Preface

This course was written by XiaoR Geek (www.xiaorgeek.com) to help more users of XiaoR Geek products to understand and use the products in the first place after purchasing them.

In today's 21st century, the Internet is driving the development of this society, and under this Internet, there is an industry that we are not unfamiliar with, and that is the IT industry; if your friend says he is an IT practitioner nowadays, what is the first thing that comes to your mind? It must be: his salary must not be low; even though IT industry is not low, but programming is not as easy as you might think, of course it is not as difficult as you might think ! As we all know: everything is difficult at the beginning. But if you have a persevering heart, you can succeed too!

This course is from zero to one and will give you a comprehensive understanding of the Raspberry Pi, how to get to know it and what Python programming is all about. We hope that you have gained something from this course.

If you are reading this, you must be one of our users. Of course, if you are not one of our customers and happen to be reading this, then I am sure you would like to know more about our products.and if you have any questions about our products, please contact us.

Our email： service@xiaorgeek.net

Our youtube： XiaoR Geek Official

Our shopify： www.xiaorgeek.net

Our Official website： www.xiaorgeek.com

Catalogue

Chapter 1 Getting to know the Raspberry Pi 4B

1.1 The mission of Raspberry Pi

Our mission is to put the power of computing and digital making into the hands of people all over the world. We do this so that more people are able to harness the power of computing and digital technologies for work, to solve problems that matter to them, and to express themselves creatively.

-From the Raspberry Pi website

Since the initial launch of the Raspberry Pi in February 2012, the Raspberry Pi has been a popular choice in the embedded development community as a cost-effective, small form factor programmable microcomputer.

“Small as it is, the sparrow has all the vital organs”. The Raspberry Pi can do all the tasks that an daily PC can do (as long as you don't mind its slowness), but with its low power consumption, mobile portability, GPIO and other features, the Raspberry Pi is perfect for many things that are difficult to do on an ordinary computer.

1.2 Performance advantages of the Raspberry Pi 4B

Compared to common embedded microcontrollers such as the 51 microcontroller and STM32, Raspberry Pi can not only perform the same IO pin control, but also run a corresponding operating system, allowing for more complex task management and scheduling, supporting a wider range of applications.For example, the choice of development language is not just limited to C. Connecting the underlying hardware with the upper layer applications allows for cloud control and cloud management of the Internet of Things, or you can ignore the IO control of the Raspberry Pi and use the Raspberry Pi to build a small web server to do some small test development and services.

Compared to a common computer platform, the Raspberry Pi can provide IO pins that enable direct control of other underlying hardware functions that a common computer cannot do, but at the same time computers can do complex computing (non-embedded) faster than the Raspberry Pi, but in terms of price, the Raspberry Pi has the better advantage.

Compared to the previous generation Raspberry Pi 3B+, the Raspberry Pi 4B offers ground-breaking increases in processor speed, multimedia performance, memory and connectivity, while retaining backwards compatibility and similar power consumption.

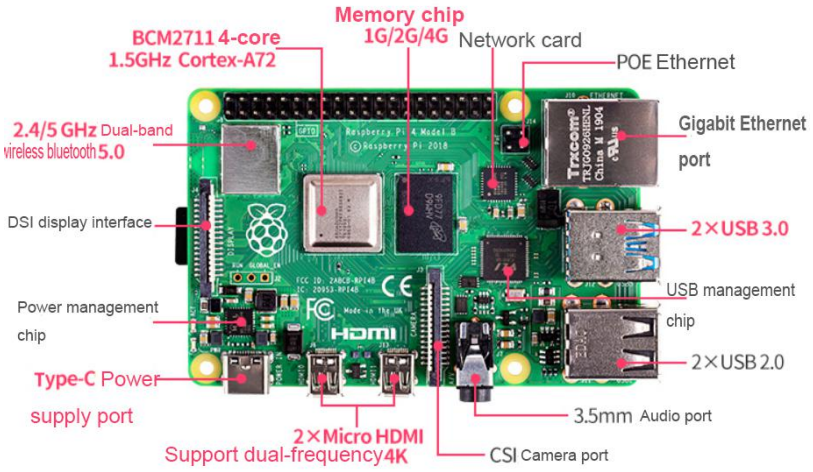
1.3 Raspberry Pi 4B Parameters

Main features of the Raspberry Pi 4B include a high-performance 64-bit quad-core processor, support for resolutions up to 4K dual displays via a pair of micro-HDMI ports， hardware video decoding up to 4Kp60, up to 4GB of RAM, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, USB 3.0 and PoE functionality (via a separate PoE HAT plug-in).

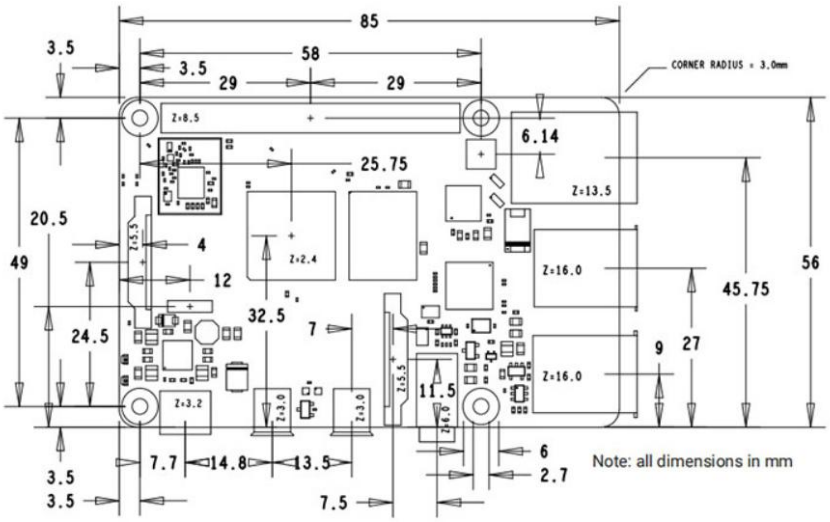
- Broadcom BCM2711, Quad-Core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz

- 1GB, 2GB or 4GB LPDDR4-3200 SDRAM (depending on model)
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports
- Raspberry Pi standard 40-pin GPIO header connector (fully backwards compatible with previous boards)
- 2 micro-HDMI ports (up to 4kp60 supported)
- 2-channel MIPI DSI display port
- 2-channel MIPI CSI camera port
- 4-pin stereo audio and composite video port
- H.265 (4kp60 decoding), H264 (1080p60 decoding, 1080p30 encoding)
- OpenGL ES 3.0 graphics
- Micro-SD card slot for loading the operating system and data storage via USB-C
- Connector provides 5V DC (min. 3A*)
- Provide 5V DC via GPIO connector (min. 3A *)
- Powered over Ethernet (PoE) is enabled (requires a separate PoE HAT) Working temperature: 0 –50 degrees Celsius

1.3.1 Interface distribution



1.3.2 Specification dimensions



Chapter 2 Equipment assembly

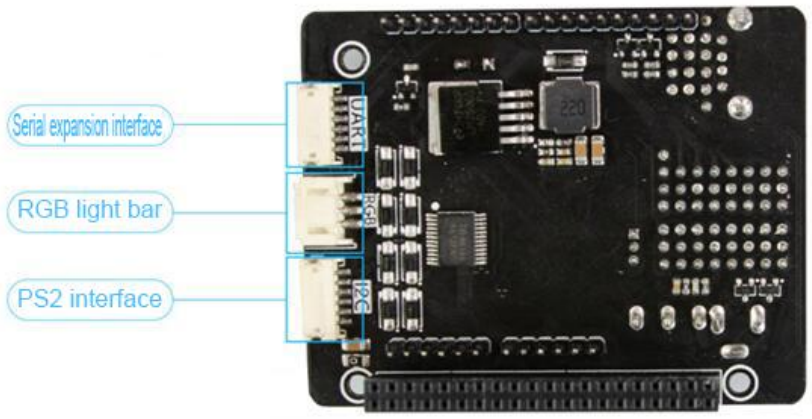
2.1 Raspberry-X Series Introduction

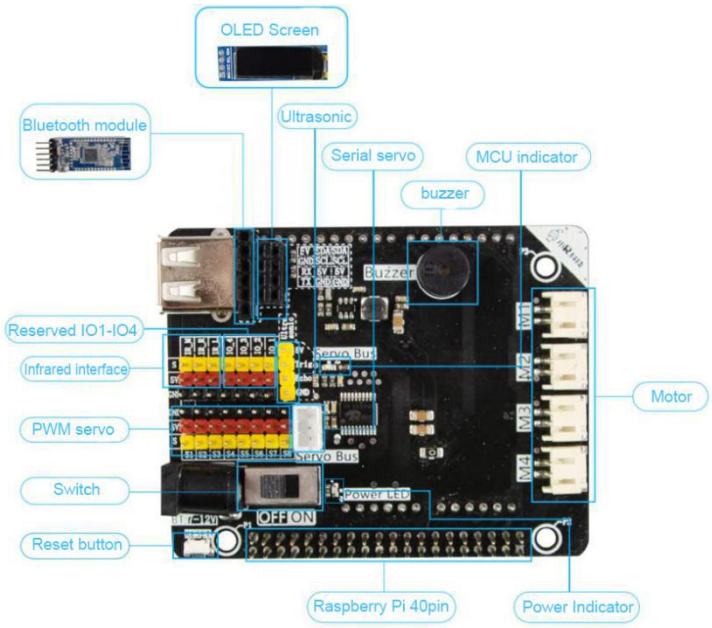
The X series is an upgraded version of the DS/GFS/TH series, although it is an upgraded version, it is also a brand new product; We have upgraded the structure and expanded the functions of the DS/GFS/TH series in order to bring a better experience to the user!

In terms of structure, we have upgraded the original PCB material to an electroplated aluminium alloy, and designed a double-layer structure to make the whole car more simple, more textured and more impact resistant!

In terms of functionality, we always keep in mind the technological innovation and keep up with the times, adding new programmable hardware and Opencv technology applications on the basis of the original DS, so that the car is no longer monotonous, and users are no longer limited in their secondary development ideas.In terms of service, we are taking a re-examination at the past, and we will be providing detailed courses from 0 to 1 to let bignner can really understand and learn programming through our products, and be able to implement all the functions of our carts, and feel the joy from learning and programming!

2.2 X Series hardware interface description





Note： The Bluetooth module is exclusive to the Arduino and STM32 series of cars, the Raspberry Pi already comes with Bluetooth.

2.3 X Series hardware assembly

First of all we strongly recommend that you watch the video to install the whole car, there are installation video and assembly manuals in our accompanying documentation or you can watch the installation video online by visiting our learning website.

If there are missing or broken parts in the installation, we hope you can understand, and at the first time to contact us so that we can quickly arrange a replacement for you, thank you for your understanding!

Chapter 3 Setting up the development environment

3.1 Preparation

The Raspberry Pi development board does not have on-board Flash, so the Raspberry Pi system needs to be installed on a TF (SD) card. By relying on the removable TF card, we can also use multiple TF cards to switch between multiple operating systems quickly and easily, without having to re-flash the system every time we change to another system.

Before starting to burn, we need to download the appropriate software and prepare the hardware to be used, the list of steps is as follows

3.1.1 Software tools

The first thing we need to do when we get the Raspberry Pi is to burn the system to the SD card which is used in the Raspberry Pi. The Raspberry Pi uses the SD card as a 'hard drive' so we have to burn the system to the SD card before it can boot properly. Before we burn the firmware, we need to prepare the following software：

1. SD Card Formatter

SD Card Formatter is a quick formatting tool for SD cards.。

Download link：https://www.sdcard.org/downloads/formatter/index.html

SD Card Formatter

Scan to download

2. Win32DiskImager

Firmware burner (img format firmware), also known as Balenaetcher

Win32DiskImage download link：https://ww.lanzous.com/i9veysb

Win32DiskImager

Scan to download

Balenaetcher download link：https://www.balena.io/etcher/

Balenaetcher

Scan to download

3. Download the firmware package that we have provided, unzip it and set it aside, after unzipping the format suffix is .img.

3.1.1 Hardware tools

For the hardware we need a card reader, a Micro SD card, a PC, a monitor if possible and a Micro HDMI cable as well as a set of keyboard and mouse without a driver.

- SD card and card reader

16g SD card (reliable brand, SD card with capacity greater than 16G)

SD card and card reader

3.2 Burning firmware

1. First we format the memory card using SD Card Formatter as follows：

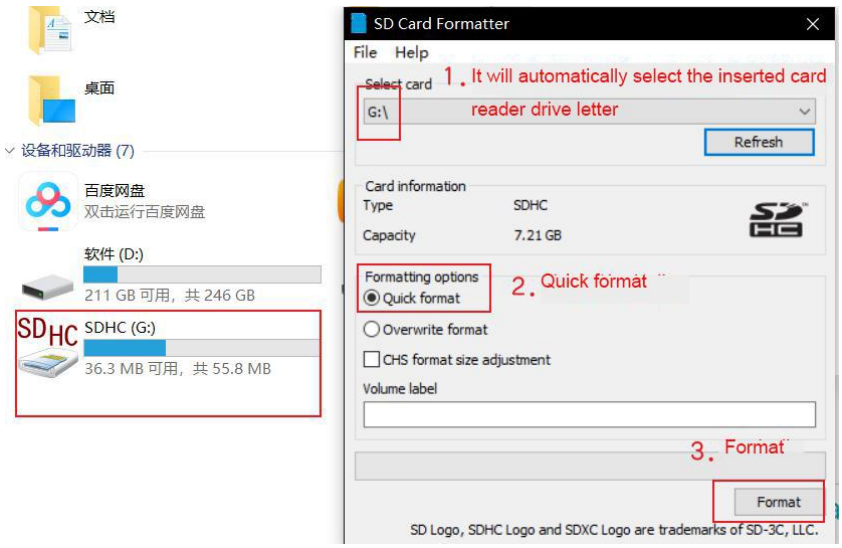
Insert the SD card into the card reader, which is plugged into the USB port of your PC, and open the SD Card Formatter software. SD Card Formatter will only read the inserted SD card and will not read your hard disk partition.

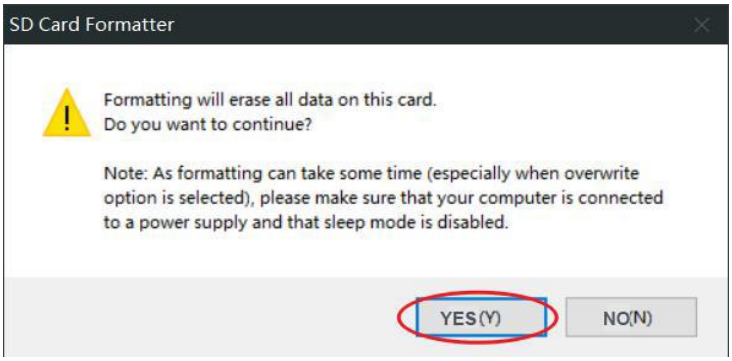
- SD card inserted into the card reader

- Card reader plugged into the USB port of the computer

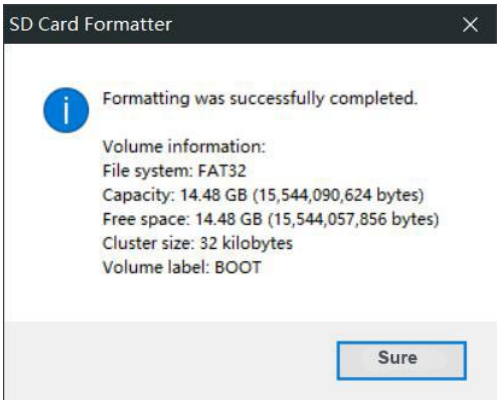
- Open the SD Card Formatter and the inserted SD card drive will be automatically selected, click on Format directly and you will be prompted if you are sure to format the SD card.

