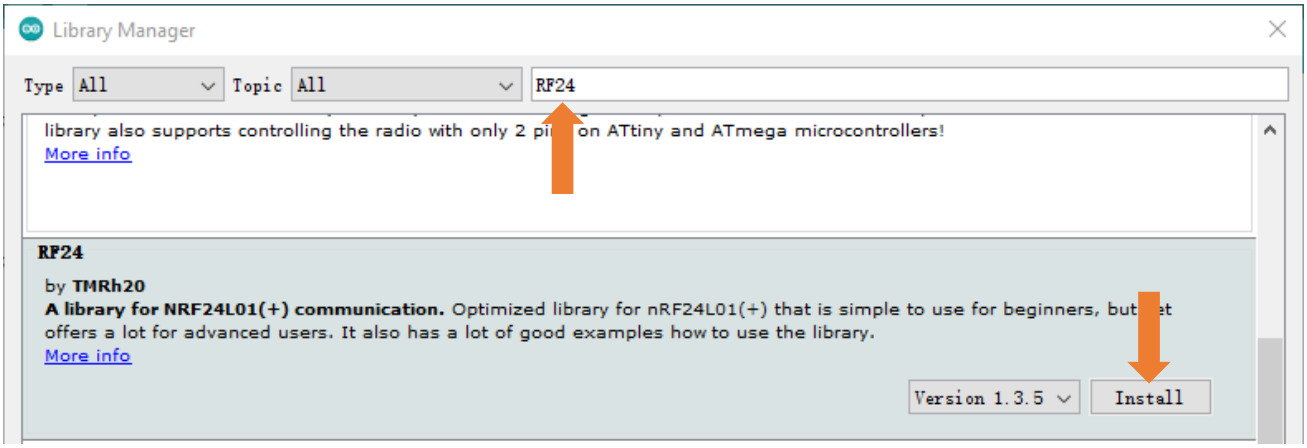
Enter "**Freenove_WS2812B_RGBLED_Controller**" and press "Enter" key to search.
Find and **install** the latest version (so far it is version 1.0.0).



Install IRtemote library.



If you purchase the version with RF remote controller, you need to install RF24.



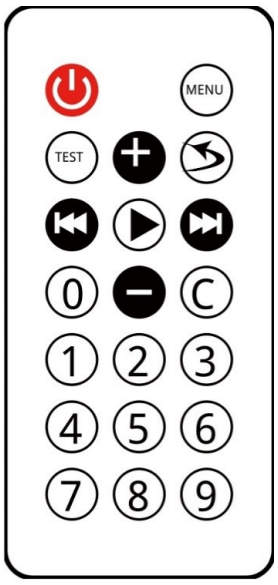
Step 2 Upload Code

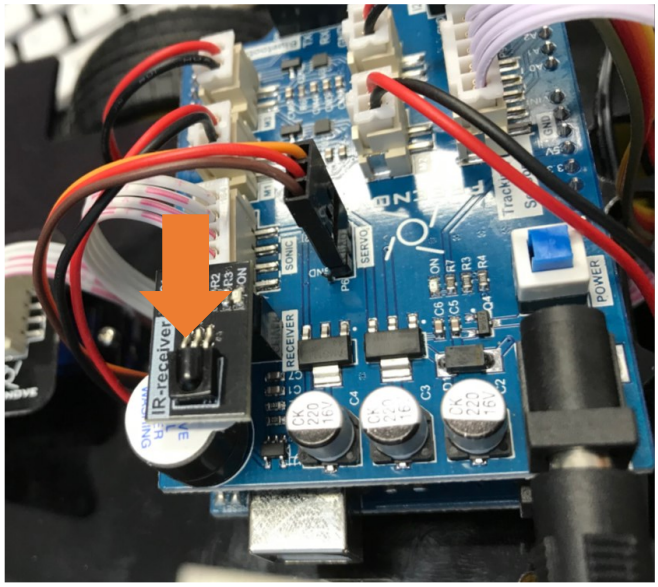
Please remove Bluetooth when you upload code.

If you want to use different control ways, you need to upload different codes.

IR remote control

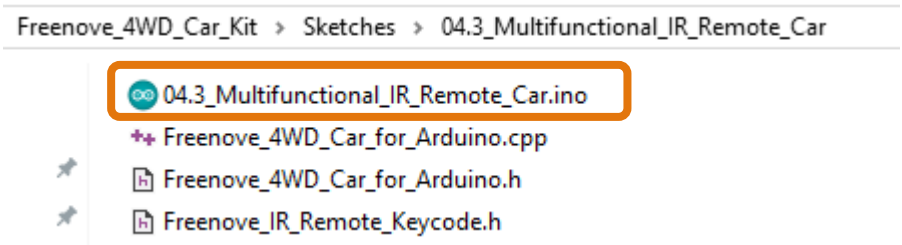
Install IR receiver to the car.





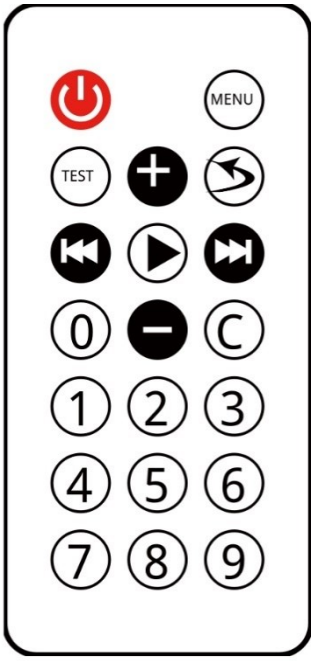
Upload following code to the car board.

Freenove_4WD_Car_Kit\Sketches\04.3_Multifunctional_IR_Remote_Car



After uploading successfully, you can use the IR remote control to control the car.

After the code is successfully uploaded, turn on the power of the car and use the infrared remote control to control the car and other functions. The corresponding keys and their functions are shown in the following table:

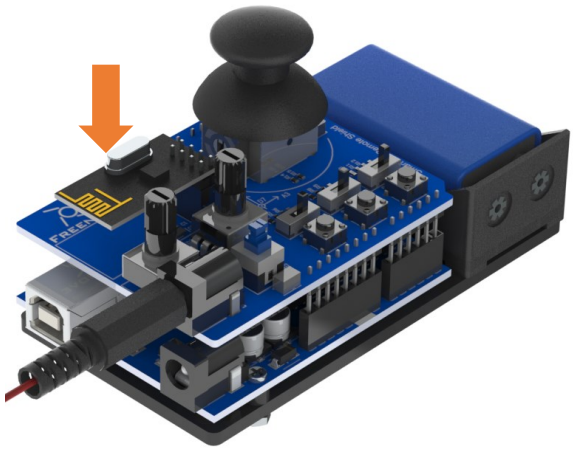


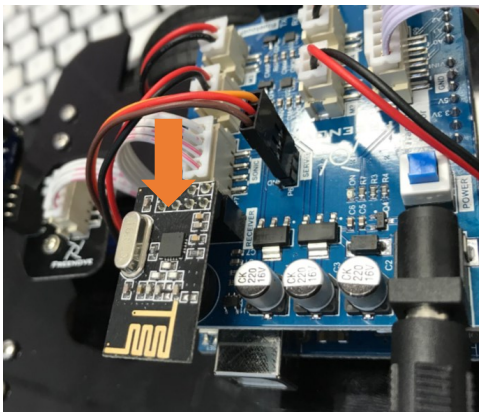
Key graph	Key define	Key code	Function
+	IR_REMOTE_KEYCODE_UP	0xFF02FD	move forward
−	IR_REMOTE_KEYCODE_DOWN	0xFF9867	move back
⏮	IR_REMOTE_KEYCODE_LEFT	0xFFE01F	Turn left
⏭	IR_REMOTE_KEYCODE_RIGHT	0xFF906F	Turn right
▶	IR_REMOTE_KEYCODE_CENTER	0xFFA857	Turn on buzzer
1	IR_REMOTE_KEYCODE_1	0xFF30CF	Make the LED run mode 1 to scroll the rainbow color.
4	IR_REMOTE_KEYCODE_4	0xFF10EF	Make LED run mode 2, changing the color of the water LED
2	IR_REMOTE_KEYCODE_2	0xFF18E7	The color of the LED bar changes faster. The color is from ColorWheel.
3	IR_REMOTE_KEYCODE_3	0xFF7A85	The color of the LED bar changes slower.
5	IR_REMOTE_KEYCODE_5	0xFF38C7	The LED bar cycle period is decreased, and the LED bar changes at a faster speed
6	IR_REMOTE_KEYCODE_6	0xFF5AA5	The LED bar cycle period is increased, and the LED bar changes at a slower speed

RF remote control

Download tutorial and code to **assemble** remote controller.
https://github.com/Freenove/Freenove_Remote_Control_Kit/raw/master/Tutorial.pdf

Remove IR reviver and install RF module.





Upload following code to the car board.
Freenove_4WD_Car_Kit\Sketches\05.4_Multifunctional_RF24_Remote_Car

Upload following code to the RF remote control.
Freenove_4WD_Car_Kit\Sketches\05.1_RF24_Remote_Controller

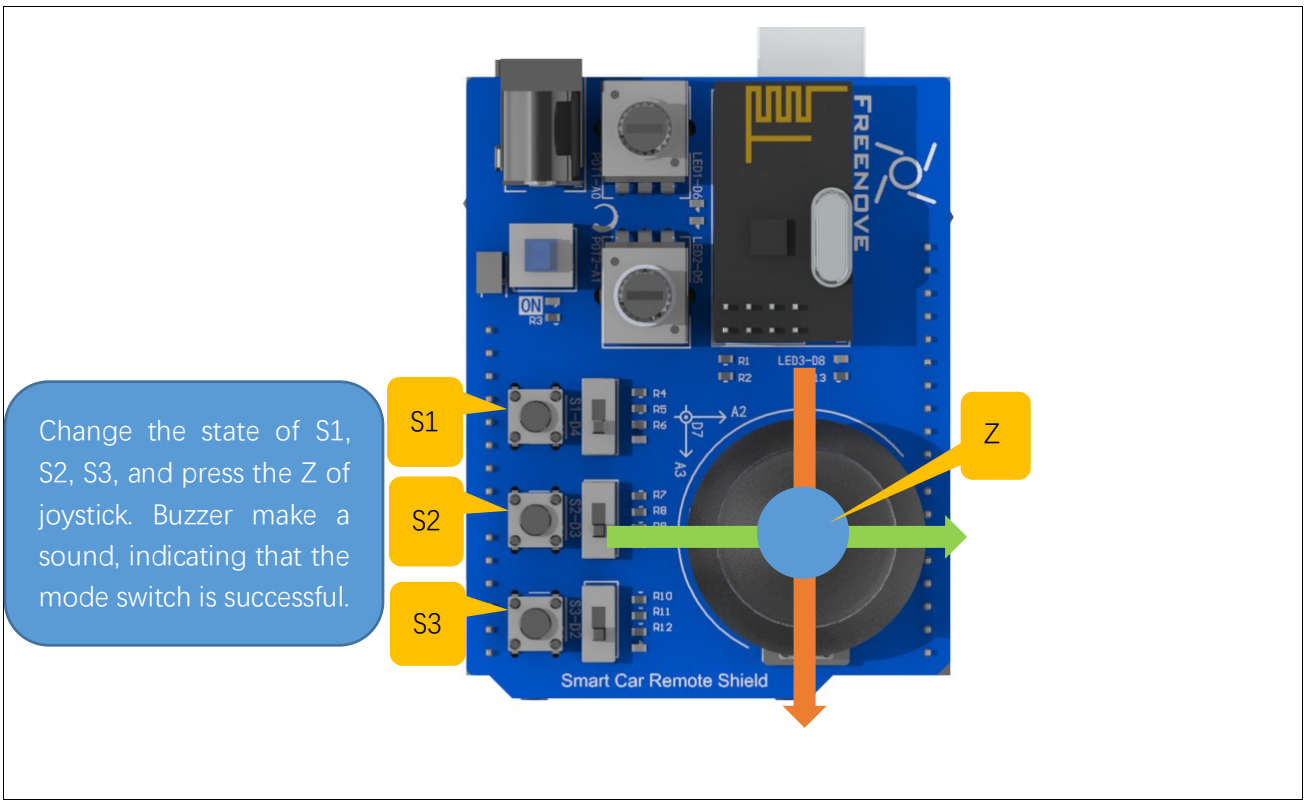
After uploading successfully, you can use the RF remote control to control the car.

Switch different modes

1, Change the switch state of S1, S2, and S3, and the car will stop moving.
2, Press the Z axis of the joystick and the buzzer B sounds to indicate that the mode is successfully switched.

The following table shows the modes indicated by the different states of the three switches S1, S2, and S3. The LED next to the switch illuminates to indicate ON and OFF state of switches. The three switches can form 2x2x2=8 modes.

S1	S2	S3	Mode No.	Mode
ON	ON	ON	0	None
ON	ON	OFF	1	Calibrate servo mode
ON	OFF	ON	2	None
ON	OFF	OFF	3	Obstacle avoidance mode
OFF	ON	ON	4	None
OFF	ON	OFF	5	Line tracking mode
OFF	OFF	ON	6	Switch LED mode
OFF	OFF	OFF	7	Manuel control mode / Default mode



Mode 0, 2, 4

Reserved. We did not assign functions for them.

Mode 1-Calibrate servo

If your servo is not accurately mounted at 90 degrees, you can use this mode for fine adjustment (+-10 degrees).

In this mode, you can adjust potentiometer 2 (POT2) to fine tune the angle of the servo. When you adjust the servo to the correct angle, press the Z-axis of joystick to save calibration data to EEPROM. It will be saved permanently unless it is modified.

Mode 3-obstical avoidance, Mode 5-line tracking mode

These two modes have been learned separately in the previous project, and their running logic and codes are consistent with the previous project.

The difference is that in this project, the car can respond to commands from the remote control at any time. Therefore, in this project, it is still necessary to communicate with the remote controller in these two modes. When the remote control signal is disconnected, the car will stop. Therefore, the normal communication between the remote control and the car should be maintained at any time. Poor communication conditions may cause these two modes to work abnormally.

Mode 6-switch LED display mode

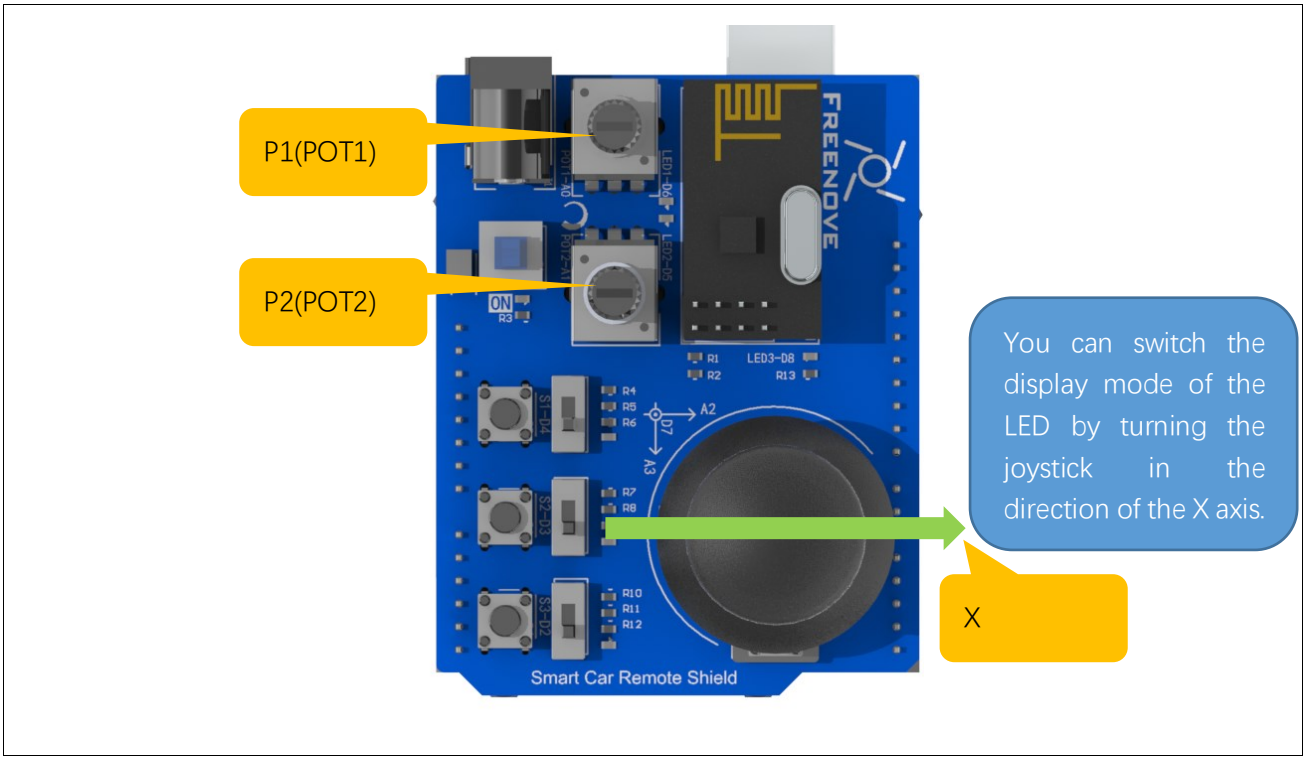
There are three display modes for the LEDs on the car, which are 0-flowing rainbow, 1-flowing water led, 2-Blink. In this mode, the display mode of the LED can be switched.

After entering this mode,

Move the joystick along the positive direction of its X-axis to switch the LED to the next mode.

Move the joystick in the negative direction of its X-axis to switch the LED to the previous mode.

In any mode, the LEDs can be adjusted with potentiometers P1 and P2. P1 is used to adjust the color change of the LED, and P2 is used to adjust the change frequency of LED.



Mode 7- manual remote mode

This mode is manual remote mode and is the default mode. This mode is consistent with the previous project "RF_Remote_Car". Use the joystick to control to move forward, move back and turn left, turn right.

The following chapters will teach you how to control components.

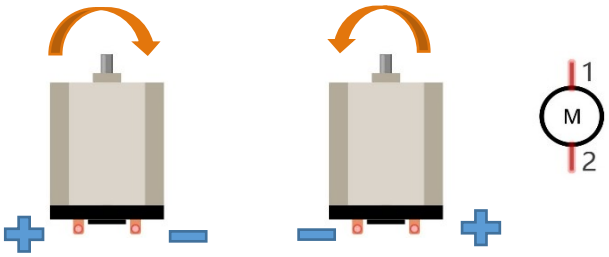
Chapter 1 Control Basic Components

This chapter will introduce how to control motor, buzzer, RGB LED and get battery level.
If you have any concerns, please feel free to contact us via support@freenove.com

1.1 Motor

When motor is connected to the power supply, it will rotate in one direction. Reverse the polarity of power supply, the motor will rotate in the opposite direction.

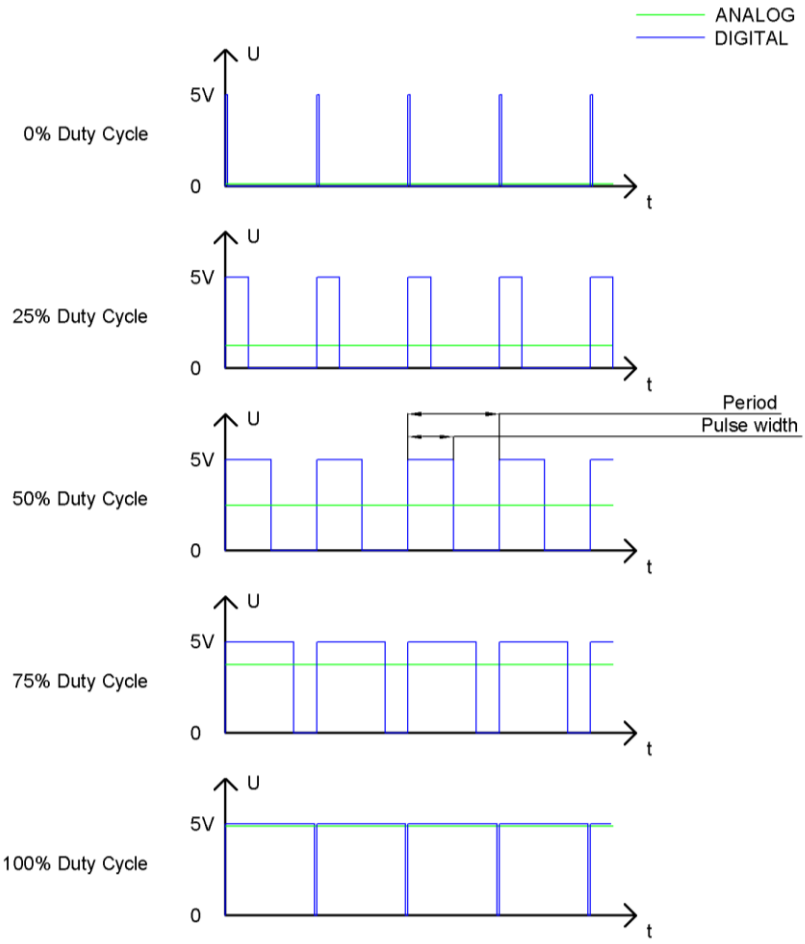
And the speed of motor depends on the voltage between two ends. The larger the voltage, the lager the speed.



PWM

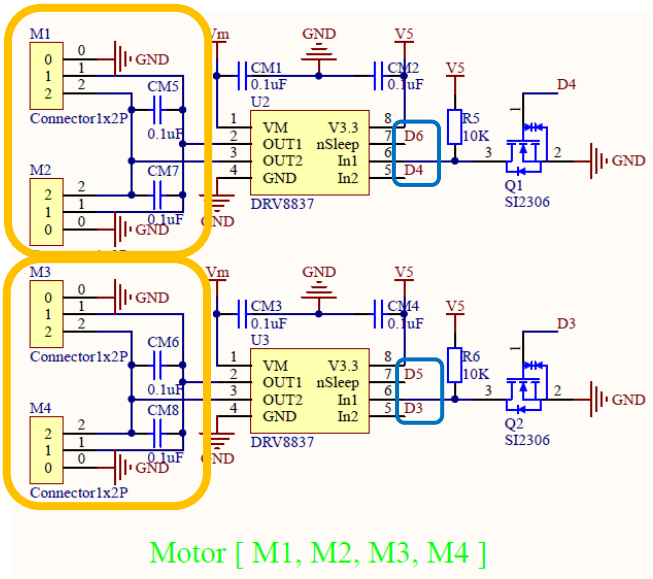
PWM, Pulse Width Modulation, uses digital pins to send certain frequencies of square waves, that is, the output of high levels and low levels, which alternately last for a while. The total time for each set of high levels and low levels is generally fixed, which is called the period (the reciprocal of the period is frequency). The time of high level outputs are generally called “pulse width”, and the duty cycle is the percentage of the ratio of pulse duration, or pulse width (PW) to the total period (T) of the waveform.

The longer the output of high levels last, the larger the duty cycle and the higher the corresponding voltage in analog signal will be. The following figures show how the analogs signal voltage vary between 0V-5V (high level is 5V) corresponding to the pulse width 0%-100%:



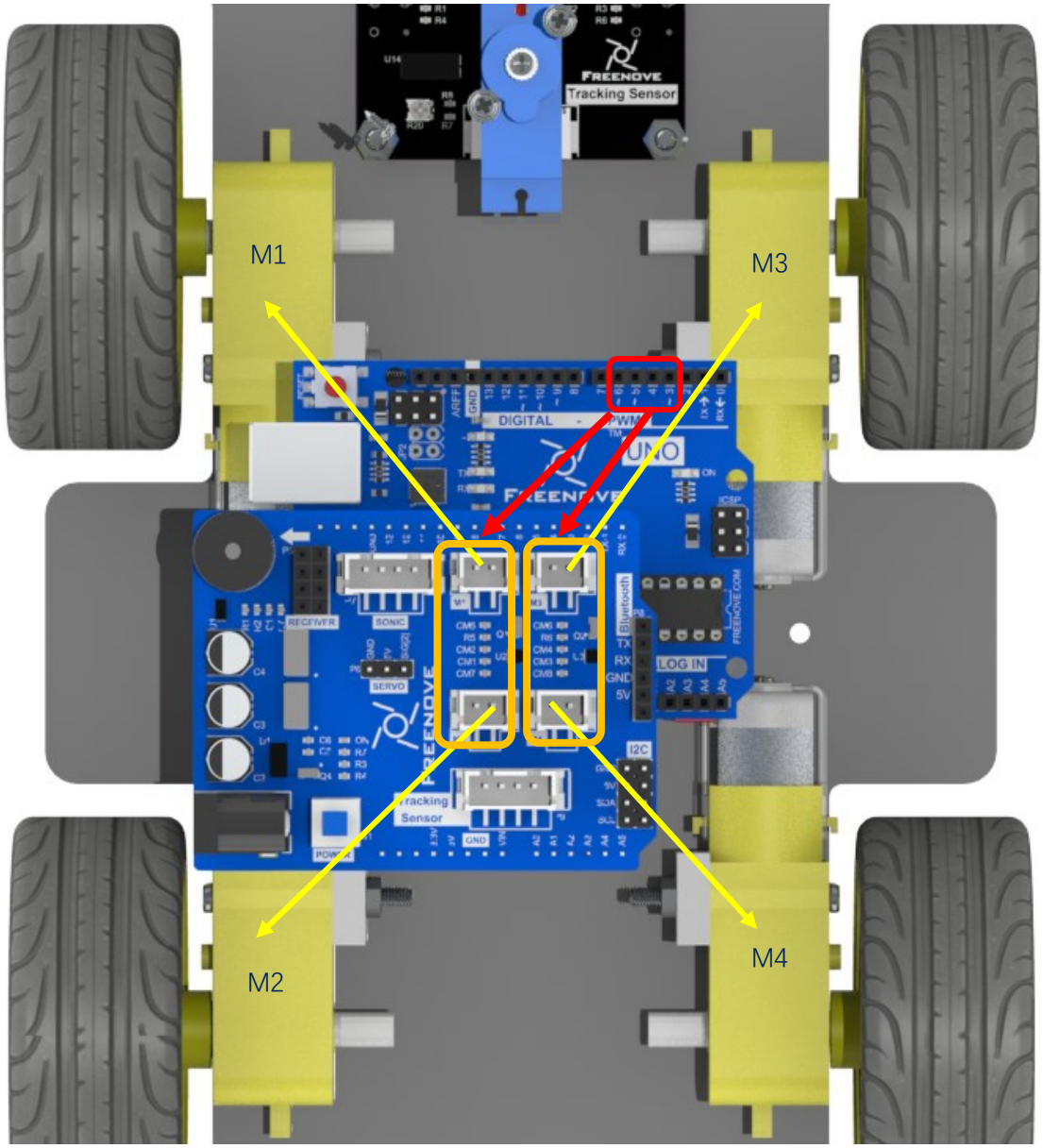
The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.

In this car, M1 and M2 are connected in parallel; M3 and M4 are connected in parallel. Schematic is below:



Motor [M1, M2, M3, M4]

D3 and D5 pins of control board control M3 and M4 respectively. D4 and D6 pins of board control M1 and M2.



Code

01.1.1_RunMotor_Left_Wheel

If you don't know how to upload code to Arduino, you can refer to First Use.

Next turn off the power. And connect control board to computer with a USB cable. Upload code in the folder: Sketches\01.1.1_RunMotor_Left_Wheel

Then disconnect USB cable and put your car in a place with enough space to move. Or make the car's head stand up and observe the wheel. **The following code also requires this step.**
Turn on the power of Car. Then the left motors and wheels of car will rotate forth and back. And you can see the speeds are different. You can turn off the power of the car to stop that.

From previous section, we have known that M1 and M2 are connected in parallel andM3 and M4 are connected in parallel. It means that **you can only control motors in the same side together but** not separately. Usually, motors of the same side don't need to rotate in different directions.

The code is below:

```
#define PIN_DIRECTION_RIGHT 3
#define PIN_DIRECTION_LEFT  4
#define PIN_MOTOR_PWM_RIGHT 5
#define PIN_MOTOR_PWM_LEFT  6

void setup() {
  pinMode(PIN_DIRECTION_LEFT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_LEFT, OUTPUT);
  pinMode(PIN_DIRECTION_RIGHT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_RIGHT, OUTPUT);
}

void loop() {
  //Left motors rotates to one direction
  digitalWrite(PIN_DIRECTION_LEFT, HIGH);
  analogWrite(PIN_MOTOR_PWM_LEFT, 100);
  delay(1000);    //delay(ms), 1000ms=1s

  analogWrite(PIN_MOTOR_PWM_LEFT, 0); //Stop
  delay(1000);

  //left motors rotates to opposite direction
  digitalWrite(PIN_DIRECTION_LEFT, LOW);
  analogWrite(PIN_MOTOR_PWM_LEFT, 255);
  delay(1000);

  analogWrite(PIN_MOTOR_PWM_LEFT, 0); //Stop
  delay(1000);
}

```

In the code, following code is uses to define pins.

```
#define PIN_DIRECTION_RIGHT 3
#define PIN_DIRECTION_LEFT  4
#define PIN_MOTOR_PWM_RIGHT 5
#define PIN_MOTOR_PWM_LEFT  6
```

As you can see in schematic above. The pin for motors cannot be modified.

#define A B, it means A is B. Use A instead of B for easy maintenance and easy reading.

You can also define pins with int type, like int PIN_DIRECTION_RIGHT=3; But this need more RAM space.

Usually, there are two basic main functions for Arduino code, void setup() and void loop().

void setup(){ }
The setup() function is called when a sketch starts, which is used to initialize variables, pin modes, start using libraries, etc. The setup() function will **only run once**, after each **power up** or **reset** of the Arduino board.
void loop(){ }
This function will loop consecutively. Code in this function will be executed again and again…

There are some other functions integrated by Arduino.

pinMode(pin, mode)
Configures the specified pin to behave either as an input or an output. Parameters pin: Arduino pin number to set the mode of. mode: INPUT, OUTPUT, or INPUT_PULLUP.
digitalWrite(pin, value)
Write a HIGH or a LOW value to a digital pin. If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW. Parameters pin: Arduino pin number. value: HIGH or LOW.
analogWrite(pin, value)
Writes an analog value (PWM wave) to a pin. You do not need to call pinMode() to set the pin as an output before calling analogWrite(). Parameters pin: Arduino pin to write to. Allowed data types: int. value: the duty cycle: between 0 (always off) and 255 (always on). Allowed data types: int.

For more details, please refer to: https://www.arduino.cc/reference/

Note, as to analogWrite(pin, value) of motors for this car, the maximum value is 255.As the motor requires a certain voltage to run, and analogwirte (pin, 1) doesn't meet such required voltage., according to different voltage, the value that makes the car run will be much greater than analogwirte (pin, 1).

01.1.2_RunMotor_Right_Wheel

Upload code in folder: Sketches\01.1.2_RunMotor_Right_Wheel.

Then put your car in a place with enough space to move. Turn ON the power of car. Then the right motors and wheels of car will rotate forth and back. And you can see that the speeds are different.

Because the left motors and right motors are mirrored. When the motors rotate with same direction, the car will react with reversed direction.

The code is below:

```
1 #define PIN_DIRECTION_RIGHT 3
2 #define PIN_DIRECTION_LEFT  4
```

```
#define PIN_MOTOR_PWM_RIGHT 5
#define PIN_MOTOR_PWM_LEFT  6

void setup() {
  pinMode(PIN_DIRECTION_LEFT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_LEFT, OUTPUT);
  pinMode(PIN_DIRECTION_RIGHT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_RIGHT, OUTPUT);
}

void loop() {
  //Right motors rotate to one direction
  digitalWrite(PIN_DIRECTION_RIGHT, HIGH);
  analogWrite(PIN_MOTOR_PWM_RIGHT, 100);
  delay(1000);
  analogWrite(PIN_MOTOR_PWM_RIGHT, 0);
  delay(1000);

  //Right motors rotate to opposite direction
  digitalWrite(PIN_DIRECTION_RIGHT, LOW);
  analogWrite(PIN_MOTOR_PWM_RIGHT, 255);
  delay(1000);
  analogWrite(PIN_MOTOR_PWM_RIGHT, 0);
  delay(1000);
}

```

01.1.3_Car_Move_and_Turn

Upload code in folder: Sketches\01.1.3_Car_Move_and_Turn.

Then put your car in a place with enough space to move. Turn ON the power of car. Then the car will move forth and back, then turn left and turn right. After stopping for 2 second, it repeat the actions.

The code is below:

```
#define PIN_DIRECTION_RIGHT 3
#define PIN_DIRECTION_LEFT  4
#define PIN_MOTOR_PWM_RIGHT 5
#define PIN_MOTOR_PWM_LEFT  6

void setup() {
  pinMode(PIN_DIRECTION_LEFT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_LEFT, OUTPUT);
  pinMode(PIN_DIRECTION_RIGHT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_RIGHT, OUTPUT);
}

```

```
13  void loop() {
14    //Move forward
15    motorRun(200, 200);
16    delay(1000);
17
18    //Move back
19    motorRun(-200, -200);
20    delay(1000);
21
22    //Turn left
23    motorRun(-200, 200);
24    delay(1000);
25
26    //Turn right
27    motorRun(200, -200);
28    delay(1000);
29
30    //Stop
31    motorRun(0, 0);
32    delay(2000);
33  }
34
35  void motorRun(int speedl, int speedr) {
36    int dirL = 0, dirR = 0;
37    if (speedl > 0) {
38      dirL = 0;
39    }
40    else {
41      dirL = 1;
42      speedl = -speedl;
43    }
44    if (speedr > 0) {
45      dirR = 1;
46    }
47    else {
48      dirR = 0;
49      speedr = -speedr;
50    }
51    digitalWrite(PIN_DIRECTION_LEFT, dirL);
52    digitalWrite(PIN_DIRECTION_RIGHT, dirR);
53    analogWrite(PIN_MOTOR_PWM_LEFT, speedl);
54    analogWrite(PIN_MOTOR_PWM_RIGHT, speedr);
55  }
56
```

Now we will use motorRun(200, -200) as an example to introduce this function.

```
35  void motorRun(int speedl, int speedr) {   //speedl=200, speedr=-200
36    int dirL = 0, dirR = 0;
37    if (speedl > 0) {    //speedl=200>0, yes
38      dirL = 0;          //so dirL =0
39    }
40    else {
41      dirL = 1;
42      speedl = -speedl;
43    }
44    if (speedr > 0) {      //speedr=-200<0
45      dirR = 1;
46    }
47    else {
48      dirR = 0;            //so dirR=0
49      speedr = -speedr;    //then speedr=200
50    }
51    digitalWrite(PIN_DIRECTION_LEFT, dirL);   // digitalWrite(PIN_DIRECTION_LEFT, 0);
52    digitalWrite(PIN_DIRECTION_RIGHT, dirR);  // digitalWrite(PIN_DIRECTION_LEFT, 0);
53    analogWrite(PIN_MOTOR_PWM_LEFT, speedl);  // analogWrite(PIN_MOTOR_PWM_LEFT, 200);
54    analogWrite(PIN_MOTOR_PWM_RIGHT, speedr); // analogWrite(PIN_MOTOR_PWM_RIGHT, 200)
55  }
```

The left motors and right motors are **mirrored**. When the motors on two side rotate with same direction, the car will turn. In order to make a car move forth or back, we need set motor on two side with different direction. So motorRun(200, -200) will make the car turn right.