Click "Yes" .



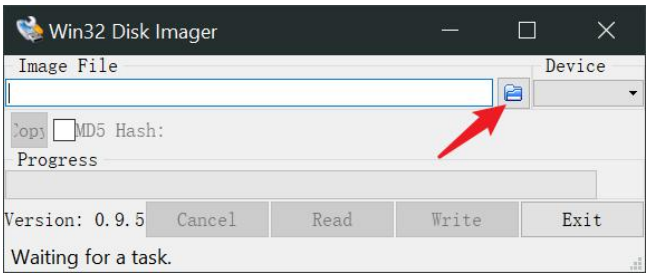


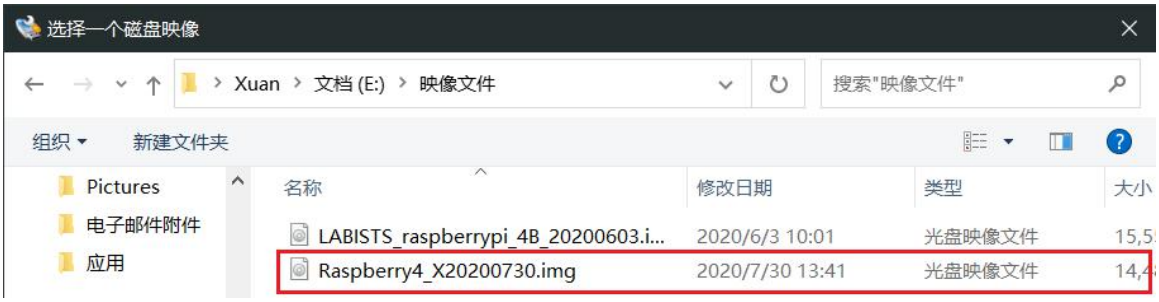
Wait a few moments and the formatting success window will pop up, as shown below



2. Using Win32DiskImager to burn the firmware

Double click to open the installed Win32DiskImager, open the window as shown below, we select the correct device disk letter and click on the blue folder button, it will open the resource manager for us to select an image file, here we need to select the .img image file that we downloaded and decompressed in advance. After selecting the image file, click on "Write" and wait patiently for the writing to complete, a write success pop-up will appear after the writing is complete.



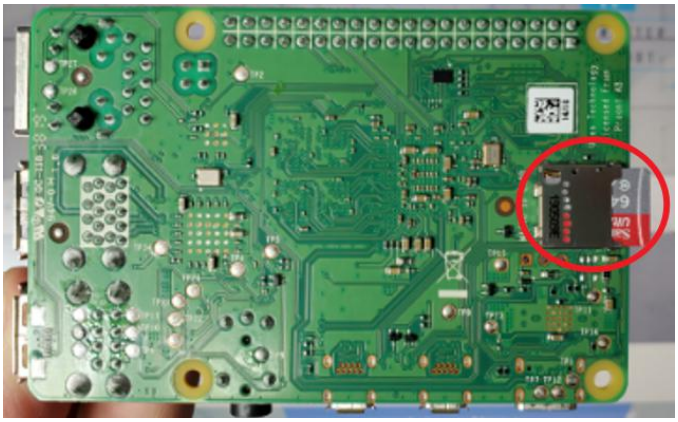


3.3 Start the Raspberry Pi

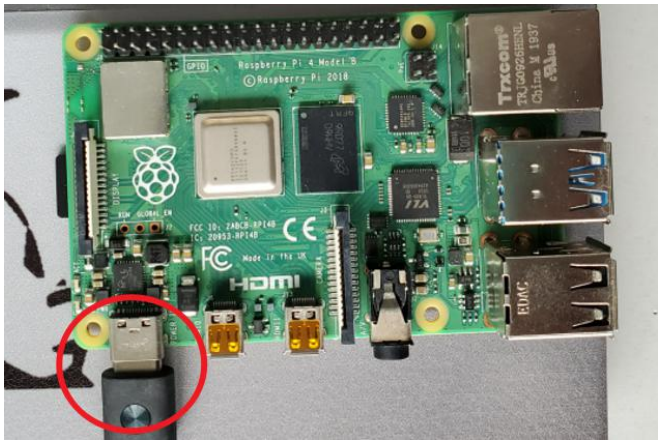
After we have burned the system is to start our Raspberry Pi, plug the Micro SD card back into the Raspberry Pi, here we use the computer USB3.0 interface power supply; we recommend users use the Raspberry Pi dedicated power adapter (rechargeable battery can also be), voltage of 5V, current recommended at least 2A (When used as a single board, the DSX is equipped with a power supply board and the lithium battery current is 6A, so please feel free to use it.） The more peripherals in the Raspberry Pi, the more current may be used, which may cause problems such as AP disconnection and reboot if the power supply is not increased.

Insert the SD card with the burned system into the SD card on the back of the Raspberry Pi.

- Insert the SD card with the system burned into the SD card slot on the back of the Raspberry Pi



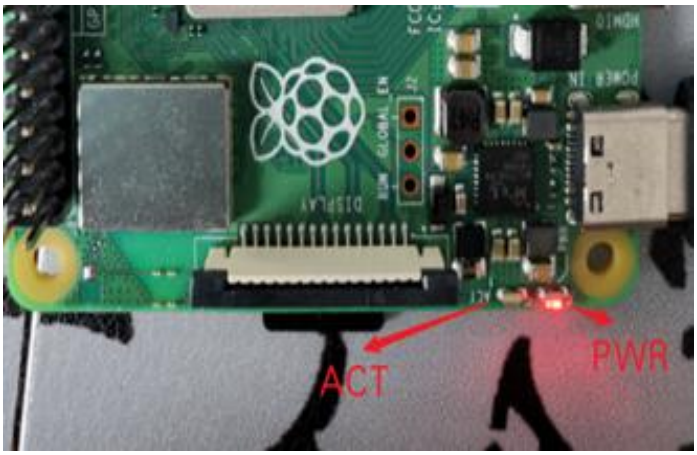
- The Raspberry Pi can be started by supplying power from the type-c connector; at this point it is powered when using the Raspberry Pi motherboard alone, when using the X-Series cars, the Raspberry Pi is powered by the driver board for backwards compatibility and there is no need to power the Raspberry Pi separately.



The Raspberry Pi will automatically power on after power is supplied, there are two indicators on the Raspberry Pi, the PWR indicator and the ACT indicator, the ACT indicator will flash green, slowly

and quickly, but as long as there is a flashing green, it means that the SD card is active (similar to the hard disk light on a PC), if you find that the ACT light is not flashing and the Raspberry Pi does not power on properly, you need to consider whether the system burned on the SD card is damaged. If you are sure that your SD card system is not corrupted but does not boot up, please contact us in the first instance for an after-sales service.

- PWR/ACT indication; PWR indicator stays on red when plugged in.



If you have a monitor with HDMI interface , then connecting the Raspberry Pi and the monitor together via the Micro HDMI cable, we are able to see the boot process of the Raspberry Pi as well as the desktop, and by plugging a drive-free keyboard and mouse into the Raspberry Pi USB port, this is a computer (with limited performance).

3.4 Raspberry Pi login

This section will teach you how to log into your Raspberry Pi by using wired and wireless (WiFi) methods, if you don't have an extra monitor, you may try to follow with the chapter tutorials.

3.4.1 Remote login username and password

User name： pi

Password： raspberry

All characters are in lower case, after logging in by Remote Desktop or Putty (not limited to these two login methods), you will need to enter the above account and password first, just like we need to enter the password when booting up a Windows operating system.

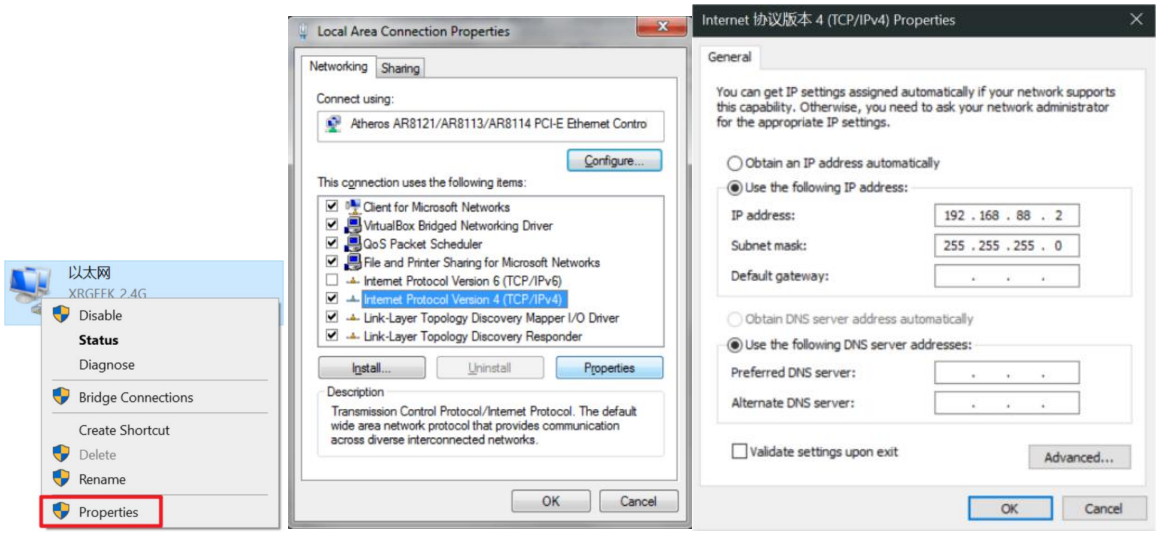
3.4.1 Logging in to the IP address

There are various login methods for Raspberry Pi, from graphical interface login to terminal login, the choice of login method varies from person to person, after choosing the login method we need to determine the IP address first.

Wireless login: If you choose to flip in our firmware, the Raspberry Pi will boot up an AP hotspot (similar to a router) and our PC will connect to this hotspot, then the login IP address will be: 192.168.1.1

Wired login: If you choose to flip in our firmware, and you may use a wired network login, then we need to change the IP address of the wired network IPV4 to 192.168.88.2, after the change the login IP address is: 192.168.88.100

A wired network login requires a network cable with one end connected to the Raspberry Pi network port and the other end connected to the computer network port.



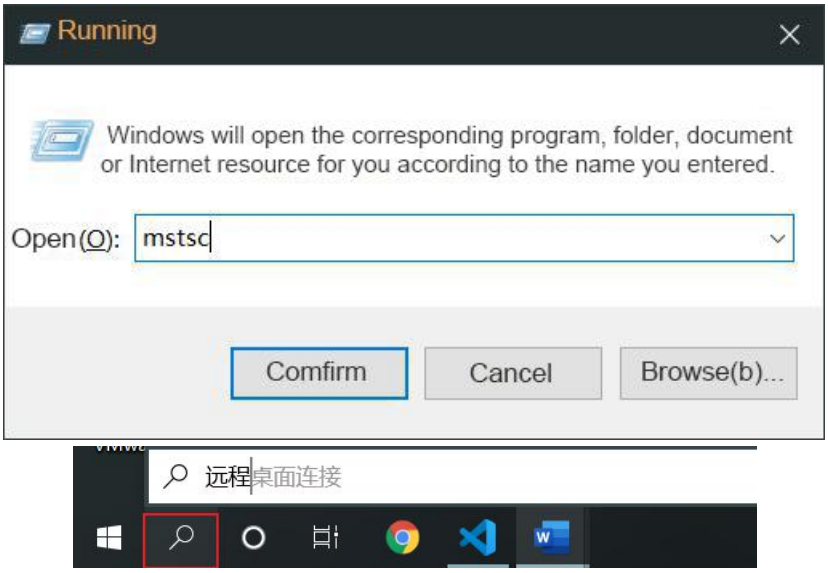
Remember to click OK after changing the IPV4 IP address, otherwise the settings will not take effect.

3.4.3 Windows comes with remote Desktop Login

Wired IP login

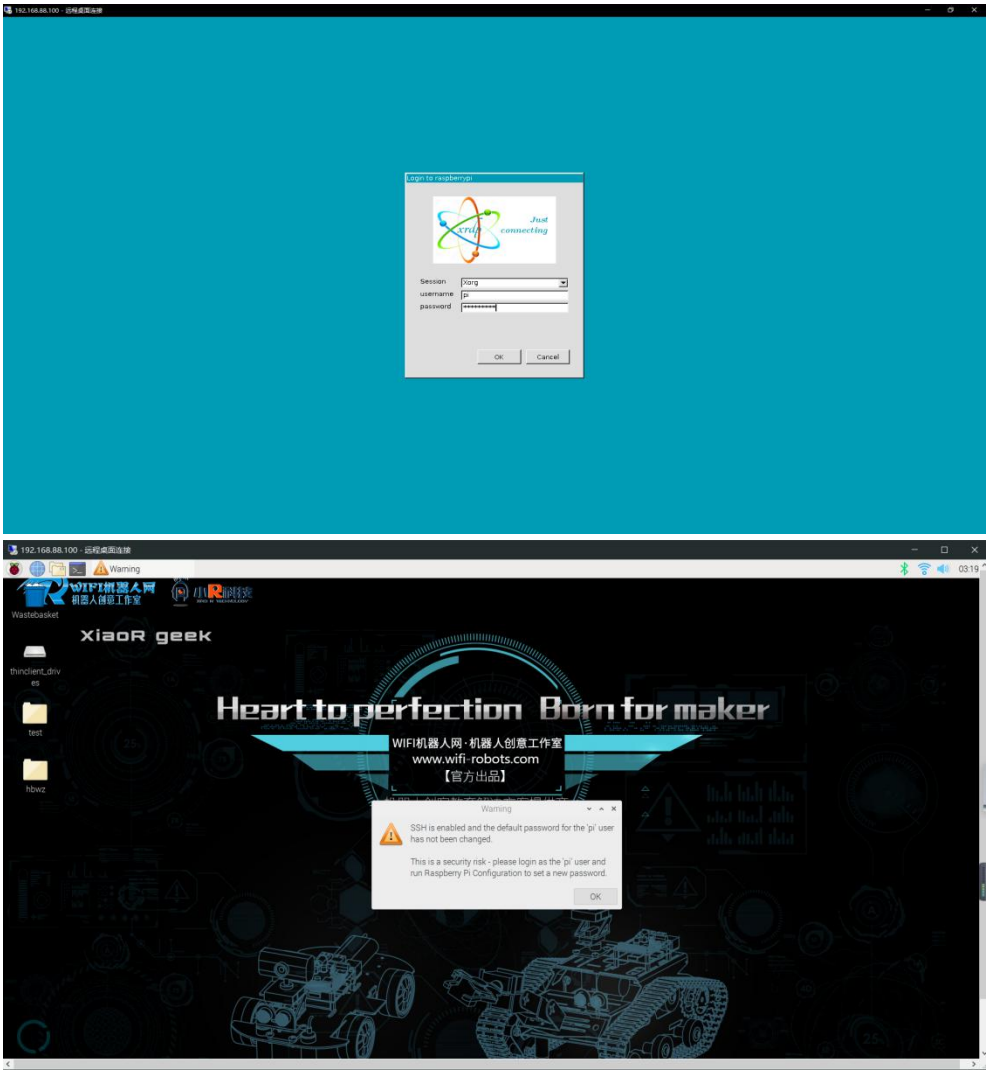
Press Win+R to wake up the Run screen and type mstsc or search for the keyword remote to start the Remote Desktop Connection function, which is not available on some systems due to system limitations, such as Windows 10 Home Edition or some lite versions of Windows systems; I am using Windows 10 Professional Edition.

Search for the remote keyword, it will automatically match "Remote Desktop Connection", type Enter to open this function.



After opening the remote desktop, enter the wired IP address 192.168.88.100 and click "Connect" to start the remote desktop.

If a security warning pops up, we can click "Yes", then a sky blue remote window will open and we will be prompted to enter our username and password, type in the username pi password raspberry to successfully log in to the Raspberry Pi desktop, as shown below.



If you do not log in successfully, please check the password that you typed is correct; secondly, when you enter the desktop, the pop-up warning above is telling you that SSH is enabled, but the default username and password have not been changed; just a security risk warning, no need to panic.

Wireless IP Login

The same process, but for wireless IP login, we need to wait for the Raspberry Pi to start up and connect to the Raspberry Pi hotspot name, the hotspot name starts with "XiaoRGEEK-PIX", when connecting to this hotspot, we just need to change the login IP address to 192.168.1.1 (wireless login does not need to change the IPV4 address). IPV4 address).

Here you need to connect to the AP hotspot emitted by the Raspberry Pi first, and then open a remote desktop connection. The hotspot emitted by the Raspberry Pi is not networked, and we connect also by using the LAN, so the absence of network is not affected.

Why you need to be taught to use two ways to remote

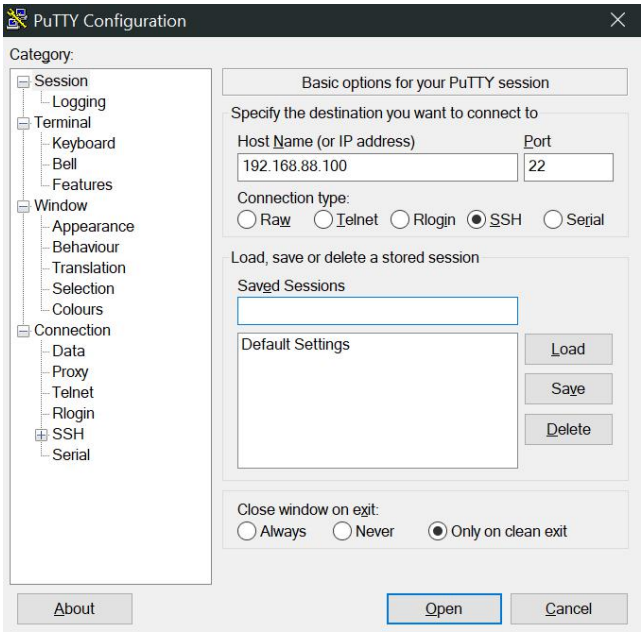
If your Raspberry Pi needs to be connected to the network (e.g. updating sources, downloading packages, etc.) then we need to use a wired IP to use the remote, and then the Raspberry Pi's WiFi function to connect to a WiFi hotspot that has a network by using client mode. Of course we can also use the wired network to connect to the router to surf the internet and the wireless network to remote.

As well we can log in via wireless IP only (WiFi), this requires our router to support viewing the IP address of the connected device, if your router is able to view the allocated address of the access device, then knowing this allocated address and entering the IP address assigned to the Raspberry Pi by your router when remote desktop will also allow you to remote to the Raspberry Pi via wireless, but only if your computer device is on the same LAN as the Raspberry Pi.

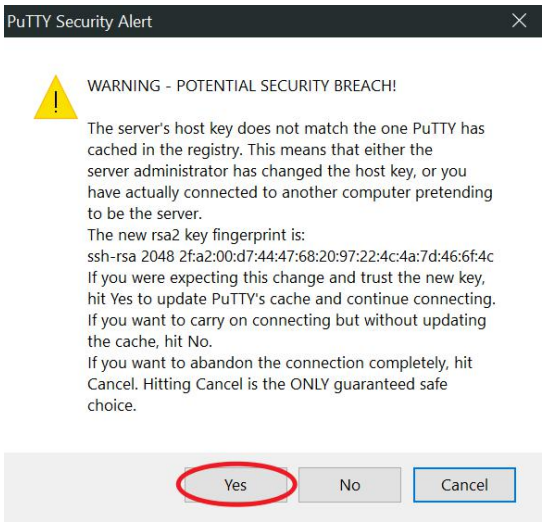
3.4.2 Putty terminal login

Putty is a Telnet, SSH, rlogin, pure TCP and serial interface connection software, here we are also used to connect to the Raspberry Pi remotely, but Putty can only use terminal commands to control the Raspberry Pi after connecting to the Raspberry Pi, no graphical interface.

The procedure is to open Putty and enter the wired IP address: 192.168.88.100 if you are using a wired remote, or the wireless IP address: 192.168.1.1 if you are using a wireless remote, port 22 remains unchanged and the connection type is SSH, as shown in the figure below.

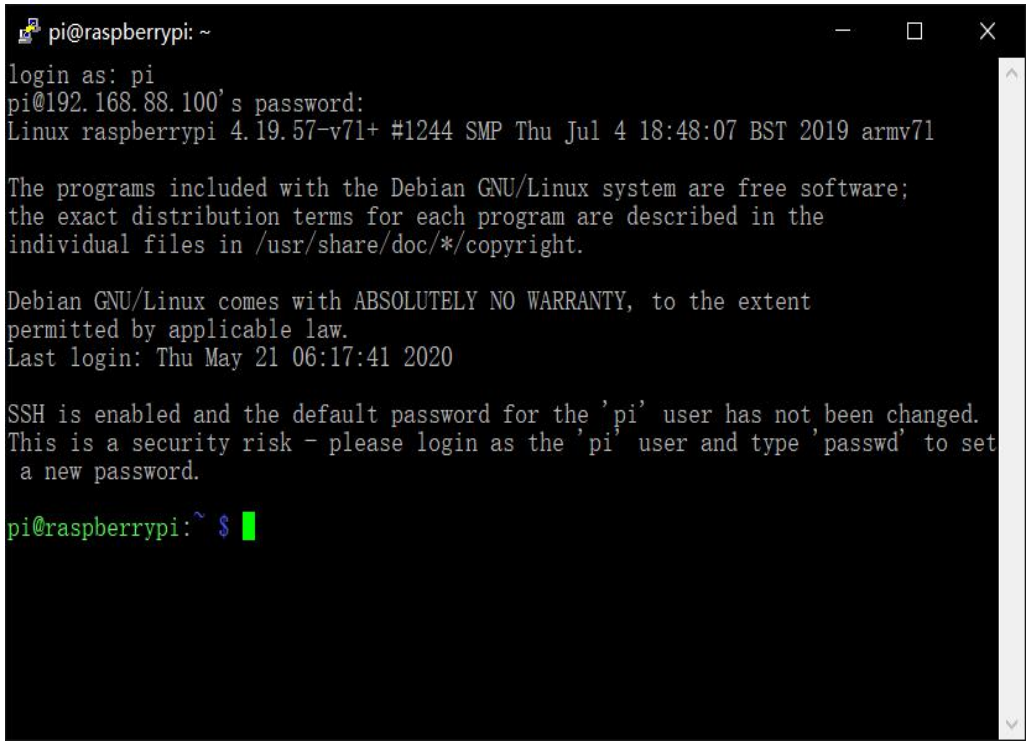


Once you have clicked open, if you are prompted with the following message, we can either click "Yes" or "No", or click cancel to end the connection.



Then in the next screen type in the username: pi password: raspberry to log in successfully, here we need to pay attention to: we can't see when we type in the password, so if there is a password error (Access denied) please type in the password again, the password account name and password are lowercase.

The successful Putty login screen is shown below.



There are many other remote software, such as VNC, xshell, etc., which I won't go into here, but if you're interested, you can search for tutorials on the internet.

3.5 Raspberry Pi and Windows transfer methods

In practice, due to the performance limitations of the Raspberry Pi, we mostly choose to do the project on the PC and verify it on the Raspberry Pi. Therefore, in practice, we write the code on the PC and transfer it to the Raspberry Pi for testing, so there is a file transfer problem. Here, we recommend Winscp, a software program whose main function is to securely copy files between the local and remote computers.

Winscp download link：https://winscp.net/eng/download.php

Installation：

Process: Receive the license agreement → Install type (typical installation is recommended) → Select user interface style → Install

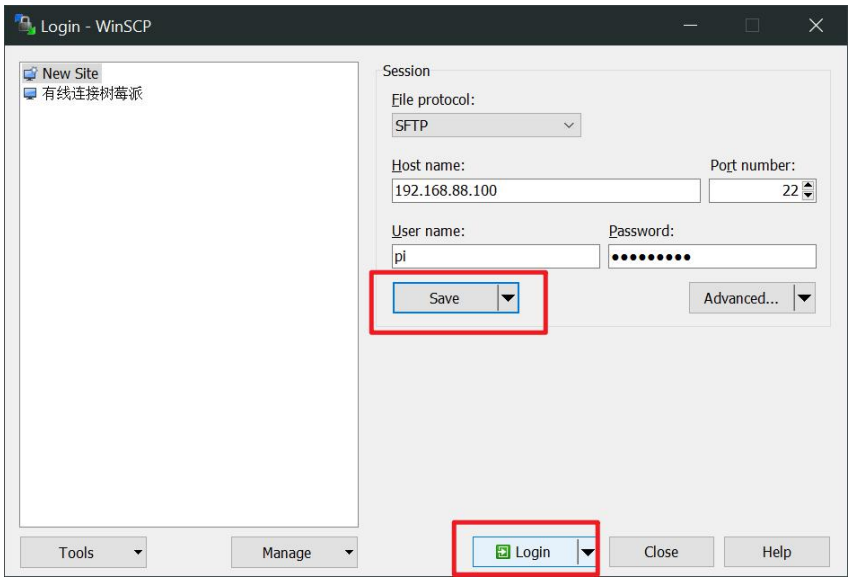
Typical installation defaults to the C drive location, if you do not want to install the C drive please select custom installation; during the installation process we only need to click on "Next" until the final "Install" to install Winscp smoothly, due to the large size of the picture and the simple installation process, here is not a screenshot to demonstrate.

Usage：

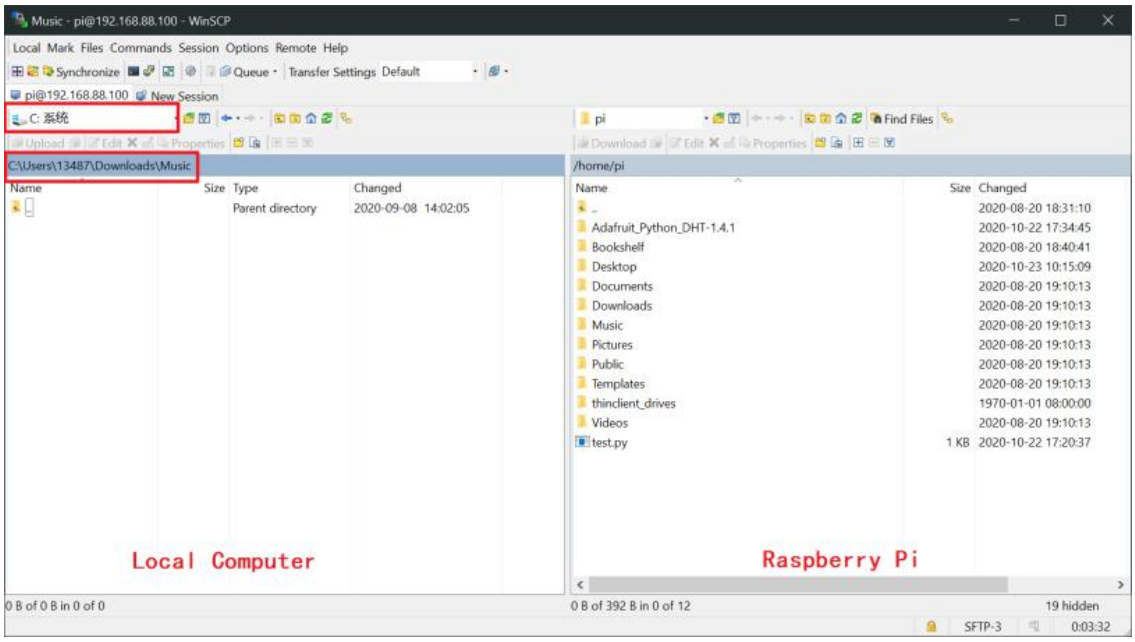
Open the Winscp which has just been successfully installed, enter the IP address, port unchanged, username pi and password raspberry; click on the green login button at the bottom after you have finished entering, if nothing else, you will be able to enter the login screen successfully.If

you fail to log in, please check the error message to see if you have entered your username and password incorrectly, as well as your IP address and port number.

- Login screen, the picture shows the IP address of the Raspberry Pi AP hotspot, please enter the correct IP address according to your actual situation.



- The screen after successful login



To transfer a file to the Raspberry Pi we just need to select the file to be copied and drag it to the right with the left mouse button

3.6 System backup

After we have used the Raspberry Pi for a while, we have done some special operations, such as reconfiguring the rc.local configuration file, but in case some of our subsequent operations may lead to some special circumstances such as system corruption, we can choose to do a system backup after we have done some major operations on the Raspberry Pi and experimented successfully.

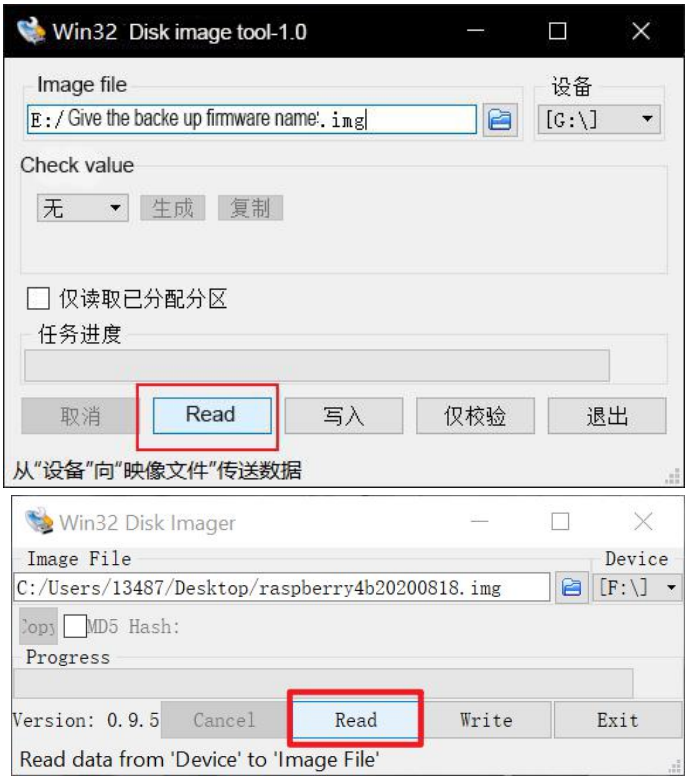
The system backup is also done by using Win32DiskImager to read out the existing image from the memory card.