1.2 Buzzer and Battery level

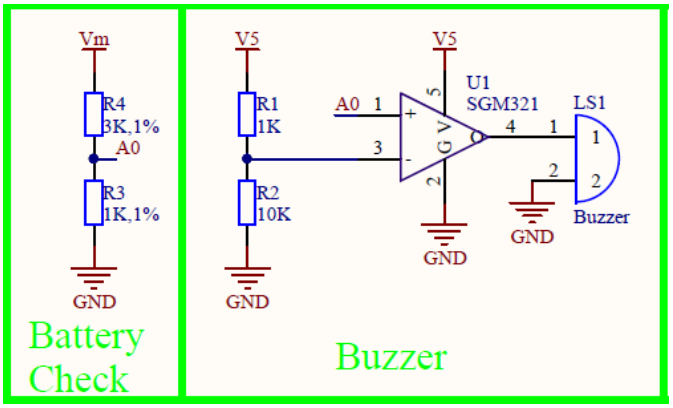
A buzzer is an audio component. There are both active and passive types of buzzers. Active buzzers have oscillator inside, these will sound as long as power is supplied. Passive buzzers require an external oscillator signal (generally using PWM with different frequencies) to make a sound.

Active buzzer Passive buzzer

(A white label is attached to the active buzzer)

Active buzzers are easier to use. Generally, they can only make a specific sound frequency.
Passive buzzers require an external circuit to make sounds, but they can be controlled to make sounds of various frequencies. The resonant frequency of the passive buzzer is 2kHz, which means the passive buzzer is the loudest when resonant frequency is 2kHz.
In this car, the buzzer is an active buzzer. So it can only make a specific frequency of sound. Function tone() of Arduino cannot be used for this kind of buzzer.

Since the control board has limited number of IOs. Buzzer and battery level are not used frequently. So we use one IO of control board controlling buzzer and detecting battery level in different time
The schematic of buzzer and battery is below:

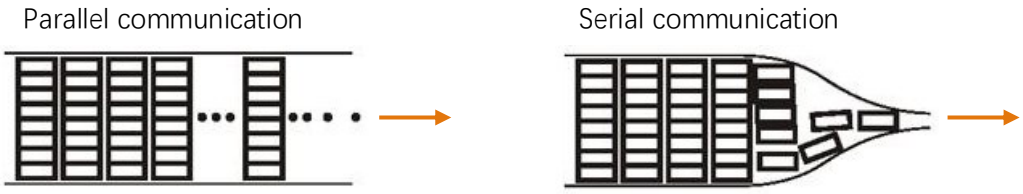


As you can see they use same IO A0.

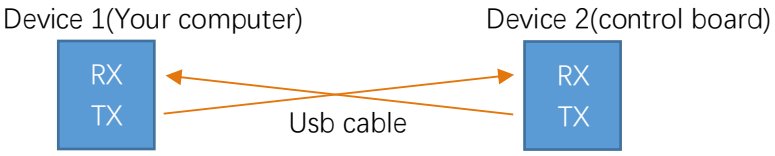
Battery level should be **Vm=4*V0**. V0 is the voltage detected by A0 of control board.
The maximum battery level is 8.4v (V0=2.1V).
The right circuit means if V0>4.5V, the buzzer will make sound.
If V0 is within(0~2.1v), it is used for battery. If V0>4.5V, it is used for buzzer.

Serial Communication

Serial communication uses one data cable to transfer data one bit by another in turn. Parallel communication means that the data is transmitted simultaneously on multiple cables. Serial communication takes only a few cables to exchange information between systems, which is especially suitable for computers to computers, long distance communication between computers and peripherals. Parallel communication is faster, but it requires more cables and higher cost, so it is not appropriate for long distance communication.



Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication connections of two devices use is as follows:



For serial communication, the **baud rate in both sides must be the same**. The baud rates commonly used are 9600 and 115200.

Computer identifies serial devices connected to your computer as COMx. We can use the Serial Monitor window of Arduino Software to communicate with Freenove control board.

Code

01.2.1_Buzzer

Upload code in Sketches\01.2.1_Buzzer. Then turn the switch of car.

You will hear b-bbbb----bbbb----bbbb---- ……

The code in setup() is only executed once. Code in loop() will be executed circularly.

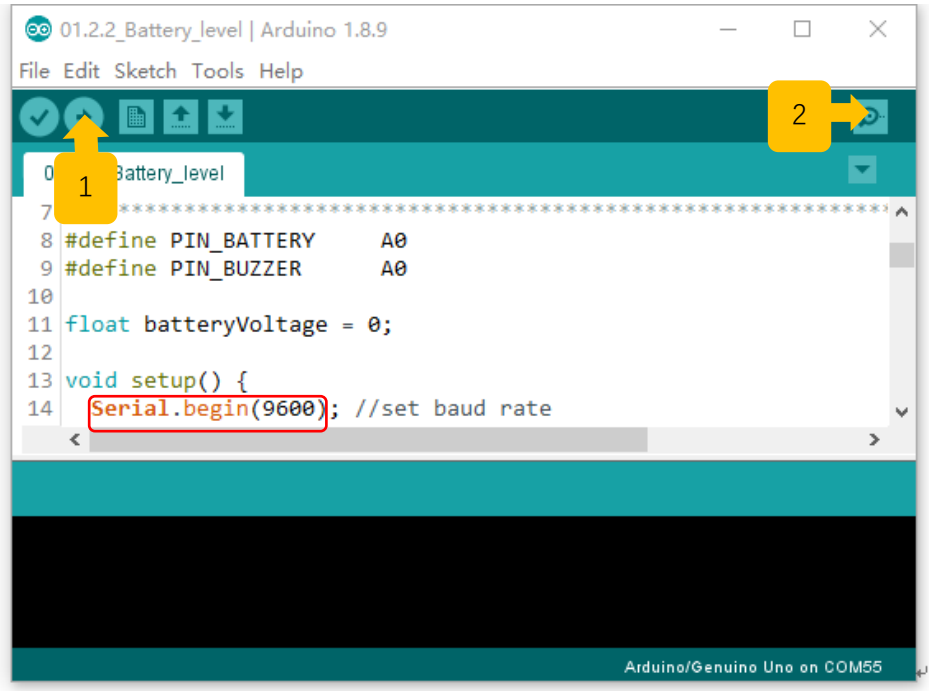
```
#define PIN_BATTERY     A0
#define PIN_BUZZER      A0

void setup() {
  pinMode(PIN_BUZZER, OUTPUT);
  digitalWrite(PIN_BUZZER, HIGH);
  delay(100);
  digitalWrite(PIN_BUZZER, LOW);
}

void loop() {
  for (int i = 0; i < 4; i++) {
    digitalWrite(PIN_BUZZER, HIGH);//turn on buzzer
```

```
14        delay(100);
15        digitalWrite(PIN_BUZZER, LOW); //turn off buzzer
16        delay(100);
17      }
18      delay(500);
19  }
```

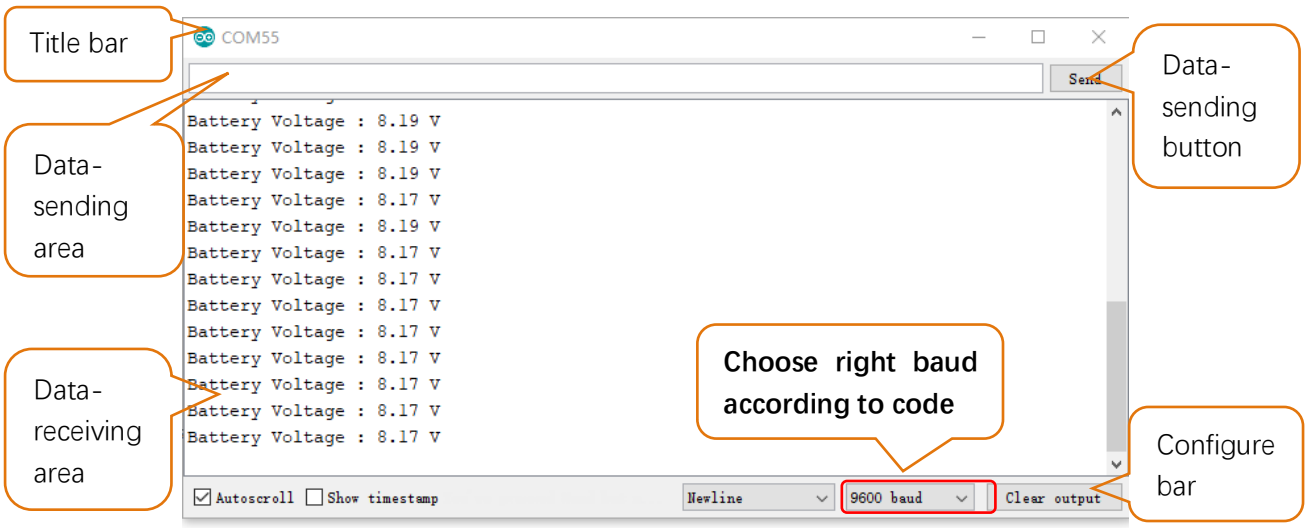
01.2.2_Battery_level

Upload code in Sketches\01.2.2_Battery_level. Don't disconnect the USB cable. Then turn on the switch of the car.

And then click the Serial Monitor icon to open the Serial Monitor window.



Then you will hear buzzer making sounds twice. And then see the Interface of Serial Monitor window as follows. If you can't open it, make sure Freenove control board had been connected to the computer, and choose the right serial port in the menu bar "Tools-Port".



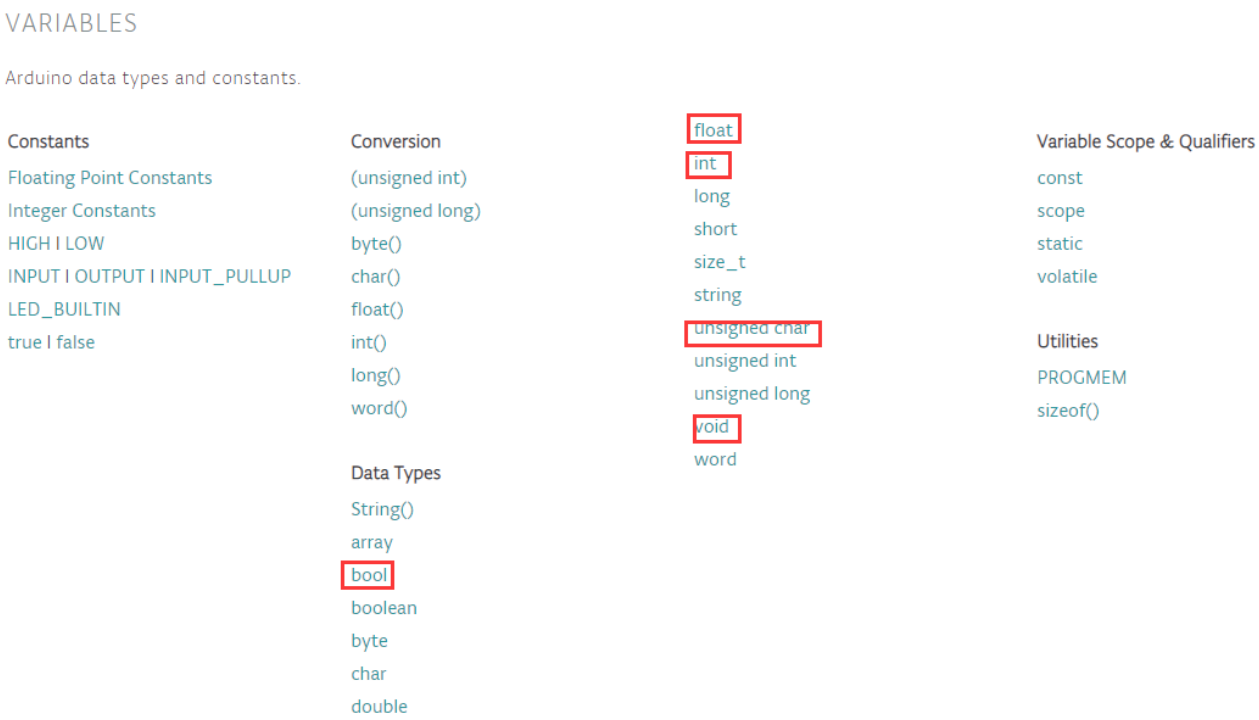
The code is below:

```
#define PIN_BATTERY     A0
#define PIN_BUZZER      A0

float batteryVoltage = 0;

void setup() {
  Serial.begin(9600); //set baud rate
  pinMode(PIN_BUZZER, OUTPUT);

  digitalWrite(PIN_BUZZER, HIGH);
  delay(500);
  digitalWrite(PIN_BUZZER, LOW);
  delay(500);
  digitalWrite(PIN_BUZZER, HIGH);
  delay(500);
  digitalWrite(PIN_BUZZER, LOW);
}

void loop() {
  //Because the buzzer and battery voltage detection use A0 pin together,
  //the buzzer must be turned off when reading battery voltage,
  //otherwise the read battery voltage is incorrect.

  //Turn off buzzer
  pinMode(PIN_BUZZER, false);
  digitalWrite(PIN_BUZZER, false);

  //print battery voltage
  if (getBatteryVoltage()) {
    Serial.print("Battery Voltage : ");
```

```
31      Serial.print(batteryVoltage);
32      Serial.println(" V");
33    }
34    else {
35      Serial.print("Port busy! Please turn off buzzer before reading battery voltage.");
36    }
37    delay(1000);
38  }
39
40  bool getBatteryVoltage() {
41      pinMode(PIN_BATTERY, INPUT);
42      int batteryADC = analogRead(PIN_BATTERY);
43      if (batteryADC < 614)             // 614/1023* 5=3V, reasonable Voltage:<2.1V
44      {
45        batteryVoltage = batteryADC / 1023.0 * 5.0 * 4;
46        return true;
47      } else {
48      return false;
49    }
50  }
```

In this code, different Data Types are mentioned: float, bool, void.

If you are not familiar with data types, please refer to https://www.arduino.cc/reference/



The function is defined in form: data type function name ().
If it is void function name(), after it is executed, it will return nothing.

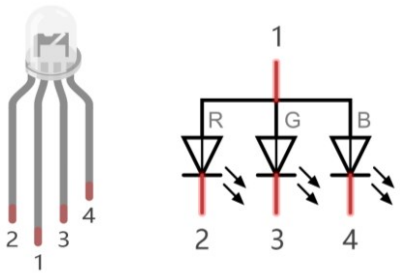
```
28    //print battery voltage
29    if (getBatteryVoltage()) {
30      Serial.print("Battery Voltage : ");
31      Serial.print(batteryVoltage);
32      Serial.println(" V");
33    }
34    else {
35      Serial.print("Port busy. Please turn off buzzer before reading battery voltage.");
36    }
37    delay(1000)
38  }
39
40  bool getBatteryVoltage() {
41      pinMode(PIN_BATTERY, INPUT);
42      int batteryADC = analogRead(PIN_BATTERY);
43      if (batteryADC < 614)              // 614/1023* 5=3V, reasonable Voltage:<2.1V
44      {
45        batteryVoltage = batteryADC / 1023.0 * 5.0 * 4;
46        return true;
47      }else {
48      return false;
49    }
50  }
```

Code is always executed line by line. When if (getBatteryVoltage()) is executed, the program will turn to execute bool getBatteryVoltage(){ }. After return, the program will execute code in if{ }.

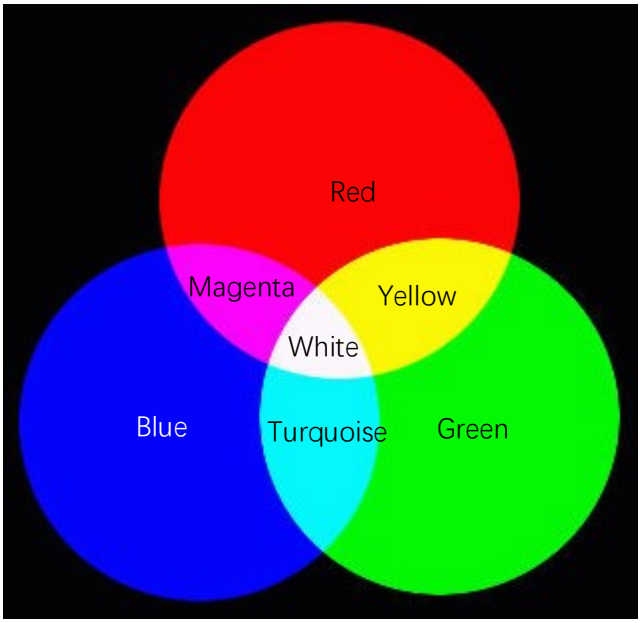
Serial.print()
Syntax is below: Serial.print(val) Serial.print(val, format) Serial: serial port object. See the list of available serial ports for each board on the Serial main page. val: the value to print. Allowed data types: any data type. The only difference is that Serial.pritnln() will send a "\n". content of next print() or println() will be shown in a new line.
analogRead(pin)
It will map input voltages between 0 and the operating voltage(5V or 3.3V) into integer values between 0 and 1023. For 5v as an example, ADCvalue= analogRead(pin) If you want to get the voltage value. It should be V=ADCvalue/1023*5V.

1.3 RGB

A RGB LED has 3 LEDs integrated into one LED component. It can respectively emit Red, Green and Blue light. In order to do this, it requires 4 pins (this is also how you identify it). The long pin (1) is the common which is the Cathode (+) or positive lead, the other 3 are the Anodes (-) or negative leads. A rendering of a RGB LED and its electronic symbol are shown below. We can make RGB LED emit various colors of light and brightness by controlling the 3 Anodes (2, 3 & 4) of the RGB LED

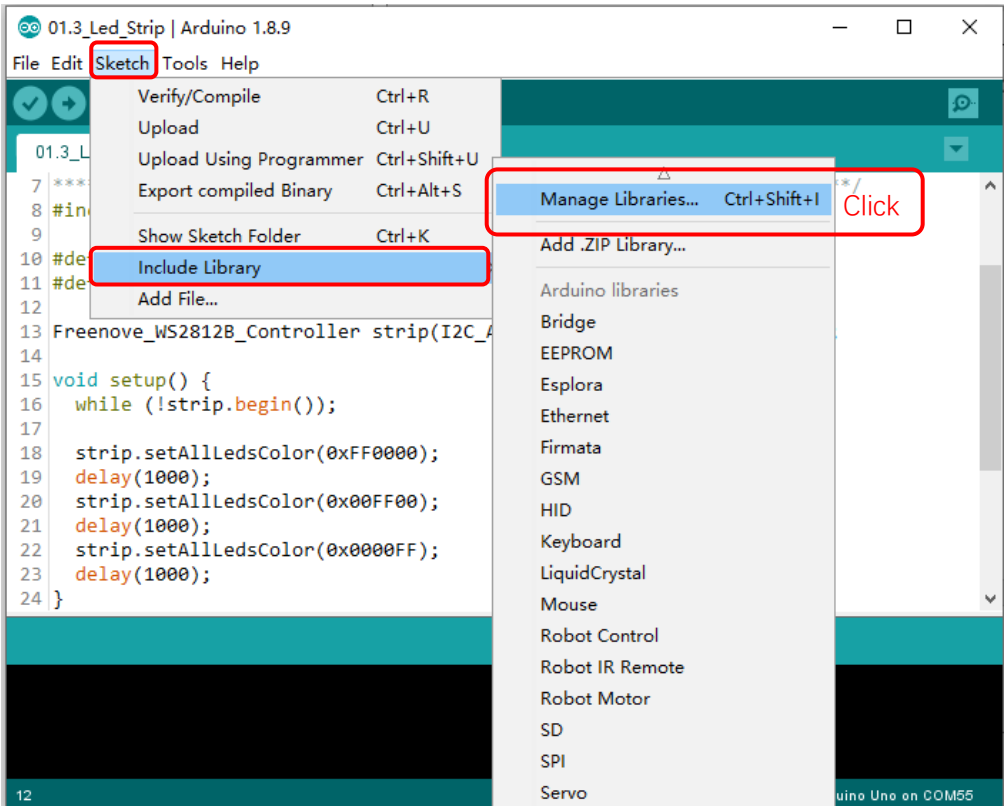


Red, Green, and Blue light are called 3 Primary Colors when discussing light (Note: for pigments such as paints, the 3 Primary Colors are Red, Blue and Yellow). When you combine these three Primary Colors of light with varied brightness, they can produce almost any color of visible light. Computer screens, single pixels of cell phone screens, neon lamps, etc. can all produce millions of colors due to this phenomenon.



We know from previous section that, control board controls LEDs to emit a total of 256(0-255) different brightness with PWM. So, through the combination of three different colors of LEDs, RGB LED can emit 256^3=16777216 Colors, 16Million colors.

Include Library



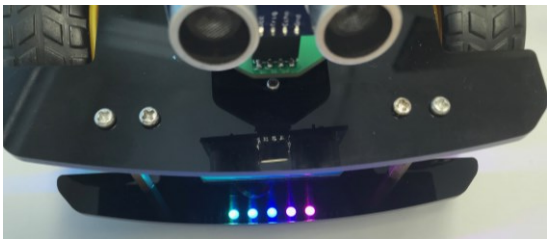
Enter "Freenove_WS2812B_RGBLED_Controller" and press "Enter" key to search.
Find and **install** the latest version (now it is version 1.0.0).

Then go back and upload the code.

Code

01.3_Led_Strip

Upload code in Sketches\01.3_Led_Strip. Then turn the power of car.
You can find a proper view to observe LED colors on the bottom acrylic board.



You will see that all LEDs show red, green, blue, yellow in turn. Then all LEDs turn off.
Then first 5 LEDs show different colors one by one. Then all LEDs show the same color and the color changes smoothly. Then different LEDs show different colors and they change smoothly.
The code is below:

```
#include "Freenove_WS2812B_RGBLED_Controller.h"

#define I2C_ADDRESS   0x20
#define LEDS_COUNT    10  //it defines number of LEDs

Freenove_WS2812B_Controller strip(I2C_ADDRESS, LEDS_COUNT, TYPE_GRB); //initialization

void setup() {
  while (!strip.begin());          //judge if initialization success

  strip.setAllLedsColor(0xFF0000); //Set all LED color to red
  delay(2000);
  strip.setAllLedsColor(0x00FF00); //set all LED color to green
  delay(2000);
  strip.setAllLedsColor(0x0000FF); //set all LED color to blue
  delay(2000);
  strip.setAllLedsColor(255, 255, 0); //set all LED color to yellow. This is just different
form of rgb value.
  delay(2000);
  strip.setAllLedsColor(0, 0, 0);     //set all LED off.
  delay(2000);

  strip.setLedColor(0, 255, 0, 0);    //set the NO.0 LED to red
  delay(1000);
  strip.setLedColor(1, 0, 255, 0);    //set the NO.1 LED to green
  delay(1000);
  strip.setLedColor(2, 0, 0, 255);    //set the NO.2 LED to blue
  delay(1000);
  strip.setLedColor(3, 255, 255, 0); //set the NO.3 LED to yellow
  delay(1000);
  strip.setLedColor(4, 255, 0, 255); //set the NO.4 LED to purple
  delay(1000);
}

```

```
35  void loop() {
36    for (int k = 0; k < 255; k = k + 2) {
37      strip.setAllLedsColorData(strip.Wheel(k)); // set color data for all LED
38      strip.show();                              // show the color set before
39      delay(50);
40    }
41    delay(3000);
42
43    for (int j = 0; j < 255; j += 2) {
44      for (int i = 0; i < LEDS_COUNT; i++) {
45        strip.setLedColorData(i, strip.Wheel(i * 256 / LEDS_COUNT + j)); //set color data for
46  LED one by one
47      }
48      strip.show();  // show the color set
49      delay(50);
50    }
51  }
```

First, you need install and include the library.

```
#include "Freenove_WS2812B_RGBLED_Controller.h"
```

Then set number of LED. The maximum is 255. But it is limited by power supply.

```
#define LEDS_COUNT    10  //it defines number of LEDs
```

There are two ways to set LED color.

A, set LED color directly.

1) set all LED colors

strip.setAllLedsColor(rgb value)

rgb value should be Hexadecimal value, like `strip.setAllLedsColor(0x0000FF)`.

strip.setAllLedsColor(rvalue, gvalue, bvalue)

This is another form, Like `strip.setAllLedsColor(255, 255, 0)`.

You can find more rgb values and colors here https://www.rapidtables.com/web/color/RGB_Color.html

2) set color for one LED

strip.setLedColor(LED_Index, rgb_value) or **strip.setLedColor(LED_Index, rvalue, gvalue, bvalue)**

LED No. starts from 0, like `strip.setLedColor(0, 255, 0, 0)`.

B, Set color data first. Then use show function.

This is a more **efficient** way to set colors for many LEDs. It will make the code run faster than before.

1) set all LED color

strip.setAllLedsColorData(rgb value) or **strip.setAllLedsColorData(rvalue, gvalue, bvalue)**

After you complete setting color, just add **strip.show()**

2) set color for one LED

strip.setLedColorData(LED_Index, rgb value) or **strip.setLedColorData(LED_Index, rvalue, gvalue, bvalue)**

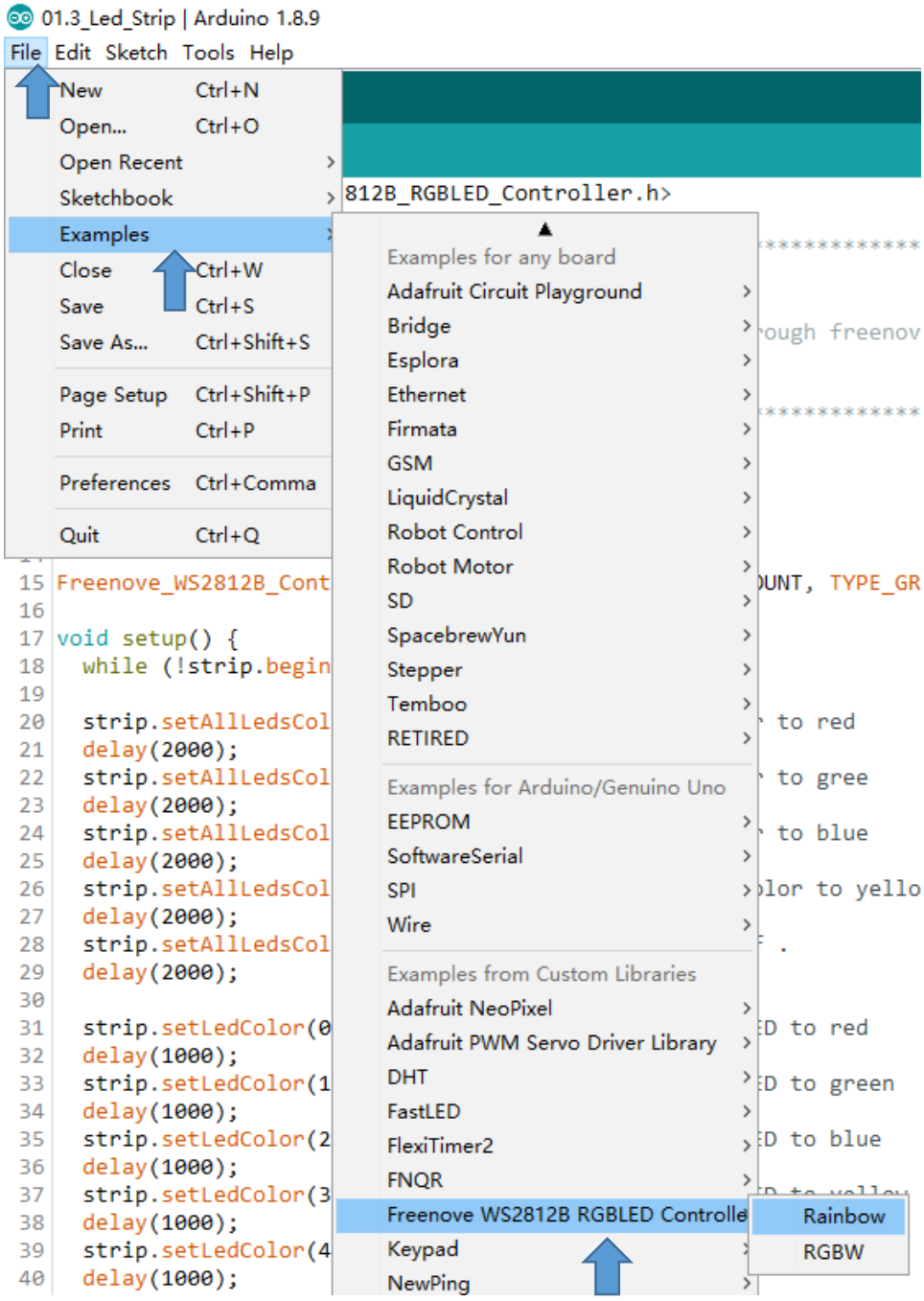
After you complete setting color, just add **strip.show()**

strip.Wheel(value) value can be 0~255.

Different values correspond to different colors. You can take **strip.Wheel(value)** as a rgb value to use in the

code, like `strip.setAllLedsColorData(strip.Wheel(k))`.

You can find some examples of the library below:



1.4 Integrate functions

Now we will write a code, which include all previous functions.

Code

01.4.1_Integrate_Functions

Upload code in Sketches\01.4.1_Integrate_Functions. **Don't separate the files in the folder.**

Then put your car in a place with enough space to move. Turn ON the power of car.

```
#include "Freenove_WS2812B_RGBLED_Controller.h"

#define I2C_ADDRESS   0x20 //define I2C address of LEDs
#define LEDS_COUNT    10   //it defines number of LEDs
//define motor pins
#define PIN_DIRECTION_LEFT  4
#define PIN_DIRECTION_RIGHT 3
#define PIN_MOTOR_PWM_LEFT  6
#define PIN_MOTOR_PWM_RIGHT 5

#define PIN_BATTERY      A0
#define PIN_BUZZER       A0
#define MOTOR_PWM_DEAD    10
float batteryVoltage = 0;
bool isBuzzered = false;

Freenove_WS2812B_Controller strip(I2C_ADDRESS, LEDS_COUNT, TYPE_GRB); //initialization

void setup() {
  while (!strip.begin()); //judge if initialization successes

  Serial.begin(9600);  //set baud rate
  pinsSetup();         //set pin mode (output or input )

  strip.setAllLedsColor(0x00FF00); //set all LED color to green
  motorRun(100, 100);   //Car move forward
  alarm(4, 1);          //4 beat, 1 repeat
  resetCarAction();     //Stop the car and turn off the buzzer

  strip.setAllLedsColor(0xFF0000); //Set all LED color to red
  motorRun(-100, -100); //Car move back
  setBuzzer(true);      //turn on buzzer
  delay(1000);
  resetCarAction();     //Stop the car and turn off the buzzer
```

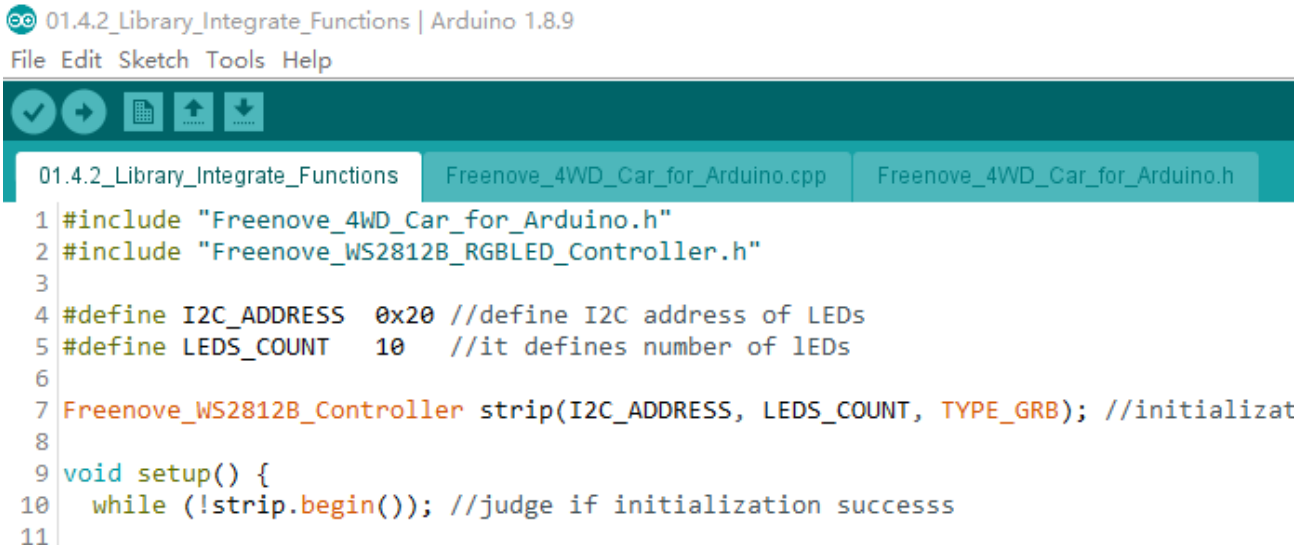
```
}

void loop() {
  if (getBatteryVoltage() == true) {
    Serial.print("voltage: ");
    Serial.println(batteryVoltage);
  }

  int r = random(0, 255);  //a random value between 0~255
  int g = random(0, 255);  //a random value between 0~255
  int b = random(0, 255);  //a random value between 0~255
  strip.setAllLedsColor(r, g, b); //Set all LED color with current r, g, b value
  delay(1000);
}

void pinsSetup() {
  pinMode(PIN_DIRECTION_LEFT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_LEFT, OUTPUT);
  pinMode(PIN_DIRECTION_RIGHT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_RIGHT, OUTPUT);
  setBuzzer(false); //Turn off the Buzzer
}

void motorRun(int speedl, int speedr) {
  int dirL = 0, dirR = 0;
  if (speedl > 0) {
    dirL = 0;
  }
  else {
    dirL = 1;
    speedl = -speedl;
  }

  if (speedr > 0) {
    dirR = 1;
  }
  else {
    dirR = 0;
    speedr = -speedr;
  }

  speedl = constrain(speedl, 0, 255);
  speedr = constrain(speedr, 0, 255);

```

```
79    if (abs(speedl) < MOTOR_PWM_DEAD && abs(speedr) < MOTOR_PWM_DEAD) {
80      speedl = 0;
81      speedr = 0;
82    }
83    digitalWrite(PIN_DIRECTION_LEFT, dirL);
84    digitalWrite(PIN_DIRECTION_RIGHT, dirR);
85    analogWrite(PIN_MOTOR_PWM_LEFT, speedl);
86    analogWrite(PIN_MOTOR_PWM_RIGHT, speedr);
87  }
88
89  bool getBatteryVoltage() {
90    if (!isBuzzered) {
91      pinMode(PIN_BATTERY, INPUT);
92      int batteryADC = analogRead(PIN_BATTERY);
93      if (batteryADC < 614)   // 3V/12V ,Voltage read: <2.1V/8.4V
94      {
95        batteryVoltage = batteryADC / 1023.0 * 5.0 * 4;
96        return true;
97      }
98    }
99    return false;
100 }
101
102 void setBuzzer(bool flag) {
103   isBuzzered = flag;
104   pinMode(PIN_BUZZER, flag);
105   digitalWrite(PIN_BUZZER, flag);
106 }
107
108 void alarm(u8 beat, u8 repeat) {
109   beat = constrain(beat, 1, 9);
110   repeat = constrain(repeat, 1, 255);
111   for (int j = 0; j < repeat; j++) {
112     for (int i = 0; i < beat; i++) {
113       setBuzzer(true);
114       delay(100);
115       setBuzzer(false);
116       delay(100);
117     }
118     delay(500);
119   }
120 }
121
122 void resetCarAction() {
```

```
123     motorRun(0, 0);
124     setBuzzer(false);
125 }
```

01.4.2_Library_Integrate_Functions

Upload code in sketches\01.4.2_Library_Integrate_Functions, which has the same function with code 01.4.1_Integrate_Functions. The only difference is that we write some codes into library. We will use this library later. The library will make programing more efficient.