Usage procedure：

● Create an empty file in .img format in any allowed folder on the computer.;

● The system that needs to be backed up (memory card) is inserted into a card reader, which is plugged into the USB port of the computer.;

●Open Win32DiskImager and select the empty file that was created in the first step, then click on Read.;

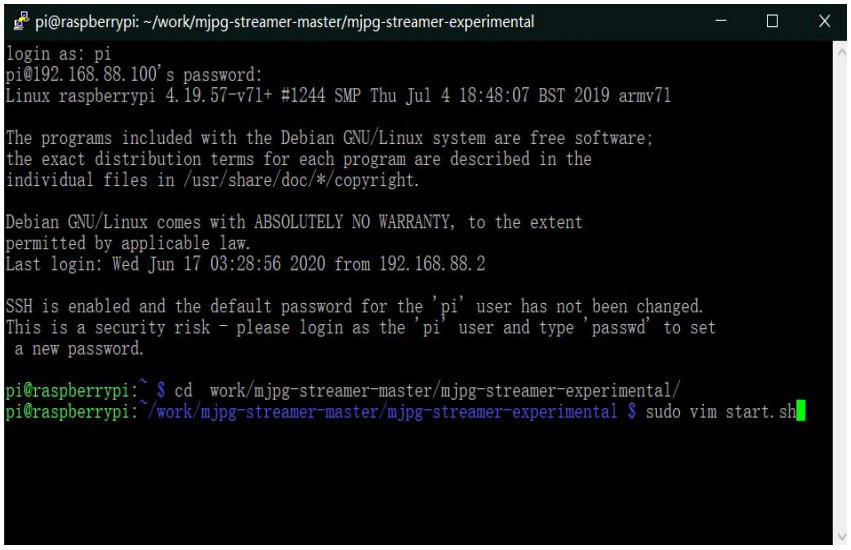


3.7 Multi-channel video configuration

The Raspberry series of robot cars support multi-channel video configurations and we only need to modify the mjpegstreamer startup script to achieve this.

1. First we need to remotely access the desktop (putty is fine) and enter the following command at the command line.

```
1. cd work/mjpg-streamer-master/mjpg-streamer-experimental/
2. o vim start.sh
```



2.Open the start.sh configuration file with sudo privileges, add the camera serial numbers in order and save (the default address port numbers in the host settings are 8080-8083):

video0： 8080

video1： 8081

video2： 8082

video3： 8083

The complete input is as follows:

```
./mjpeg_streamer -i "./input_uvc.so -d /dev/video3"-o"./output_http.so -p 8083 -
w ./www"&
```

3.8 Raspberry Pi WiFi Configuration AP/Client Mode

The system used in the Raspberry-X series of carts is based on the official Raspberry system, which integrates the mjpg-streamer push-streaming service and configures the Raspberry Pi's wireless card to AP hotspot mode by default. However, in the process of use, we may upgrade the Python package version or install new package files and other needs, in short, we need to set the WiFi to Client mode; AP mode is like our home router, which emits WiFi hotspots, and Client mode is like our mobile phone, which is to connect to WiFi hotspots.

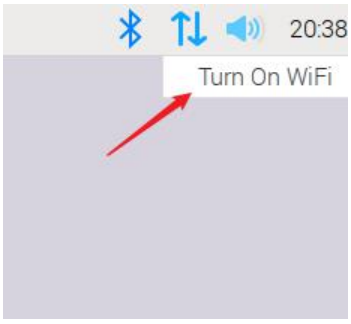
Here we will learn how to switch the Raspberry Pi wireless card back and forth between AP mode and Client mode.

By default you have already watched the Raspberry Pi login tutorial in chapter 3.4 and by default you have learn how to log in remotely, if you do not know how to log in remotely we suggest you watch the tutorial in chapter 3.4 first; of course you can also use the monitor and keyboard and mouse to do this.

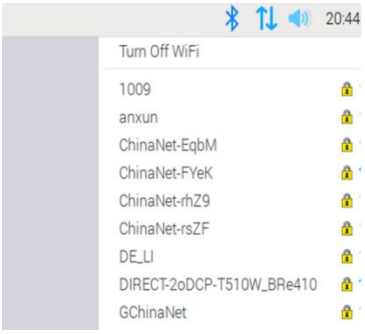
3.8.1 Enabling Client Mode

The XiaoR Geek factory firmware wireless card defaults to AP mode, the AP mode wireless card logo is the data arrow, it is blue as I am using a wired network remote desktop here, if you are using a direct monitor connection, the logo should be grey here.

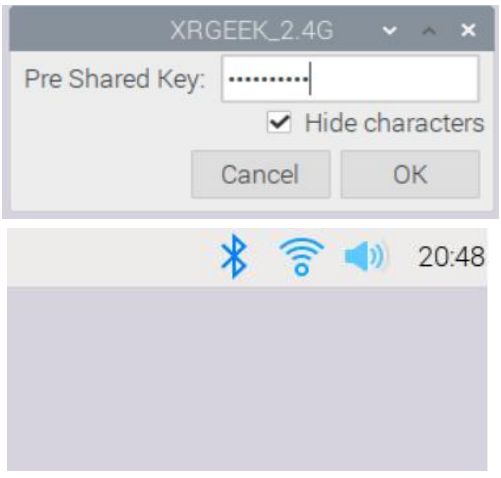
At this point the Raspberry Pi is in AP mode, we can search for the Raspberry Pi's WiFi hotspot, we click on the left mouse button on this blue data symbol, the option: Turn On WIFI will appear, click Turn On WIFI again and we have enabled Client mode.



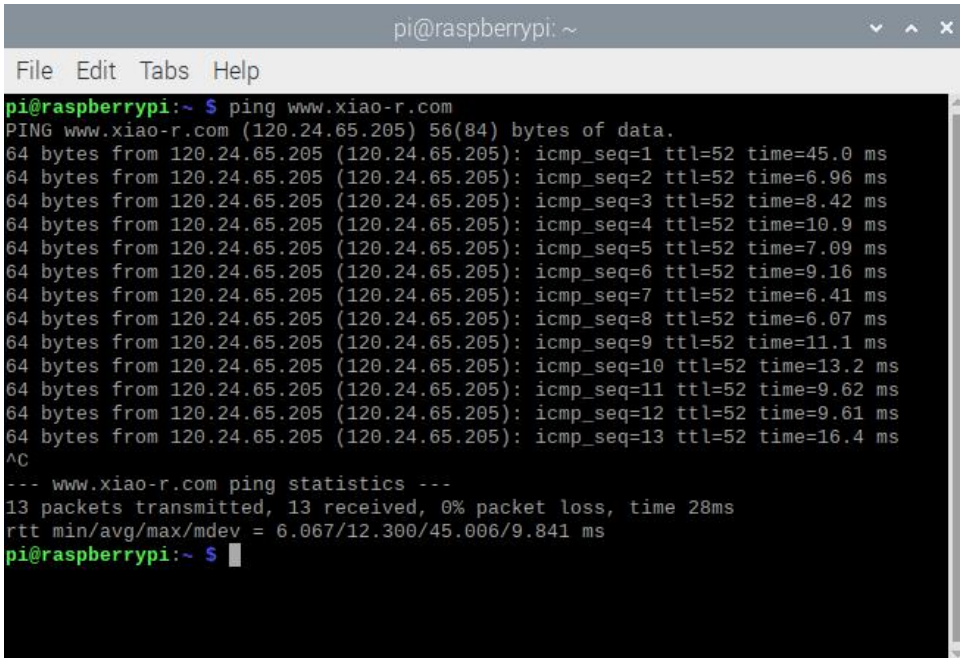
After a short wait of 5-8 seconds, left mouse click on the data flag and we can see the available WiFi connections nearby.



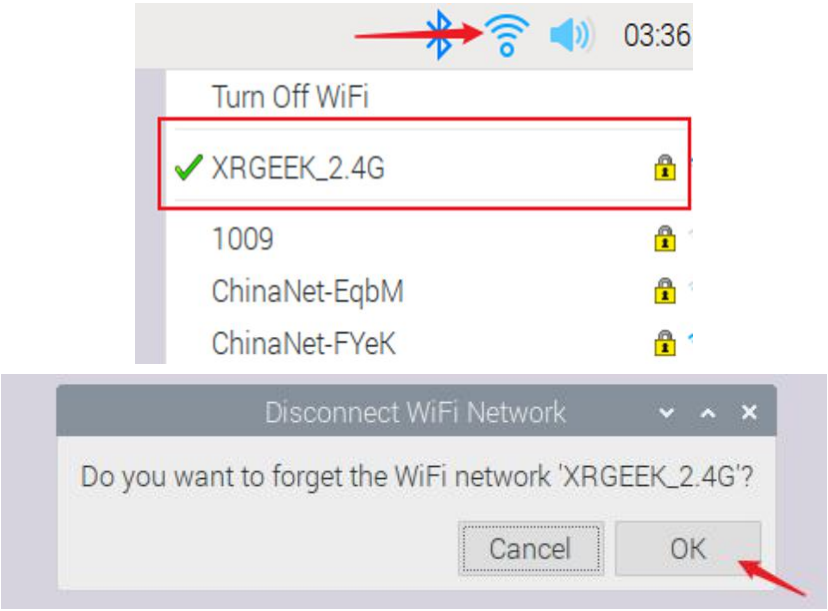
Left click to select the WiFi hotspot that you need to connect to, enter the password and we will see the data flag bit change to the WiFi flag, this means you are connected to this hotspot and the AP hotspot signal on the Raspberry Pi itself will disappear because you have enabled Client mode.



After connecting to our home route and testing the internet, it is absolutely fine.



If you want to revert to AP mode, just disconnect the hotspot and reboot the Raspberry Pi. We just need to left click the WiFi logo on the top right corner of the desktop to disconnect the hotspot, then left click the connected hotspot, the pop-up box we select OK.



3.8.2 Setting up permanent Client mode

There are also users who may need to connect their Raspberry Pi to a home route, set up port forwarding and use it remotely for a long period of time. However, after we may reboot the Raspberry Pi once, the Raspberry Pi does not connect to the same SSID properly or cannot access the

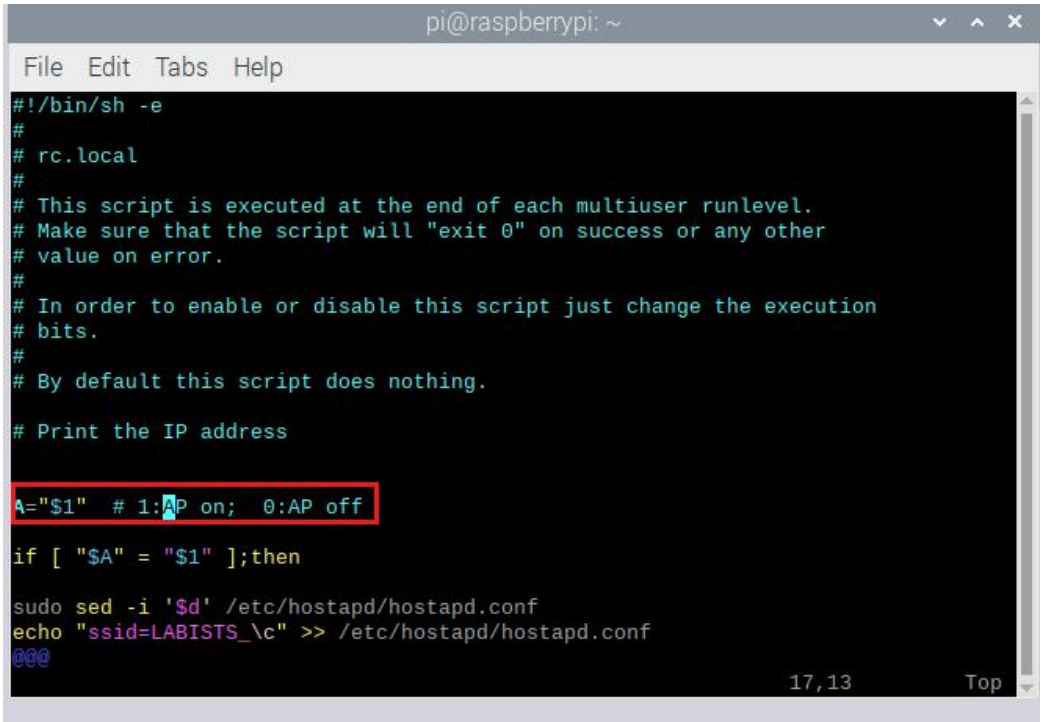
internet properly when connected; this is because our firmware will start hostapd at boot to configure the wireless network card.

So we need to modify the re.local file to comment out the hostapd configuration on boot.

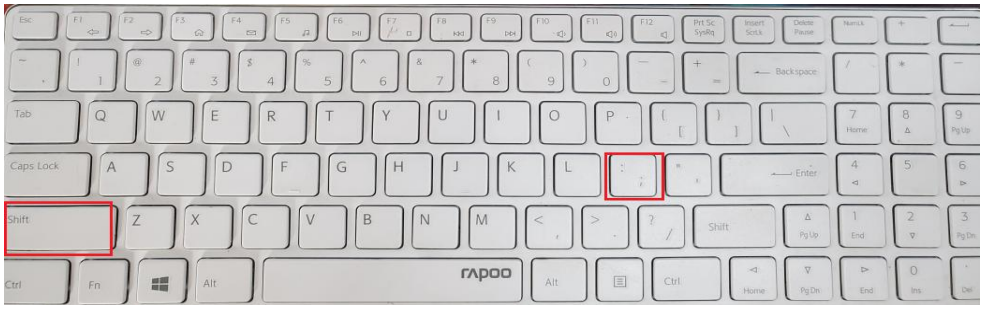
1. Start the command terminal and enter the following command：

```
sudo vim /etc/rc.local
```

2.Change 1 to 2 and save to exist.



Vim command: first move the cursor to the number '1', press the letter i key, you can see the bottom left corner of the window becomes INSERT means that now you can edit the inserted characters, and pressing Shift + : key after modified , you can see the bottom left corner of the window appears ':', then type wq, it will automatically save to exit.



If you want to revert to AP mode, do the same way and change 2 to 1.

3.9 Writing Hello World on RPi

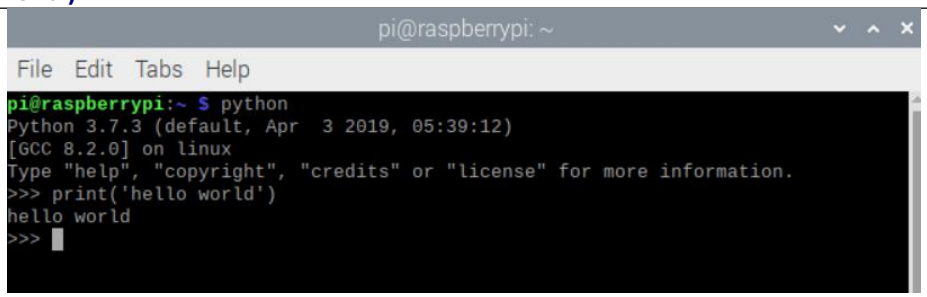
The classic hello word is a necessary part of learning a language, so let's try printing out hello world in Raspberry Pi using Python.Next we'll walk you through the HelloWorld project with detailed steps to get started with the joy of programming in Python.