The main code is as below. The Freenove_4WD_Car_for_Arduino.h has defined some pins, variables and functions. We can use them directly. We do not need redefine them again.

```
1   #include "Freenove_4WD_Car_for_Arduino.h"
2   #include "Freenove_WS2812B_RGBLED_Controller.h"
3
4   #define I2C_ADDRESS  0x20 //define I2C address of LEDs
5   #define LEDS_COUNT   10   //it defines number of LEDs
6   Freenove_WS2812B_Controller strip(I2C_ADDRESS, LEDS_COUNT, TYPE_GRB); //initialization
7
8   void setup() {
9     while (!strip.begin()); //judge if initialization successes
10    Serial.begin(9600);  //set baud rate
11    pinsSetup();          //set pin mode (output or input )
12
13    strip.setAllLedsColor(0x00FF00); //set all LED color to green
14    motorRun(100, 100);   //Car move forward
15    alarm(4, 1);          //4 beat, 1 repeat
16    resetCarAction();     //Stop the car and turn off the buzzer
17
18    strip.setAllLedsColor(0xFF0000); //Set all LED color to red
19    motorRun(-100, -100); //Car move back
```

```
20    setBuzzer(true);      //turn on buzzer
21    delay(1000);
22    resetCarAction();     //Stop the car and turn off the buzzer
23  }
24
25  void loop() {
26    if (getBatteryVoltage() == true) {
27      Serial.print("voltage: ");
28      Serial.println(batteryVoltage);
29    }
30    int r = random(0, 255);  //a random value between 0~255
31    int g = random(0, 255);  //a random value between 0~255
32    int b = random(0, 255);  //a random value between 0~255
33    strip.setAllLedsColor(r, g, b); //Set all LED color with current r, g, b value
34    delay(1000);
35  }
```

In this code, Freenove_4WD_Car_for_Arduino library is used.

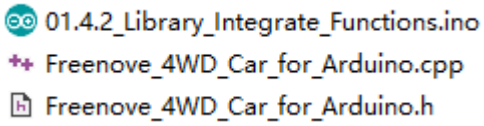
```
#include "Freenove_4WD_Car_for_Arduino.h"
```

And you can see, we did not define pins and functions in the main. But we can use them.

Now let's learn a new skill, using library.

Open the directory Sketches\01.4.2_Library_Integrate_Functions. You will see .cpp and .h file. They are **library files**.

When use the library in new code, you just need put this two files into the same directory of your code.



This library will be used latter, which will make programing more efficient.

Now we will introduce content in the library.

First, check **Freenove_4WD_Car_for_Arduino.h**.

This file defines some pins and function.

After you write the code `#include "Freenove_4WD_Car_for_Arduino.h"`

You will be able to use the pins define and functions in Freenove_4WD_Car_for_Arduino.h. And you don't need to define the pins and build the functions any more in your code.

```
1  // Freenove_4WD_Car_for_Arduino.h
2  #ifndef _FREENOVE_4WD_CAR_FOR_ARDUINO_h
3  #define _FREENOVE_4WD_CAR_FOR_ARDUINO_h
4  #if defined(ARDUINO) && ARDUINO >= 100
5      #include "arduino.h"
6  #else
7      #include "WProgram.h"
```

```
8  #endif
9  #define PIN_SERVO          2
10 //define motor pin
11 #define PIN_DIRECTION_LEFT 4
12 #define PIN_DIRECTION_RIGHT    3
13 #define PIN_MOTOR_PWM_LEFT 6
14 #define PIN_MOTOR_PWM_RIGHT    5
15
16 //define ultrasonic module pin
17 #define PIN_SONIC_TRIG     7
18 #define  PIN_SONIC_ECHO        8
19
20 #define PIN_IRREMOTE_RECV  9
21 //define SPI pin
22 #define PIN_SPI_CE             9
23 #define PIN_SPI_CSN            10
24 #define PIN_SPI_MOSI       11
25 #define PIN_SPI_MISO       12
26 #define PIN_SPI_SCK            13
27
28 #define PIN_BATTERY            A0
29 #define PIN_BUZZER             A0
30
31 //define tracking sensor pin
32 #define PIN_TRACKING_LEFT  A1
33 #define PIN_TRACKING_CENTER    A2
34 #define PIN_TRACKING_RIGHT A3
35 #define MOTOR_PWM_DEAD     5
36
37 //define functions
38 // all following content are used in integrated code above
39 extern float batteryVoltage;
40 void pinsSetup();
41 void motorRun(int speedl, int speedr);
42 bool getBatteryVoltage();
43 void setBuzzer(bool flag);
4  void alarm(u8 beat, u8 repeat);
45 void resetCarAction();
46 #endif
```

Then check **Freenove_4WD_Car_for_Arduino.cpp**.

Its name should be the same as with .h file. We define functions in .h file and implement its content in .cpp. You can see how the function work.

```
1  #include "Freenove_4WD_Car_for_Arduino.h"
```

```
float batteryVoltage = 0;
bool isBuzzered = false;

void pinsSetup() {
  //define motor pin
  pinMode(PIN_DIRECTION_LEFT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_LEFT, OUTPUT);
  pinMode(PIN_DIRECTION_RIGHT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_RIGHT, OUTPUT);

  //define ultrasonic module pin
  pinMode(PIN_SONIC_TRIG, OUTPUT);
  pinMode(PIN_SONIC_ECHO, INPUT);

  //define tracking sensor pin
  pinMode(PIN_TRACKING_LEFT, INPUT);
  pinMode(PIN_TRACKING_RIGHT, INPUT);
  pinMode(PIN_TRACKING_CENTER, INPUT);
  setBuzzer(false); // turn off buzzer
}

void motorRun(int speedl, int speedr) {
  int dirL = 0, dirR = 0;
  if (speedl > 0) {
    dirL = 0;
  }
  else {
    dirL = 1;
    speedl = -speedl;
  }

  if (speedr > 0) {
    dirR = 1;
  }
  else {
    dirR = 0;
    speedr = -speedr;
  }
  speedl = constrain(speedl, 0, 255); // speedl absolute value should be within 0~255
  speedr = constrain(speedr, 0, 255); // speedr absolute value should be within 0~255
  digitalWrite(PIN_DIRECTION_LEFT, dirL);
  digitalWrite(PIN_DIRECTION_RIGHT, dirR);
  analogWrite(PIN_MOTOR_PWM_LEFT, speedl);
  analogWrite(PIN_MOTOR_PWM_RIGHT, speedr);
```

```
}

bool getBatteryVoltage() {
  if (!isBuzzered) {
    pinMode(PIN_BATTERY, INPUT);
    int batteryADC = analogRead(PIN_BATTERY);
    if (batteryADC < 614)      // 3V/12V ,Voltage read: <2.1V/8.4V
    {
      batteryVoltage = batteryADC / 1023.0 * 5.0 * 4;
      return true;
    }
  }
  return false;
}
void setBuzzer(bool flag) {
  isBuzzered = flag;
  pinMode(PIN_BUZZER, flag);
  digitalWrite(PIN_BUZZER, flag);
}
void alarm(u8 beat, u8 repeat) {
  beat = constrain(beat, 1, 9);
  repeat = constrain(repeat, 1, 255);
  for (int j = 0; j < repeat; j++) {
    for (int i = 0; i < beat; i++) {
      setBuzzer(true);
      delay(100);
      setBuzzer(false);
      delay(100);
    }
    delay(500);
  }
}
void resetCarAction() {
  motorRun(0, 0);
  setBuzzer(false);
}
```

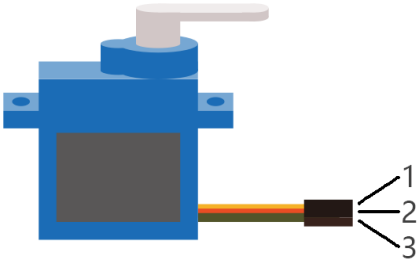
Chapter 2 Obstacle avoidance

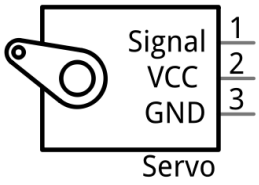
In this chapter, we will introduce servo and ultrasonic module. And then make a car that can automatically obstacles.

If you have any concerns, please feel free to contact us via support@freenove.com

2.1 Servo

Servo is a compact package which consists of a DC Motor, a set of reduction gears to provide torque, a sensor and control circuit board. Most Servos only have a 180-degree range of motion via their "horn". Servos can output higher torque than a simple DC Motor alone and they are widely used to control motion in model cars, model airplanes, robots, etc. Servos have three wire leads which usually terminate to a male or female 3-pin plug. Two leads are for electric power: Positive (2-VCC, Red wire), Negative (3-GND, Brown wire), and the signal line (1-Signal, Orange wire) as represented in the Servo provided in your Kit.





We will use a 50Hz PWM signal with a duty cycle in a certain range to drive the servo. The lasting time 0.5ms-2.5ms of PWM single cycle high level corresponds to the servo angle 0 degrees - 180 degrees linearly. Part of the corresponding values are as follows:

High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	90 degree
2ms	135 degree
2.5ms	180 degree

When you change the servo signal, the servo will rotate to the designated position.

Code

02.1_Servo

Upload code in Sketches\02.1_Servo. Then turn on the power of car.

Then you will see servo sweep among 45°, 90° and 135°.

The code is below:

```
1  #include "Servo.h"      //servo library
2
```

```
#define PIN_SERVO  2   //define servo pin

Servo;            //create servo object to control a servo
char servoOffset = 0;   // change the value to Calibrate servo

void setup() {
  servo.attach(PIN_SERVO);        //initialize servo
  servo.write(90 + servoOffset);  //Calibrate servo
}

void loop() {
  servo.write(45);      //make servo rotate to 45°
  delay(1000);          //delay 1000ms

  servo.write(90);      //make servo rotate to 90°
  delay(1000);

  servo.write(135);    //make servo rotate to 135°
  delay(1000);

  servo.write(90);      //make servo rotate to 90°
  delay(1000);
}

```

Servo library

There are some functions of this library:

servo.attach(pin)

Attach the Servo variable to a pin.

pin: servo pin it is an initialization step to control a servo.

servo.write(angle)

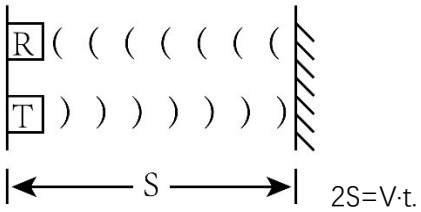
Writes a value to the servo, controlling the shaft accordingly.

angle: the value to write to the servo, from 0 to 180

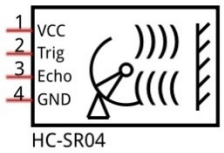
For more details, please refer to: https://www.arduino.cc/en/Reference/Servo

2.2 Ultrasonic ranging module

The Ultrasonic Ranging Module uses the principle that ultrasonic waves will reflect when they encounter any obstacles. This is possible by counting the time interval between when the ultrasonic wave is transmitted to when the ultrasonic wave reflects back after encountering an obstacle. Time interval counting will end after an ultrasonic wave is received, and the time difference (delta) is the total time of the ultrasonic wave's journey from being transmitted to being received. Because the speed of sound in air is a constant, and is about v=340m/s, we can calculate the distance between the Ultrasonic Ranging Module and the obstacle: s=vt/2.



The HC-SR04 Ultrasonic Ranging Module integrates a both an ultrasonic transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into high frequency (beyond human hearing) sound waves (mechanical energy) and the function of the receiver is opposite of this. The picture and the diagram of the HC SR04 Ultrasonic Ranging Module are shown below:

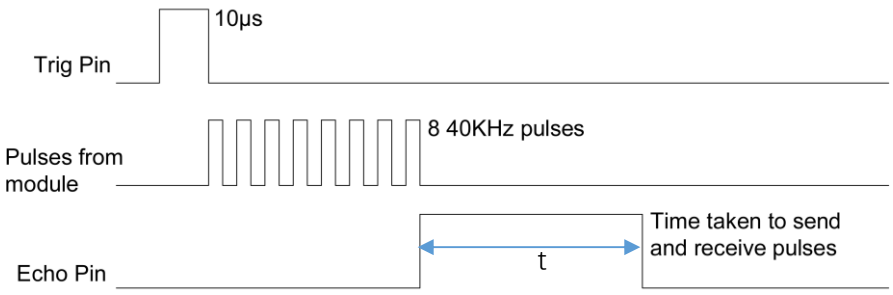


Pin description:

Pin name	Pin number	Description
Vcc	1	Positive electrode of power supply, the voltage is 5V
Trig	2	Triger pin
Echo	3	Echo pin
Gnd	4	Negative electrode of power supply

Instructions for use:

Output a high-level pulse in Trig pin lasting for least 10uS, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled **up**. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled **down**. The duration of high level in the Echo pin is the total time of the ultrasonic wave from transmitting to receiving, **s=vt/2**. This is done constantly.



Code

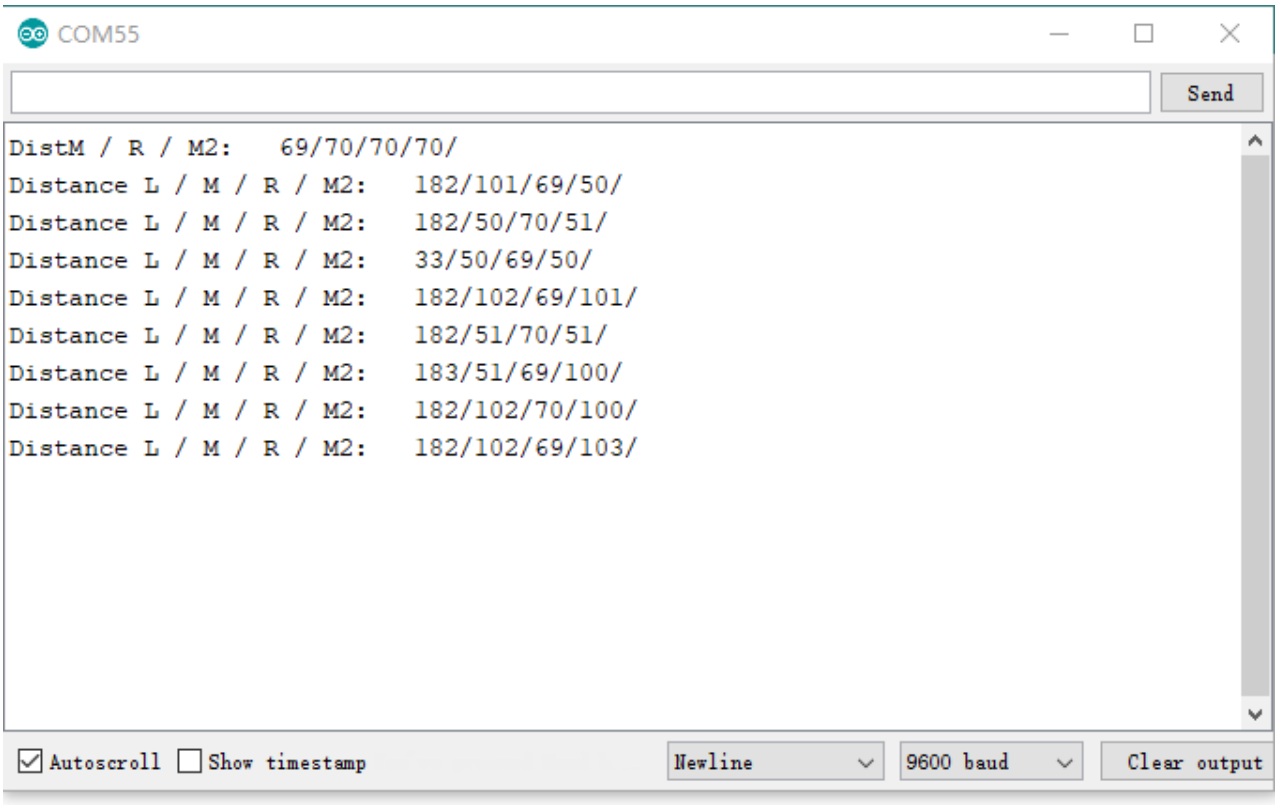
02.2_Ultrasonic_Ranging

This project is used to set servo different angles and detect obstacle distance at each angle.

Upload code in Sketches\02.2_Ultrasonic_Ranging.

Then turn on the power of car. Then open the serial monitor.

Then you will see servo sweep among 45°, 90°, 135°. And ultrasonic module detects different distance at different angles. All the distance values are printed below:



The code is below:

```
#include "Servo.h"              //include servo library

#define PIN_SERVO      2        //define servo pin

#define PIN_SONIC_TRIG    7     //define Trig pin
#define PIN_SONIC_ECHO    8     //define Echo pin

#define MAX_DISTANCE    300   //cm
#define SONIC_TIMEOUT   (MAX_DISTANCE*60) // calculate timeout
#define SOUND_VELOCITY  340  //sound Velocity: 340m/s

Servo servo;              //create servo object
char servoOffset = 0;     //change the value to Calibrate servo
u8 distance[4];           //define an array with type u8(same to unsigned char)

void setup() {
  Serial.begin(9600);
  pinMode(PIN_SONIC_TRIG, OUTPUT);// set trigPin to output mode
  pinMode(PIN_SONIC_ECHO, INPUT); // set echoPin to input mode
  servo.attach(PIN_SERVO);        //initialize servo
  servo.write(90 + servoOffset);  // change servoOffset to Calibrate servo
}
void loop() {
  servo.write(45);
  delay(1000);
  distance[0] = getSonar();   //get ultrasonic value and save it into distance[0]

  servo.write(90);
  delay(1000);
  distance[1] = getSonar();

  servo.write(135);
  delay(1000);
  distance[2] = getSonar();

  servo.write(90);
  delay(1000);
  distance[3] = getSonar();

  Serial.print("Distance L / M / R / M2:   ");  //Left/Middle/Right/Middle2
  for (int i = 0; i < 4; i++) {
    Serial.print(distance[i]);     //print ultrasonic in 45° , 90° , 135° , 90°
    Serial.print("/");
```

```
44    }
45    Serial.print('\n');  //next content will be printed in new line
46  }
47
48  float getSonar() {
49    unsigned long pingTime;
50    float distance;
51    digitalWrite(PIN_SONIC_TRIG, HIGH); // make trigPin output high level lasting for 10 μs to
52  triger HC_SR04,
53    delayMicroseconds(10);
54    digitalWrite(PIN_SONIC_TRIG, LOW);
55    pingTime = pulseIn(PIN_SONIC_ECHO, HIGH, SONIC_TIMEOUT); // Wait HC-SR04 returning to the
56  high level and measure out this waiting time
57    if (pingTime != 0)
58      distance = (float)pingTime * SOUND_VELOCITY / 2 / 10000; // calculate the distance
59  according to the time
60    else
61      distance = MAX_DISTANCE;
62    return distance; // return the distance value
63  }
```

```
#define MAX_DISTANCE    300   //cm
#define SONIC_TIMEOUT   (MAX_DISTANCE*60) // calculate timeout
#define SOUND_VELOCITY  340  //sound Velocity: 340m/s
```

First calculate t for max distance.

340 (t/10000000)/2= MAX_DISTANCE /100 t= 58.8*MAX_DISTANCE (Unit of t is μs)

We set timeout to MAX_DISTANCE*60, a little larger, since the maximum distance of ultrasonic module is much larger.

```
pingTime = pulseIn(PIN_SONIC_ECHO, HIGH, SONIC_TIMEOUT); // Wait HC-SR04 returning to the high level
```

If time of high level lasting time of echo is larger than SONIC_TIMEOUT, it will return 0. Then pingTime=0.

pulseIn(pin, value, timeout)

pin: the number of the Arduino pin on which you want to read the pulse.

value: type of pulse to be read: either HIGH or LOW.

timeout (optional): the number of microseconds to wait for the pulse to start; default is one second.

For more details, please refer to:

https://www.arduino.cc/reference/en/language/functions/advanced-io/pulsein/

Array

An array is a collection of variables that are accessed with an index number.

Define an array in Arduino.

Data type Array name [Number of elements]

`u8 distance[3];` define an array named distance, its data type is u8, which has 3 elements.

The index starts from 0.

u8= **unsigned char** Its range is 0~2^8, namely 0~255.

Range of u16 is 0~65535.

For more details about array, please refer to:
https://www.arduino.cc/reference/en/language/variables/data-types/array/

2.3 Automatic Obstacle Avoidance Car

Code

02.3.1_Automatic_Obstacle_Avoidance

Upload code in Sketches\02.3.1_Automatic_Obstacle_Avoidance.

Then disconnect USB cable. Turn on the power of car. And put the car in a place with enough space to move.

The code is below:

```
#include <Servo.h>
#define PIN_SERVO           2

#define PIN_DIRECTION_LEFT  4
#define PIN_DIRECTION_RIGHT 3
#define PIN_MOTOR_PWM_LEFT  6
#define PIN_MOTOR_PWM_RIGHT 5

#define PIN_SONIC_TRIG      7
#define PIN_SONIC_ECHO      8

#define PIN_BATTERY         A0

#define OBSTACLE_DISTANCE   40
#define OBSTACLE_DISTANCE_LOW 15

#define MAX_DISTANCE    300   //cm
#define SONIC_TIMEOUT   (MAX_DISTANCE*60)
#define SOUND_VELOCITY    340   //soundVelocity: 340m/s

Servo servo;
char servoOffset = 0;
int speedOffset;//batteryVoltageCompensationToSpeed

void setup() {
  pinMode(PIN_DIRECTION_LEFT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_LEFT, OUTPUT);
  pinMode(PIN_DIRECTION_RIGHT, OUTPUT);
  pinMode(PIN_MOTOR_PWM_RIGHT, OUTPUT);

  pinMode(PIN_SONIC_TRIG, OUTPUT);// set trigPin to output mode
  pinMode(PIN_SONIC_ECHO, INPUT); // set echoPin to input mode
  servo.attach(PIN_SERVO);
  calculateVoltageCompensation();
}
```

```
36
37  void loop() {
38    updateAutomaticObstacleAvoidance();
39  }
40
41  void updateAutomaticObstacleAvoidance() {
42    int distance[3], tempDistance[3][5], sumDisntance;
43    static u8 leftToRight = 0, servoAngle = 0, lastServoAngle = 0;  //
44    const u8 scanAngle[2][3] = { {150, 90, 30}, {30, 90, 150} };
45
46    for (int i = 0; i < 3; i++)
47    {
48      servoAngle = scanAngle[leftToRight][i];
49      servo.write(servoAngle);
50      if (lastServoAngle != servoAngle) {
51        delay(130);
52      }
53      lastServoAngle = servoAngle;
54      for (int j = 0; j < 5; j++) {
55        tempDistance[i][j] = getSonar();
56        delayMicroseconds(2 * SONIC_TIMEOUT);
57        sumDisntance += tempDistance[i][j];
58      }
59      if (leftToRight == 0) {
60        distance[i] = sumDisntance / 5;
61      }
62      else {
63        distance[2 - i] = sumDisntance / 5;
64      }
65      sumDisntance = 0;
66    }
67    leftToRight = (leftToRight + 1) % 2;
68
69    if (distance[1] < OBSTACLE_DISTANCE) {        //Too little distance ahead
70      if (distance[0] > distance[2] && distance[0] > OBSTACLE_DISTANCE) {     //Left distance is
71  greater than right distance
72        motorRun(-(150 + speedOffset), -(150 + speedOffset)); //Move back
73        delay(100);
74        motorRun(-(150 + speedOffset), (150 + speedOffset));
75      }
76      else if (distance[0] < distance[2] && distance[2] > OBSTACLE_DISTANCE)
77  {                   //Right distance is greater than left distance
78        motorRun(-(150 + speedOffset), -(150 + speedOffset)); //Move back
79        delay(100);
```

```
80        motorRun((150 + speedOffset), -(150 + speedOffset));
81      }
82      else {                      //Get into the dead corner, move back, then turn.
83        motorRun(-(150 + speedOffset), -(150 + speedOffset));
84        delay(100);
85        motorRun(-(150 + speedOffset), (150 + speedOffset));
86      }
87    }
88    else {                        //No obstacles ahead
89      if (distance[0] <  OBSTACLE_DISTANCE_LOW) {      //Obstacles on the left front.
90        motorRun(-(150 + speedOffset), -(150 + speedOffset)); //Move back
91        delay(100);
92        motorRun((180 + speedOffset), (50 + speedOffset));
93      }
94      else if (distance[2] <  OBSTACLE_DISTANCE_LOW) {     //Obstacles on the right front.
95        motorRun(-(150 + speedOffset), -(150 + speedOffset)); //Move back
96        delay(100);
97        motorRun((50 + speedOffset), (180 + speedOffset));
98      }
99      else {                        //Cruising
100       motorRun((80 + speedOffset), (80 + speedOffset));
101     }
102   }
103 }
104
105 float getSonar() {
106   unsigned long pingTime;
107   float distance;
108   digitalWrite(PIN_SONIC_TRIG, HIGH); // make trigPin output high level lasting for 10μs to
109 triger HC_SR04,
110   delayMicroseconds(10);
111   digitalWrite(PIN_SONIC_TRIG, LOW);
112   pingTime = pulseIn(PIN_SONIC_ECHO, HIGH, SONIC_TIMEOUT); // Wait HC-SR04 returning to the
113 high level and measure out this waiting time
114   if (pingTime != 0)
115     distance = (float)pingTime * SOUND_VELOCITY / 2 / 10000; // calculate the distance
116 according to the time
117   else
118     distance = MAX_DISTANCE;
119   return distance; // return the distance value
120 }
121
122 void calculateVoltageCompensation() {
123   float voltageOffset = 8.4 - getBatteryVoltage();
```

```
124     speedOffset = voltageOffset * 20;
125   }
126
127   void motorRun(int speedl, int speedr) {
128     int dirL = 0, dirR = 0;
129     if (speedl > 0) {
130       dirL = 0;
131     }
132     else {
133       dirL = 1;
134       speedl = -speedl;
135     }
136
137     if (speedr > 0) {
138       dirR = 1;
139     }
140     else {
141       dirR = 0;
142       speedr = -speedr;
143     }
144     digitalWrite(PIN_DIRECTION_LEFT, dirL);
145     digitalWrite(PIN_DIRECTION_RIGHT, dirR);
146     analogWrite(PIN_MOTOR_PWM_LEFT, speedl);
147     analogWrite(PIN_MOTOR_PWM_RIGHT, speedr);
148   }
149
150
151   float getBatteryVoltage() {
152     pinMode(PIN_BATTERY, INPUT);
153     int batteryADC = analogRead(PIN_BATTERY);
154     float batteryVoltage = batteryADC / 1023.0 * 5.0 * 4;
155     return batteryVoltage;
156   }
```

We use analogWrite(PIN_MOTOR, PWM value) to make motor work. The PWM value represents different speeds when the voltage of battery changes. So we set a speedOffset to compensate the difference.

```
1   void calculateVoltageCompensation() {
2     float voltageOffset = 8.4 - getBatteryVoltage();
3     speedOffset = voltageOffset * 20;
4   }
```

20 is tested value. You can try to test what is the most proper value.

We need control servo to 150°, 90°, 30°, 90°, 150°…
Get 5 ultrasonic module values (distance) at one angle. Then calculate the average distance.

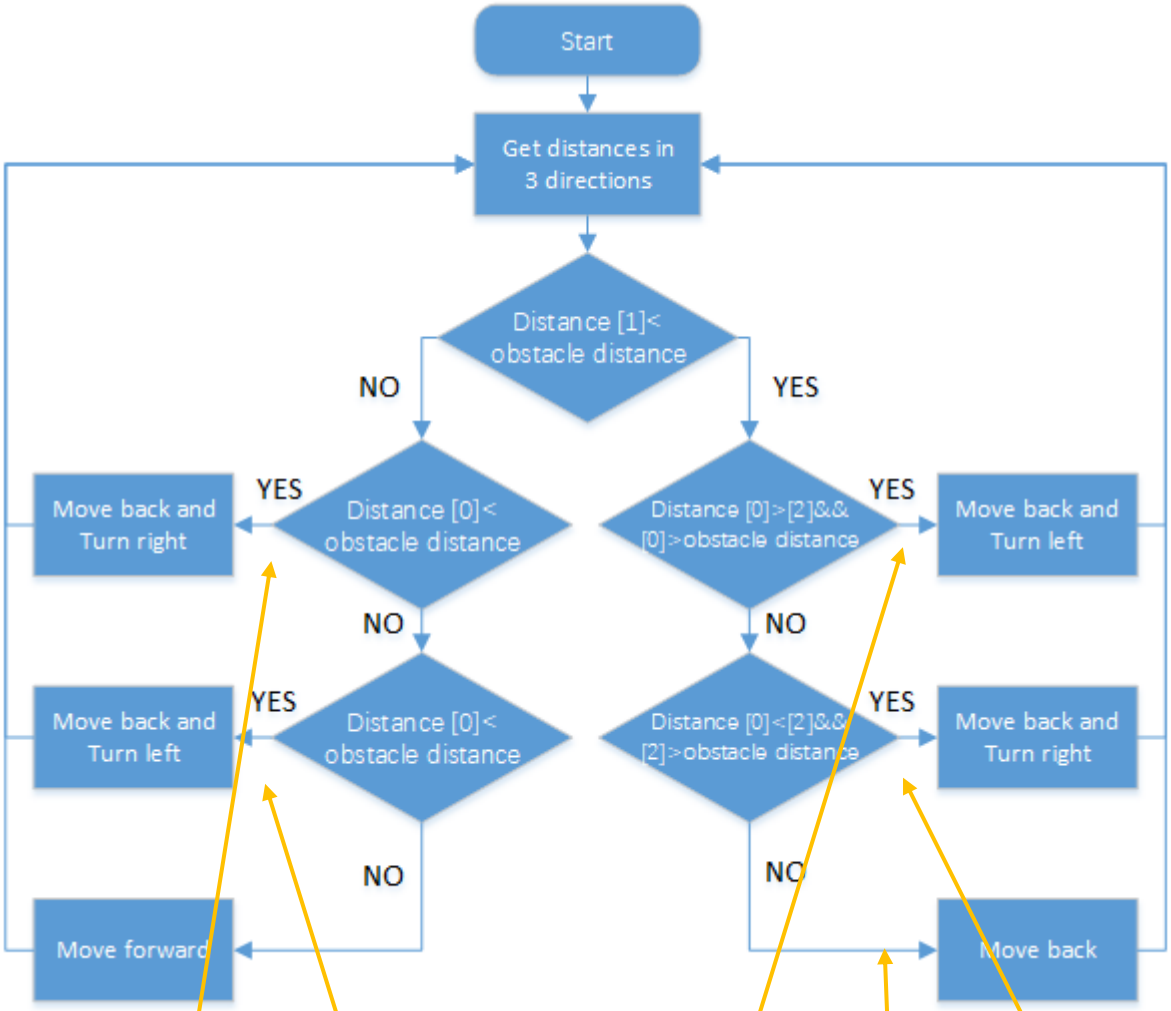
```
int distance[3], tempDistance[3][5], sumDisntance;
  static u8 leftToRight = 0, servoAngle = 0, lastServoAngle = 0;  //
  const u8 scanAngle[2][3] = { {150, 90, 30}, {30, 90, 150} };
  for (int i = 0; i < 3; i++)
  {
    servoAngle = scanAngle[leftToRight][i];
    servo.write(servoAngle);
    if (lastServoAngle != servoAngle) {
      delay(130);
    }
    lastServoAngle = servoAngle;
    for (int j = 0; j < 5; j++) {
      tempDistance[i][j] = getSonar();
      delayMicroseconds(2 * SONIC_TIMEOUT);
      sumDisntance += tempDistance[i][j];
    }
    if (leftToRight == 0) {
      distance[i] = sumDisntance / 5;
    }
    else {
      distance[2 - i] = sumDisntance / 5;
    }
    sumDisntance = 0;
  }
  leftToRight = (leftToRight + 1) % 2;

```

Then make the car react according to the distances above. About its logic, please refer to flow chart.