3.9.1 Terminal writing

When we open the terminal command line and type python, the XiaoR Geek factory firmware default python command enables python 3.7.3; we type the following code and type enter to see the following result：

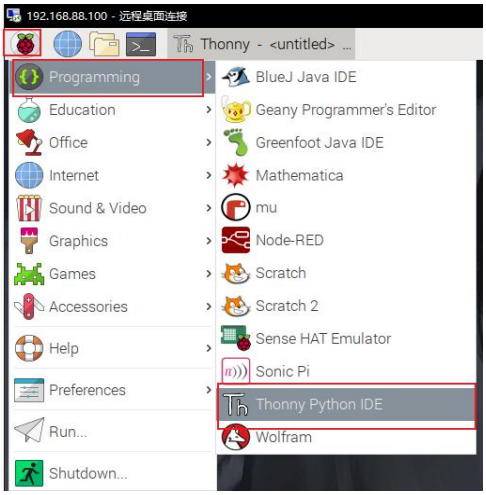
：

```
print('hello world')
```



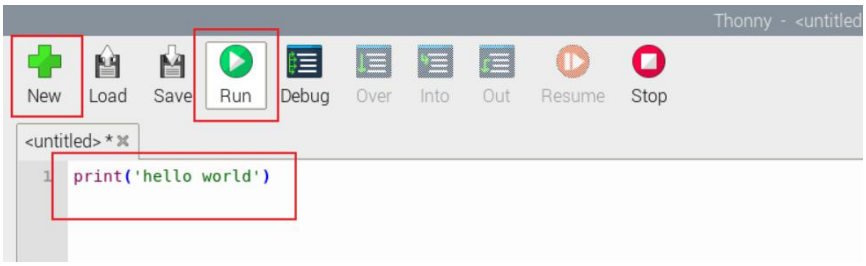
3.9.2 Thonny Python IDE

Thonny is a multi-platform (windows, Mac, Linux) python IDE which is based on the python built-in graphics library tkinter, and supports syntax coloring, code auto-completion, debug and more. This IDE is a Python IDE that comes with the Raspberry Pi firmware.

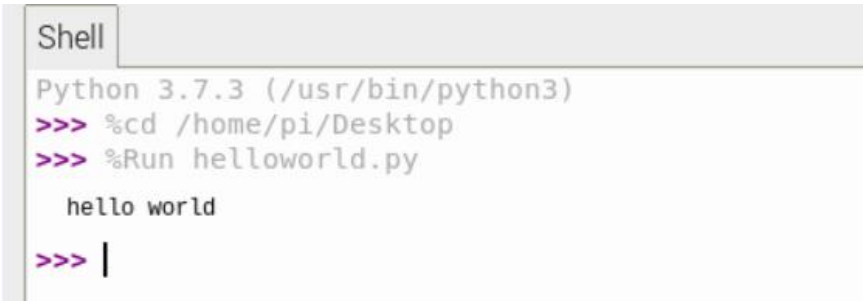


Open the IDE and do the following in sequence

NEW→Type the
code→RUN

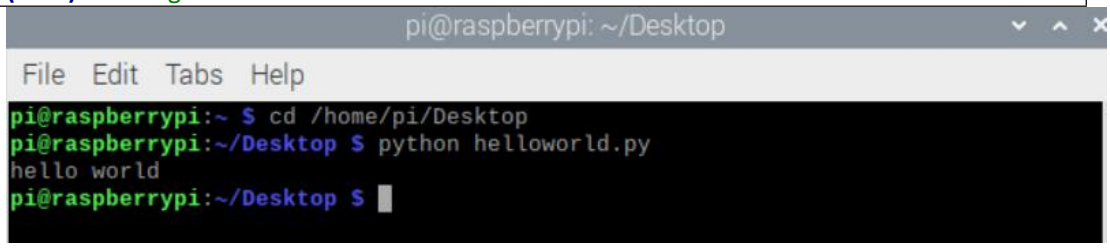


We will be prompted to save the file first, and once we have saved the file the results of the run will be displayed



Of course, when the amount of code is large enough we can edit it first in an IDE on the PC (e.g. pycharm) and then use winscp to transfer the code file to the Raspberry Pi to run it, double-clicking on the .py file on the Raspberry Pi will open the code by default using the Thonny IDE and run it directly; at the same time we can also use the command line in the terminal to use the use the cd command to switch to the directory where the code is located and use the python command to run the code.

```
x = 3          # Define variables x
print(x * 3)   # Using variables x
```



Hello World is just a very, very simple project and later on, we will take you step by step through how GPIO programming is done on the Raspberry Pi and how the whole Raspberry-X project is built.

Chapter 4 Basic Syntax of Python

This section provides a brief introduction to the basic syntax of Python. If you want to learn more about Python's syntax and features, check it out online, watch a video, or buy a book.

A beginner's coursePython3.x: https://www.runoob.com/python3/python3-tutorial.html

4.1 Variable

Variables are easy to understand in Python and represent (or point to) the name of a specific value. For example, you might use the name x to represent 2, which is consistent with the equation variable for Middle School Algebra; for example, for the equation y = x * X, x is the variable. When X = 2, the result is 4; when x = 5, the result is 25.

The difference is that in Python, a variable can be not only a number, but also any data type. Define variables as integers and strings in code as follows:

```
x = 3          # Integral type
y = '12345'    # String
```

This is called an assignment, and we assign the value 2 to the variable X. In other words, you associate the variable x with the value 2, and after you assign a value to the variable, you can use it in an expression.

The output is 9; Unlike in other languages, you must assign a value to a Python variable before using it, because there is no default value for a variable in Python.

Note: Variable Names (identifiers) in Python can be made up of letters, numbers, and underscores , and can not begin with numbers; number1 is the legal variable name and 1number is not.Identifiers are case sensitive and can not be named using keywords.

View keyword method:

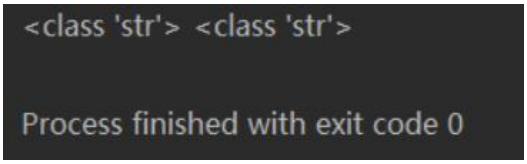
```
import keyword
  print(keyword.kwlist)   # Run these  code to see the keywords
```

4.2 String

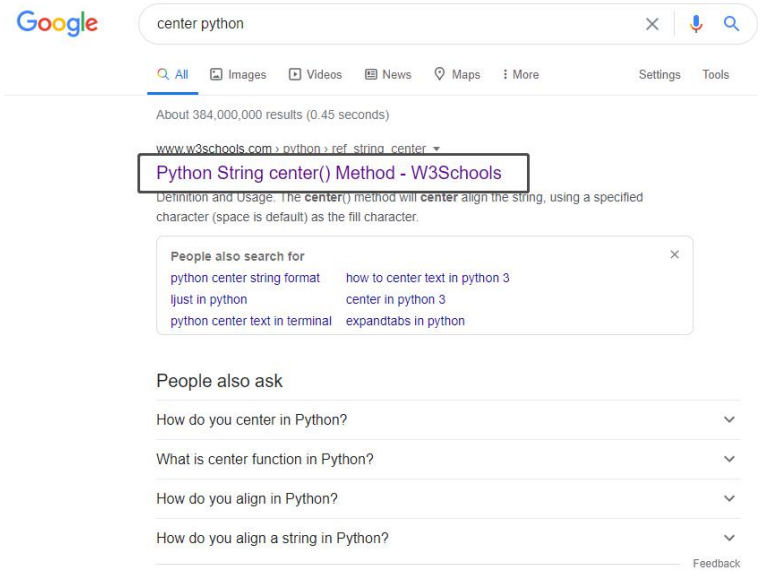
Strings are common data type in Python. We can use quotes'or'to create a string. Creating a string is as simple as assigning a value to a variable. For example:

```
# Single quotation mark
var1 = 'XiaoRGeek'
# Double quotation marks
var2 = "123456"
# Null string
var3 = ''
# If you want to see the variable type, use the type () built-in function
  print(type(var1),type(var2))
```

We can see that both output expenditures var1 and var2 are str types:



There are many ways to use strings, listed in the center, find, join, lower, replace, split, strip, translate, and so on, we do not discuss the use of methods here, you can refer to relevant books or Google search.



4.3 List

A list consists of a series of elements arranged in a particular order. You can create a list that contains all the letters of the alphabet, the numbers 0-9, or the names of all family members; you can add anything to the list, and the elements can be unrelated.

In Python, a list is represented by '[] ' and its elements are separated by ', ' . Here's a simple example of a list that includes some graphics card brand names:

```
# Create a list
display_cards = ['ASUS', 'MSI', 'iGAME', 'ZOTAC', 'AORUS']
# Create an empty list and use append to add elements
lists = []
# Print list
print(display_cards)
```

```
['ASUS', 'MSI', 'iGAME', 'ZOTAC', 'AORUS']

Process finished with exit code 0
```

4.3.1 Access list elements

The list is an ordered collection, so to access the values in the list, we just need to give Python the index of the corresponding value, and if we want to access 'iGAME' in display, we know that the index in Python starts at zero, then we know that 'iGAME' has an index value of 2:

```
# Prints a value with an index of 2 in the display list
print(display_cards[2])
```

The output is:

```
iGAME

Process finished with exit code 0
```

4.3.2 Modifying, adding, and deleting elements

Modifying

Modifying the list is easy, just use the normal assignment, but instead of using an assignment statement like x = 2, use index notation to assign values to elements at a particular location, such as x [1] = 2.

```
# Creat a list
display_cards = ['ASUS', 'MSI', 'iGAME', 'ZOTAC', 'AORUS']
# Print list
print(display_cards)
# Modify the element with index 3
display_cards[3] = 'GALAX'
# Print the modified list
print(display_cards)
```

Code output:

```
['ASUS', 'MSI', 'iGAME', 'ZOTAC', 'AORUS']
['ASUS', 'MSI', 'iGAME', 'GALAX', 'AORUS']

Process finished with exit code 0
```

You can see that 'Zotac' with an index of 3 in the list has been changed to 'GALAX' .

You can see that 'Zotac' with an index of 3 in the list has been changed to 'GALAX' .

Adding

To add an element, you can use the append () built-in function to add an element at the end. Append () is used as follows:

```
# Creat a list
display_cards = ['ASUS', 'MSI', 'iGAME', 'ZOTAC', 'AORUS']
# Use the append function to add an element to the end of the list 使
display_cards.append('GALAX')
# Print list
print(display_cards)
```

Code output:

```
['ASUS', 'MSI', 'iGAME', 'ZOTAC', 'AORUS', 'GALAX']

Process finished with exit code 0
```

You can see that the'GALAX'element has been added to the end of the list. I'm sure some of you are wondering, if I wanted to add an element anywhere in the list , how to do? We can use the

insert () function, which takes two parameters, the first is the insertion position (index value) and the second is the insertion value.

Here's how to use it:

```
# Creat a list
display_cards = ['ASUS', 'MSI', 'iGAME', 'ZOTAC', 'AORUS']
# Insert the GALAX element using the insert function with the Insert Index at 1
display_cards.insert(1, 'GALAX')
# Print list
print(display_cards)
```

Code output:

```
['ASUS', 'GALAX', 'MSI', 'iGAME', 'ZOTAC', 'AORUS']

Process finished with exit code 0
```

Delete

Delete the specified element in the list, we use the del Keyword, also need to use the specified index value, let's see how to use it:

Code output:

```
['ASUS', 'MSI', 'iGAME', 'ZOTAC', 'AORUS']

Process finished with exit code 0
```

GALAX has completely disappeared, and the list has gone from 6 to 5. The DEL statement can be used to delete something other than a list element. You can use it for dictionaries and even variables.

Slice assignment

Slicing is an extremely powerful feature, and the ability to assign values to slices makes it even more powerful.

```
# The list built-in function converts other sequences to lists
name = list('Perl')
# print list
print(name)
# Check the slice that needs to be changed before doing a slice assignment print(name[1:])
# Slice assignment with index 1 beginning to end changed to ython
name[1:] = list('ython')
# Assignment result
print(name)
```

Code output:

```
['P', 'e', 'r', 'l']
['e', 'r', 'l']
['P', 'y', 't', 'h', 'o', 'n']

Process finished with exit code 0
```

As you can see from the example above, multiple elements can be assigned at the same time through slices, and slice assignments can replace slices with sequences of different lengths

You can also use slice assignments to insert new elements without replacing existing ones:

```
numbers = [1, 5]
# numbers[1:1] is blank slice
numbers[1:1] = [2, 3, 4]
print(numbers)
```

Code output:

Here I "replaced" an empty slice, which is equivalent to inserting a new sequence, and we can do the opposite to remove the slice:

```
numbers = [1, 2, 3, 4, 5]
# The slice is also left-closed and right-open, Numbers [1:4] represents the three elements in the numbers list that have an index value of 123 numbers[1:4] = []
print(numbers)
```

Code output:

These are the basic operations for creating, modifying, adding, and deleting lists, as well as some common methods for insert, append, which we've covered, clear, copy, count, extend, index, pop, remove, reverse, sort, etc. , but here we do not discuss too much , you can refer to relevant books or Baidu search.

Note: Some methods are sequential, not list-specific; the two most common types of sequences are lists and tuples, and strings are also sequences.

4.4 Dictionary

A dictionary consists of a key and its corresponding value. This key-value pair is called an item. The key and the value are separated by':', the items are separated by commas, and the whole dictionary is enclosed by'{}'. If we go to the key of the dictionary, we can get the corresponding value of the key:

```
# Create an empty dictionary
names = {}
# Create A name-age dictionary
name_ages = {'cx': '23', 'hx': '27', 'yhd': '22'}
# Access key, return value
print(name_ages['cx'])
```

The code returns 23:

4.4.1 Basic Dictionary operation

The basic behavior of a dictionary is similar to a sequence in many ways.

D is a dictionary：

- Len (d) returns the number of items (key-value Pairs) that Dictionary d contains
- d [k] returns the value associated with the key K
- d[k] = v Associates the Value V to the key K
- del d[k] deletes the key K
- k in d checks if the Dictionary d contains an entry with the key K

The dictionary also has methods like clear, copy, fromkeys, get, items, keys, pop, popitem, setdefault, update, etc. , but here we do not discuss too much , you can refer to relevant books or Baidu search.

4.5 Tuple

Like lists, tuples are sequences, the only difference is that tuples are not modifiable (strings are immutable) . The syntax for a tuple is simple, just separate some values with commas and enclose them in parentheses.

```
# Creat a tuple
age = (23,25,27,29)
```

How to create a tuple that contains only one value? Is it like this:

```
age = (23)
print(type(age))
```

Let's see the output:

Python tells us that this is an int, not a tuple.

Correctly create a tuple with only one element:

```
age = (23,)
print(type(age))
```

```
<class 'tuple'>

Process finished with exit code 0
```

Now you know the importance of this comma!

4.6 Conditional Statement

In the context of conditionals and while loops, let's start with boolean values: True and False. When used as a Bourg expression, the following values are treated as false by the interpreter:

```
False, None, '', "", (), [], {}
```

In other words, the standard values False and None, the various numeric values 0, the empty sequence, and the empty map are all treated as false, while other values are treated as true, including the special value True.

This means that any Python value can be interpreted as true. The standard values are True and False, although there are many truth values to choose. In some languages, the standard truth values are 0(false) and 1(true) . In fact, True and False are aliases for 0 and 1, which look different, but do the same thing:

```
# They're equal
print(True == 1)
print(False == 0)
```

Output:

```
True
True

Process finished with exit code 0
```

Let's see the following code:

```
number = input(' Please enter an integer：')
if number:
   print(' The number you entered is：%s' % number)
```

Run the code above , we can print out any number,because the built-in function input () converts any character entered by the user to a string, the string is not null. So the if number assertion is always true, running the print ('you entered a number of:% S'% number) statement. When we press the enter key without typing anything, the program ends up typing nothing.

Because you don't type anything, the string is empty, and the if judgment is false, the result is nothing.

```
number = input(' Please enter a non-zero integer：')
if number:
   print(' The number you have entered is：%s' % number)
else:
   print(' You haven't entered anything！')
```

We added an else decision to run the print statement under the else when you enter an empty string. So we can see that the else is an if clause, and if the if condition is not met, then the

statement under else is run.

Again, let's look at the elif clause; elif is an abbreviation for else if and is used when we need to check multiple conditions:

```
num = input(' Please enter a natural number： ')
num = int(num)
if num > 0:
   print(' The number you entered is greater than 0')
elif num < 0:
   print(' The number you entered is less than 0')
else:
   print(' The number you entered is equal to 0')
```

4.7 Looping statement

A looping statement, as its name implies, is always doing something in a loop.

In many practical problems there are many regular repeat operation, so the program needs to repeat the execution of certain statements. A set of statements that are repeatedly executed is called the body of the loop. Whether the loop can be repeated or not determines the termination condition of the loop. A loop structure is a process structure that executes a program repeatedly under certain conditions. A program that is repeatedly executed is called a loop body. The looping statement consists of the body of the loop and the termination condition of the loop.

4.7.1 While loop

Print all the numbers 1-100, you might have to do the following before learning the loop statement:

```
print(1)
print(2)
print(3)
...
print(100)
```

But we can use a while loop to do this:

```
x = 1
#  a loop condition followed while
while x <= 100:
   # print
   print(x)  # looping body
   x += 1
```

Soon after we execute the code, Python prints out all the numbers 1-100; isn't that a lot simpler? We can also use a loop to make sure the user has entered a name with the following code:

```
name = ''
while not name:
   name = input('Please enter your name： ')
print('Hello，{}!'.format(name))
```

Try to run the code and press enter directly while you're asked to enter a name. You'll see the prompt again, so name is still an empty string, which is false. Where not is a logical predicate for Boolean True and False, not True is False, not False is True.

In a while statement, there are two other important commands, continue, break to skip the loop,

Continue is used to skip the loop, break is used to exit the loop, and the "condition" can also be a constant to indicate that the loop must hold, Here's how it works:

```
# Continue Usage
i = 1
while i < 10:
   i += 1
   # % Module (remainder)
   if i%2 > 0:    # Skip output when not an even number
      continue
   print(i)       # Output even numbers 2,4,6,8,10

# Break usage
i = 1
while 1:          # The circular condition 1 must be ture
   print(i)       # Output 1~10
   i += 1
   if i > 10:    # Jump Out of loop when i is greater than 10
      break      # Exit loop
```

4.7.2 For loop

We learned while loop earlier, while loop statement is flexible enough to iterate through the body of the loop when the condition is true. This is usually fine, but sometimes you need to customize it to your code requirements, such as executing code blocks for each element in the sequence.

Thus,you can use the for loop statement:

```
number = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in number:
   print(i)
```

The for loop iterates through all the elements in the number list and prints them out. This brings us to the built-in range () function, which creates a list of integers, typically in a for loop.

```
for i in range(11):
   print(i)
```

We can see the same effect as the for loop example above.

The range function takes three arguments, range (start position, end position, and step size) , and the step size defaults to 1, and the resulting list contains start positions but not end positions. Here's an example:

```
# Print even numbers for 2-9, starting with 2, ending with 10, and step length with 2
for i in range(2, 10, 2):
   print(i)
```

4.8 Function

Functions are like small programs that can be used to perform specific operations. Function definition uses the def keyword. Here's how to define a function:

```
#  Define a test function
def test():
   # The function functions to output a sentence
   print(' This is a test function ')


# Calling function
test()
```

Code output:

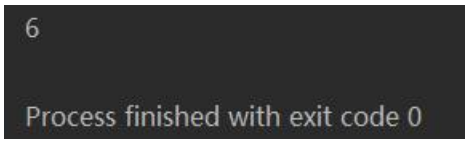
As you can see from the example above, functions are organized, reusable pieces of code that implement a single, or related function. Function can improve the modularity of the application and the reuse of the code.

We've already talked about built-in functions, the sequential methods append () , insert () , and print () , which we used a lot. Our custom functions are called custom functions and are created to do what we need.

By the way, functions can take arguments. Here's a function with parameters:

```
#  Define a test function
def test(x):
   a = x + 3
   # The function ends with return and does not run down again
   # Function returns the value of a
   return a
   # It's not going to be printed
   print(' Does the function run here?')


# Define variable b
b = 3
# Call the test function to pass in parameter b and print it out with the print function
print(test(b))
```

Code output:



Of course, the function can be passed in a number of parameters and parameters can have a default value, but we do not have a detailed overview here, you can look up relevant information on the Internet, for systematic learning.

```
#  Parameter with default value
```

```
#  Define a test function
def test(x, y=3):
   a = x + 3
   b = y + 3
   # The function ends with return and does not run down again
   # Function returns the value of a
   return a + b


# Define variable b
b = 3
# Call the test function to pass in parameter b and print it out with the print function
# When no second argument is passed to the function, y uses the default value 3
print(test(b))
# Direct pass parameter 2 and 3
print(test(2, 4))
```

4.9 Class

In the last section of this chapter, we talk about classes. What are classes? A class is an object. Each object belongs to a specific class and is called an instance of that class.

Did you get a little lost there? Just to clarify, you are an object. What kind of object do you belong to? You belong to the human race, which is a very general class, because it has different skin colors, different languages and so on. You are the object of human instantiation. Does that make it clearer?

Let's create a class and see what features it has:

```
class Person:
   def __init__(self,name,age):
      self.name = name
      self.age = age

   def eat(self):
      print(self.name + ' Eat ')

   def sleep(self):
      print(self.name + ' Sleep ')

   def get_age(self):
      print(self.name + ' This year has been.. ' + str(self.age))


# Instantiate object
A = Person('XiaoMing', 25)
B = Person('XiaoHong', 18)
# Object to call class methods
A.eat()
A.sleep()
A.get_age()
B.eat()
B.sleep()
```

B.get_age()

Through the example, we know：

1. Classes are defined through the keyword class.

2. __ init __ () is a special method that Python automatically runs whenever you create a new instance based on the Person class. In the name of this method, there are two underscores at the beginning and two at the end, which is a convention designed to avoid name conflicts between Python default methods and normal methods.

3. Method __init__ () is defined to have three parameters: Self, name, and age. In the definition of this method, the parameter self is necessary and must precede the other parameters. Why do you have to include the parameter self in the method definition? Because Python calls the __init__() method to create the Person instance automatically passes in the argument self. Each method call associated with a class passes the argument self, which is a reference to the instance itself, giving the instance access to properties and methods in the class. When we create the Person instance, Python calls method __init__() of the Person class. We will pass the name and age to Person () through the argument; self will pass it automatically, so we don't need to pass it. Whenever we create an instance based on the Person class, we only need to provide a value for the last two formal parameters (name and age) .

4.Eat、 sleep and get_age， Since these methods require no additional information, such as name or age, they have only one parameter, self; the instance we will create later would able to access these methods, in other words, humans will eat and sleep.

Of course, we're just going to take you through the basics of classes. Classes are very important part of Python, and it's not something you can explain in a minute or two, because of the relatively complex concepts of inheritance and polymorphism, need everybody to come down to consult the related material to carry on the system study.

4.10 Summary

This chapter is just a brief introduction to the basics of Python, so that if you are zero-based users, you can learn a little bit about Python and then look at the underlying machine code below. But if you want to get a good grasp of the car's Python Code, you'll probably need to know a bit about the basics of Python and its syntax.

At the same time, the x series code comments are very detailed and easy to understand. Believe your abilities!

At this point we have a basic understanding of the creation and use of variables, we talked about the number, string, list, tuple, dictionary are Python variables, in which list and dictionary are common data type, you need to be skilled with it. Sometimes, we need to convert the data to the built-in type, the data type conversion, you just need to use the data type as the function name. You can look it up, or search for the Keyword: Python data type conversion.

Conditional statements and circular statements are also the basic knowledge that we need when writing logical structures, and the conditional judgment grammar itself is also very simple, as long as your logical thinking is clearly, there is no problem.

The functions avoid a lot of duplication of effort, improve Code Reuse, reduce the amount of code written, if you need to write functions that need to be used in multiple places, we can consider

writing functions, use function to reduce the amount of code you write.

The only thing that is hard for beginners to understand is classes, which exist not only in Python, but also in the commonly used C + + Java concept of classes. In a nutshell, a class is a high-level abstraction, a high-level data type, a blueprint for an object, that defines the properties and behavior of the object you want to use Think of it as a template from which we can instantiate objects to use, which in turn are abstracted into classes. Class knowledge is too broad, a sentence or two is not clear, there are inheritance, polymorphism, rewrite knowledge points, class methods are divided into instantiation methods, class methods (@classmethod) , static methods (@staticmethod) . If you want to learn thoroughly, you need to come down and look up the relevant information. I won't go into it here. I won't go into it here.

Chapter 5 Lower computer source code learning

5.1 Raspberry GPIO programming

Before learning the source code, we need to understand one thing, the program is diverse; as we learn mathematics when we are young, there is only one answer, but the solution is diverse. The same is true for programs, where 20 programmers working on the same project can come up with many different scenarios; the way the code is expressed is different because everyone understands it differently. So if you have a different idea in the next chapter, or if you have a simpler implementation, you can redevelop the code as you see fit, also congratulates you to encounter the problem to be able to have the different angle different direction thinking

Raspberry programming generally goes straight to efficient Python ,for GPIO programming, Python also has a library for this, the best known and most commonly is RPI.GPIO. This library is specifically designed for programming the Raspberry, which lets you easily control the GPIO pins as easily as the Arduino.

5.1.1 Install libraries

Install Python3 RPI.GPIO

To install the GPIO library, type the following in the Raspberry Terminal Command interface.

```
# Install RPI.GPIO，install it here python3 GPIO！Emphasis
sudo apt-get install python-rpi.gpio python3-rpi.gpio
```

Query GPIO corresponding pins

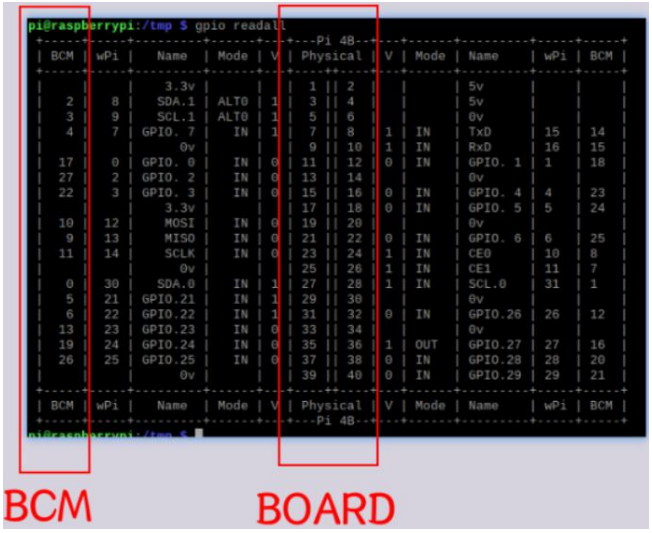
The GPIO pin diagram can be queried by the Terminal Input Instruction Gpio readall. The 4B motherboard may report the following error when using this instruction.

```
Oops - unable to determine board bype... model: 17
```

This error is due to the fact that PI4 is not currently supported in version, so you need to update it ,you can enter the following commands in turn:

```
1. cd /tmp
2. wget https://project-downloads.drogon.net/wiringpi-latest.deb
3. sudo dpkg -i wiringpi-latest.deb
```

After completing the above update, type GPIO readall again to query, as shown below:



5.1.2 Basic GPIO usage

Import the RPI.GPIO module

As anyone with a Python programming background knows, to use a library file (package) , you first need to import the library file. If we want to program the Raspberry Pie GPIO in Python, the first thing we need to do is the guided package operation.

```
# Import RPI. GPIO library, and use the GPIO as an alias
 import RPi.GPIO as GPIO
```

If you want to see how successful the introduction is, just run the code, no error means OK, or you can use the try statement:

```
try:
   import RPi.GPIO as GPIO
except RuntimeError:
   print("Introduce error ")
```

Stitch number

At the RPI. GPIO ,the two pin numbers on the Raspberry Pi are also supported in the GPIO. The first is the BOARD number, which corresponds to the physical pin number on the Raspberry Pi Board. The advantage of using this number is that your hardware will always be available and you won't have to worry about the Raspberry Pi version. Therefore, after the board upgrade, you do not need to rewrite the connector or code. The second number is the BCM rule, which is a more low-level way of working, and corresponds to the channel number in Broadcom's on-chip systems. When using a pin, you need to find the corresponding rule between the channel number and the physical pin number. For different versions of Raspberry Pi, the script files you write may not be common

By querying the pin number with the GPIO readall directive, we can see where the two pins, BOARD and BCM, correspond, so the two patterns can be determined to your liking. Most beginners prefer to write code in BOARD mode.

```
# Import RPI. GPIO library, and use the GPIO as an alias
import RPi.GPIO as GPIO
# The pin numbering rules use BOARD mode
GPIO.setmode(GPIO.BOARD)
```

Pin Initialization

Like the Raspberry Pattie Arduino, before you use a pin, you need to set the pin: As input or output.

```
# Set the pin to input mode
GPIO.setup(pin, GPIO.IN)
# Set the pin to output mode
GPIO.setup(pin, GPIO.OUT)
# Set the default high level for the output pins, the low level initial=GPIO.LOW
GPIO.setup(pin, GPIO.OUT, initial=GPIO.HIGH)
```

Read Pin value

We often encounter some common combination of sensors in our lives. For example, the voice-activated light, if there is sound, the corridor lights will be lit up, this is a very simple case, the actual voice sensor will have a threshold (critical value) , when the sensor exceeds this threshold, it returns a state (with sound) , which we judge to be the next step, so we need to read the GPIO's return value.

```
# The input () method can read the value of the pin pin
```

```
GPIO.input(pin)
 # If your pin is connected to a number of sensors, such as a temperature sensor, light sensor, etc. .
```

Pull it Up and down

First of all, what is pull up, pull down? Pull-up and pull-down refer to whether the GPIO outputs a high potential (pull-up) or a low potential (pull-down) . Pull-up is the input high level, and then a pull-up resistance (for protection) , until the pull-up means that the port in the default input is high level. A drop-down is an input low and a drop-down resistance is applied. When we do the up and down of the pin, we usually add the down and up resistors in the circuit design, but can we control the circuit in Raspberry without changing the circuit in Python? For Python'S GPIO initialization:

```
GPIO.setup(pinx,GPIO.IN,pull_up_down=GPIO.PUD_UP/GPIO.DOWN)
```

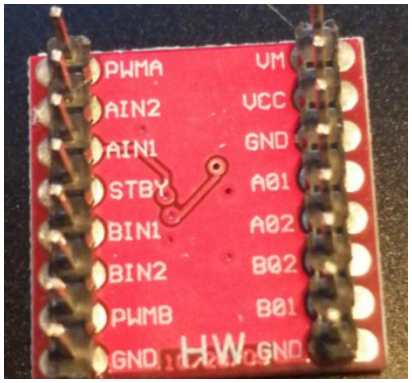
The above is the basic GPIO, but it is not limited to the above knowledge. If you are interested, you can search relevant information to expand your knowledge base.