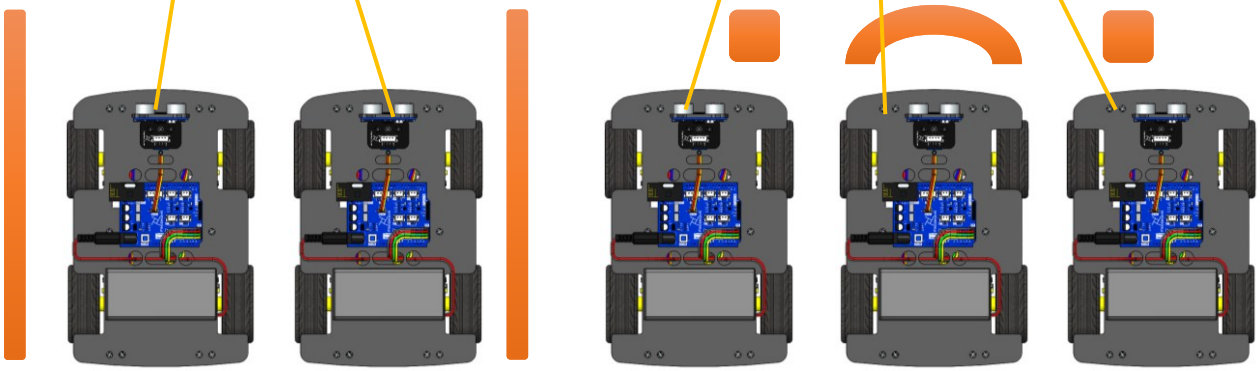
```
  //make the car react according to the distances
  if (distance[1] < OBSTACLE_DISTANCE) {        //obstacle in front
    if (distance[0] > distance[2] && distance[0] > OBSTACLE_DISTANCE) {     //Left distance is
greater than right distance
      motorRun(-(150 + speedOffset), -(150 + speedOffset)); //Move back a little
      delay(100);
      motorRun(-(150 + speedOffset), (150 + speedOffset));
      delay(50);
    }
    else if (distance[0] < distance[2] && distance[2] > OBSTACLE_DISTANCE)
{                   //Right distance is greater than left distance
      motorRun(-(150 + speedOffset), -(150 + speedOffset)); //Move back a little
      delay(100);
      motorRun((150 + speedOffset), -(150 + speedOffset)); //Turn right
      delay(50);
    }
```

```
    else {                      //Get into the dead corner, move back a little, then spin.
      motorRun(-(150 + speedOffset), -(150 + speedOffset)); //Move back
      delay(100);
      motorRun(-(150 + speedOffset), (150 + speedOffset));   //Turn left
      delay(50);
    }
  }
  else {                        //No obstacles ahead
    if (distance[0] < OBSTACLE_DISTANCE_LOW) {      //Obstacles on the left front.
      motorRun(-(150 + speedOffset), -(150 + speedOffset)); //Move back
      delay(100);
      motorRun((180 + speedOffset), (50 + speedOffset));   //Turn right with large radius
      delay(50);
    }
    else if (distance[2] < OBSTACLE_DISTANCE_LOW) {     //Obstacles on the right front.
      motorRun(-(150 + speedOffset), -(150 + speedOffset)); //Move back
      delay(100);
      motorRun((50 + speedOffset), (180 + speedOffset));   //Turn left with large radius
      delay(50);
    }
    else {                        //Cruising, no obstacle closed
      motorRun((80 + speedOffset), (80 + speedOffset));
    }
  }
}

```

Code logic is as flow chart below:
Distance[0] is the left distance to obstacle.
Distance[1] is the center distance to obstacle.
Distance[2] is the right distance to obstacle.



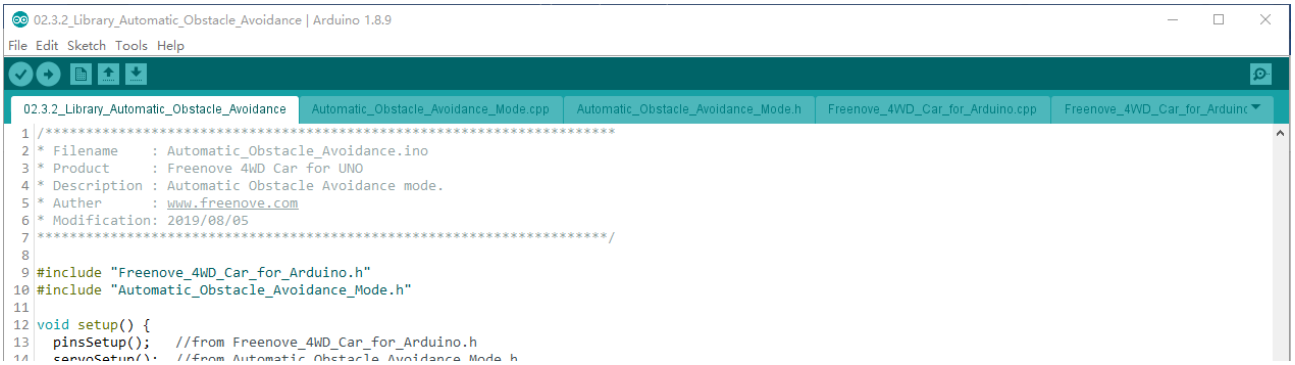
This project consider following



02.3.2_Liberary_Automatic_Obstacle_Avoidance

Code in sketches\02.3.2_Liberary_Automatic_Obstacle_Avoidance includes a library

The code function and content is similar to previous section. There are little difference. Previous code is more simple.



There are 4 labels.

Freenove_4WD_Car_for_Arduino.h has introduced in section 01.4.2_Library_Integrate_Functions

The main label is **02.3.2_Liberary_Automatic_Obstacle_Avoidance.ino**

Its code is below.

```
#include "Freenove_4WD_Car_for_Arduino.h"
#include "Automatic_Obstacle_Avoidance_Mode.h"

void setup() {
  pinsSetup(); //from Freenove_4WD_Car_for_Arduino.h
  servoSetup();//from Automatic_Obstacle_Avoidance_Mode.h
  oa_CalculateVoltageCompensation();//Automatic_Obstacle_Avoidance_Mode.h
}

void loop() {
  updateAutomaticObstacleAvoidance();//Automatic_Obstacle_Avoidance_Mode.h
}
```

The code is very simple. The function from other labels.

Code in label **Automatic_Obstacle_Avoidance_Mode.h**

```
#ifndef _AUTOMATIC_OBSTACLE_AVOIDANCE_h
#define _AUTOMATIC_OBSTACLE_AVOIDANCE_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "arduino.h"
#else
#include "WProgram.h"
#endif
#include <EEPROM.h>
#include "Servo.h"
#include "Freenove_4WD_Car_for_Arduino.h"

```

```
13  #define OA_SERVO_CENTER                     (90)
14  #define OA_SCAN_ANGLE_INTERVAL 50
15  #define OA_SCAN_ANGLE_MIN       (OA_SERVO_CENTER - OA_SCAN_ANGLE_INTERVAL)
16  #define OA_SCAN_ANGLE_MAX       (OA_SERVO_CENTER + OA_SCAN_ANGLE_INTERVAL)
17  #define OA_WAITTING_SERVO_TIME 130
18
19  #define OA_CRUISE_SPEED             (110 + oa_VoltageCompensationToSpeed)
20
21  #define OA_ROTATY_SPEED_LOW         (120 + oa_VoltageCompensationToSpeed)
22  #define OA_ROTATY_SPEED_NORMAL      (150 + oa_VoltageCompensationToSpeed)
23  #define OA_ROTATY_SPEED_HIGH        (180 + oa_VoltageCompensationToSpeed)
24
25  #define OA_TURN_SPEED_LV4           (180 + oa_VoltageCompensationToSpeed)
26  #define OA_TURN_SPEED_LV1           (50 + oa_VoltageCompensationToSpeed )
27
28  #define OA_BACK_SPEED_LOW           (110 + oa_VoltageCompensationToSpeed)
29  #define OA_BACK_SPEED_NORMAL        (150 + oa_VoltageCompensationToSpeed)
30  #define OA_BACK_SPEED_HIGH          (180 + oa_VoltageCompensationToSpeed)
31
32  #define OA_OBSTACLE_DISTANCE        40
33  #define OA_OBSTACLE_DISTANCE_LOW    15
34
35  #define OA_SPEED_OFFSET_PER_V  35
36
37  #define OA_SERVO_OFFSET_ADDR_IN_EEPROM      0
38
39  #define MAX_DISTANCE       300      //cm
40  #define SONIC_TIMEOUT      (MAX_DISTANCE*60)
41  #define SOUND_VELOCITY     340      //soundVelocity: 340m/s
42
43
44  extern Servo servo;
45  extern int servoOffset;
46
47  void servoSetup(); //initialize the servo
48  void setServoOffset(int offset); //Set servo offset
49  void writeServo(u8 n); //set servo to an angle, offset has been considered
50  void writeServoOffsetToEEPROM();
51  void getServoOffsetFromEEPROM();
52
53  float getSonar();
54  void oa_CalculateVoltageCompensation(); // Calculate Voltage Compensation
55  void updateAutomaticObstacleAvoidance();
56  #endif
```

.h file used to define pins, variables and functions.
You can get information of what you can use from .h file.

The detailed implement code of the functions is included in .cpp file.

```
#include "Automatic_Obstacle_Avoidance_Mode.h"

Servo servo;
char servoOffset = 0;
int oa_VoltageCompensationToSpeed;

void servoSetup() {
    getServoOffsetFromEEPROM();
    servo.attach(PIN_SERVO);
    servo.write(90 + servoOffset);
}

void setServoOffset(char offset) {
    servoOffset = offset = constrain(offset, -100, 100);
    servo.write(90 + offset);
}

void writeServo(u8 n)
{
    servo.write(90 + servoOffset);
}

void writeServoOffsetToEEPROM() {
    servo.write(90 + servoOffset);
    EEPROM.write(OA_SERVO_OFFSET_ADDR_IN_EEPROM, servoOffset);
}

void getServoOffsetFromEEPROM() {
    servoOffset = EEPROM.read(OA_SERVO_OFFSET_ADDR_IN_EEPROM);
    servoOffset = constrain(servoOffset, -10, 10);
}

float getSonar() {
    unsigned long pingTime;
    float distance;
    digitalWrite(PIN_SONIC_TRIG, HIGH); // make trigPin output high level lasting for 10μs
to triger HC_SR04,
    delayMicroseconds(10);
    digitalWrite(PIN_SONIC_TRIG, LOW);
    pingTime = pulseIn(PIN_SONIC_ECHO, HIGH, SONIC_TIMEOUT); // Wait HC-SR04 returning to the
```

```
high level and measure out this waiting time
    if (pingTime != 0)
        distance = (float)pingTime * SOUND_VELOCITY / 2 / 10000; // calculate the distance
according to the time
    else
        distance = MAX_DISTANCE;
    return distance; // return the distance value
}

void oa_CalculateVoltageCompensation() {
    getBatteryVoltage();
    float voltageOffset = BAT_VOL_STANDARD - batteryVoltage;
    oa_VoltageCompensationToSpeed = voltageOffset * OA_SPEED_OFFSET_PER_V;
}

void updateAutomaticObstacleAvoidance() {
    int distance[3], tempDistance[3][5], sumDisntance;
    static u8 cnt = 0, servoAngle = 0, lastServoAngle = 0;    //
    if (cnt == 0) {
        for (int i = 0; i < 3; i++) {
            servoAngle = OA_SCAN_ANGLE_MAX - i * OA_SCAN_ANGLE_INTERVAL + servoOffset;
            servo.write(servoAngle);
            if (lastServoAngle != servoAngle) {
                delay(OA_WAITTING_SERVO_TIME);
            }
            lastServoAngle = servoAngle;
            for (int j = 0; j < 5; j++) {
                tempDistance[i][j] = getSonar();
                delayMicroseconds(2 * SONIC_TIMEOUT);
                sumDisntance += tempDistance[i][j];
            }
            distance[i] = sumDisntance / 5;
            sumDisntance = 0;
        }
        cnt++;
    }
    else {
        for (int i = 2; i > 0; i--) {
            servoAngle = OA_SCAN_ANGLE_MAX - i * OA_SCAN_ANGLE_INTERVAL + servoOffset;
            servo.write(servoAngle);
            if (lastServoAngle != servoAngle) {
                delay(OA_WAITTING_SERVO_TIME);
            }
            lastServoAngle = servoAngle;
```

```
85              for (int j = 0; j < 5; j++) {
86                  tempDistance[i][j] = getSonar();
87                  delayMicroseconds(2 * SONIC_TIMEOUT);
88                  sumDisntance += tempDistance[i][j];
89              }
90              distance[i] = sumDisntance / 5;
91              sumDisntance = 0;
92          }
93          cnt = 0;
94      }
95
96      if (distance[1] < OA_OBSTACLE_DISTANCE) {                //Too little distance ahead
97          if (distance[0] > OA_OBSTACLE_DISTANCE || distance[2] > OA_OBSTACLE_DISTANCE) {
98              motorRun(-OA_BACK_SPEED_LOW, -OA_BACK_SPEED_LOW);    //Move back a little
99              delay(100);
100             if (distance[0] > distance[2]) {            //Left distance is greater than
101 right distance
102                 motorRun(-OA_ROTATY_SPEED_LOW, OA_ROTATY_SPEED_LOW);
103             }
104             else {                                      //Right distance is greater than
105 left distance
106                 motorRun(OA_ROTATY_SPEED_LOW, -OA_ROTATY_SPEED_LOW);
107             }
108         }
109         else {                                          //Get into the dead corner, move
110 back a little, then spin.
111             motorRun(-OA_BACK_SPEED_HIGH, -OA_BACK_SPEED_HIGH);
112             delay(100);
113             motorRun(-OA_ROTATY_SPEED_NORMAL, OA_ROTATY_SPEED_NORMAL);
114         }
115     }
116 // following code is different from last section. Last section is more simple
117     else {                                              //No obstacles ahead
118         if (distance[0] < OA_OBSTACLE_DISTANCE) {       //Obstacles on the left front.
119             if (distance[0] < OA_OBSTACLE_DISTANCE_LOW) {    //Very close to the left front
120 obstacle.
121                 motorRun(-OA_BACK_SPEED_LOW, -OA_BACK_SPEED_LOW);    //Move back
122                 delay(100);
123             }
124             motorRun(OA_TURN_SPEED_LV4, OA_TURN_SPEED_LV1);
125         }
126         else if (distance[2] < OA_OBSTACLE_DISTANCE) {      //Obstacles on the right
127 front.
128             if (distance[2] < OA_OBSTACLE_DISTANCE_LOW) {    //Very close to the right
```

```
129  front obstacle.
130                  motorRun(-OA_BACK_SPEED_LOW, -OA_BACK_SPEED_LOW);     //Move back
131                  delay(100);
132              }
133              motorRun(OA_TURN_SPEED_LV1, OA_TURN_SPEED_LV4);
134          }
135          else {                                                        //Cruising
136              motorRun(OA_CRUISE_SPEED, OA_CRUISE_SPEED);
137          }
138      }
139
     }
```

Since the servo may not be installed to 90°, we need adjust it. But we don't want to adjust it every time, so we can save the servoOffset in a place and next time, we can use it directly.. The microcontroller on the Arduino and Genuino boards have 1024 bytes of EEPROM.
It is useful when we use remote control to calibrate the servo.

```
1  void writeServoOffsetToEEPROM() {
2      servo.write(90 + servoOffset);
3      EEPROM.write(OA_SERVO_OFFSET_ADDR_IN_EEPROM, servoOffset);
4  }
5
6  void getServoOffsetFromEEPROM() {
7      servoOffset = EEPROM.read(OA_SERVO_OFFSET_ADDR_IN_EEPROM);
8      servoOffset = constrain(servoOffset, -10, 10);
9  }
```

EEPROM

The microcontroller on the Arduino and Genuino boards have 1024 bytes of EEPROM.
EEPROM.write(address, val); write val to address
value = EEPROM.read(address); read the data saved in address

for example,
EEPROM.write(0, 3);
EEPROM.write(1, 5);
EEPROM.write(2, 8);
Value0= EEPROM.read(0); then Value0 = 3
Value1 = EEPROM.read(2); then Value2 = 8
For more information, refer to https://www.arduino.cc/en/Reference/EEPROM

Other code logic is similar to previous section.

Chapter 3 Line tracking

In this chapter, we will introduce line-tracking sensor and make a line-tracking car.
If you have any concerns, please feel free to contact us via support@freenove.com

3.1 Line tracking sensor

There are three Reflective Optical Sensors on this car. When the infrared light emitted by infrared diode shines on the surface of different objects, the sensor will receive light with different intensities after reflection.
As we know, black objects absorb light better. So when black lines are drawn on the white plane, the sensor can detect the difference. The sensor can also be called Line Tracking Sensor.
Warning:
Reflective Optical Sensor (including Line Tracking Sensor) should be avoided using in environment with infrared interference, like sunlight. Sunlight contains a lot of invisible light such as infrared and ultraviolet. Under environment with intense sunlight, Reflective Optical Sensor cannot work normally.

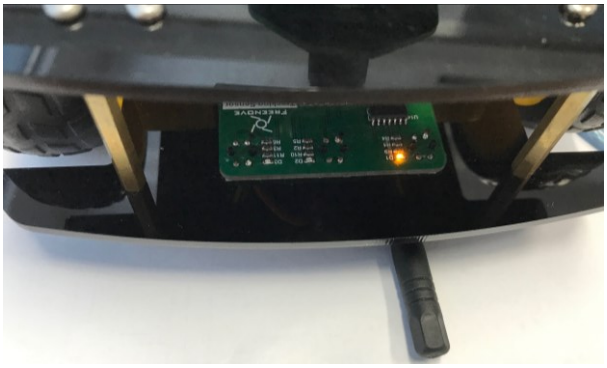
The following table shows the values of all cases when three Tracking Sensors detect objects of different colors. Among them, black objects or no objects were detected to represent 1, and white objects were detected to represent 0.

Left	Middle	Right	Value(binary)	Value(decimal)
0	0	0	000	0
0	0	1	001	1
0	1	0	010	2
0	1	1	011	3
1	0	0	100	4
1	0	1	101	5
1	1	0	110	6
1	1	1	111	7

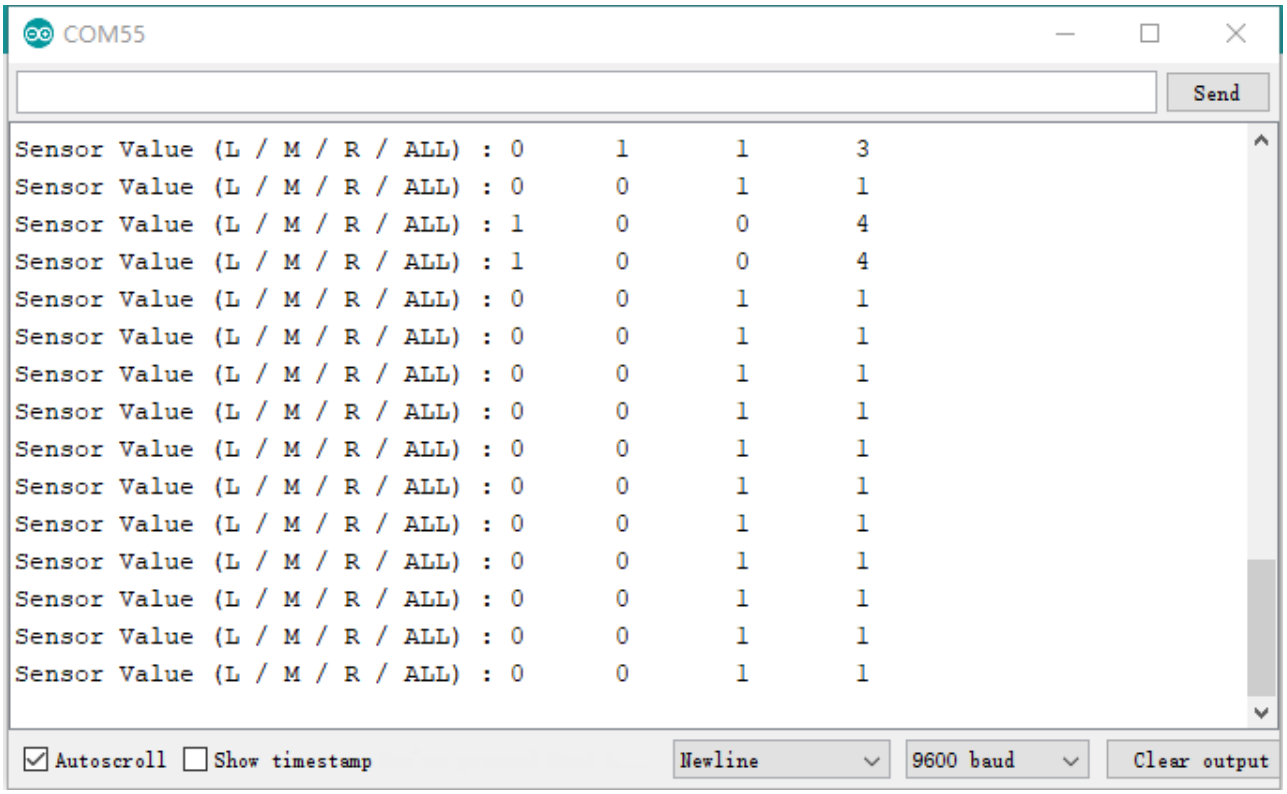
Code

03.1_Tracking_Sensor

Upload code in Sketches\03.1_Tracking_Sensor, turn on the power, and open the serial monitor. Put something black under the sensor. Move it and you will see different LEDs light up.



The monitor is shown below:



The code is below:

```
#define PIN_TRACKING_LEFT   A1
#define PIN_TRACKING_CENTER A2
#define PIN_TRACKING_RIGHT  A3

u8 sensorValue[4];  //define a array, u8 = unsigned char, 0~255.

void setup() {
  Serial.begin(9600); //set baud rate
  pinMode(PIN_TRACKING_LEFT, INPUT); //
  pinMode(PIN_TRACKING_RIGHT, INPUT); //
  pinMode(PIN_TRACKING_CENTER, INPUT); //
}

```

```
14  void loop() {
15    sensorValue[0] = digitalRead(PIN_TRACKING_LEFT);
16    sensorValue[1] = digitalRead(PIN_TRACKING_CENTER);
17    sensorValue[2] = digitalRead(PIN_TRACKING_RIGHT);
18    sensorValue[3] = sensorValue[0] << 2 | sensorValue[1] << 1 | sensorValue[2];
19    Serial.print("Sensor Value (L / M / R / ALL) : ");
20    for (int i = 0; i < 4; i++) {
21      Serial.print(sensorValue[i]);
22      Serial.print('\t');  //means Tab
23    }
24    Serial.print('\n');    //means new line
25    delay(500);
26  }
27
```

Bitwise Operators

There are some Bitwise Operators.

<< (bitshift left)

If sensorValue[0] =1, sensorValue[1]= 1, sensorValue[2]=1

sensorValue[0] << 2 then sensorValue[0]=100,(Binary), namely 4(Decimal)

sensorValue[1] << 1 then sensorValue[0]=010,(Binary), namely 2(Decimal)

sensorValue[2]=001(Binary)

| (bitwise or)

The code turns to: 100 | 010 | 001 =111(Binary), namely 7(Decimal)

>> (bitshift right)

& (bitwise and)

^ (bitwise xor)

~ (bitwise not)

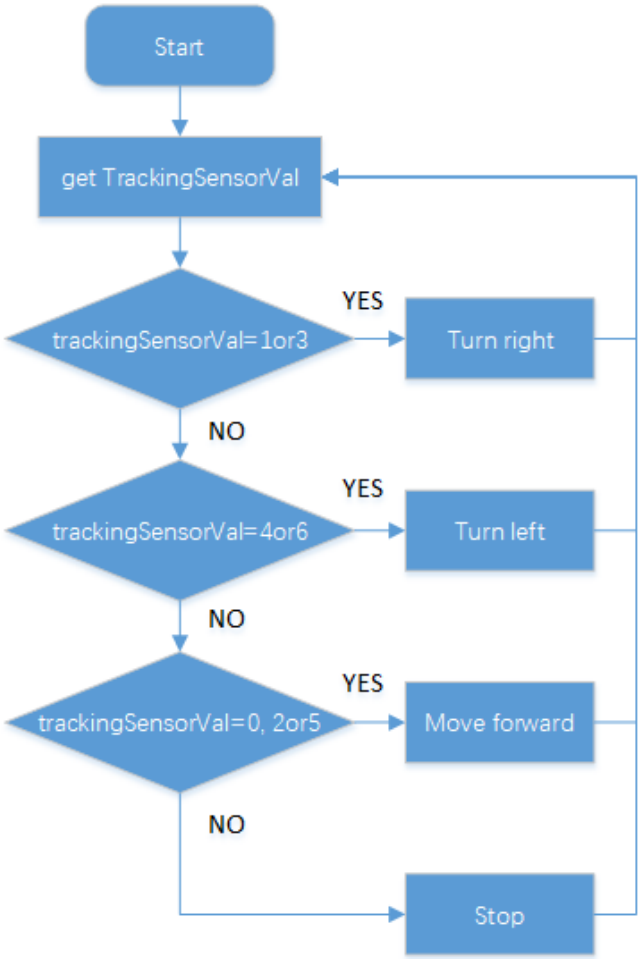
For more information, please refer to: https://www.arduino.cc/reference/

3.2 Line tracking car

The car will make different actions according to the value transmitted by the line-tracking sensor.
When

Left	Middle	Right	Value(binary)	Value(decimal)	Action
0	0	0	000	0	move forward
0	0	1	001	1	Turn Right
0	1	0	010	2	Move Forward
0	1	1	011	3	Turn Right
1	0	0	100	4	Turn Left
1	0	1	101	5	Move Forward
1	1	0	110	6	Turn Left
1	1	1	111	7	Stop

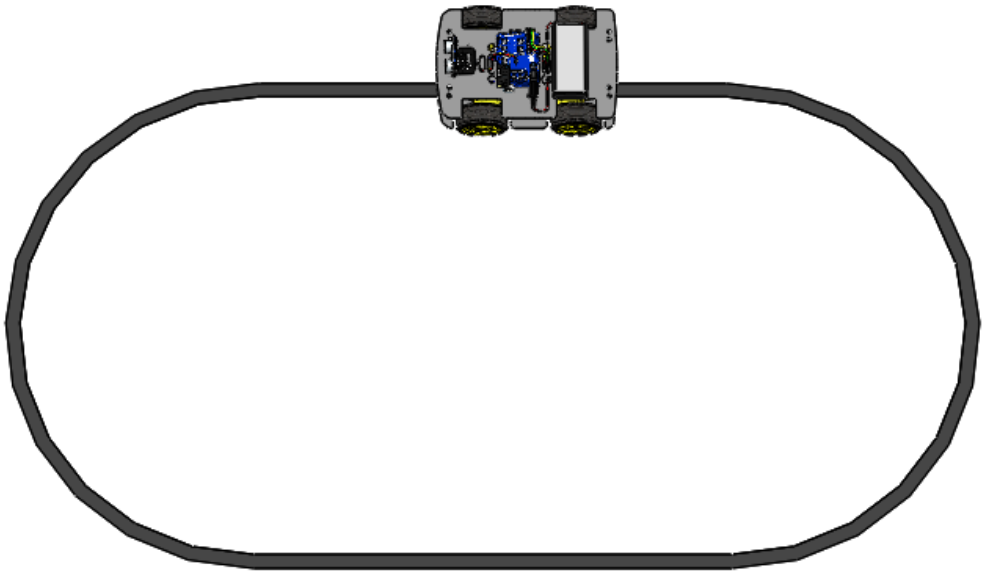
Flow chart of line tracking car is as below:



Code

03.2_Automatic_Tracking_Line

Upload code in Sketches\03.2_Automatic_Tracking_Line, turn on the power. Use a black tape to build a line and then put your car on it as below.



The code is below.

```
#include "Freenove_4WD_Car_for_Arduino.h"

#define TK_STOP_SPEED          0
#define TK_FORWARD_SPEED        (90 + tk_VoltageCompensationToSpeed    )

//define different speed levels
#define TK_TURN_SPEED_LV4       (180 + tk_VoltageCompensationToSpeed   )
#define TK_TURN_SPEED_LV3       (150 + tk_VoltageCompensationToSpeed   )
#define TK_TURN_SPEED_LV2       (-140 + tk_VoltageCompensationToSpeed  )
#define TK_TURN_SPEED_LV1       (-160 + tk_VoltageCompensationToSpeed  )

int tk_VoltageCompensationToSpeed;  //define Voltage Speed Compensation

void setup() {
  pinsSetup(); //set up pins, from Freenove_4WD_Car_for_Arduino.h library
  getTrackingSensorVal();//Calculate Voltage speed Compensation
}

void loop() {
  u8 trackingSensorVal = 0;
  trackingSensorVal = getTrackingSensorVal(); //get sensor value

```

```
  switch (trackingSensorVal)
  {
    case 0:   //000
      motorRun(TK_FORWARD_SPEED, TK_FORWARD_SPEED); //car move forward
      break;
    case 7:   //111
      motorRun(TK_STOP_SPEED, TK_STOP_SPEED); //car stop
      break;
    case 1:   //001
      motorRun(TK_TURN_SPEED_LV4, TK_TURN_SPEED_LV1); //car turn
      break;
    case 3:   //011
      motorRun(TK_TURN_SPEED_LV3, TK_TURN_SPEED_LV2); //car turn right
      break;
    case 2:   //010
    case 5:   //101
      motorRun(TK_FORWARD_SPEED, TK_FORWARD_SPEED);  //car move forward
      break;
    case 6:   //110
      motorRun(TK_TURN_SPEED_LV2, TK_TURN_SPEED_LV3); //car turn left
      break;
    case 4:   //100
      motorRun(TK_TURN_SPEED_LV1, TK_TURN_SPEED_LV4); //car turn right
      break;
    default:
      break;
  }
}

void tk_CalculateVoltageCompensation() {
  getBatteryVoltage(); //from Freenove_4WD_Car_for_Arduino.h library
  float voltageOffset = 7 - batteryVoltage;
  tk_VoltageCompensationToSpeed = 30 * voltageOffset;
}

//when black line on one side is detected, the value of the side will be 0, or the value is 1
u8 getTrackingSensorVal() {
  u8 trackingSensorVal = 0;
  trackingSensorVal = (digitalRead(PIN_TRACKING_LEFT) == 1 ? 1 : 0) << 2 |
(digitalRead(PIN_TRACKING_CENTER) == 1 ? 1 : 0) << 1 | (digitalRead(PIN_TRACKING_RIGHT) == 1 ?
1 : 0) << 0;
  return trackingSensorVal;
}
```

Exp1 ? Exp2 : Exp3;

If Exp1 is true, the result of this code is Exp2.
If Exp1 is false, the result of this code is Exp3.
For example
If y=8;
var = (y < 10) ? 30 : 40; then var=30
If y=10
var = (y < 10) ? 30 : 40; then var=40

switch...case

```
switch (var) {
  case 1:
    //do something when var equals 1
    break;
  case 2:
    //do something when var equals 2
    break;
  default:
    // if nothing else matches, do the default
    // default is optional
    break;
}
```

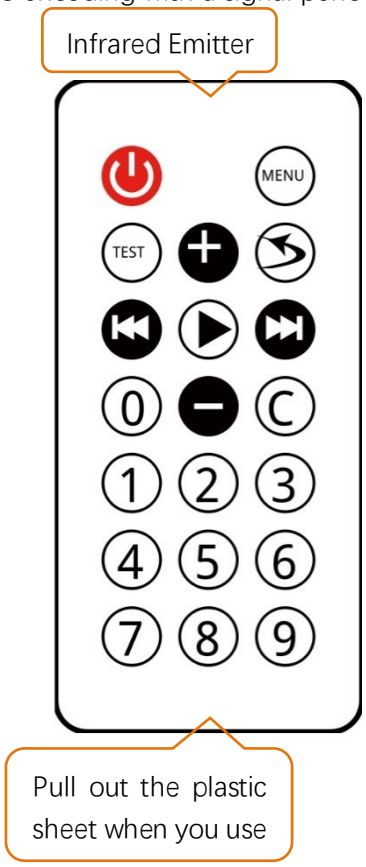
For more information, please refer to:
https://www.arduino.cc/reference/en/language/structure/control-structure/switchcase/

Chapter 4 IR control

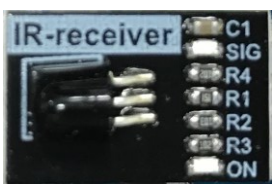
In this chapter, we will introduce infrared remote and receiver and make an IR remote control car.
If you have any concerns, please feel free to contact us via support@freenove.com

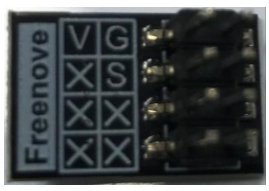
4.1 Infrared remote and receiver

Infrared(IR) remote control is a device with a certain number of buttons. Pressing down different buttons will make the infrared emission tube, which is located in the front of the remote control, send infrared with different encodings. Infrared remote control technology is widely used, such as TV, air conditioning, etc. Thus, it makes it possible for you to switch TV programs and adjust the temperature of the air conditioning wheb you are away from them. The remote control we use is shown below:
This infrared remote control uses NEC encoding with a signal period of 108ms.

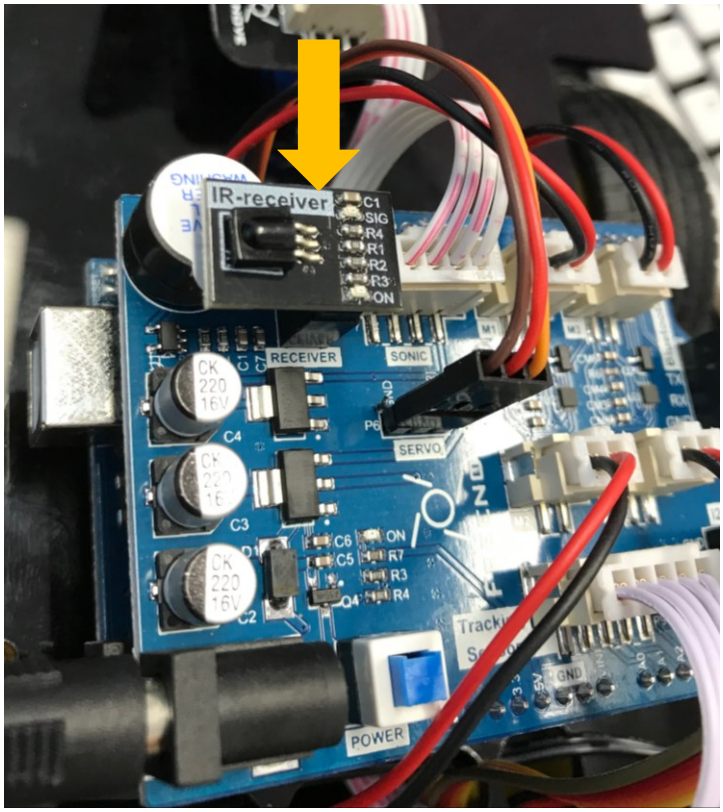


Infrared(IR) receiver is a component, which can receive the infrared light, so we can use it to detect the signal emitted by the infrared remote control. DATA pin here outputs the received infrared signal.





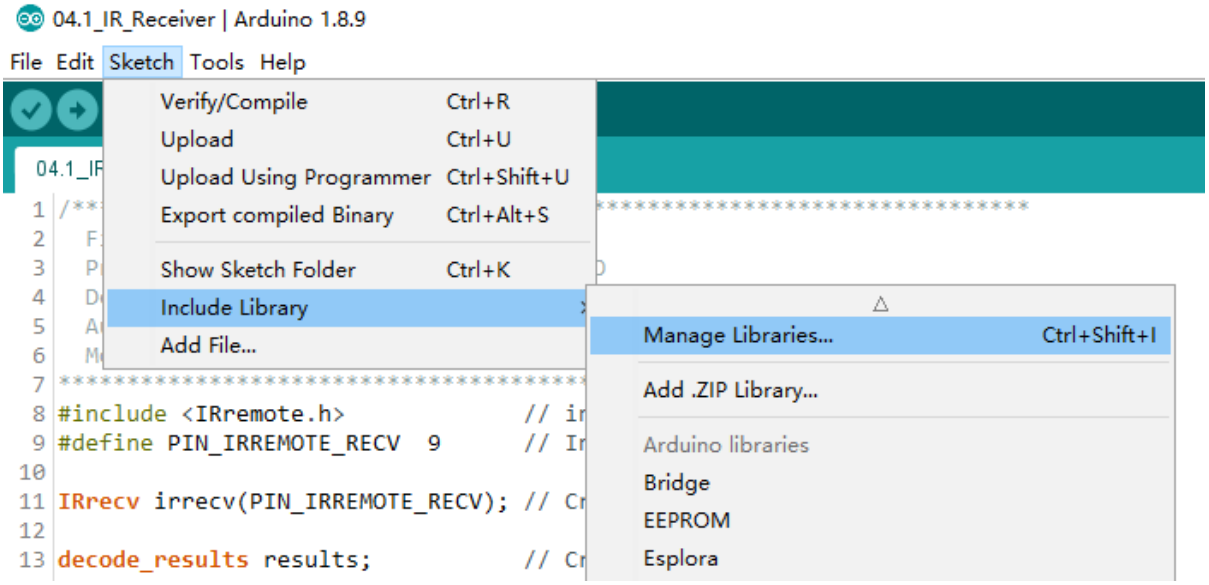
Install IR receiver on the car as below:



Code

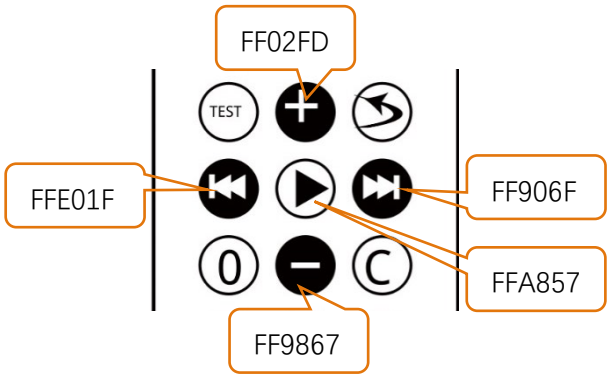
04.1_IR_Receiver

An IRremote library will be used in the code. Therefore, we need to include it.

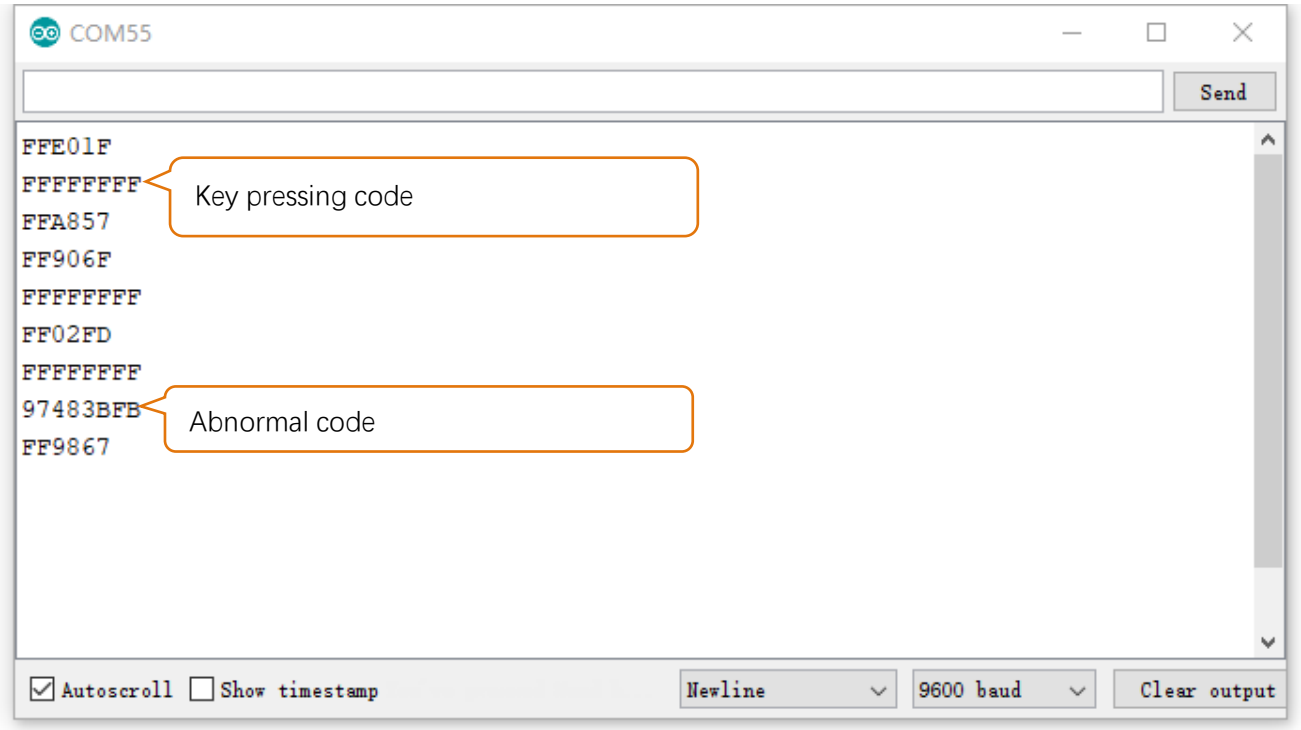


Search IRremtoe and install it.

Then upload code in Sketches\04.1_IR_Receiver, turn on the power and open the serial monitor.
And then **pull off the plastic piece closed to the battery,** and press following keys in IR remote.



After receiving the signal from Infrared Emitter, the controller will recognize which key has been pressed. And print it in the monitor as below



When you press one key lasting for some time, FFFFFFFF will be shown. It indicates that the key is in pressing state.

IR communication is not very stable, so it is easily to be interfered. When used, the transmitting LED of remote control must be aimed at the receiver. Otherwise, it may result in no data being received or incorrect data being received.

In addition, sometimes the code received is abnormal. Except key codes and FFFFFFFF, other codes received are abnormal. When we process the code, we need skip this kind of data.

Different key code corresponds to different key.

```
#include <IRremote.h>              // include IRremote library
#define PIN_IRREMOTE_RECV  9       // Infrared receiving pin
IRrecv irrecv(PIN_IRREMOTE_RECV); // Create a class object used to receive
decode_results results;           // Create a decoding results class object

void setup()
{
  Serial.begin(9600);         // Initialize the serial port and set the baud rate to 9600
  irrecv.enableIRIn();        // Start the receiver
}

void loop() {
  if (irrecv.decode(&results)) {        // Waiting for decoding
    Serial.println(results.value, HEX); // Print out the decoded results
    irrecv.resume();                    // Receive the next value
  }
  delay(100);
}
```

4.2 IR Remote Car

We will use IR remote to control the car to make some actions.

Code

04.2_IR_Remote_Car

First, upload the code in Sketches\04.2_IR_Remote_Car, and turn on the power of car.

Key graph	Key define	Key code	Function
	IR_REMOTE_KEYCODE_UP	0xFF02FD	move forward
	IR_REMOTE_KEYCODE_DOWN	0xFF9867	move back
	IR_REMOTE_KEYCODE_LEFT	0xFFE01F	Turn left
	IR_REMOTE_KEYCODE_RIGHT	0xFF906F	Turn right
	IR_REMOTE_KEYCODE_CENTER	0xFFA857	Turn on buzzer

The code is below:

```
#include <IRremote.h>
#include "Freenove_4WD_Car_for_Arduino.h"

//define key, the code cannot be changed.
#define IR_REMOTE_KEYCODE_UP       0xFF02FD
#define IR_REMOTE_KEYCODE_DOWN     0xFF9867
#define IR_REMOTE_KEYCODE_LEFT     0xFFE01F
#define IR_REMOTE_KEYCODE_RIGHT    0xFF906F
#define IR_REMOTE_KEYCODE_CENTER   0xFFA857

#define IR_UPDATE_TIMEOUT      120
#define IR_CAR_SPEED           180

IRrecv irrecv(PIN_IRREMOTE_RECV);
decode_results results;
u32 currentKeyCode, lastKeyCode;
bool isStopFromIR = false;
u32 lastIRUpdateTime = 0;

void setup() {
  irrecv.enableIRIn(); // Start the receiver
```

```
}

void loop() {
  if (irrecv.decode(&results)) {
    isStopFromIR = false;
    currentKeyCode = results.value;
    if (currentKeyCode != 0xFFFFFFFF) {
      lastKeyCode = currentKeyCode;
    }
    switch (lastKeyCode) {
      case IR_REMOTE_KEYCODE_UP:
        motorRun(IR_CAR_SPEED, IR_CAR_SPEED); //move forward
        break;
      case IR_REMOTE_KEYCODE_DOWN:
        motorRun(-IR_CAR_SPEED, -IR_CAR_SPEED); //move back
        break;
      case IR_REMOTE_KEYCODE_LEFT:
        motorRun(-IR_CAR_SPEED, IR_CAR_SPEED);  //turn left
        break;
      case IR_REMOTE_KEYCODE_RIGHT:
        motorRun(IR_CAR_SPEED, -IR_CAR_SPEED);  //turn right
        break;
      case IR_REMOTE_KEYCODE_CENTER:
        setBuzzer(true);              //turn on buzzer
        break;
      default:
        break;
    }
    irrecv.resume(); // Receive the next value
    lastIRUpdateTime = millis(); //write down current time
  }
  else {
    if (millis() - lastIRUpdateTime > IR_UPDATE_TIMEOUT) {
      if (!isStopFromIR) {
        isStopFromIR = true;
        motorRun(0, 0);
        setBuzzer(false);
      }
      lastIRUpdateTime = millis();
    }
  }
}

```

Since the emission period of the infrared remote control is 108 ms, the signal timeout time is set to a rough

value of 120 ms. If the timeout is less than 108 ms, the data will be read ahead of time and cannot be read. It will cause a misunderstanding that the signal is lost. If the timeout time is set too large, such as 1000ms, the delay of car action will be more serious.

```
11  #define IR_UPDATE_TIMEOUT      120
    ......
54      if (millis() - lastIRUpdateTime > IR_UPDATE_TIMEOUT) {
55        if (!isStopFromIR) {
56          isStopFromIR = true;
57          motorRun(0, 0);
58          setBuzzer(false);
59        }
60        lastIRUpdateTime = millis();
61      }
    ......
```

You can change value of IR_CAR_SPEED to change speed of the car in range (0~255). In fact, since motor need enough voltage to make the car move so the minimum value is not 0.

```
12  #define IR_CAR_SPEED           180
```

millis()

Returns the number of milliseconds passed since the Arduino board began running the **current** program

```
    lastIRUpdateTime = millis(); //write down current time
  }
  else {
    if (millis() - lastIRUpdateTime)
```

millis() – lastIRUpdateTime is the time from lastIRUpdateTime to current line.

For more details, please refer to:
https://www.arduino.cc/reference/en/language/functions/time/millis/

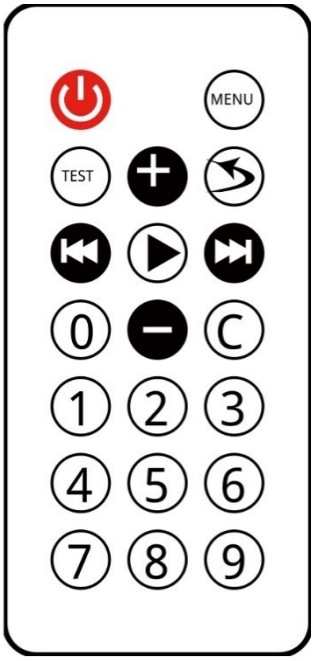
4.3 Multifunctional IR Remote Car

In this project, we will add a LED bar function compared to the last section, so the infrared remote control car will have more functions.

Upload Code and Running

Open the code in Sketches\04.3_ Multifunctional_IR_Remote_Car.ino. Upload it into car.

After the code is successfully uploaded, turn on the power of the car and use the infrared remote control to control the car and other functions. The corresponding keys and their functions are shown in the following table:



Key graph	Key define	Key code	Function
	IR_REMOTE_KEYCODE_UP	0xFF02FD	move forward
	IR_REMOTE_KEYCODE_DOWN	0xFF9867	move back
	IR_REMOTE_KEYCODE_LEFT	0xFFE01F	Turn left
	IR_REMOTE_KEYCODE_RIGHT	0xFF906F	Turn right
	IR_REMOTE_KEYCODE_CENTER	0xFFA857	Turn on buzzer
	IR_REMOTE_KEYCODE_1	0xFF30CF	Make the LED run mode 1 to scroll the rainbow color.
	IR_REMOTE_KEYCODE_4	0xFF10EF	Make LED run mode 2, changing the color of the water LED
	IR_REMOTE_KEYCODE_2	0xFF18E7	The color of the LED bar changes faster. The color is from ColorWheel.
	IR_REMOTE_KEYCODE_3	0xFF7A85	The color of the LED bar changes slower.
	IR_REMOTE_KEYCODE_5	0xFF38C7	The LED bar cycle period is decreased, and the LED bar changes at a faster speed
	IR_REMOTE_KEYCODE_6	0xFF5AA5	The LED bar cycle period is increased, and the LED bar changes at a slower speed

Code

This project has many labels.

“Freenove_IR_Remote_Keycode.h” is used to save key code of IR remote control.

Freenove_IR_Remote_Keycode.h

```
// Freenove_IR_Remote_Keycode.h
#ifndef _FREENOVE_IR_REMOTE_KEYCODE_h
#define _FREENOVE_IR_REMOTE_KEYCODE_h
#if defined(ARDUINO) && ARDUINO >= 100
    #include "arduino.h"
#else
    #include "WProgram.h"
#endif
#include <IRremote.h>

#define IR_REMOTE_KEYCODE_POWER     0xFFA25D
#define IR_REMOTE_KEYCODE_MENU      0xFF629D
#define IR_REMOTE_KEYCODE_MUTE      0xFFE21D

#define IR_REMOTE_KEYCODE_MODE      0xFF22DD
#define IR_REMOTE_KEYCODE_UP        0xFF02FD
#define IR_REMOTE_KEYCODE_BACK      0xFFC23D

#define IR_REMOTE_KEYCODE_LEFT      0xFFE01F
#define IR_REMOTE_KEYCODE_CENTER    0xFFA857
#define IR_REMOTE_KEYCODE_RIGHT     0xFF906F

#define IR_REMOTE_KEYCODE_0         0xFF6897
#define IR_REMOTE_KEYCODE_DOWN      0xFF9867
#define IR_REMOTE_KEYCODE_OK        0xFFB04F

#define IR_REMOTE_KEYCODE_1             0xFF30CF
#define IR_REMOTE_KEYCODE_2             0xFF18E7
#define IR_REMOTE_KEYCODE_3             0xFF7A85

#define IR_REMOTE_KEYCODE_4             0xFF10EF
#define IR_REMOTE_KEYCODE_5             0xFF38C7
#define IR_REMOTE_KEYCODE_6             0xFF5AA5

#define IR_REMOTE_KEYCODE_7             0xFF42BD
#define IR_REMOTE_KEYCODE_8             0xFF4AB5
#define IR_REMOTE_KEYCODE_9             0xFF52AD
#endif
```

04.3_ Multifunctional_IR_Remote_Car.ino

```
#include "Freenove_4WD_Car_for_Arduino.h"
#include "Freenove_IR_Remote_Keycode.h"
#include "Freenove_WS2812B_RGBLED_Controller.h"
//////////////////////////////////////////
////Definitions related to IR-remote
#define IR_UPDATE_TIMEOUT     110
#define IR_CAR_SPEED          180

IRrecv irrecv(PIN_IRREMOTE_RECV);
decode_results results;
u32 currentKeyCode, lastKeyCode;
bool isStopFromIR = false;
u32 lastIRUpdateTime = 0;
//////////////////////////////////////////
////Definitions related to Led-strip
#define STRIP_I2C_ADDRESS  0x20
#define STRIP_LEDS_COUNT   10

u8 colorPos = 0;
u8 colorStep = 50;
u8 stripDisplayMode = 1;
u8 currentLedIndex = 0;
u16 stripDisplayDelay = 100;
u32 lastStripUpdateTime = 0;
Freenove_WS2812B_Controller strip(STRIP_I2C_ADDRESS, STRIP_LEDS_COUNT, TYPE_GRB);
//////////////////////////////////////////

void setup() {
    strip.begin();
    irrecv.enableIRIn(); // Start the receiver
}

void loop() {
    if (irrecv.decode(&results)) {
        isStopFromIR = false;
        currentKeyCode = results.value;
        if (currentKeyCode != 0xFFFFFFFF) {
            lastKeyCode = currentKeyCode;
        }
        switch (lastKeyCode) {
        case IR_REMOTE_KEYCODE_UP:
            motorRun(IR_CAR_SPEED, IR_CAR_SPEED);
            break;
```

```
44              case IR_REMOTE_KEYCODE_DOWN:
45                  motorRun(-IR_CAR_SPEED, -IR_CAR_SPEED);
46                  break;
47              case IR_REMOTE_KEYCODE_LEFT:
48                  motorRun(-IR_CAR_SPEED, IR_CAR_SPEED);
49                  break;
50              case IR_REMOTE_KEYCODE_RIGHT:
51                  motorRun(IR_CAR_SPEED, -IR_CAR_SPEED);
52                  break;
53              case IR_REMOTE_KEYCODE_CENTER:
54                  setBuzzer(true);
55                  break;
56              case IR_REMOTE_KEYCODE_0:
57                  break;
58              case IR_REMOTE_KEYCODE_1:
59                  stripDisplayMode = 1;
60                  break;
61              case IR_REMOTE_KEYCODE_2:
62                  colorStep += 5;
63                  if (colorStep > 100)
64                  {
65                      colorStep = 100;
66                  }
67                  break;
68              case IR_REMOTE_KEYCODE_3:
69                  colorStep -= 5;
70                  if (colorStep < 5)
71                  {
72                      colorStep = 5;
73                  }
74                  break;
75              case IR_REMOTE_KEYCODE_4:
76                  stripDisplayMode = 0;
77                  break;
78              case IR_REMOTE_KEYCODE_5:
79                  stripDisplayDelay -= 20;
80                  if (stripDisplayDelay < 20)
81                  {
82                      stripDisplayDelay = 20;
83                  }
84                  break;
85              case IR_REMOTE_KEYCODE_6:
86                  stripDisplayDelay += 20;
87                  if (stripDisplayDelay > 300)
```

```
88                {
89                    stripDisplayDelay = 300;
90                }
91                break;
92            }
93            irrecv.resume(); // Receive the next value
94            lastIRUpdateTime = millis();
95        }
96        else {
97            if (millis() - lastIRUpdateTime > IR_UPDATE_TIMEOUT)
98            {
99                if (!isStopFromIR) {
100                   isStopFromIR = true;
101                   motorRun(0, 0);
102                   setBuzzer(false);
103               }
104               lastIRUpdateTime = millis();
105           }
106       }
107       switch (stripDisplayMode)
108       {
109       case 0:
110           if (millis() - lastStripUpdateTime > stripDisplayDelay)
111           {
112               for (int i = 0; i < STRIP_LEDS_COUNT; i++) {
113                   strip.setLedColorData(i, strip.Wheel(colorPos + i * 25));
114               }
115               strip.show();
116               colorPos += colorStep;
117               lastStripUpdateTime = millis();
118           }
119           break;
120       case 1:
121           if (millis() - lastStripUpdateTime > stripDisplayDelay)
122           {
123               strip.setLedColor(currentLedIndex, strip.Wheel(colorPos));
124               currentLedIndex++;
125               if (currentLedIndex == STRIP_LEDS_COUNT)
126               {
127                   currentLedIndex = 0;
128                   colorPos += colorStep;
129               }
130               lastStripUpdateTime = millis();
131           }
```

```
            break;
        default:
            break;
        }
}
```

Compared to the previous project, this code has more switch-case branches of the remote key code for controlling the light bar.

```
……
            case IR_REMOTE_KEYCODE_1:
                stripDisplayMode = 1;
                break;
            case IR_REMOTE_KEYCODE_2:
                colorStep += 5;
                if (colorStep > 100)
                {
                    colorStep = 100;
                }
                break;
            case IR_REMOTE_KEYCODE_3:
                colorStep -= 5;
                if (colorStep < 5)
                {
                    colorStep = 5;
                }
                break;
            case IR_REMOTE_KEYCODE_4:
                stripDisplayMode = 0;
                break;
            case IR_REMOTE_KEYCODE_5:
                stripDisplayDelay -= 20;
                if (stripDisplayDelay < 20)
                {
                    stripDisplayDelay = 20;
                }
                break;
            case IR_REMOTE_KEYCODE_6:
                stripDisplayDelay += 20;
                if (stripDisplayDelay > 300)
                {
                    stripDisplayDelay = 300;
                }
                break;
……
```

In the loop(), control the behavior of the light bar based on these variables above.
The variable **stripDisplayMode** is used to control the display mode of the LED.
The variable **stripDisplayDelay** is used to control the cycle time of the LED, that is, the speed of change.
The variable **colorStep** is used to control the color value change for each iteration.

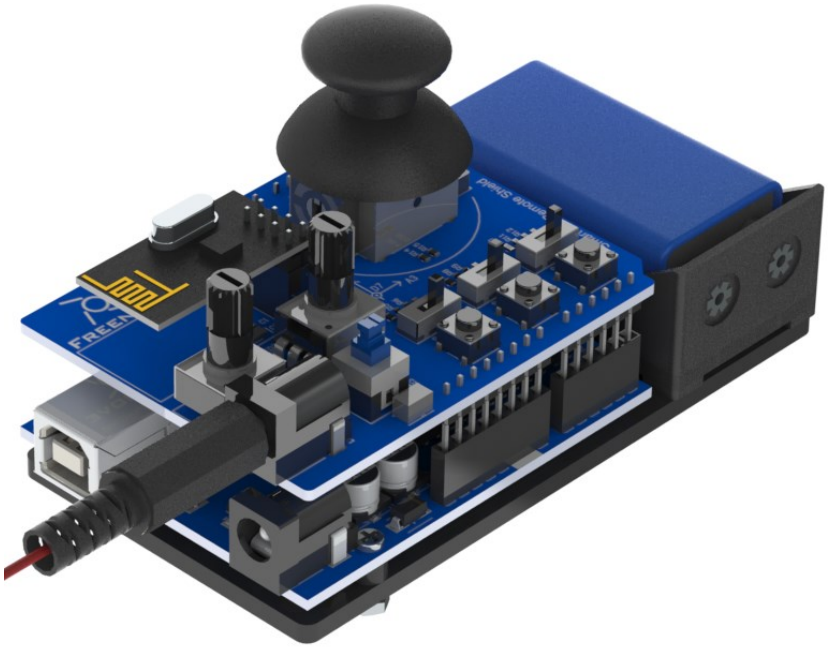
```
switch (stripDisplayMode)
    {
    case 0:
        if (millis() - lastStripUpdateTime > stripDisplayDelay)
        {
            for (int i = 0; i < STRIP_LEDS_COUNT; i++) {
                strip.setLedColorData(i, strip.Wheel(colorPos + i * 25));
            }
            strip.show();
            colorPos += colorStep;
            lastStripUpdateTime = millis();
        }
        break;
    case 1:
        if (millis() - lastStripUpdateTime > stripDisplayDelay)
        {
            strip.setLedColor(currentLedIndex, strip.Wheel(colorPos));
            currentLedIndex++;
            if (currentLedIndex == STRIP_LEDS_COUNT)
            {
                currentLedIndex = 0;
                colorPos += colorStep;              }
            lastStripUpdateTime = millis();
        }
        break;
    default:
        break;
    }

```

Chapter 5 RF Remote Control

If you purchased the kit without RF remote control, you can skip this chapter.
If you have any concerns, please feel free to contact us via support@freenove.com

5.1 Remote Control

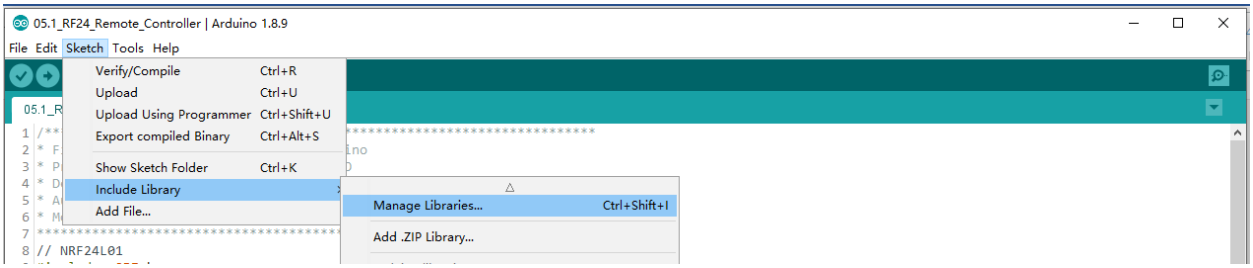
Download tutorial and code to **assemble** remote controller.
https://github.com/Freenove/Freenove_Remote_Control_Kit/raw/master/Tutorial.pdf



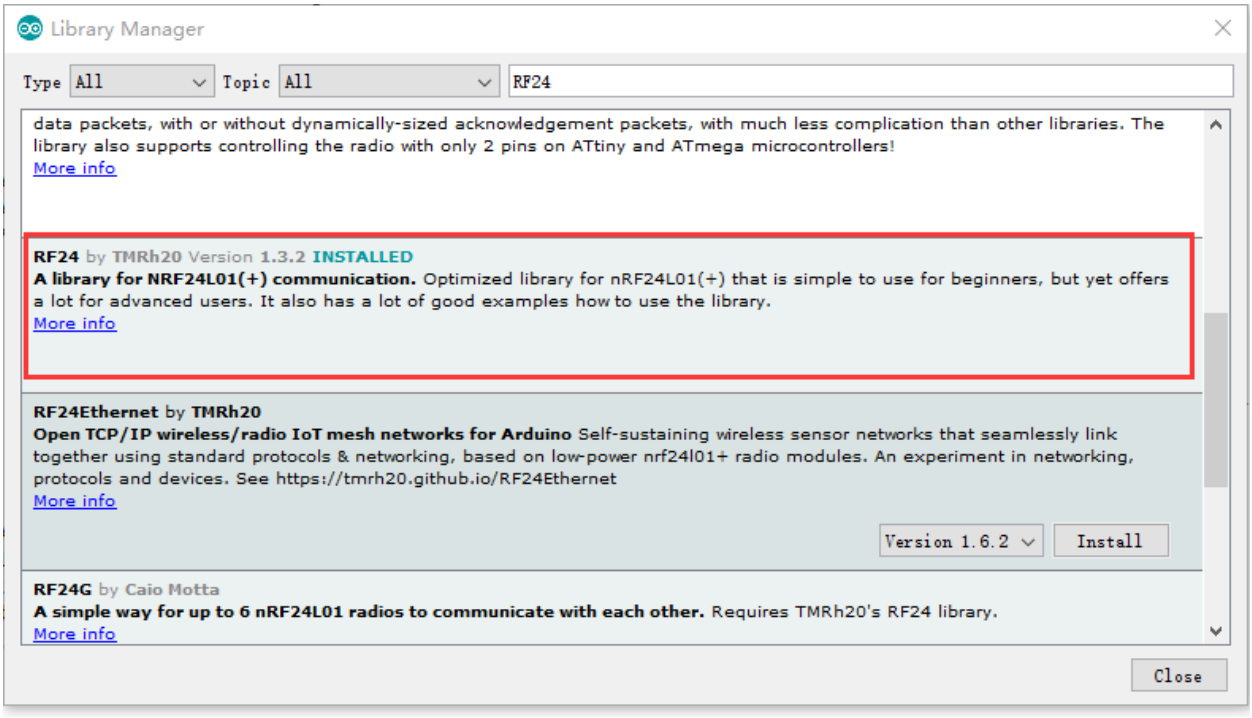
For all resources, please refer to https://github.com/Freenove/Freenove_Remote_Control_Kit

Upload Code and Running

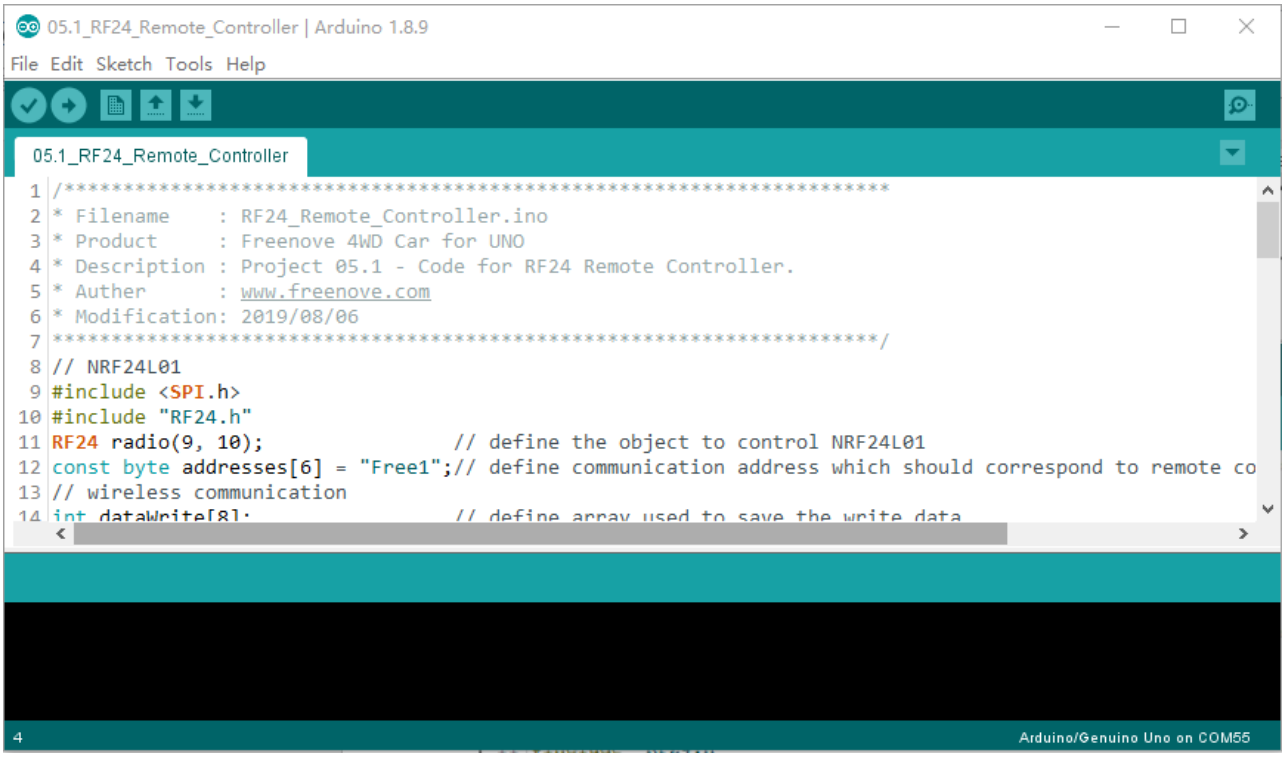
First, install library. SPI is library of Arduino, which will be included automatically. You only need to install RF24.



Find RF24 and install it.



Connect remote controller to computer. And upload code in Sketches\05.1_RF24_Remote_Controller. Then turn on the power of RF remote control.