5.2 Motor

We must be familiar with electric motor, four-wheel-drive car is the toy we often play in childhood, we will take down the motor when it is broken down, in fact, this motor is what we now call the motor. We connect a battery directly to the motor and it turns. Then we connect the two wires of the motor back to the battery and it turns the same way. We change the wiring manually, which is equivalent to the motor drive function that we can use now.

Motor drive can control the positive and negative rotation of the motor, the principle is that the chip has an h bridge to change the direction of the current .L298, TB6612 and other motor drive are Common Motor Drive , our car motor drive is TB6612.

Simply , TB6612 supports two-way motor output, each output has three states, positive, negative, stop .So if you put two different inputs on either side, the motor turns



Interface	Stop	Forward rotation	Inverts
AIN1	0	0	1
AIN2	0	1	0
BIN1	0	0	1
BIN2	0	1	0

5.2.1 Single Motor experiment

Looking at the above table, we can see that as long as we give different signals to the two input

ports, the motor will turn forward or backward; in addition to the input port, there is an enable pin EN, which can be used for speed regulation.

Start with the experimental code:

```
# Import package operation
import RPi.GPIO as GPIO
import time

# Defines the GPIO interface
ENA = 13  # TB6612 enable A
IN1 = 19  # Motor Interface 1
IN2 = 16  # Motor Interface 2

# Remove warning messages
GPIO.setwarnings(False)
# BCM mode
GPIO.setmode(GPIO.BCM)
# Initialize pin to output mode and set to low level
GPIO.setup(ENA, GPIO.OUT, initial=GPIO.LOW)
# Create a pwm instance
ENA_pwm = GPIO.PWM(ENA, 100)
# Turn on pwm
ENA_pwm.start(0)
# Changing the frequency is equivalent to adjusting the speed
ENA_pwm.ChangeDutyCycle(50)
GPIO.setup(IN1, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(IN2, GPIO.OUT, initial=GPIO.LOW)

try:
  while 1:
    GPIO.output(IN1, True)
    GPIO.output(IN2, False)
    time.sleep(2)
    GPIO.output(IN1, False)
    GPIO.output(IN2, True)
    time.sleep(2)
except KeyboardInterrupt:
    print(' You terminated the program \n Program terminated ! ')
    GPIO.cleanup()
```

Tips： Here's how to run the lab code for the car; The lab code for the car behind us will run in Raspberry, pasted or winscp. The Py File is transferred to the Raspberry desktop (it's up to you where you want it to go) .

A word of caution:

1.Warning：

The Raspberry Gpio may have more than one script/circuit manipulating the GPIO. If the Raspberry detects that the pin is not the default input state, it gives a warning, which can be avoided with a single line of code:

```
# Remove warning messages
```

```
GPIO.setwarnings(False)
```

2. Use RPI. Pulse Width modulation (PWM) function of GPIO module

Pulse Width modulation (PWM) is a method of digital encoding of analog signal level, which uses digital output of microprocessor to control analog circuit. On Raspberry Pi, PWM can be realized by programming GPIO.

Create a PWM instance:

```
p=GPIO.PWM(channel, frequency) (Pin ,Frequency）
```

Turn on PWM：

```
p.start(dc) # dc stands for duty cycle（Range：0.0 <= dc >= 100.0）
```

Change Frequency：

```
p.ChangeFrequency(freq) # freq is the new frequency set, in Hz
```

Change duty cycle：

```
p.ChangeDutyCycle(dc)  # range：0.0 <= dc >= 100.0
```

Stop PWM:

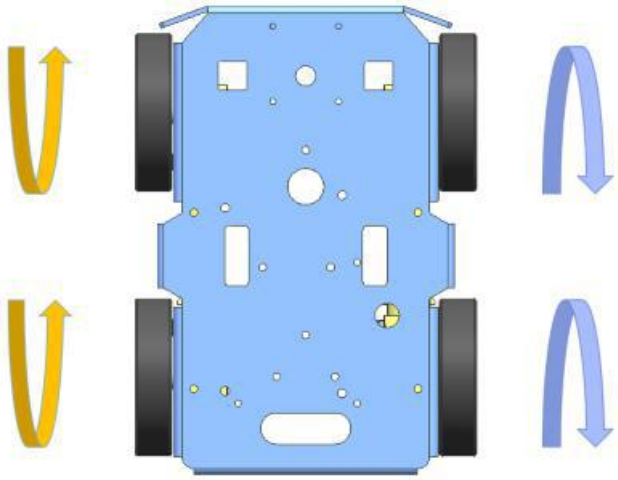
```
p.stop()
```

Note that if the variable "P" in the instance is out of range, it will also cause the PWM to stop.

5.2.2 Differential steering principle

What is a differential steering?

Differential steering is when a vehicle turns by controlling the difference in speed between the left and right wheels. When the wheels rotate at different speeds, or when one of the wheels does not move, the car rotates.



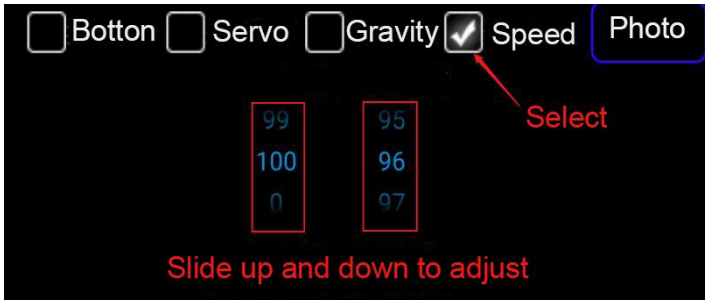
Turn clockwise on the left and counterclockwise on the right

The speed on both sides will form a differential speed and turn in place

As you can see from the diagram, the right-hand drive wheel stops or turns clockwise, but the speed is slower than the left-hand drive wheel, and the car turns right or deviates to the right.

Here we also give you a point, because the left and right motor due to environmental (friction)

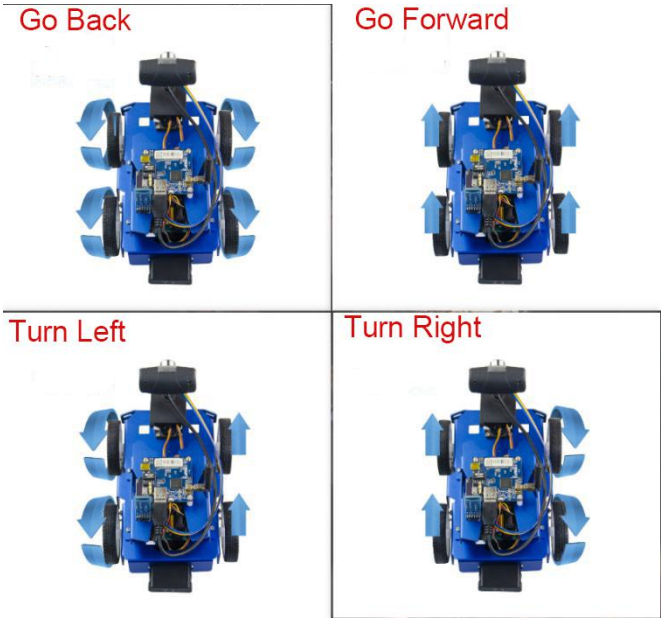
and motor internal and other factors will cause the left and right side of the driving wheel speed difference, the speed difference is very slight, but the longer the car goes straight, the more we will see that the car will have a deviation when it goes straight. This can not be completely avoided. When there is a big difference in going straight, we can compensate (but not completely avoid) with Left-and Right-motor speed control; if the car is moving to the left, then turn down the right-hand speed.



5.2.3 Analysis of the principle of car motion

On the x series car, there are only two motors; based on the differential turn principle in the previous bar, we can analyze the four motion states of the car, which are as follows：

1. Forward: Two Motors turn forward simultaneously;
2. Back: Two Motors turn back at the same time；
3. Left turn: Left Motor to turn back, right motor to turn forward；
4. Right turn: the right motor turns back, the left motor turns forward.



5.2.4 Raspberry-x motor program

Class:RobotDirection File：xr_motor.py

In the Raspberry-X series, we first created a cart direction class, the RobotDirection class,

created six functions in the RobotDirection class to set the state of TB6612's two-way motor, using this six states to set the cart's motion state.

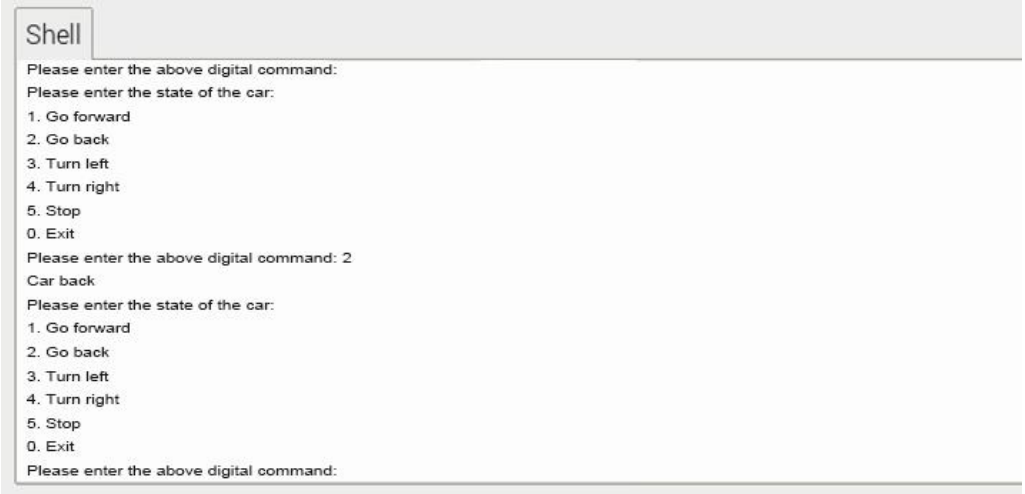
The following code can be tested, combined with separate motor experiments in section 5.1.1 and motion state analysis in section 5.1.3, and is easy to understand:

```
import RPi.GPIO as GPIO

ENA = 13  # TB6612 enable A
ENB = 20  # TB6612 enable B

IN1 = 16  # Motor Interface 1
IN2 = 19  # Motor Interface 2

IN3 = 26  # Motor Interface 3
IN4 = 21  # Motor Interface 4


# GPIO output settings
def digital_write(gpio, status):
    GPIO.output(gpio, status)


# Create a cart direction class
class RobotDirection(object):
    def __init__(self):
        pass

    # The two motors on the left are connected in series together
    # The two motors on the right are connected in series with the other one
    # Left Motor turning forward
    def m1m2_forward(self):
        digital_write(IN1, True)
        digital_write(IN2, False)

    # Reverse left motor
    def m1m2_reverse(self):
        digital_write(IN1, False)
        digital_write(IN2, True)

    # Left Motor stopped
    def m1m2_stop(self):
        digital_write(IN1, False)
        digital_write(IN2, False)

    # Right Motor turning forward
    def m3m4_forward(self):
        digital_write(IN3, True)
        digital_write(IN4, False)

    # Reverse right motor
```

```
    def m3m4_reverse(self):
        digital_write(IN3, False)
        digital_write(IN4, True)

    # Rjght Motor stopped
    def m3m4_stop(self):
        digital_write(IN3, False)
        digital_write(IN4, False)

    # The above 6 functions simply set the running state of the two-way motor
    # The following five functions set the running state of the cart
    def car_forward(self):
        print(' The car goes forward ')
        self.m1m2_forward()
        self.m3m4_forward()

    def car_reverse(self):
        print(' The car goes back ')
        self.m1m2_reverse()
        self.m3m4_reverse()

    def car_left(self):
        print(' The car turns left ')
        self.m1m2_reverse()
        self.m3m4_forward()

    def car_right(self):
        print(' The car turns right ')
        self.m1m2_forward()
        self.m3m4_reverse()

    def car_stop(self):
        print(' The car stop')
        self.m1m2_stop()
        self.m3m4_stop()

# Program entry
if __name__ == '__main__':
    # Remove warning messages
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    # Initializing motor
    GPIO.setup(ENA, GPIO.OUT, initial=GPIO.LOW)
    ENA_pwm = GPIO.PWM(ENA, 1000)
    ENA_pwm.start(0)
    ENA_pwm.ChangeDutyCycle(50)
    GPIO.setup(IN1, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(IN2, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(ENB, GPIO.OUT, initial=GPIO.LOW)
    ENB_pwm = GPIO.PWM(ENB, 1000)
```

```
ENB_pwm.start(0)
ENB_pwm.ChangeDutyCycle(50)
GPIO.setup(IN3, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(IN4, GPIO.OUT, initial=GPIO.LOW)
# Instantiate object
A = RobotDirection()
try:
  while True:
    information = input(' Please enter car status：\n'
                ' 1. Forward \n'
                ' 2. Back \n'
                ' 3.Turn left\n'
                ' 4.Turn right \n'
                ' 5.Stop\n'
                ' 0. Quit \n'
                ' Please enter the numeric command above:')
    if information == '1':
      # Calling the car function using an object
      A.car_forward()
    elif information == '2':
      A.car_reverse()
    elif information == '3':
      A.car_left()
    elif information == '4':
      A.car_right()
    elif information == '5':
      A.car_stop()
    elif information == '0':
      A.car_stop()
      print('Exit')
      # Exit loop
      break
except KeyboardInterrupt:
  print(' You terminated the program. Program n terminated！ ')
  # Clear the GPIO settings
  GPIO.cleanup()
```

We run the above code line in Raspberry Pi:



You can see that we send the corresponding digital command, the car will carry out the corresponding digital command. Now let's go back to the file xr_motor.py,it will be very clear.

In the experiment, we omitted the link of speed setting, in the motor program, there is a link of speed adjustment. This we briefly mention, in the socket we will determine whether there is a speed adjustment command, if there is a speed adjustment, we will call the corresponding function to adjust the speed.

5.3 Infrared sensor

The infrared sensor on the x series is a digital sensor, the output state is 0,1, namely in the digital circuit high level and low level.

Infra-red Colour and interface table：

Wire color	Interface description
Red	GND （Black stitching）
Green	5V （Red stitching）
Orange	IR_R （Yellow stitching）
Black	IR_L （Yellow stitching）

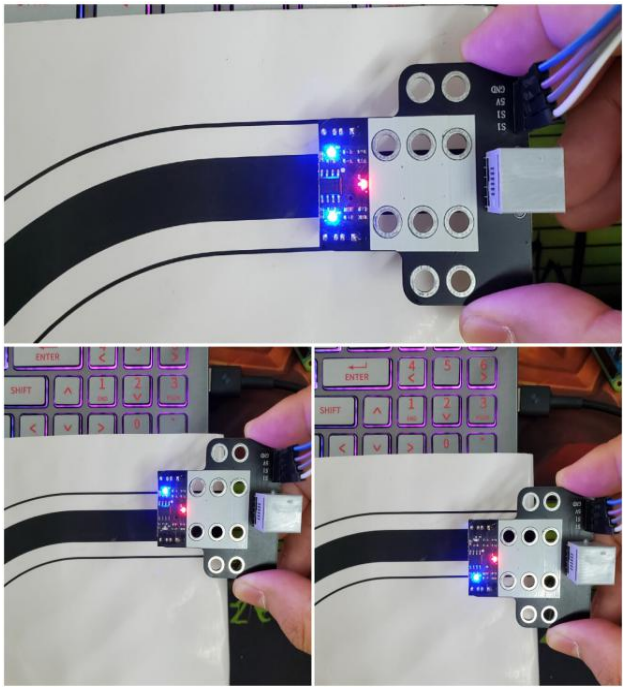
5.3.1 Use of infrared sensors

How to use infrared sensor? We still use experiment to explain, experiment can let a person feel most!