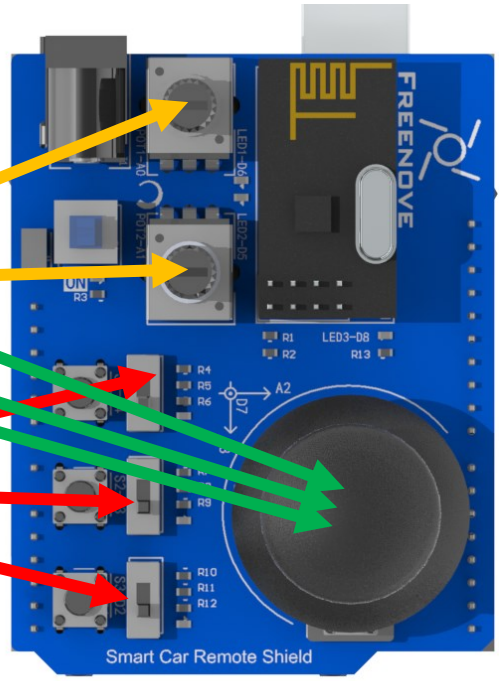


05.1_RF24_Remote_Controller

The code is below:

```
// RF24L01
#include <SPI.h>
#include "RF24.h"
RF24 radio(9, 10);                  // define the object to control RF24L01
const byte addresses[6] = "Free1";// define communication address which should correspond to
the control board of car
// wireless communication
int dataWrite[8];                   // define array used to save the write data
// pin
const int pot1Pin = A0,             // define POT1 Potentiometer
          pot2Pin = A1,             // define POT2 Potentiometer
          joystickXPin = A2,        // define pin for direction X of joystick
          joystickYPin = A3,        // define pin for direction Y of joystick
          joystickZPin = 7,         // define pin for direction Z of joystick
          s1Pin = 4,                // define pin for S1
          s2Pin = 3,                // define pin for S2
          s3Pin = 2,                // define pin for S3
          led1Pin = 6,              // define pin for LED1 which is close to POT1 and used to
indicate the state of POT1
          led2Pin = 5,              // define pin for LED2 which is close to POT2 and used to
indicate the state of POT2
```

```
22          led3Pin = 8;            // define pin for LED3 which is close to RF24L01 and used to
23  indicate the state of RF24L01
24
25  void setup() {
26    // RF24L01
27    radio.begin();                    // initialize RF24
28    radio.setPALevel(RF24_PA_MAX);    // set power amplifier (PA) level
29    radio.setDataRate(RF24_1MBPS);    // set data rate through the air
30    radio.setRetries(0, 15);          // set the number and delay of retries
31    radio.openWritingPipe(addresses);   // open a pipe for writing
32    radio.openReadingPipe(1, addresses);// open a pipe for reading
33    radio.stopListening();            // stop listening for incoming messages
34
35  // pin
36    pinMode(joystickZPin, INPUT);     // set led1Pin to input mode
37    pinMode(s1Pin, INPUT);            // set s1Pin to input mode
38    pinMode(s2Pin, INPUT);            // set s2Pin to input mode
39    pinMode(s2Pin, INPUT);            // set s3Pin to input mode
40    pinMode(led1Pin, OUTPUT);         // set led1Pin to output mode
41    pinMode(led2Pin, OUTPUT);         // set led2Pin to output mode
42    pinMode(led3Pin, OUTPUT);         // set led3Pin to output mode
43  }
44
45  void loop()
46  {
47    // put the values of rocker, switch and pot
48    dataWrite[0] = analogRead(pot1Pin); // save
49    dataWrite[1] = analogRead(pot2Pin); // save
50    dataWrite[2] = analogRead(joystickXPin);  /
51    dataWrite[3] = analogRead(joystickYPin);  /
52    dataWrite[4] = digitalRead(joystickZPin);  /
53    dataWrite[5] = digitalRead(s1Pin);         /
54    dataWrite[6] = digitalRead(s2Pin);         /
55    dataWrite[7] = digitalRead(s3Pin);         /
56
57    // write radio data
58    if (radio.write(dataWrite, sizeof(dataWrite
59    {
60      digitalWrite(led3Pin, HIGH);
61      delay(20);
62      digitalWrite(led3Pin, LOW);
63    }
64
65    // make LED emit different brightness of light according to analog of potentiometer
```



```
66    analogWrite(led1Pin, map(dataWrite[0], 0, 1023, 0, 255));
67    analogWrite(led2Pin, map(dataWrite[1], 0, 1023, 0, 255));
68  }
```

```
5   const byte addresses[6] = "Free1";
```

You can change the address if you have two Freenove remote to control different cars, like Free2. But it is **necessary** to keep it same to the address in code in car.

```
48    dataWrite[0] = analogRead(pot1Pin);      //(0~1023) save data of Potentiometer 1
49    dataWrite[1] = analogRead(pot2Pin);      //(0~1023) save data of Potentiometer 2
50    dataWrite[2] = analogRead(joystickXPin); //(0~1023) save data of direction X of joystick
51    dataWrite[3] = analogRead(joystickYPin); //(0~1023)save data of direction Y of joystick
52    dataWrite[4] = digitalRead(joystickZPin); //(0,1) save data of direction Z of joystick
53    dataWrite[5] = digitalRead(s1Pin);        //(0,1) save data of switch 1
54    dataWrite[6] = digitalRead(s2Pin);        // (0,1)save data of switch 2
55    dataWrite[7] = digitalRead(s3Pin);        // (0,1)save data of switch 3
```

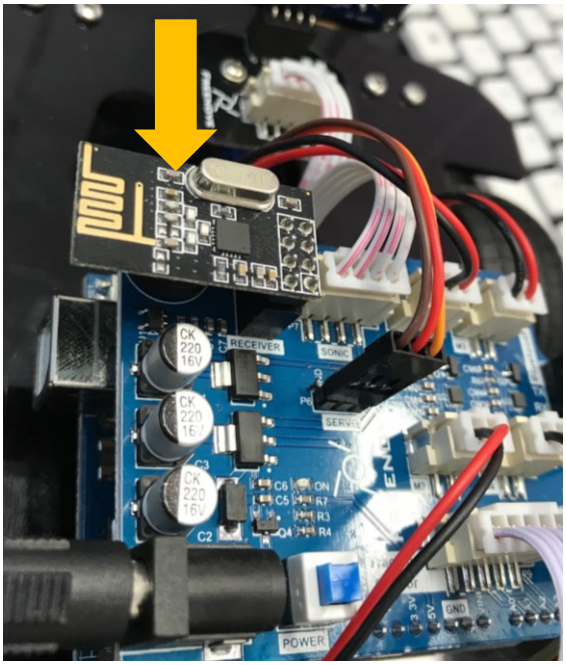
When joystick is in center place, its value is about x=512, y=512

When joystick is pressed, z=0; when it is not pressed, z=1.

Switches are the same, when switch is on, s=0; when switch is off, s=1.

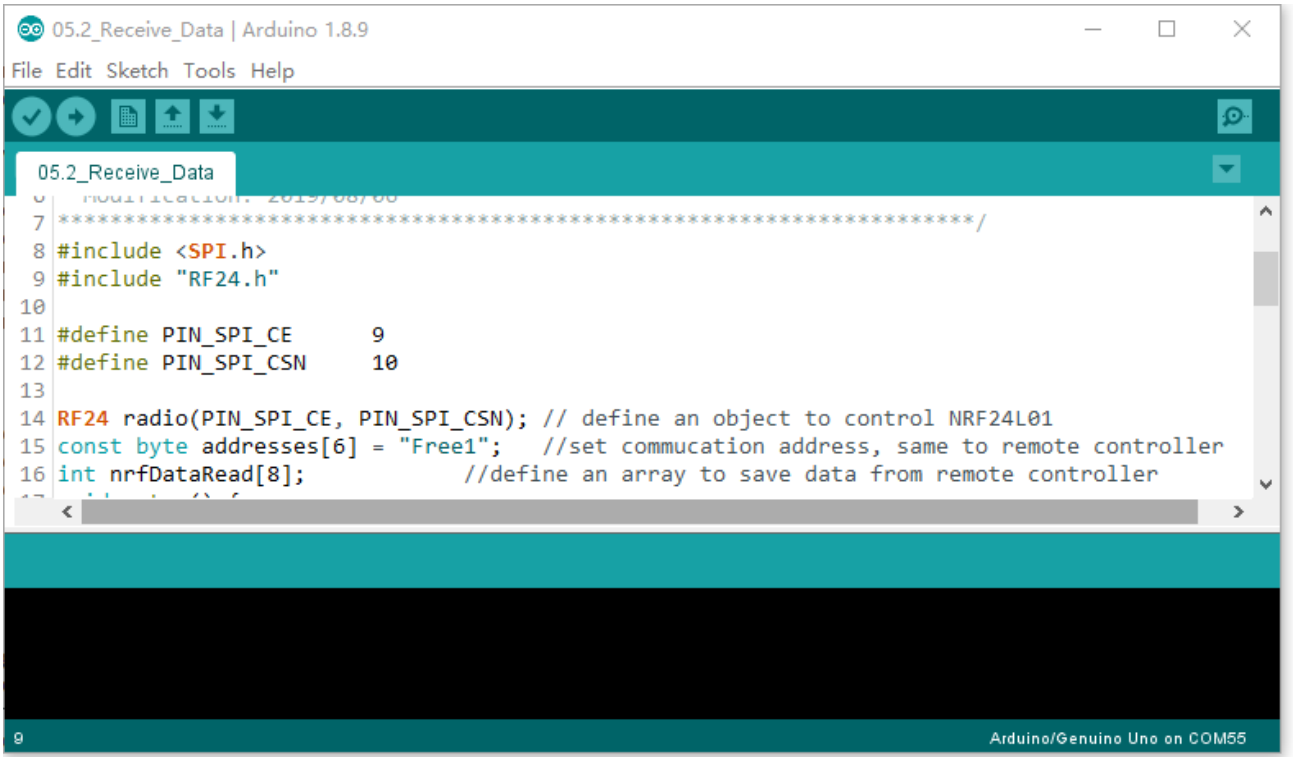
5.2 Receive RF Remote Control Data

Install RF module on the car. RF module and IR receiver use the same receiver pins.



Upload Code and Running

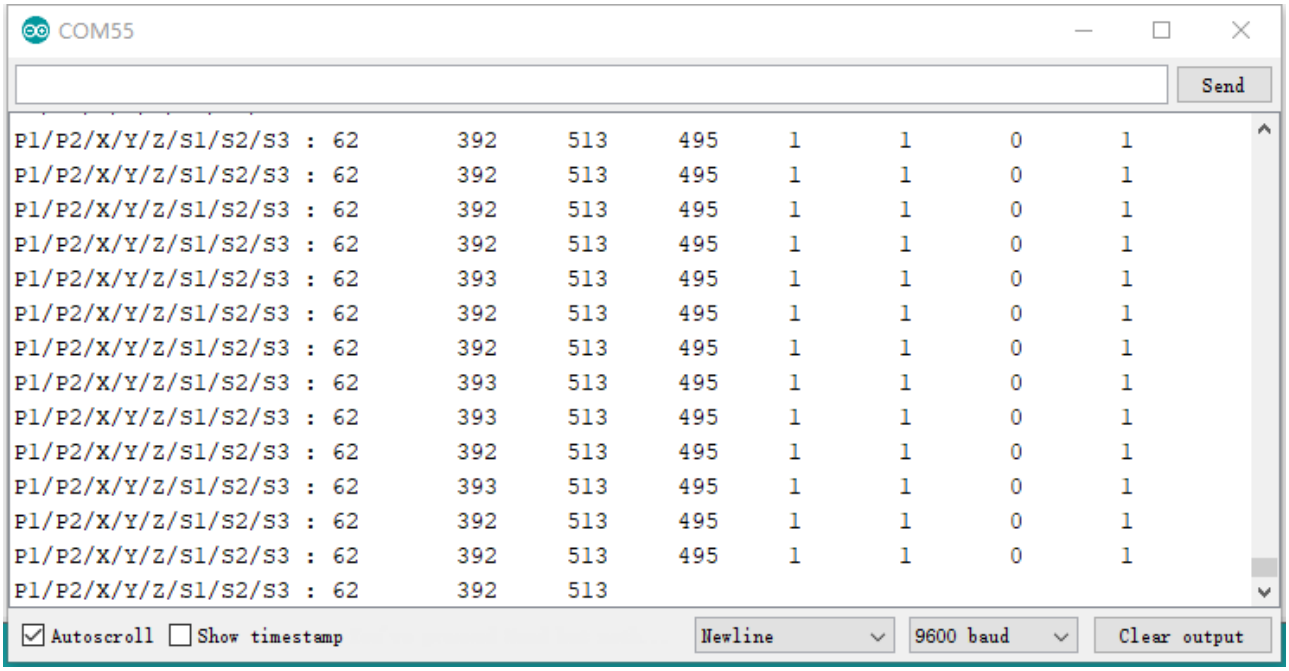
Then connect car control board to computer. And upload code in Sketches\05.2_Receive_Data.



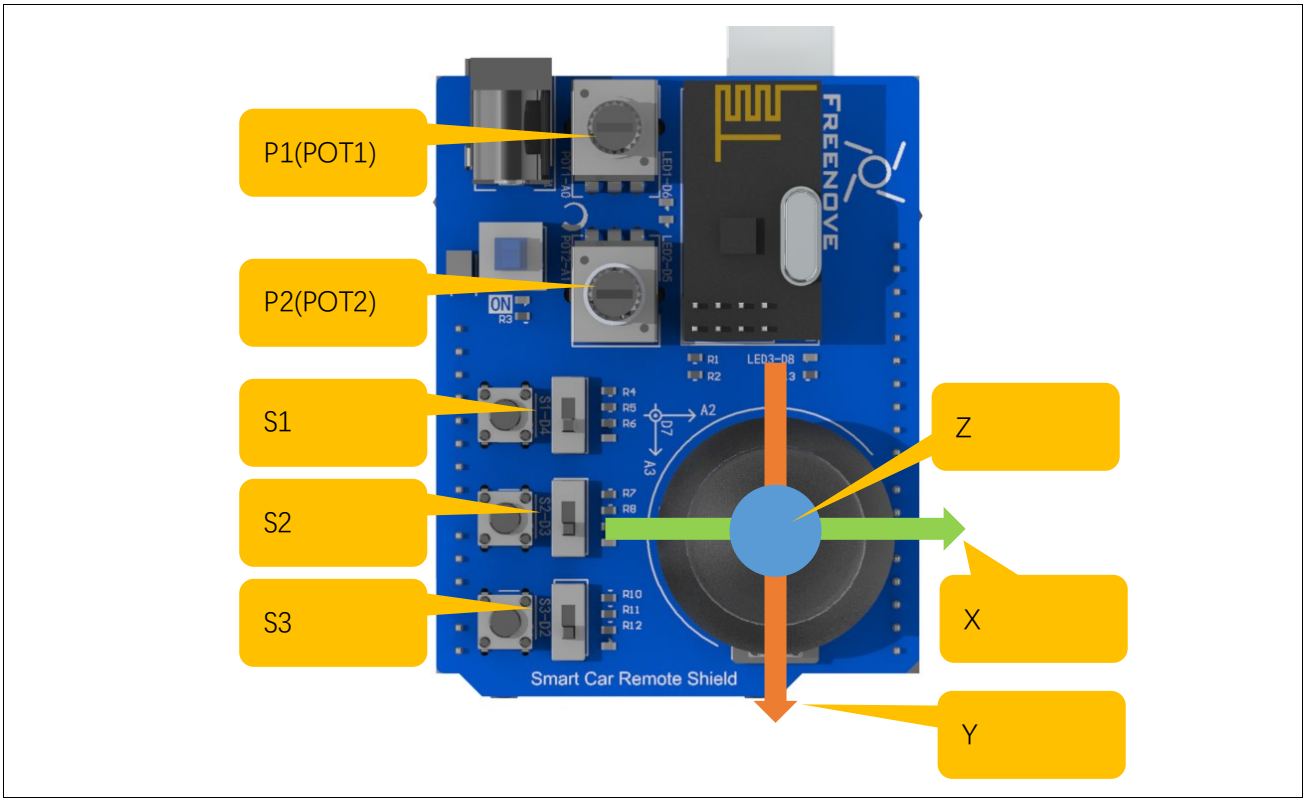
Power the remote controller.

Keep car control board connected to computer via USB cable and open the serial monitor. You will see contents as below.

Operate the remote controller and observe the data change.



Data to components



Code

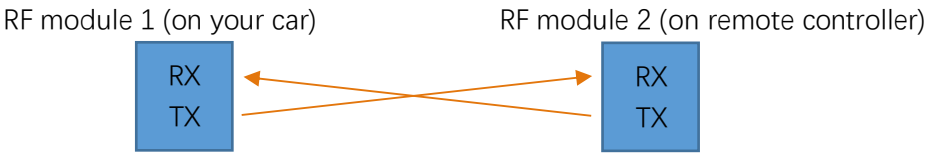
05.2_Receive_Data

The code is below.

```
#include <SPI.h>
#include "RF24.h"

#define PIN_SPI_CE      9
#define PIN_SPI_CSN     10

RF24 radio(PIN_SPI_CE, PIN_SPI_CSN); // define an object to control RF24L01
const byte addresses[6] = "Free1";   //set commutation address, same to remote controller
int RFDataRead[8];                   //define an array to save data from remote controller
void setup() {
  Serial.begin(9600);

  // RF24L01 initialization steps
  if (radio.begin()) {                  // initialize RF24
    radio.setPALevel(RF24_PA_MAX);      // set power amplifier (PA) level
    radio.setDataRate(RF24_1MBPS);      // set data rate through the air
    radio.setRetries(0, 15);            // set the number and delay of retries
    radio.openWritingPipe(addresses);   // open a pipe for writing
    radio.openReadingPipe(1, addresses);// open a pipe for reading
    radio.startListening();             // start monitoringtart listening on the pipes opened
    Serial.println("Start listening remote data ... ");
  }
  else {
    Serial.println("Not found the RF chip!");
  }
}

void loop() {
  if (radio.available()) {              // if receive the data
    while (radio.available()) {         // read all the data
      radio.read(RFDataRead, sizeof(RFDataRead));   // read data
    }
    Serial.print("P1/P2/X/Y/Z/S1/S2/S3 : ");
    for (int i = 0; i < sizeof(RFDataRead) / 2; i++) {
      Serial.print(RFDataRead[i]);
      Serial.print('\t');
    }
    Serial.print('\n');
  }
```

```
40  }
41
```

The communication is as follows:



`radio.write(dataWrite, sizeof(dataWrite)` Transfer the data.

Both of the codes for car and for remote have following contents:

```
5   const byte addresses[6] = "Free1";
    ……
18      radio.openWritingPipe(addresses);    // open a pipe for writing
19      radio.openReadingPipe(1, addresses);// open a pipe for reading
```

This is used to set writing (transmitting) and reading (receiving) addresses of the RF24L01 module. Module B can receive the data of module A, only when the writing address of module A is the same as the reading address of module B. The module's own writing and reading addresses can be the same or different. When there are many RF modules around, the writing and reading addresses can be changed to avoid interference from other devices.

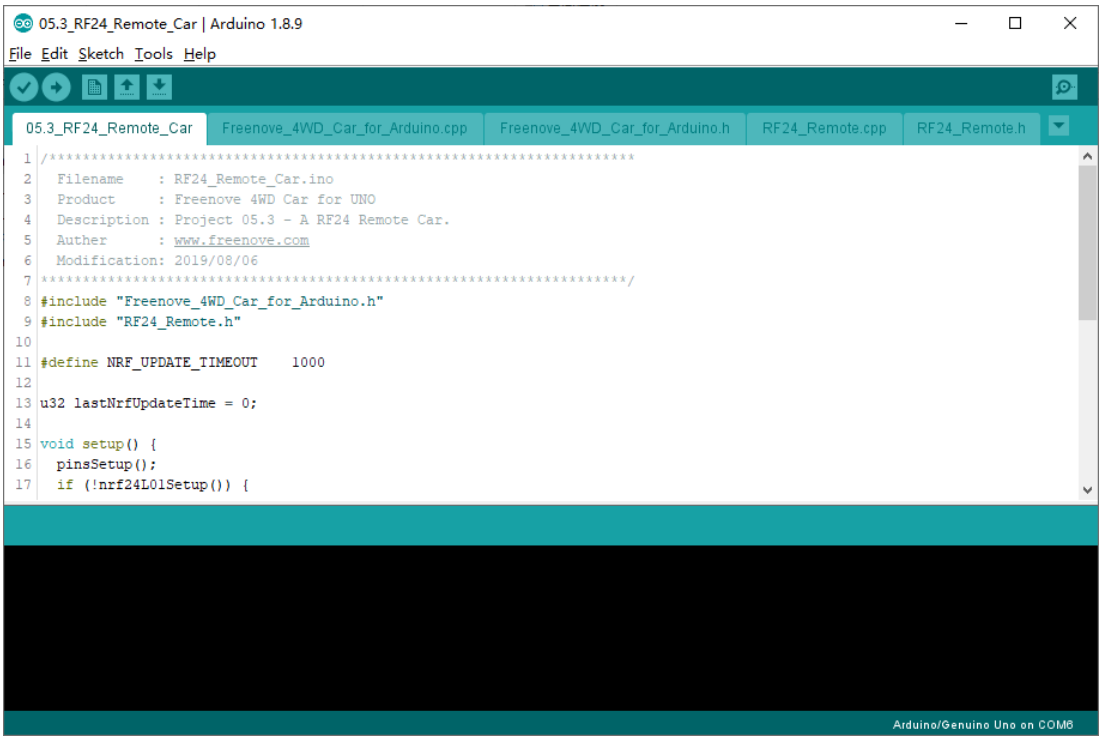
After we receive data from remote, we will recognize which component is operated according to the data change.

5.3 RF Remote Car

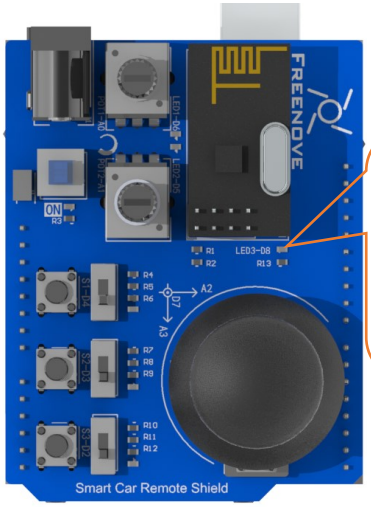
After successfully receiving the data sent by the RF remote control, we can use the data to control car movement, buzzer sounds, etc.

Upload Code and Running

Connect Car to computer with USB cable, and upload code in Sketches\05.3_RF24_Remote_Car.ino.



Then disconnect the USB cable and turn on the RF Remote Control and Car power switches. Operate the joystick to control the movement of the car. Press the joystick to turn on the buzzer.



LED on state indicate the communication successes.
LED off state indicates the communication fails.

When it is off, you need restart power of remote controller and car.

Code

The project has five labels, of which "RF24_Remote.h" and "RF24_Remote.cpp" store contents about data reading and data processing for the RF24 remote control. These contents are called in "05.3_RF24_Remote_Car.ino".

RF24_Remote.h

```
#ifndef _RF_REMOTE_h
#define _RF_REMOTE_h

#include "Freenove_4WD_Car_for_Arduino.h"
#include "RF24.h"

extern RF24 radio;

enum RemoteData
{
    POT1 = 0,
    POT2 = 1,
    JOYSTICK_X = 2,
    JOYSTICK_Y = 3,
    JOYSTICK_Z = 4,
    S1 = 5,
    S2 = 6,
    S3 = 7
};
enum RemoteMode {
    ON_ON_ON = 0,
    ON_ON_OFF = 1,          //Servo calibration mode.
    ON_OFF_ON = 2,
    ON_OFF_OFF = 3,         //Ultrasonic obstacle avoidance mode.
    OFF_ON_ON = 4,
    OFF_ON_OFF = 5,         //Line tracking mode.
    OFF_OFF_ON = 6,         //
    OFF_OFF_OFF = 7             //Remote control mode.
};
enum RemoteModeSwitchState {
    MODE_SWITCHING_IS_INITIALIZING = 0,
    MODE_SWITCHING_IS_PROCESSING = 1,
    MODE_SWITCHING_IS_CONFIRMING = 2,
    MODE_SWITCHING_WAS_FINISHED = 3
};

#define MODE_REMOTE_CONTROL                 OFF_OFF_OFF
```

```
#define MODE_LINE_TRACKING              OFF_ON_OFF
#define MODE_OBSTACKE_AVOIDANCE             ON_OFF_OFF

extern int RFDataRead[8];

bool RF24L01Setup();
bool getRF24L01Data();
void updateCarActionByRFRemote();
void resetRFDataBuf();
u8 updateRFCarMode();

#endif
```

Where enum RemoteData defines the order of the remote control data received, using more meaningful variable names instead of numbers. For example, if you want to use the X-axis data of the remote control, you can use RFDataRead[JOYSTICK_X] instead of the no meaningless RFDataRead[2]. It like define

```
enum RemoteData
{
    POT1 = 0, // similar to #define POT1 0
    POT2 = 1,
    JOYSTICK_X = 2,
    JOYSTICK_Y = 3,
    JOYSTICK_Z = 4,
    S1 = 5,
    S2 = 6,
    S3 = 7
};
```

The enum RemoteMode defines all states of the remote control switches S1, S2, S3. The enum RemoteModeSwitchState defines the state when the remote control switches modes. These contents will also be used in the next project.

```
enum RemoteMode {
    ON_ON_ON = 0,
    ON_ON_OFF = 1,          //Servo calibration mode.
    ON_OFF_ON = 2,
    ON_OFF_OFF = 3,         //Ultrasonic obstacle avoidance mode.
    OFF_ON_ON = 4,
    OFF_ON_OFF = 5,         //Line tracking mode.
    OFF_OFF_ON = 6,         //
    OFF_OFF_OFF = 7             //Remote control mode.
};
enum RemoteModeSwitchState {
    MODE_SWITCHING_IS_INITIALIZING = 0,
    MODE_SWITCHING_IS_PROCESSING = 1,
```

```
    MODE_SWITCHING_IS_CONFIRMING = 2,
    MODE_SWITCHING_WAS_FINISHED = 3
};
```

bool RF24L01Setup();
Initialize RF24L01. If success, return true, if not, return false.
bool getRF24L01Data();
Read data from remote control. If there is data read, return true; if not, return false.
void updateCarActionByRFRemote();
Control car actions (mainly for motor and buzzer) according to data received.
void resetRFDataBuf();
Set values of RFData array to default values.
u8 updateRFCarMode();
Combine data of S1 S2 S3 of remote control into one value and return it.

05.3_RF24_Remote_Car.ino

```
#include "Freenove_4WD_Car_for_Arduino.h"
#include "RF24_Remote.h"

#define RF_UPDATE_TIMEOUT     1000
u32 lastRFUpdateTime = 0;

void setup() {
    pinsSetup();
    if (!RF24L01Setup()) {
        alarm(4, 2);
    }
}

void loop() {
    if (getRF24L01Data()) {
        updateCarActionByRFRemote();
        lastRFUpdateTime = millis();
    }
    if (millis() - lastRFUpdateTime > RF_UPDATE_TIMEOUT) {
        lastRFUpdateTime = millis();
        resetRFDataBuf();
        updateCarActionByRFRemote();
    }
}
```

In setup(), initialize pins and RF24L01. In loop(), receive data from the remote control and process the data. If receiving data is time out, the remote control signal is considered to be lost and the car will be placed in the initialization state.

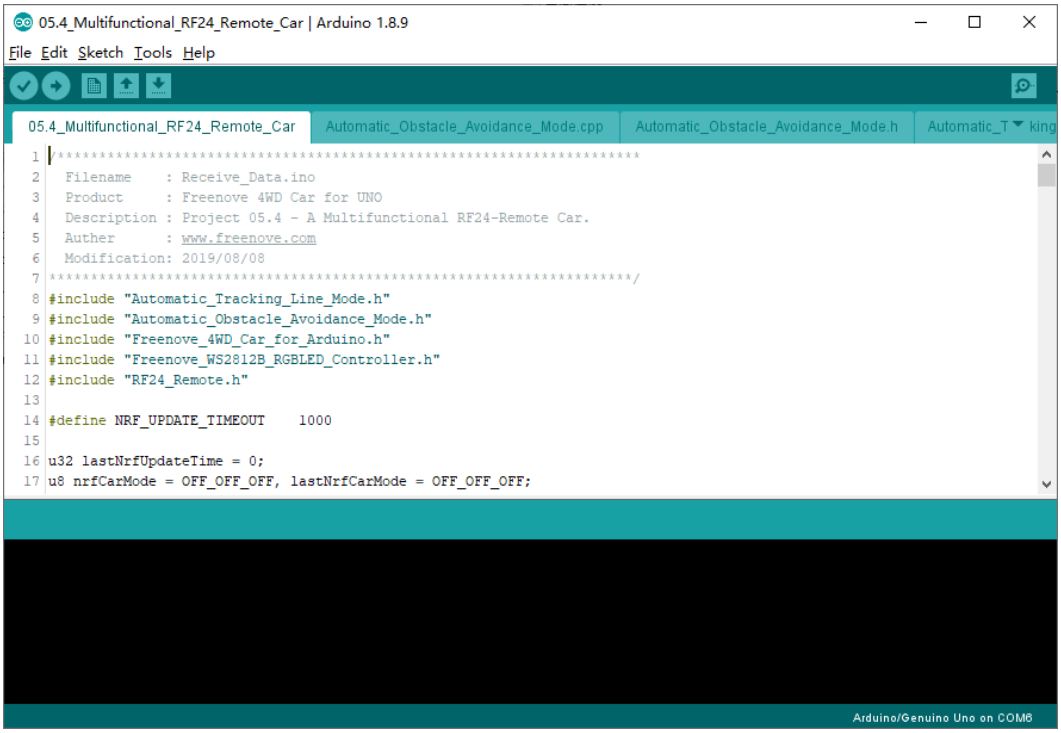
5.4 Multifunctional RF24 Remote Car

This project combines almost all functions of the car, such as obstacle avoidance mode, line tracking mode, remote control mode, RGB LED display mode, servo position calibration and so on. The switching among different functions is realized by switching the states of S1, S2, and S3 on the remote controller.

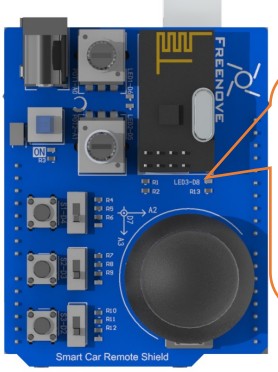
The project code is more complicated, but the components and knowledge used have been introduced in the previous project. The only difference is that they are integrated into one project, which is quite large.

Upload Code and Running

Connect Car to computer with USB cable.
And upload code in Sketches\05.4_Multifunctional_RF24_Remote_Car.ino.



Disconnect the USB cable. Turn on the power of the remote control and the car. By default, the car is in manual remote control mode, and the S1, S2, and S3 switches on the remote control are off.



LED ON state indicates the communication successes.
LED OFF state indicates the communication fails.

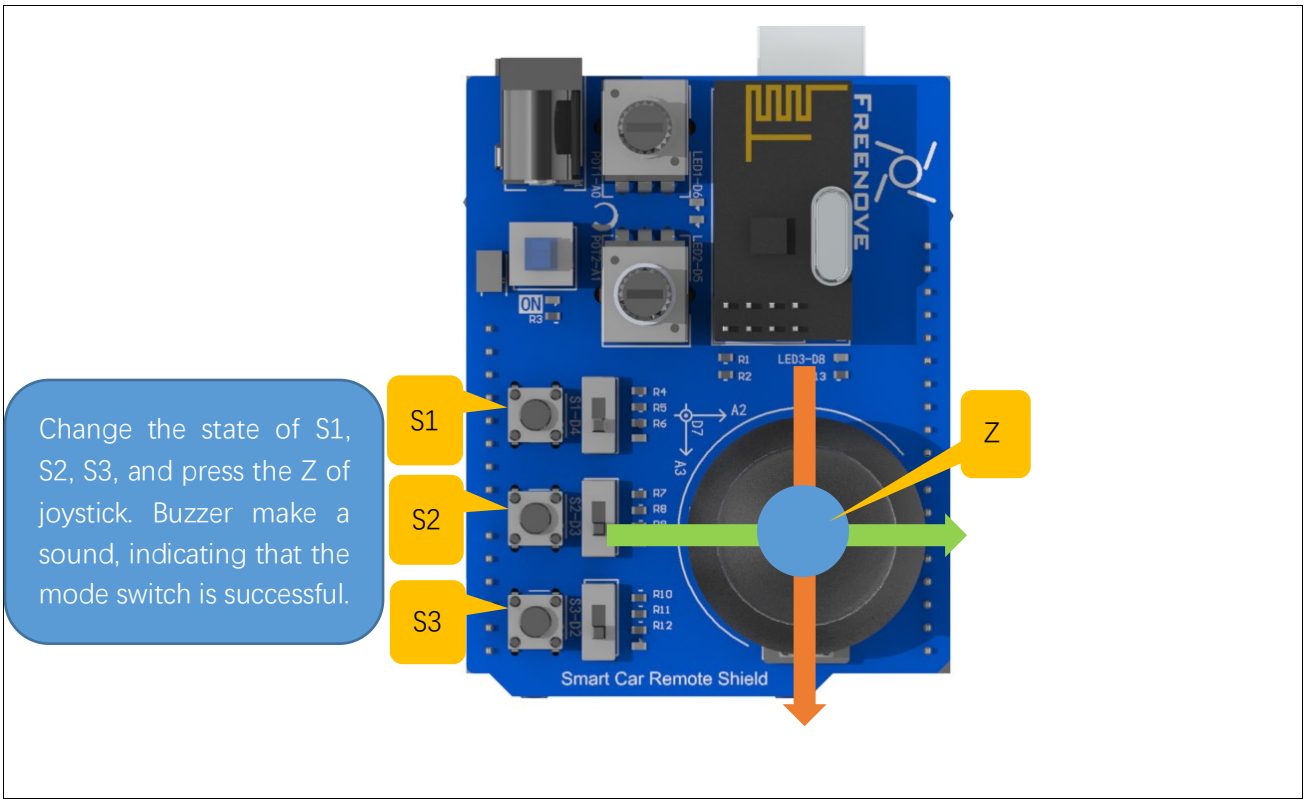
When it is OFF, you need to restart power of remote controller and car.

Switch different modes

1, Change the switch state of S1, S2, and S3, and the car will stop moving.
2, Press the Z axis of the joystick and the buzzer B sounds to indicate that the mode is successfully switched.

The following table shows the modes indicated by different states of the three switches S1, S2, and S3. The LED next to the switch illumination to indicate ON and OFF state of switches. The three switches can form 2x2x2=8 modes.

S1	S2	S3	Mode No.	Mode
ON	ON	ON	0	None
ON	ON	OFF	1	Calibrate servo mode
ON	OFF	ON	2	None
ON	OFF	OFF	3	Obstacle avoidance mode
OFF	ON	ON	4	None
OFF	ON	OFF	5	Line tracking mode
OFF	OFF	ON	6	Switch LED mode
OFF	OFF	OFF	7	Manuel control mode / Default mode



Mode 0, 2, 4

Reserved. We did not assign functions for them.

Mode 1-Calibrate servo

If your servo is not accurately mounted at 90 degrees, you can use this mode for fine adjustment (+-10 degrees).

In this mode, you can adjust potentiometer 2 (POT2) to fine tune the angle of the servo. When you adjust the servo to the correct angle, press the Z-axis of joystick to save calibration data to EEPROM. It will be saved permanently unless it is modified.

Mode 3-obstical avoidance, Mode 5-line tracking mode

These two modes have been learned separately in the previous project, and their running logic and codes are consistent with the previous project.

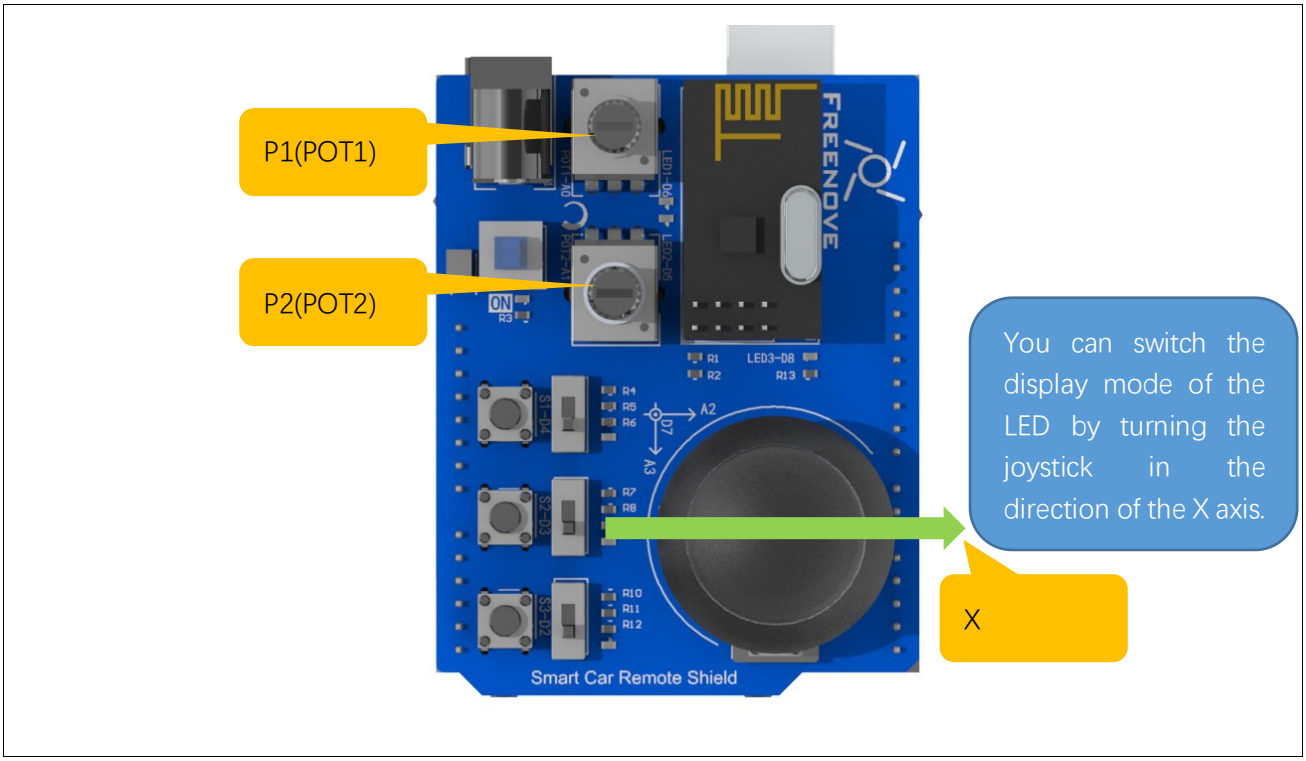
The difference is that in this project, the car can respond to commands from the remote control at any time. Therefore, in this project, it is still necessary to communicate with the remote controller in these two modes. When the remote control signal is disconnected, the car will stop. Therefore, the normal communication between the remote control and the car should be maintained at any time. Poor communication conditions may cause these two modes to work abnormally.

Mode 6-switch LED display mode

There are three display modes for the LEDs on the car, which are 0-flowing rainbow, 1-flowing water led, 2-Blink. In this mode, the display mode of the LED can be switched.

After entering this mode,
Move the joystick along the positive direction of its X-axis to switch the LED to the next mode.
Move the joystick in the negative direction of its X-axis to switch the LED to the previous mode.

In any mode, the LEDs can be adjusted with potentiometers P1 and P2. P1 is used to adjust the color change of the LED, and P2 is used to adjust the change frequency of LED.



Mode 7- manual remote mode

This mode is manual remote mode and is the default mode. This mode is consistent with the previous project "RF_Remote_Car". Use the joystick to control to move forward, move back and turn left, turn right.

Code

There are 9 (file) labels for this project, 8 of which have appeared in previous projects. In the file "05.4_Multifunctional_RF24_Remote_Car.ino", it is mainly the logical management of each module function, such as switching of the car mode, switching of the LED display mode.

05.4_Multifunctional_RF24_Remote_Car.ino

```
#include "Automatic_Tracking_Line_Mode.h"
#include "Automatic_Obstacle_Avoidance_Mode.h"
#include "Freenove_4WD_Car_for_Arduino.h"
#include "Freenove_WS2812B_RGBLED_Controller.h"
#include "RF24_Remote.h"

#define RF_UPDATE_TIMEOUT     1000

u32 lastRFUpdateTime = 0;
u8 RFCarMode = OFF_OFF_OFF, lastRFCarMode = OFF_OFF_OFF;
u8 switchModeState = MODE_SWITCHING_WAS_FINISHED;
u8 joystickSwitchState = MODE_SWITCHING_WAS_FINISHED;

#define STRIP_I2C_ADDRESS  0x20
#define STRIP_LEDS_COUNT   10

#define MAX_NUMBER_OF_DISP_MODES  3

u8 colorPos = 0;
u8 colorStep = 50;
u8 stripDisplayMode = 1;
u8 currentLedIndex = 0;
u16 stripDisplayDelay = 100;
u32 lastStripUpdateTime = 0;
Freenove_WS2812B_Controller strip(STRIP_I2C_ADDRESS, STRIP_LEDS_COUNT, TYPE_GRB);

void setup() {
    pinsSetup();
    if (!RF24L01Setup()) {
        alarm(4, 2);
    }
    servoSetup();
    while (!strip.begin());
    strip.setAllLedsColor(0xFF0000);
}

void loop() {
```

```
    if (getRF24L01Data()) {
        RFCarMode = updateRFCarMode();
        if (RFCarMode != lastRFCarMode) {
            if (switchModeState == MODE_SWITCHING_WAS_FINISHED) {
                switchModeState = MODE_SWITCHING_IS_INITIALIZING;
            }
            switch (switchModeState)
            {
            case MODE_SWITCHING_IS_INITIALIZING:
                //Serial.println("Switching Mode Init...");
                resetCarAction();
                writeServo(90);
                switchModeState = MODE_SWITCHING_IS_PROCESSING;
                break;
            case MODE_SWITCHING_IS_PROCESSING:
                if (RFDataRead[JOYSTICK_Z] == 0) {
                    //Serial.println("Switching Mode ...");
                    setBuzzer(true);
                    switchModeState = MODE_SWITCHING_IS_CONFIRMING;
                }
                break;
            case MODE_SWITCHING_IS_CONFIRMING:
                if (RFDataRead[JOYSTICK_Z] == 1) {
                    //Serial.println("Comfirm Switched.");
                    setBuzzer(false);

                    switchModeState = MODE_SWITCHING_WAS_FINISHED;
                    lastRFCarMode = RFCarMode;
                    switch (RFCarMode)
                    {
                    case MODE_OBSTACKE_AVOIDANCE:
                        oa_CalculateVoltageCompensation();
                        break;
                    case MODE_LINE_TRACKING:
                        tk_CalculateVoltageCompensation();
                        break;
                    default:
                        break;
                    }
                    //Serial.println("Switch mode finished !");
                }
                break;
            case MODE_SWITCHING_WAS_FINISHED:
                break;
```

```
82                default:
83                    break;
84                }
85            }
86            else {
87                if (switchModeState != MODE_SWITCHING_WAS_FINISHED) {
88                    switchModeState = MODE_SWITCHING_WAS_FINISHED;
89                }
90                switch (RFCarMode)
91                {
92                case ON_ON_ON:
93                    break;
94                case ON_ON_OFF:   //S1,S2 ON, S3 OFF
95                    setServoOffset(map(nrfDataRead[1], 0, 1023, -20, 20));
96                    if (RFDataRead[JOYSTICK_Z] == 0) {
97                        setBuzzer(true);
98                        writeServoOffsetToEEPROM();
99                    }
100                   else {
101                       setBuzzer(false);
102                   }
103                   break;
104               case ON_OFF_ON:
105                   break;
106               case ON_OFF_OFF:    ////Sonic Obstacle Avoidance Mode, S1 is ON and S2, S3 are
107   OFF
108                   updateAutomaticObstacleAvoidance();
109                   break;
110               case OFF_ON_ON:
111                   break;
112               case OFF_ON_OFF:    //Tracking Mode,S2 is ON and S1, S3 are OFF.
113                   updateAutomaticTrackingLine();
114                   break;
115               case OFF_OFF_ON:    //S3 is ON and S1, S2 are OFF
116                 //stripDisplayMode = RFDataRead[POT2] / 512;
117                   switch (joystickSwitchState)
118                   {
119                       static u8 switchCounter = 0;
120                   case MODE_SWITCHING_IS_INITIALIZING:
121                       if (RFDataRead[JOYSTICK_X] > 900)
122                       {
123                           setBuzzer(true);
124                           switchCounter++;
125                           joystickSwitchState = MODE_SWITCHING_IS_PROCESSING;
```

```
126                         }
127                         else if (RFDataRead[JOYSTICK_X] < 200)
128                         {
129                             setBuzzer(true);
130                             switchCounter--;
131                             joystickSwitchState = MODE_SWITCHING_IS_PROCESSING;
132                         }
133                         break;
134                     case MODE_SWITCHING_IS_PROCESSING:
135                         if ((RFDataRead[JOYSTICK_X] < 600) && (RFDataRead[JOYSTICK_X] > 400))
136                         {
137                             setBuzzer(false);
138                             joystickSwitchState = MODE_SWITCHING_IS_CONFIRMING;
139                         }
140                         break;
141                     case MODE_SWITCHING_IS_CONFIRMING:
142                         stripDisplayMode += switchCounter;
143                         if (stripDisplayMode == 0xff)
144                         {
145                             stripDisplayMode = MAX_NUMBER_OF_DISP_MODES - 1;
146                         }
147                         stripDisplayMode %= MAX_NUMBER_OF_DISP_MODES;
148                         joystickSwitchState = MODE_SWITCHING_WAS_FINISHED;
149                         break;
150                     case MODE_SWITCHING_WAS_FINISHED:
151                         switchCounter = 0;
152                         joystickSwitchState = MODE_SWITCHING_IS_INITIALIZING;
153                         break;
154                     default:
155                         break;
156                     }
157                     break;
158                 case OFF_OFF_OFF:   //Remote Mode, all switch leds are OFF
159                     updateCarActionByRFRemote();
160                     break;
161                 default:
162                     break;
163                 }
164             }
165             colorStep = map(RFDataRead[POT1], 0, 1023, 0, 255);
166             stripDisplayDelay = RFDataRead[POT2];
167             lastRFUpdateTime = millis();
168         }
169
```

```
170     if (millis() - lastRFUpdateTime > RF_UPDATE_TIMEOUT) {
171         lastRFUpdateTime = millis();
172         resetRFDataBuf();
173         updateCarActionByRFRemote();
174         RFCarMode = lastRFCarMode = MODE_REMOTE_CONTROL;
175     }
176     switch (stripDisplayMode)
177     {
178     case 0:
179         if (millis() - lastStripUpdateTime > stripDisplayDelay)
180         {
181             for (int i = 0; i < STRIP_LEDS_COUNT; i++) {
182                 strip.setLedColorData(i, strip.Wheel(colorPos + i * 25));
183             }
184             strip.show();
185             colorPos += colorStep;
186             lastStripUpdateTime = millis();
187         }
188         break;
189     case 1:
190         if (millis() - lastStripUpdateTime > stripDisplayDelay)
191         {
192             strip.setLedColor(currentLedIndex, strip.Wheel(colorPos));
193             currentLedIndex++;
194             if (currentLedIndex == STRIP_LEDS_COUNT)
195             {
196                 currentLedIndex = 0;
197                 colorPos += colorStep; //
198             }
199             lastStripUpdateTime = millis();
200         }
201         break;
202     case 2:
203         colorPos = colorStep;
204         if (millis() - lastStripUpdateTime > stripDisplayDelay)
205         {
206             static bool ledState = true;
207             if (ledState)
208             {
209                 strip.setAllLedsColor(strip.Wheel(colorPos));
210             }
211             else
212             {
213                 strip.setAllLedsColor(0x00);
```

```
214                 }
215                 ledState = !ledState;
216                 lastStripUpdateTime = millis();
217             }
218             break;
219         default:
220             break;
221         }
222 }
```

In setup(), initialize pins and RF24L01 and LED module. In loop(), receive data from the remote control and process the data. If receiving data is time out, the remote control signal is considered to be lost. LED display code is added in this code, which is different from code in the previous project.

```
void setup() {
    pinsSetup();
    if (!RF24L01Setup()) {
        alarm(4, 2);
    }
    servoSetup();
    while (!strip.begin());
    strip.setAllLedsColor(0xFF0000);
}

void loop() {
    if (getRF24L01Data()) {
……
    }

    if (millis() - lastRFUpdateTime > RF_UPDATE_TIMEOUT) {
        lastRFUpdateTime = millis();
        resetRFDataBuf();
        updateCarActionByRFRemote();
        RFCarMode = lastRFCarMode = MODE_REMOTE_CONTROL;
    }
    switch (stripDisplayMode)
    {
……
    }
}
```

After reading the remote control data, judge whether the mode of the car has changed according to the state of S1, S2, S3. If the mode is changed, the code of the corresponding mode will be executed. Otherwise, the subsequent code will be executed.

```
if (getRF24L01Data()) {
        RFCarMode = updateRFCarMode();
        if (RFCarMode != lastRFCarMode) {
            if (switchModeState == MODE_SWITCHING_WAS_FINISHED) {
                switchModeState = MODE_SWITCHING_IS_INITIALIZING;
            }
            switch (switchModeState)
            {
......
            }
......
        }
        else {
......
        }
```

When the state of S1, S2, S3 changes, it means that the user wants to change the mode of the car, but in order to ensure that this change is not an erroneous operation, the user needs to press the Z axis of the joystick to confirm.

This series of actions can be implemented using the idea of a finite state machine. When the switch state changes, the state "MODE_SWITCHING_IS_INITIALIZING" is entered, the car's power module, motor and servo are initialized to prevent the car from running uncontrolledly.

```
            case MODE_SWITCHING_IS_INITIALIZING:
                //Serial.println("Switching Mode Init...");
                resetCarAction();
                writeServo(90);
                switchModeState = MODE_SWITCHING_IS_PROCESSING;
                break;
```

Then enter next state "MODE_SWITCHING_IS_PROCESSING" and wait for the user to press the rocker Z axis to confirm. If the user presses the Z axis of the joystick, the buzzer sounds and enters the next state "MODE_SWITCHING_IS_CONFIRMING", waiting for the user to release the joystick. When the user releases the joystick, the buzzer stops sounding and changes the value of the variable switchModeState that holds the car mode. After the mode is switched, the completion status "MODE_SWITCHING_WAS_FINISHED" is entered.

```
            case MODE_SWITCHING_IS_PROCESSING:
                if (RFDataRead[JOYSTICK_Z] == 0) {
                    //Serial.println("Switching Mode ...");
                    setBuzzer(true);
                    switchModeState = MODE_SWITCHING_IS_CONFIRMING;
                }
                break;
```

```
            case MODE_SWITCHING_IS_CONFIRMING:
                if (RFDataRead[JOYSTICK_Z] == 1) {
                    //Serial.println("Comfirm Switched.");
                    setBuzzer(false);
                    switchModeState = MODE_SWITCHING_WAS_FINISHED;
                    lastRFCarMode = RFCarMode;
                    switch (RFCarMode)
                    {
                    case MODE_OBSTACKE_AVOIDANCE:
                        oa_CalculateVoltageCompensation();
                        break;
                    case MODE_LINE_TRACKING:
                        tk_CalculateVoltageCompensation();
                        break;
                    default:
                        break;
                    }
                    //Serial.println("Switch mode finished!");
                }
                break;
            case MODE_SWITCHING_WAS_FINISHED:
                break;
            default:
                break;
            }
```

Next, use the switch-case statement to let the car perform the corresponding function or action according to the value of the car mode variable RFCarMode.

```
            switch (RFCarMode)
            {
            case ON_ON_ON:
                break;
            case ON_ON_OFF:   //S1,S2 ON, S3 OFF
                ……//Set Servo offset
            case ON_OFF_ON:
                break;
            case ON_OFF_OFF:  ////Sonic Obstacle Avoidance Mode, S1 is ON and S2, S3 are OFF
                updateAutomaticObstacleAvoidance();
                break;
            case OFF_ON_ON:
                break;
            case OFF_ON_OFF:    //Tracking Mode, S2 is ON and S1, S3 are OFF.
                updateAutomaticTrackingLine();
                break;
```

```
            case OFF_OFF_ON:    //S3 is ON and S1, S2 are OFF
                ……//Change LED Mode
                break;
            case OFF_OFF_OFF:   //Remote Mode, all switch leds are OFF
                updateCarActionByRFRemote();
                break;
            default:
                break;
            }
```

In the calibration servo mode, the angle of the servo is adjusted according to the value of the potentiometer POT2. When the rocker Z-axis is pressed, the adjusted data is written into the EEPROM, and the buzzer will make a sound to confirm the writing.

```
            case ON_ON_OFF:   //S1,S2 ON, S3 OFF
                setServoOffset(RFDataRead[1] / 100);
                if (RFDataRead[JOYSTICK_Z] == 0) {
                    setBuzzer(true);
                    writeServoOffsetToEEPROM();
                }
                else {
                    setBuzzer(false);
                }
                break;
```

In the mode of switching the LED display, the idea of finite state machine is still used.
First, enter the initialization state, waiting for the joystick to move in the direction of the X axis. If there is, the variable switchCounter will be self-added or self-decremented according to the positive and negative directions. Then enter next state, wait for the joystick to return to the middle. When the joystick returns to the middle, enter the next state, process the data, after the data processing is completed, enter the completion state.

```
            switch (joystickSwitchState)
            {
                static u8 switchCounter = 0;
            case MODE_SWITCHING_IS_INITIALIZING:
                if (RFDataRead[JOYSTICK_X] > 900)
                {
                    setBuzzer(true);
                    switchCounter++;
                    joystickSwitchState = MODE_SWITCHING_IS_PROCESSING;
                }
                else if (RFDataRead[JOYSTICK_X] < 200)
                {
                    setBuzzer(true);
                    switchCounter--;
```

```
                    joystickSwitchState = MODE_SWITCHING_IS_PROCESSING;
                }
                break;
            case MODE_SWITCHING_IS_PROCESSING:
                if ((RFDataRead[JOYSTICK_X] < 600) && (RFDataRead[JOYSTICK_X] > 400))
                {
                    setBuzzer(false);
                    joystickSwitchState = MODE_SWITCHING_IS_CONFIRMING;
                }
                break;
            case MODE_SWITCHING_IS_CONFIRMING:
                stripDisplayMode += switchCounter;
                if (stripDisplayMode == 0xff)
                {
                    stripDisplayMode = MAX_NUMBER_OF_DISP_MODES - 1;
                }
                stripDisplayMode %= MAX_NUMBER_OF_DISP_MODES;
                joystickSwitchState = MODE_SWITCHING_WAS_FINISHED;
                break;
            case MODE_SWITCHING_WAS_FINISHED:
                switchCounter = 0;
                joystickSwitchState = MODE_SWITCHING_IS_INITIALIZING;
                break;
            default:
                break;
            }
            break;
```

The variable that controls the LED display color change and the variable of the display frequency are updated every time according to remote control data.

```
        colorStep = map(RFDataRead[POT1], 0, 1023, 0, 255);
        stripDisplayDelay = RFDataRead[POT2];
```

Finally, the LED display mode variable stripDisplayMode, the color change amplitude variable colorStep, the display period variable stripDisplayDelay.

```
    switch (stripDisplayMode)
    {
    case 0:
        if (millis() - lastStripUpdateTime > stripDisplayDelay)
        {
            for (int i = 0; i < STRIP_LEDS_COUNT; i++) {
                strip.setLedColorData(i, strip.Wheel(colorPos + i * 25));
            }
            strip.show();
```

```
            colorPos += colorStep;
            lastStripUpdateTime = millis();
        }
        break;
    case 1:
        if (millis() - lastStripUpdateTime > stripDisplayDelay)
        {
            strip.setLedColor(currentLedIndex, strip.Wheel(colorPos));
            currentLedIndex++;
            if (currentLedIndex == STRIP_LEDS_COUNT)
            {
                currentLedIndex = 0;
                colorPos += colorStep; //
            }
            lastStripUpdateTime = millis();
        }
        break;
    case 2:
        colorPos = colorStep;
        if (millis() - lastStripUpdateTime > stripDisplayDelay)
        {
            static bool ledState = true;
            if (ledState)
            {
                strip.setAllLedsColor(strip.Wheel(colorPos));
            }
            else
            {
                strip.setAllLedsColor(0x00);
            }
            ledState = !ledState;
            lastStripUpdateTime = millis();
        }
        break;
    default:
        break;
    }
```

Chapter 6 Bluetooth control

This section requires an Android or iPhone device with Bluetooth to control the car.
If you have any concerns, please feel free to contact us via support@freenove.com

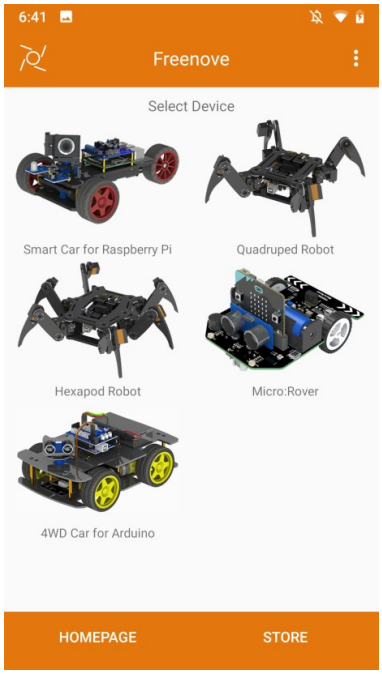
6.1 Set Bluetooth and Receive Data

Android and iPhone iOS App

Download and install app. You can use app to control the robot.
You can download Freenove App through below ways:

View or download on Google Play:
https://play.google.com/store/apps/details?id=com.freenove.suhayl.Freenove
Download APK file directly
https://github.com/Freenove/Freenove_App_for_Android/raw/master/freenove.apk
Then install it on your Android phone.

For **iPhone app**, please serach **freenove** in **App Store**.



Instructions for using the Bluetooth module

The Bluetooth module uses the serial port to communicate with the Arduino. The serial communication is also used to upload the program to Arduino. **Therefore, when uploading the program to Arduino, the Bluetooth module should be unplugged; otherwise, the program upload will fail!**

When the Bluetooth module is not connected by other device, the Bluetooth module can be configured using the AT command. Once connected, the Bluetooth module acts as a data pipe and cannot be configured.

By default, the Bluetooth module has a baud rate of 9600, no parity, 8 data bits, and 1 stop bit. Bluetooth name "BT05", role mode is slave mode.

Set name of bluetooth

If you have multiple Bluetooth modules with the same name around you, you will be confused when you connect. Which one is the Bluetooth module I want to connect to?

In the next project, we will introduce how to use the AT command to modify the name of the Bluetooth module and the master-slave role in the program.

For AT commands and more information about the Bluetooth module, refer to the documentation in the package for Datasheets/BT05-Instruction.pdf

This project uses the AT command to make some settings for the Bluetooth module, then uses the Bluetooth module to receive data from the app and print the data on the serial monitor.

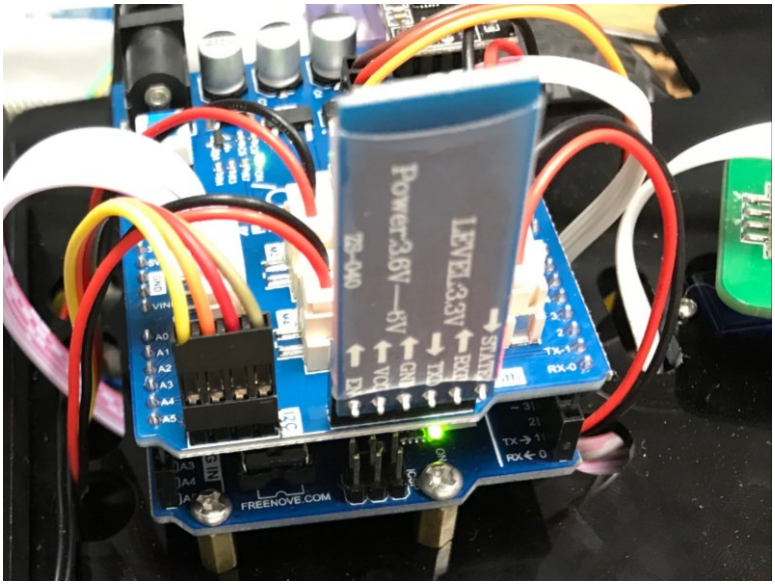
Upload Code and Running

Unplug Bluetooth. Then upload code in Sketches/06.1_Receive_Bluetooth_Data.ino.

Note, when upload code, we need unplug Bluetooth first. Or the uploading will fail.

After the code is successfully uploaded, follow the steps below.

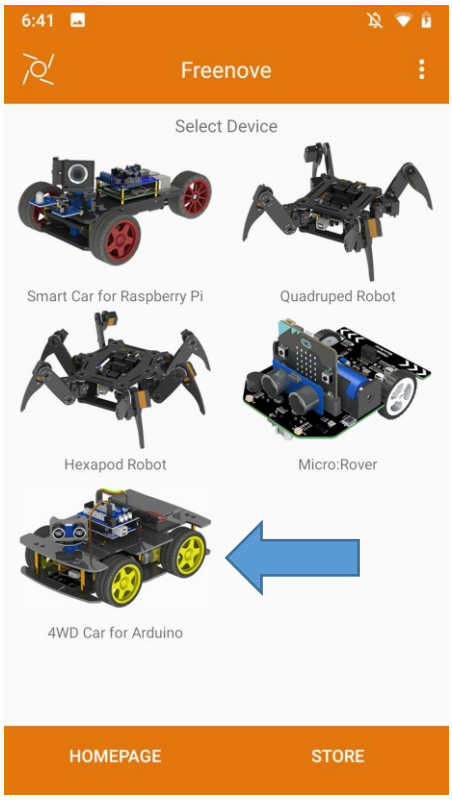
1. Plug the Bluetooth module on car as shown below. **Don't reverse it. The wrong connection may damage your hardware.**



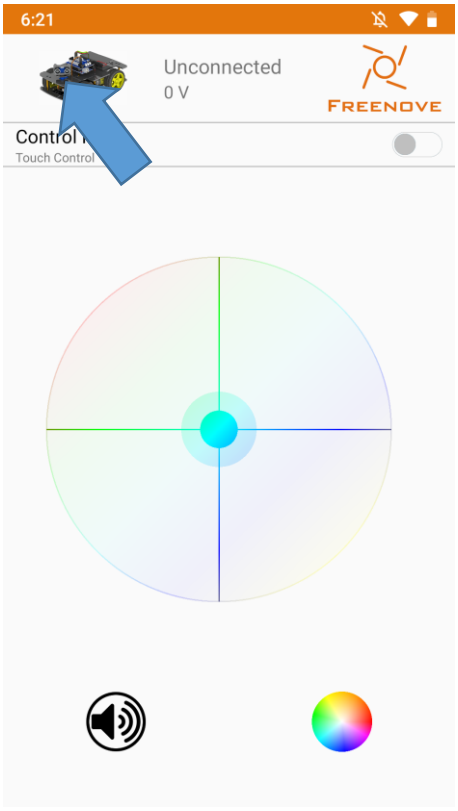
2. Open Arduino monitor.

As shown in the figure below, the Bluetooth name is set to "BT05" and the Bluetooth mode is slave mode (ROLE=0).

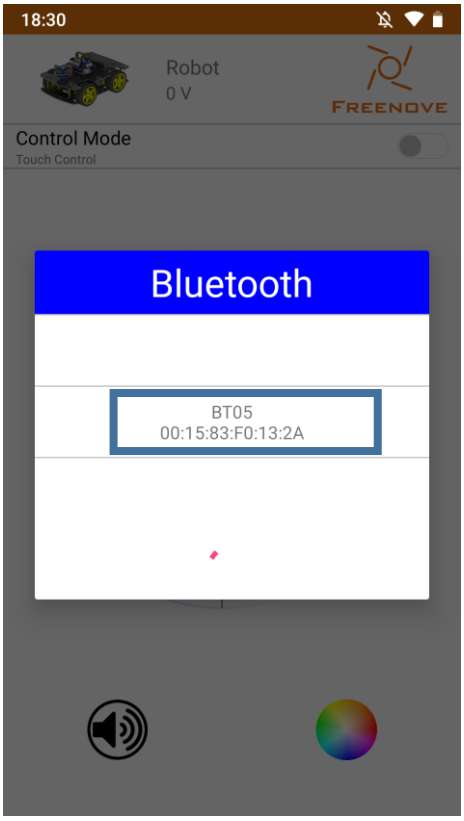
Next, open the app and click the 4WD Car for Arduino.



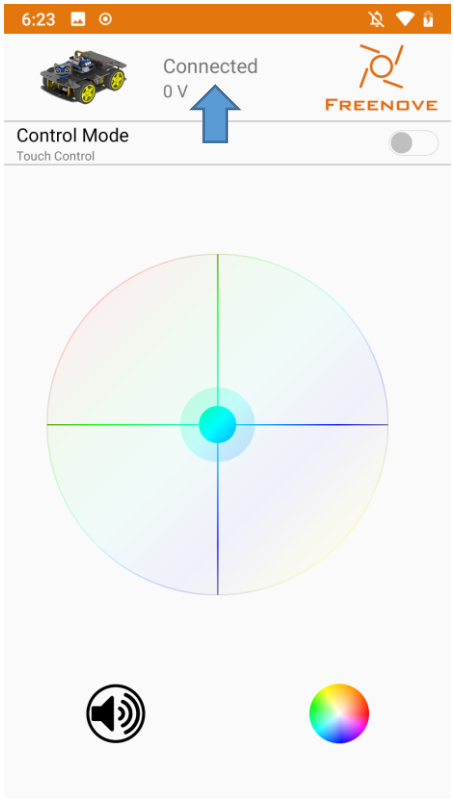
Click following icon.



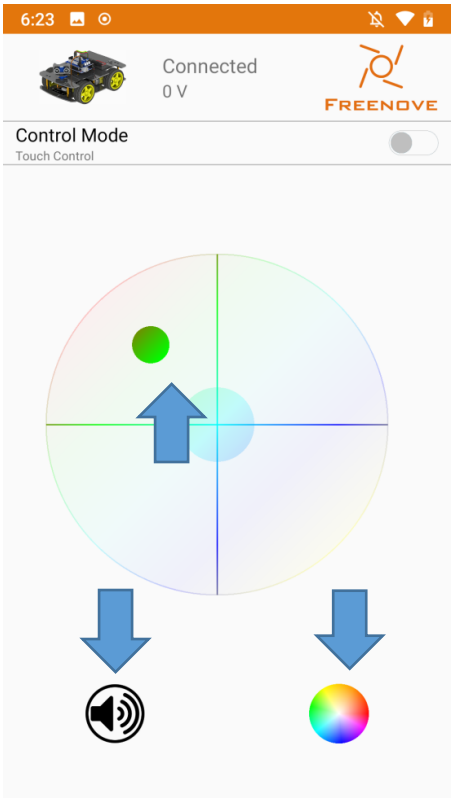
Click BT05.



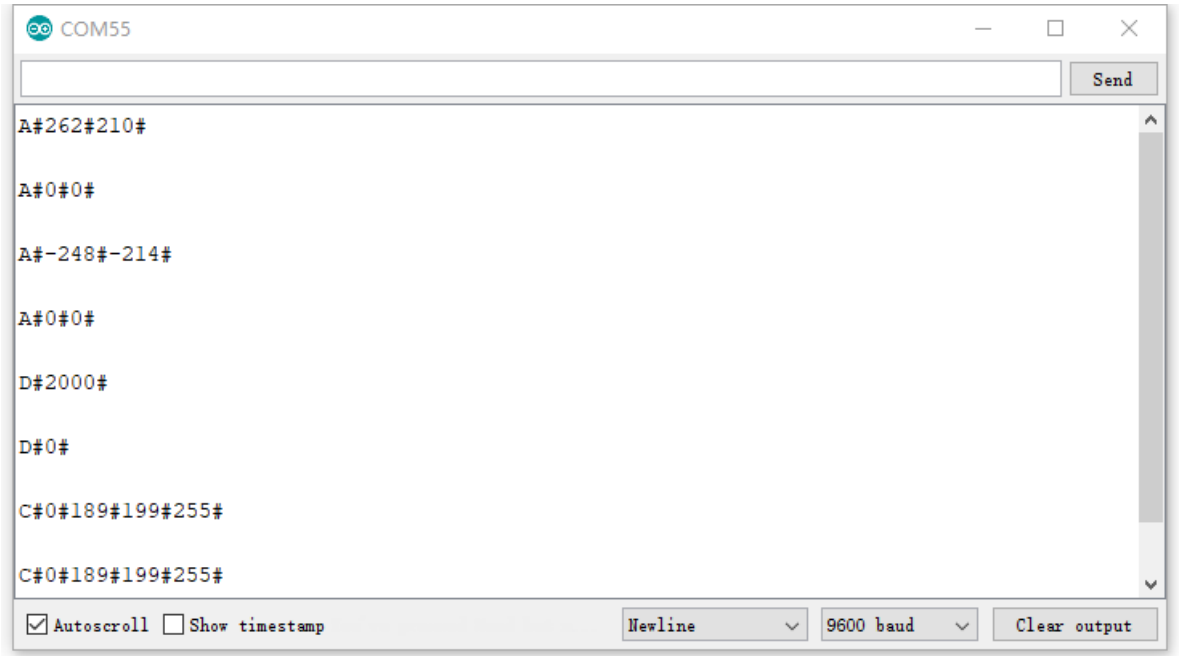
If the connection successes, it will be connected.



Operate the app. And observe the monitor.



The monitor will show content below:



Bluetooth data- car action

The command format for communication between app and car is A#xxx#xxx#...xxx#, where # is a separator, the first character A represents the action command, it can be other characters, such as B, C, D…The xxx represents the parameters of the action command. And different commands carry different parameters. The command list is as below:

Action command	Description	Command character	Number of parameters (app send/receive)	Format example (app send/receive)
MOVE	Move, parameters are speeds of the two motors	A	2	A#100#100#
STOP	Stop moving	B	0	B#
LED_RGB	Control RGBLED, parameters respectively are serial number of LED mode, red value, green value, blue value.	C	3	C#2#100#150#200#
BUZZER	Control buzzer, and parameter is the frequency.	D	1	D#2000#
VOLTAGE	Get the batteries voltage, parameter is the voltage value, unit mV	I	0/1	I# /I#4100#

Code

06.1_Receive_Bluetooth_Data

The data of the Bluetooth module is sent to the Arduino through the serial port. The project code comes from the Arduino example code "SerialEvent". Receive data from the serial port and print the data when '\n' is received.

The code is below:

```
1  String inputString = "";         // a String to hold incoming data
2  bool stringComplete = false;  // whether the string is complete
3
4  void setup() {
5      // initialize serial:
6      Serial.begin(9600); //default data transmission baud rate of the Bluetooth module is 9600
7      Serial.println("AT+NAMEBT05");// Or use Serial.print("AT+NAMEBT05\r\n");
8      delay(200);
9      Serial.println("AT+ROLE0");
10     delay(200);
11     // reserve 200 bytes for the inputString:
12     inputString.reserve(200);
13 }
14
```

```
15  void loop() {
16      // print the string when a newline arrives:
17      if (stringComplete) {
18          Serial.println(inputString);
19          // clear the string:
20          inputString = "";
21          stringComplete = false;
22      }
23  }
24
25  /*
26    SerialEvent occurs whenever a new data comes in the hardware serial RX. This
27    routine is run between each time loop() runs, so using delay inside loop can
28    delay response. Multiple bytes of data may be available.
29  */
30  void serialEvent() {
31      while (Serial.available()) {
32          // get the new byte:
33          char inChar = (char)Serial.read();
34          // add it to the inputString:
35          inputString += inChar;
36          // if the incoming character is a newline, set a flag so the main loop can
37          // do something about it:
38          if (inChar == '\n') {
39              stringComplete = true;
40          }
41      }
42  }
```

The default data transmission baud rate of the Bluetooth module is 9600, so set serial port baud rate to 9600.

```
    Serial.begin(9600); //default data transmission baud rate of the Bluetooth module is 9600
```

Then use the AT command to set the name of the Bluetooth module to "BT05". The format of the command is "AT+NAMExxx\r\n", where "AT+NAME" is a fixed format, and the following "xxx" is the set name. The maximum length of the name is 18 characters. The command must be followed by "\r\n" ends. Since the Serial.pirntln() function adds "\r\n" to the end, there is no need to add it in the program. The delay is 200ms to ensure that the Bluetooth module has enough time to complete the setup.

```
    Serial.println("AT+NAMEBT05");// Or use Serial.print("AT+NAMEBT05\r\n");
    delay(200);
```

Similarly, continue to use the AT command to set the Bluetooth module's role mode to slave mode. The Bluetooth module can be set to master mode or slave mode. In master mode, the Bluetooth module can actively search for and connect to other Bluetooth devices, but cannot be searched and connected by other devices. In order to be searched by the mobile app via Bluetooth, the Bluetooth role module needs to be set to the slave mode. According to the Bluetooth manufacturer's manual, 0 is the slave mode and 1 is the master mode.

```
    Serial.println("AT+ROLE0");
    delay(200);
```

In loop(), keep querying if stringComplete is true. If it is true, it indicates that the complete data has been received. The data will be printed on the serial monitor, and then the data is cleared. And then the array is ready to receive new data.

```
    if (stringComplete) {
        Serial.println(inputString);
        // clear the string:
        inputString = "";
        stringComplete = false;
    }
```

Function serialEvent() is used to receive the data transmitted by the serial port. When the newline character is received, the flag stringComplete is set to true, and the received data will be printed in loop().

```
void serialEvent() {
    while (Serial.available()) {
        // get the new byte:
        char inChar = (char)Serial.read();
        // add it to the inputString:
        inputString += inChar;
        // if the incoming character is a newline, set a flag so the main loop can
        // do something about it:
        if (inChar == '\n') {
            stringComplete = true;
        }
    }
}
```

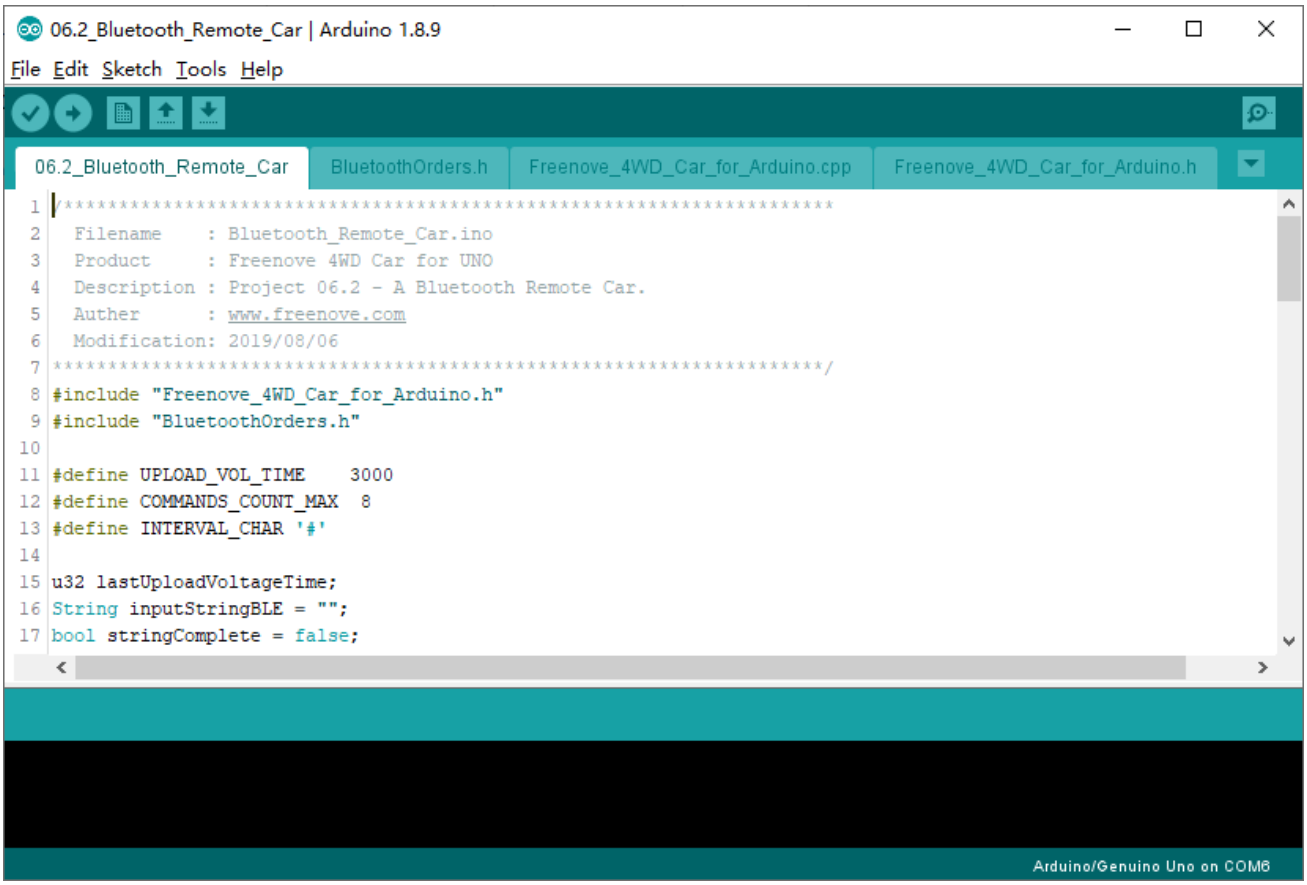
Bluetooth AT Command
AT command mode when the module is not connected. AT command, which belongs to the character line instruction, is parsed according to the line (That is, AT command must be returned by carriage return or \r\n, hexadecimal number is 0D0A). For more details about AT command, please refer to Datasheets/BT05-Instruction.pdf.
void serialEvent()
SerialEvent occurs whenever a new data comes in the hardware serial RX. This routine is run between each time loop() runs, so using delay inside loop can delay response. Multiple bytes of data may be available.

6.2 Bluetooth Remote Car

After learning how to receive the app's data via Bluetooth and the meaning of data formats, use the data to let the car do something. Control the car movement, the buzzer sounds, and the LED switches the display mode.

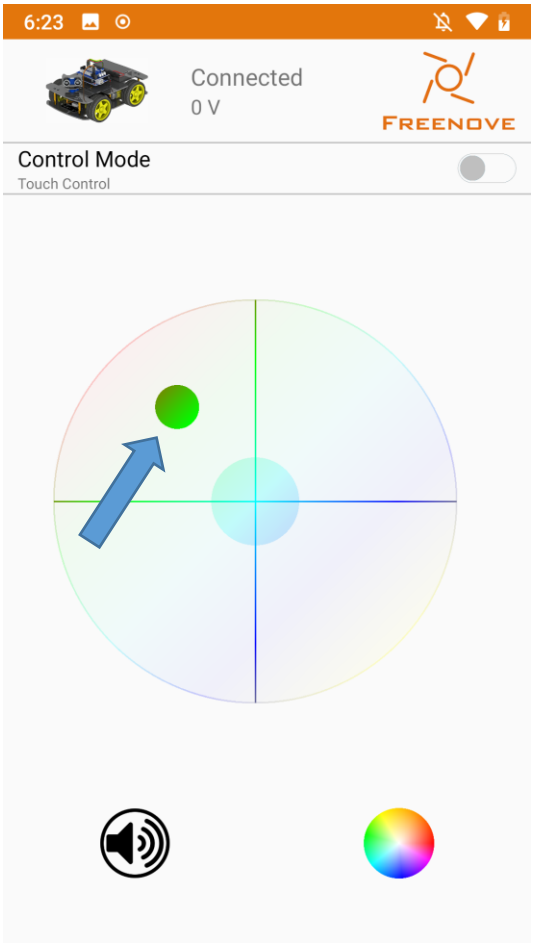
Upload Code and Running

Unplug Bluetooth. And connect car to computer with USB cable.
Then upload code in Sketches/06.2_Bluetooth_Remote_Car.ino.



After the code is successfully uploaded, unplug the USB cable and plug the Bluetooth module on car. Turn on the car power switch.

According to the previous way, the Bluetooth module of the car is connected through app, then the car can be controlled by clicking or sliding the operation panel on the APP.



Code

The project has 4 labels, the label "BluetoothOrders.h" stores Bluetooth communication, and command characters are used to control the car, only a part of which is used. The main label "06.2_Bluetooth_Remote_Car.ino" is the main content of this project.

BluetoothOrders.h

```
#ifndef _BLUETOOTHORDERS_h
#define _BLUETOOTHORDERS_h

#define ACTION_MOVE    'A'
#define ACTION_STOP    'B'
#define ACTION_RGB    'C'
#define ACTION_BUZZER    'D'
#define ACTION_ULTRASONIC    'E'
#define ACTION_LIGHT_TRACING    'F'
#define ACTION_TRACKING    'G'
#define ACTION_CAR_MODE    'H'
```

```
#define ACTION_GET_VOLTAGE   'I'
#define ECHO_OK   'J'
#define ACTION_NONE   'K'

#endif
```

06.2_Bluetooth_Remote_Car.ino

```
#include "Freenove_4WD_Car_for_Arduino.h"
#include "BluetoothOrders.h"

#define UPLOAD_VOL_TIME    3000
#define COMMANDS_COUNT_MAX  8
#define INTERVAL_CHAR '#'

u32 lastUploadVoltageTime;
String inputStringBLE = "";
bool stringComplete = false;

void setup() {
    pinsSetup();
    Serial.begin(9600);

}

void loop() {
    if (millis() - lastUploadVoltageTime > UPLOAD_VOL_TIME) {
        upLoadVoltageToApp();
        lastUploadVoltageTime = millis();
    }
    if (stringComplete) {
        String inputCommandArray[COMMANDS_COUNT_MAX];
        int paramters[COMMANDS_COUNT_MAX], paramterCount = 0;
        String inputStringTemp = inputStringBLE;
        for (u8 i = 0; i < COMMANDS_COUNT_MAX; i++) {
            int index = inputStringTemp.indexOf(INTERVAL_CHAR);
            if (index < 0) {
                break;
            }
            paramterCount = i;  //
            inputCommandArray[i] = inputStringTemp.substring(0, index);
            inputStringTemp = inputStringTemp.substring(index + 1);
            paramters[i] = inputCommandArray[i].toInt();
        }
        stringComplete = false;
```

```
38          inputStringBLE = "";
39
40          char commandChar = inputCommandArray[0].charAt(0);
41          switch (commandChar)
42          {
43          case ACTION_MOVE:
44              if (paramterCount == 2) {
45                  motorRun(paramters[1], paramters[2]);
46              }
47              break;
48          case ACTION_BUZZER:
49              if (paramterCount == 1) {
50                  setBuzzer(paramters[1]);
51              }
52              break;
53          default:
54              break;
55          }
56      }
57  }
58
59  void upLoadVoltageToApp() {
60      int voltage = 0;
61      if (getBatteryVoltage()) {
62          voltage = batteryVoltage * 1000;
63      }
64      String sendString = String(ACTION_GET_VOLTAGE) + String(INTERVAL_CHAR) +
65  String((voltage)) + String(INTERVAL_CHAR);
66      Serial.println(sendString);
67  }
68
69  void serialEvent() {
70      while (Serial.available()) {
71          char inChar = (char)Serial.read();
72          inputStringBLE += inChar;
73          if (inChar == '\n') {
74              stringComplete = true;
75          }
76      }
77  }
```

In loop (), the voltage value is uploaded to app at intervals.

```
      if (millis() - lastUploadVoltageTime > UPLOAD_VOL_TIME) {
          upLoadVoltageToApp();
```

```
        lastUploadVoltageTime = millis();
    }
```

In the sub-function upLoadVoltageToApp(), first read the battery voltage, then convert the voltage unit to mv, and send it in the format of "I#xxx#". Use Serial.println() to send a Newline character '\n', after sending the previous data,

```
void upLoadVoltageToApp() {
    int voltage = 0;
    if (getBatteryVoltage()) {
        voltage = batteryVoltage * 1000;
    }
    String sendString = String(ACTION_GET_VOLTAGE) + String(INTERVAL_CHAR) +
String((voltage)) + String(INTERVAL_CHAR);
    Serial.println(sendString);
}
```

Then in loop(), if the required data is received, the data will be parsed into commands and parameters and will be saved in an array.

Suppose you receive a command string of "A#100#200#". First use the String.indexOf() function to find the position of the separator "#". And then use the String.substring() function to split the command string into "A" and "100#200#", and save the first element "A" that is split in the array "inputCommandArray".

And then continue to repeat this operation on the split string "100#200#". You can split the element "A" into "100" and "200" respectively and save them in arrays. Finally, convert these elements to integer types and save them in the parameter array paramters.

```
    if (stringComplete) {
        String inputCommandArray[COMMANDS_COUNT_MAX];
        int paramters[COMMANDS_COUNT_MAX], paramterCount = 0;
        String inputStringTemp = inputStringBLE;
        for (u8 i = 0; i < COMMANDS_COUNT_MAX; i++) {
            int index = inputStringTemp.indexOf(INTERVAL_CHAR);
            if (index < 0) {
                break;
            }
            paramterCount = i;  //
            inputCommandArray[i] = inputStringTemp.substring(0, index);
            inputStringTemp = inputStringTemp.substring(index + 1);
            paramters[i] = inputCommandArray[i].toInt();
        }
.....
```

Finally, let the car perform different actions according to the commands and parameters.

```
        char commandChar = inputCommandArray[0].charAt(0);
        switch (commandChar)
        {
```

```
        case ACTION_MOVE:
            if (paramterCount == 2) {
                motorRun(paramters[1], paramters[2]);
            }
            break;
        case ACTION_BUZZER:
            if (paramterCount == 1) {
                setBuzzer(paramters[1]);
            }
            break;
        default:
            break;
        }
```

indexOf()

Description

Locates a character or String within another String. By default, it searches from the beginning of the String, but it can also start from a given index, allowing for the locating of all instances of the character or String.

Syntax

myString.indexOf(val)

myString.indexOf(val, from)

Parameters

myString: a variable of type String.

val: the value to search for. Allowed data types: char, String.

from: the index to start the search from.

Returns

The index of val within the String, or -1 if not found.

substring()

Description

Get a substring of a String. The starting index is inclusive (the corresponding character is included in the substring), but the optional ending index is exclusive (the corresponding character is not included in the substring). If the ending index is omitted, the substring continues to the end of the String.

Syntax

myString.substring(from)

myString.substring(from, to)

Parameters

myString: a variable of type String.

from: the index to start the substring at.

to (optional): the index to end the substring before.

Returns

The substring.

toInt()

Description

Converts a valid String to an integer. The input String should start with an integer number. If the String

contains non-integer numbers, the function will stop performing the conversion.

Syntax

myString.toInt()

Parameters

myString: a variable of type String.

Returns

If no valid conversion could be performed because the String doesn't start with an integer number, a zero is returned. Data type: long.

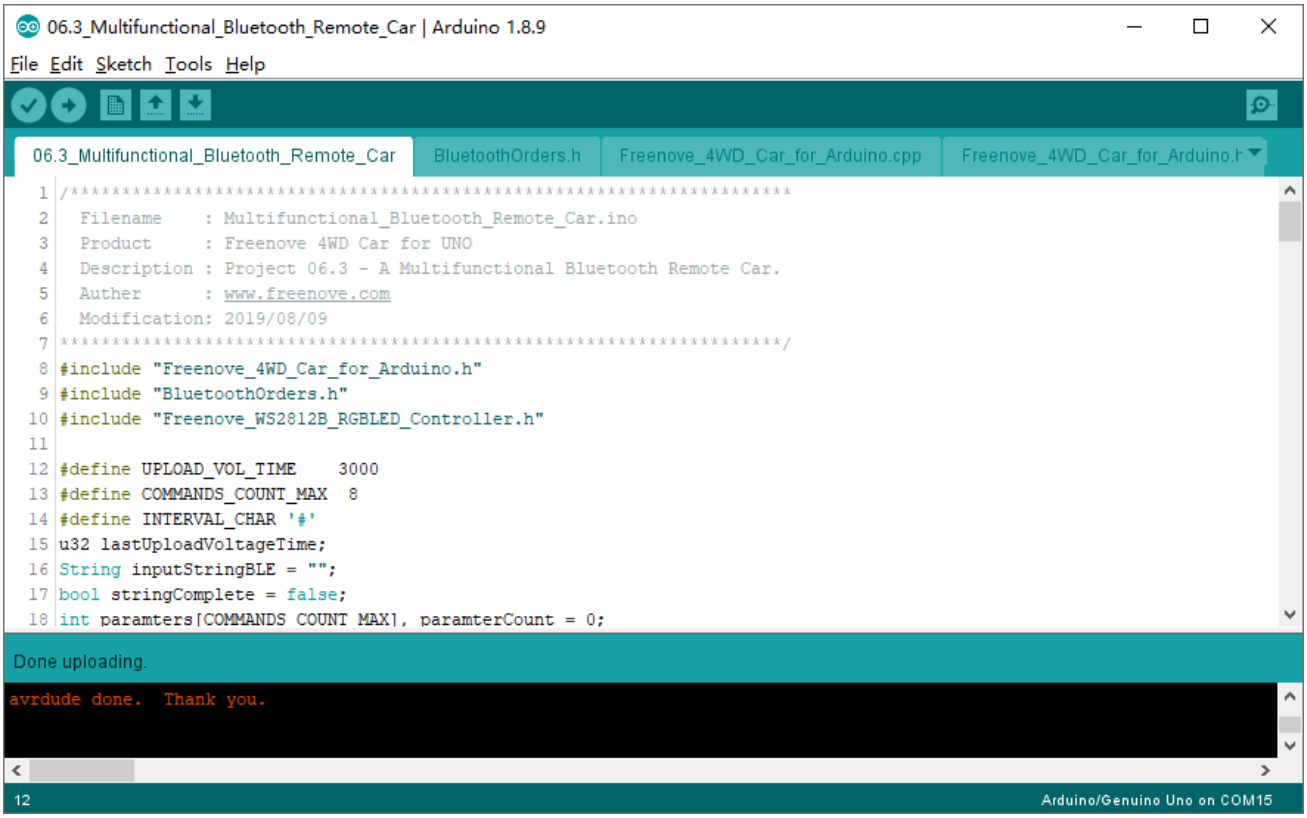
More information refer to: https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/

6.3 Multifunctional Bluetooth Remote Car

This project adds the function of LED display and the function of LED display mode switching on the basis of the previous project.

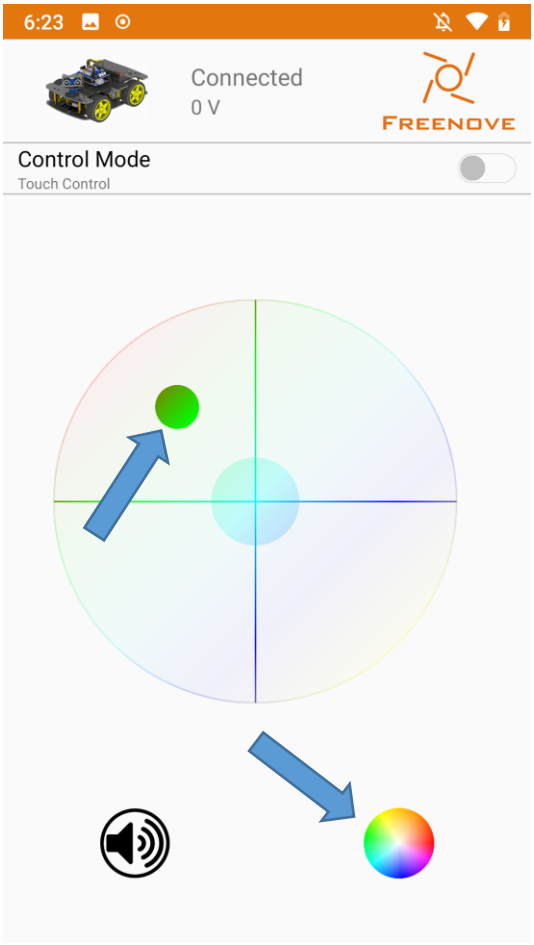
Upload Code and Running

Unplug Bluetooth. And connect car to computer with USB cable.
And upload code in Sketches/06.3_Multifunctional_Bluetooth_Remote_Car.ino
Don't separate the files in the folder.



After the code is successfully uploaded, unplug the USB cable and plug the Bluetooth module on car. Turn on the car power switch.

According to the previous method, connect Bluetooth module of the car with the app, and the operation panel of the APP can be clicked or swiped to control the movement of the car. And the color palette of the lower right corner can be clicked to control the display mode and color of the LED. Among them,
Mode 0 is a flowing rainbow,
Mode 1 is a flowing water LED with color changing.
Mode 2 is a color-adjustable Blink.
Mode 3 displays the selected color of the current color picker for all LEDs.



Code

The code for this project is based on the previous engineering code, adding code related to LED control. Nothing else has changed.

06.3_Multifunctional_Bluetooth_Remote_Car

```
#include "Freenove_4WD_Car_for_Arduino.h"
#include "Automatic_Obstacle_Avoidance_Mode.h"
#include "Automatic_Tracking_Line_Mode.h"
#include "BluetoothOrders.h"
#include "Freenove_WS2812B_RGBLED_Controller.h"

#define UPLOAD_VOL_TIME     3000
```

```
#define COMMANDS_COUNT_MAX  8
#define INTERVAL_CHAR '#'
u32 lastUploadVoltageTime;
String inputStringBLE = "";
bool stringComplete = false;
int paramters[COMMANDS_COUNT_MAX], paramterCount = 0;
int bleCarMode = MODE_NONE;

#define STRIP_I2C_ADDRESS  0x20
#define STRIP_LEDS_COUNT   10

u8 colorPos = 0;
u8 colorStep = 50;
u8 stripDisplayMode = 1;
u8 currentLedIndex = 0;
u16 stripDisplayDelay = 100;
u32 lastStripUpdateTime = 0;
Freenove_WS2812B_Controller strip(STRIP_I2C_ADDRESS, STRIP_LEDS_COUNT, TYPE_GRB);

void setup() {
  pinsSetup();
  Serial.begin(9600);
  servoSetup();
  while (!strip.begin());
  strip.setAllLedsColor(0xFF0000);
}

void loop() {
  if (millis() - lastUploadVoltageTime > UPLOAD_VOL_TIME) {
    upLoadVoltageToApp();
    lastUploadVoltageTime = millis();
  }
  if (stringComplete) {
    String inputCommandArray[COMMANDS_COUNT_MAX];

    String inputStringTemp = inputStringBLE;
    for (u8 i = 0; i < COMMANDS_COUNT_MAX; i++) {
      int index = inputStringTemp.indexOf(INTERVAL_CHAR);
      if (index < 0) {
        break;
      }
      paramterCount = i;  //
      inputCommandArray[i] = inputStringTemp.substring(0, index);
      inputStringTemp = inputStringTemp.substring(index + 1);
```

```
52          paramters[i] = inputCommandArray[i].toInt();
53        }
54        stringComplete = false;
55        inputStringBLE = "";
56
57        char commandChar = inputCommandArray[0].charAt(0);
58        switch (commandChar)
59        {
60          case ACTION_MOVE:
61            if (paramterCount == 2) {
62              motorRun(paramters[1], paramters[2]);
63            }
64            break;
65          case ACTION_CAR_MODE:
66            if (paramterCount == 1) {
67              bleCarMode = paramters[1];
68              switch (bleCarMode)
69              {
70                case MODE_NONE: case MODE_GRAVITY:
71                  resetCarAction();
72                  writeServo(OA_SERVO_CENTER);
73                  break;
74                case MODE_ULTRASONIC:
75                  oa_CalculateVoltageCompensation();
76                  break;
77                case MODE_TRACKING:
78                  tk_CalculateVoltageCompensation();
79                  break;
80                default:
81                  break;
82              }
83            }
84            break;
85          case ACTION_BUZZER:
86            if (paramterCount == 1) {
87              setBuzzer(paramters[1]);
88            }
89            break;
90          case ACTION_RGB:
91            if (paramterCount == 4) {
92              stripDisplayMode = paramters[1];
93              switch (stripDisplayMode)
94              {
95                case 0:
```

```
96                colorStep = 5;
97                stripDisplayDelay = 100;
98                break;
99              case 1:
100               colorStep = 50;
101               stripDisplayDelay = 50;
102               break;
103             case 2:
104               colorStep = 5;
105               stripDisplayDelay = 500;
106               break;
107             case 3:
108               break;
109             default:
110               break;
111           }
112         }
113         break;
114       default:
115         break;
116     }
117   }
118   switch (bleCarMode)
119   {
120     case MODE_NONE: case MODE_GRAVITY:
121       break;
122     case MODE_ULTRASONIC:
123       upLoadSonarValueToApp();
124       updateAutomaticObstacleAvoidance();
125       break;
126     case MODE_TRACKING:
127       updateAutomaticTrackingLine();
128       break;
129     default:
130       break;
131   }
132   static u8 lastColor[3];
133   switch (stripDisplayMode)
134   {
135     case 0:
136       if (millis() - lastStripUpdateTime > stripDisplayDelay)
137       {
138         for (int i = 0; i < STRIP_LEDS_COUNT; i++) {
139           strip.setLedColorData(i, strip.Wheel(colorPos + i * 25));
```

```
        }
        strip.show();
        colorPos += colorStep;
        lastStripUpdateTime = millis();
      }
      break;
    case 1:
      if (millis() - lastStripUpdateTime > stripDisplayDelay)
      {
        strip.setLedColor(currentLedIndex, strip.Wheel(colorPos));
        currentLedIndex++;
        if (currentLedIndex == STRIP_LEDS_COUNT)
        {
          currentLedIndex = 0;
          colorPos += colorStep; //
        }
        lastStripUpdateTime = millis();
      }
      break;
    case 2:
      colorPos = colorStep;
      if (millis() - lastStripUpdateTime > stripDisplayDelay)
      {
        static bool ledState = true;
        if (ledState)
        {
          strip.setAllLedsColor(paramters[2], paramters[3], paramters[4]);
        }
        else
        {
          strip.setAllLedsColor(0x00);
        }
        ledState = !ledState;
        lastStripUpdateTime = millis();
      }
      break;
    case 3:
      if (lastColor[0] != paramters[2] || lastColor[1] != paramters[3] || lastColor[2] !=
paramters[4])
      {
        strip.setAllLedsColor(paramters[2], paramters[3], paramters[4]);
        lastColor[0] = paramters[2];
        lastColor[1] = paramters[3];
        lastColor[2] = paramters[4];
```

```
184         }
185         break;
186       default:
187         break;
188     }
189 }
190 
191 void upLoadVoltageToApp() {
192   int voltage = 0;
193   if (getBatteryVoltage()) {
194     voltage = batteryVoltage * 1000;
195   }
196   String sendString = String(ACTION_GET_VOLTAGE) + String(INTERVAL_CHAR) + String((voltage)) +
197 String(INTERVAL_CHAR);
198   Serial.println(sendString);
199 }
200 
201 extern int distance[3];
202 void upLoadSonarValueToApp() {
203   String sendString = String(ACTION_ULTRASONIC) + String(INTERVAL_CHAR) +
204 String((distance[1])) + String(INTERVAL_CHAR);
205   Serial.println(sendString);
206 }
207 
208 void serialEvent() {
209   while (Serial.available()) {
210     char inChar = (char)Serial.read();
211     inputStringBLE += inChar;
212     if (inChar == '\n') {
213       stringComplete = true;
214     }
215   }
216 }
```

There are three action modes. MODE_GAVITY, MODE_ULTRASONIC and MODE_TRACKING.

```
      case ACTION_CAR_MODE:
        if (paramterCount == 1) {
          bleCarMode = paramters[1];
          switch (bleCarMode)
          {
            case MODE_NONE: case MODE_GRAVITY:
              resetCarAction();
              writeServo(OA_SERVO_CENTER);
              break;
```

```
            case MODE_ULTRASONIC:
              oa_CalculateVoltageCompensation();
              break;
            case MODE_TRACKING:
              tk_CalculateVoltageCompensation();
              break;
            default:
              break;
          }
        }
        break;
```

In the app, four LED display modes can be switched and different color change ranges and color change periods are set according to different modes. That is, the values of the variables colorStep and stripDisplayDelay.

```
switch (commandChar)
    {
      case ACTION_MOVE:
        if (paramterCount == 2) {
          motorRun(paramters[1], paramters[2]);
        }
        break;
      case ACTION_CAR_MODE:
        if (paramterCount == 1) {
          bleCarMode = paramters[1];
          switch (bleCarMode)
          {
            case MODE_NONE: case MODE_GRAVITY:
              resetCarAction();
              writeServo(OA_SERVO_CENTER);
              break;
            case MODE_ULTRASONIC:
              oa_CalculateVoltageCompensation();
              break;
            case MODE_TRACKING:
              tk_CalculateVoltageCompensation();
              break;
            default:
              break;
          }
        }
        break;
      case ACTION_BUZZER:
        if (paramterCount == 1) {
```

```
            setBuzzer(paramters[1]);
          }
          break;
        case ACTION_RGB:
          if (paramterCount == 4) {
            stripDisplayMode = paramters[1];
            switch (stripDisplayMode)
            {
              case 0:
                colorStep = 5;
                stripDisplayDelay = 100;
                break;
              case 1:
                colorStep = 50;
                stripDisplayDelay = 50;
                break;
              case 2:
                colorStep = 5;
                stripDisplayDelay = 500;
                break;
              case 3:
                break;
              default:
                break;
            }
          }
          break;
        default:
          break;
      }
    }
```

In loop(), the LED is displayed based on the received mode and parameters.

```
    static u8 lastColor[3];
    switch (stripDisplayMode)
    {
    case 0:
        if (millis() - lastStripUpdateTime > stripDisplayDelay)
        {
            for (int i = 0; i < STRIP_LEDS_COUNT; i++) {
                strip.setLedColorData(i, strip.Wheel(colorPos + i * 25));
            }
            strip.show();
            colorPos += colorStep;
```

```
            lastStripUpdateTime = millis();
        }
        break;
    case 1:
        if (millis() - lastStripUpdateTime > stripDisplayDelay)
        {
            strip.setLedColor(currentLedIndex, strip.Wheel(colorPos));
            currentLedIndex++;
            if (currentLedIndex == STRIP_LEDS_COUNT)
            {
                currentLedIndex = 0;
                colorPos += colorStep; //
            }
            lastStripUpdateTime = millis();
        }
        break;
    case 2:
        colorPos = colorStep;
        if (millis() - lastStripUpdateTime > stripDisplayDelay)
        {
            static bool ledState = true;
            if (ledState)
            {
                strip.setAllLedsColor(paramters[2], paramters[3], paramters[4]);
            }
            else
            {
                strip.setAllLedsColor(0x00);
            }
            ledState = !ledState;
            lastStripUpdateTime = millis();
        }
        break;
    case 3:
        if (lastColor[0] != paramters[2] || lastColor[1] != paramters[3] || lastColor[2] !=
paramters[4])
        {
            strip.setAllLedsColor(paramters[2], paramters[3], paramters[4]);
            lastColor[0] = paramters[2];
            lastColor[1] = paramters[3];
            lastColor[2] = paramters[4];
        }
        break;
    default:
```

```
            break;
        }
    }
```

What is next?

THANK YOU for participating in this learning experience!

We have reached the end of this Tutorial. If you find errors, omissions or you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us: **support@freenove.com**
We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you want to learn more about Arduino, Raspberry Pi, Smart Cars, Robotics and other interesting products in science and technology, please continue to visit our website. We will continue to launch fun, cost-effective, innovative and exciting products.
http://www.freenove.com/

Thank you again for choosing Freenove products.