Here are the two states:

Module does not detect black line, module status for light, level output: 0

Module detected black line, module status for the lights out, level output: 1



The experimental code is as follows:

```
# !/usr/bin/env python
import RPi.GPIO as GPIO
import time

IR_R = 18  # The right side of the car patrol infrared
IR_L = 27  # The left side of the car patrol infrared


def setup():
  # Remove warning messages
  GPIO.setwarnings(False)
    # Set the pin mode to BCM mode
    GPIO.setmode(GPIO.BCM)
  # Initialize setting to input mode, up
  GPIO.setup(IR_R, GPIO.IN, pull_up_down=GPIO.PUD_UP)
  GPIO.setup(IR_L, GPIO.IN, pull_up_down=GPIO.PUD_UP)


def loop():
  while True:
```

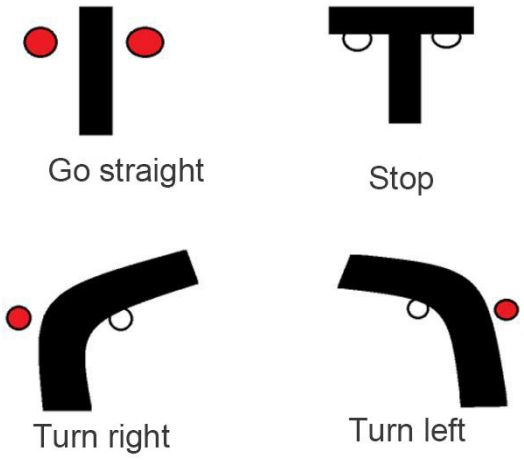
```
        print('IR_R The current state is:' + str(GPIO.input(IR_R)))
        print('IR_L The current state is:' + str(GPIO.input(IR_L)))
        time.sleep(1)


if __name__ == '__main__':
    print(' Program running ')
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        # Capture Keyboard ^ C exit program
        print(' You terminated the Program n Program End!')
        GPIO.cleanup()
```

From the above code experiment, we know that we can read the level change of the infrared sensor by the code, so we can use the if judgment statement to control something by the level change of the infrared; in short, when the infrared sensor lights off (output is 1) we use the code to turn on the headlights and other experiments, this is a very simple statement of judgment, meet the conditions will be executed. For the next lesson, we will use infrared sensors to control the positive and negative rotation of the motor.

5.3.2 Infrared scanning function

Class：Infrared File：xr_infrared.py

Under the file xr_infrared.py,there is the Infrared function, we created an Infrared class, the class contains Infrared patrol, Infrared follow, Infrared obstacle avoidance, Infrared fall function.



As can be seen from the above diagram, infrared line inspection principle, the analysis of the State is as follows:

Straight: The left and right infrared sensors do not detect a black line (bright blue) , so the car should go straight

Stop: Both right and left infrared sensors detect a black line (both extinguished) , so the car should stop.

Right turn： As shown in the picture, the black line is on the right-hand side of the road,

but the car is in a straight line before entering the corner, so the infrared sensor that first touches the black line should be the infrared sensor on the right, at this point we should correct the direction of the car, or it will rush out； The car turn right, throgh the difference speed turn (3.2.1 chapter) we mentioned， we know the car should be the state: left positive turn (clockwise) , right reverse (counter-clockwise) .

Left turn: As you can see, the first sensor to detect the black line on the left is the infrared sensor on the left; then the car is the exact opposite of the right turn.

Line inspection is still a relatively simple function, we only need to read the infrared pin returned high and low levels to determine whether the infrared above the black line or not , Two infrared return of the state only the above four possibilities, and then according to the four states to perform the corresponding car state can maintain patrol function.

Next is the Raspberry-X series of Infrared patrol code execution process, through the socket code communication received Infrared patrol instructions, we will call the Infrared class trackline patrol function.

Perform host instruction parsing in the communicationdecode function in the socket.py， When the infrared patrol command is recognized, we change the value of the function mode loop flag bit CRUISINGFLAG to enter a different function mode by comparing the value of the flag bit.

```
# Communication decoding instruction of upper computer
def communication_decode(self, buffer):
  # ···In front
  # ···Omitted, available in source code
    elif buffer[0] == 0x13:
      if buffer[1] == 0x01:      # Infrared follow command
        cfg.CRUISING_FLAG = 1   # Go to infrared follow mode and change the CRUISING value
      elif buffer[1] == 0x02:    # Infrared line-tracking command
        cfg.CRUISING_FLAG = 2   # Go to the infrared cruising command and change the
CRUISING value
  # ···Follow-up
  # ···Omitted
```

After changing the CRUISINGFLAG value, we check the CRUISINGFLAG value in the main loop to enter different functional patterns; Special note here: The Socket Class Communication Function Bluetooth Server (Bluetooth Communication) and tcpserver (network line, WiFi communication) is started through threads (Python multithreading) , Socket communication will call the instruction parsing function, this means that the communication instruction parsing function is running at the same time as the main loop cruising function and the PS2 control function; here is the key to understanding the meaning of this passage.

Multi threaded startup with main loop cruisingmode and PS2 control in xrstartmain.py .

The following code shows how to enter the corresponding function mode after we change the CRUISING value:

```
def cruising_mode():
  """
  Mode switching function
  """
```

```
    # print('pre_CRUISING_FLAG：  {}'.format(cfg.PRE_CRUISING_FLAG))
    time.sleep(0.05)
    # If the pre-loop Mode 0 is not equal to the current mode loop (the first time the program runs is 0, the first time it does not enter this if judgment)
    # Cruising has a value, not 0, when it is in a single function mode
    if cfg.PRE_CRUISING_FLAG != cfg.CRUISING_FLAG:
        # If the pre-loop mode is not 0
        # Similarly, if it is the first time into a function, the car is already at rest, will not enter this if judgment
if cfg.PRE_CRUISING_FLAG != 0:
            # Stop the cart before moving on to the next mode
            go.stop()
        # The pre-loop pattern flag is then set to the current pattern flag
        cfg.PRE_CRUISING_FLAG = cfg.CRUISING_FLAG

        if cfg.CRUISING_FLAG == 1:  # Go to infrared follow mode
            # print("Infrared.irfollow")
        infrared.irfollow()
        time.sleep(0.05)

    # Determines whether the CRUISING value is 2, where the function is parsed
        # We've already assigned a value to CRUISING in different functional modes
        elif cfg.CRUISING_FLAG == 2:  # Go to infrared patrol mode
        # print("Infrared.trackline")
        # Call the infrared line detection function
        infrared.trackline()
        time.sleep(0.05)
    # ···Follow-up
    # ···Omitted
```

This is when we call the infrared crutch function:

```
class Infrared(object):
    def __init__(self):
        pass
    # Infrared line inspection
    def trackline(self):
        print('ir_trackline run...')
        # No black lines on either side
        # digital_read()Read Pin Status
        if (gpio.digital_read(gpio.IR_L) == 0) and (gpio.digital_read(gpio.IR_R) == 0):
            go.forward()
        # Infrared sensors on the right are picking up a black line
        elif (gpio.digital_read(gpio.IR_L) == 0) and (gpio.digital_read(gpio.IR_R) == 1):
            go.right()
        # Sensors on the left are picking up a black line
        elif (gpio.digital_read(gpio.IR_L) == 1) and (gpio.digital_read(gpio.IR_R) == 0):
            go.left()
        # Black lines are detected on both sides
        elif (gpio.digital_read(gpio.IR_L) == 1) and (gpio.digital_read(gpio.IR_R) == 1):
            go.stop()
```

Now that you have a rough idea of how the whole framework works, let's use the code to do the

infrared patrol function alone, which is the basic knowledge of the sensor application.

Combine with the previous motor program, do a separate line inspection function:

```
# !/usr/bin/env python
# encoding: utf-8

import RPi.GPIO as GPIO

ENA = 13  # TB6612 enable A
ENB = 20  # TB6612 enable B
IN1 = 16  # Motor Interface 1
IN2 = 19  # Motor Interface 2
IN3 = 26  # Motor Interface 3
IN4 = 21  # Motor Interface 4
IR_R = 18  # The right side of the car patrol infrared
IR_L = 27  # The left side of the car patrol infrared

# GPIO output settings
def digital_write(gpio, status):
    GPIO.output(gpio, status)


# Create a cart direction class
class RobotDirection(object):
    def __init__(self):
        pass

    # The two motors on the left are connected in series together
    # The two motors on the right are connected in series with the other one
    # Left Motor turning forward
    def m1m2_forward(self):
        digital_write(IN1, True)
        digital_write(IN2, False)

    # Reverse left motor
    def m1m2_reverse(self):
        digital_write(IN1, False)
        digital_write(IN2, True)

    # Left Motor stopped
    def m1m2_stop(self):
        digital_write(IN1, False)
        digital_write(IN2, False)

    # Right Motor turning forward
    def m3m4_forward(self):
        digital_write(IN3, True)
        digital_write(IN4, False)

    # Reverse Right motor
    def m3m4_reverse(self):
```

```
        digital_write(IN3, False)
        digital_write(IN4, True)

    # Right Motor stopped
    def m3m4_stop(self):
        digital_write(IN3, False)
        digital_write(IN4, False)

    # The above 6 functions simply set the running state of the two-way motor
    # The following 5 functions set the running state of the cart
    def car_forward(self):
        print(' The car goes forward ')
        self.m1m2_forward()
        self.m3m4_forward()

    def car_reverse(self):
        print(' The car goes back ')
        self.m1m2_reverse()
        self.m3m4_reverse()

    def car_left(self):
        print(' The car turns left ')
        self.m1m2_reverse()
        self.m3m4_forward()

    def car_right(self):
        print('he car turns right')
        self.m1m2_forward()
        self.m3m4_reverse()

    def car_stop(self):
        print('The car stops')
        self.m1m2_stop()
        self.m3m4_stop()


if __name__ == '__main__':
    # Remove warning messages
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    # Initializing motor
    GPIO.setup(ENA, GPIO.OUT, initial=GPIO.LOW)
    ENA_pwm = GPIO.PWM(ENA, 1000)
    ENA_pwm.start(0)
    ENA_pwm.ChangeDutyCycle(50)
    GPIO.setup(IN1, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(IN2, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(ENB, GPIO.OUT, initial=GPIO.LOW)
    ENB_pwm = GPIO.PWM(ENB, 1000)
    ENB_pwm.start(0)
```

```
ENB_pwm.ChangeDutyCycle(50)
GPIO.setup(IN3, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(IN4, GPIO.OUT, initial=GPIO.LOW)
# Infrared initialization
GPIO.setup(IR_R, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(IR_L, GPIO.IN, pull_up_down=GPIO.PUD_UP)
# Instantiate object
A = RobotDirection()
try:
  while True:
    while 1:
      # No readings on either side
      if (GPIO.input(IR_R) == 0) and (GPIO.input(IR_L) == 0):
        A.car_forward()
        # Infrared sensors on the right are picking up a black line
      elif (GPIO.input(IR_R) == 0) and (GPIO.input(IR_L) == 1):
        A.car_right()
        # Sensors on the left are picking up a black line
      elif (GPIO.input(IR_R) == 1) and (GPIO.input(IR_L) == 0):
        A.car_left()
        # Black lines are detected on both sides
      elif (GPIO.input(IR_R) == 1) and (GPIO.input(IR_L) == 1):
        A.car_stop()
except KeyboardInterrupt:
  print(' You terminated the program. Program n terminated ! ')
  GPIO.cleanup()
```

Before the program is the motor control program, but in the main cycle, we no longer input different numbers to control the motor's operation, by judging the infrared output state to change the motor's operation state, just a change in judgment.

5.3.3 Infrared cliff fall prevention

We can actually test it. When we put the car on the ground, the car is in the light state. If we hang the car or put the infrared sensor on the edge of the table, we can see that the infrared sensor is in the light-out state, we looked at chapter 4.3.1 and found that when the lamp is on, the value of the Ir Pin is 0 and the lamp is off by 1. Therefore, the cliff fall prevention function only needs to judge the infrared state.

The calling method also follows the same procedure as described in section 5.2.1, and if you don't understand it, you can go back and look again.

The infrared cliff fall protection code is similar to the infrared patrol line, but the difference is the trigger condition and the function of the call after the trigger condition is any infrared hanging, that is to return to the high level; and the function of the call after the trigger is to stop all motor rotation.

```
class Infrared(object):
  # Infrared fall protection
  def avoiddrop(self):

  # First should set the speed, speed will cause the reaction is not timely
```

```
    if cfg.RIGHT_SPEED > 30 or cfg.LEFT_SPEED > 30:
        cfg.LEFT_SPEED = 30
        cfg.RIGHT_SPEED = 30
    # When both sides of the infrared detect the ground
    if gpio.digital_read(gpio.IR_L) == True or gpio.digital_read(gpio.IR_R) == True:
        # Set the IR drop mark at one
        cfg.AVOIDDROP_CHANGER = 1
    else:
        # Only when the last state obtained is normal state will stop the operation, to avoid repeated
execution stop can not be remote control
        if cfg.AVOIDDROP_CHANGER == 1:
            go.stop()
            cfg.AVOIDDROP_CHANGER = 0
```

Here we add a flag bit AVOIDDROP_CHANGER, which is used to indicate whether the cart has reached the cliff edge or not. When the car do not reach the edge of the cliff the flag bit is 1(default value is 1), if the car do not reach the cliff edge that is the judgment condition in the code whether the two infrared sensors are equal to 0;The keyword and requires both results to be 0 before the conditional expression is true; the other results are assumed to be false, and we set AVOIDDROP_CHANGER to 0.

Finally, AVOIDDROP_CHANGER works on the communication parsing function in the socket communication code, so we'll extract a section to illustrate it. The section is as follows:

```
def communication_decode(self, buffer):
    print(buffer)
    if buffer[0] == 0x00:  # Buffer [0] means type bit, equal to 0x00 means that this packet is
motor control instruction packet
        if buffer[1] == 0x01:  # Buffer [1] is the control bit, equal to 0x01 is the forward packet
            # To judge the state of ultrasonic obstacle avoidance and infrared drop prevention
            if cfg.AVOID_CHANGER == 1 and cfg.AVOIDDROP_CHANGER == 1:
                go.forward()  # Forward
        elif buffer[1] == 0x02:
            go.back()  # Step back

        elif buffer[1] == 0x03:
            if cfg.AVOID_CHANGER == 1 and cfg.AVOIDDROP_CHANGER == 1:
                cfg.LIGHT_STATUS = cfg.TURN_LEFT
                go.left()  # Turn left

        elif buffer[1] == 0x04:
            if cfg.AVOID_CHANGER == 1 and cfg.AVOIDDROP_CHANGER == 1:
                cfg.LIGHT_STATUS = cfg.TURN_RIGHT
                go.right()  # Turn right

        elif buffer[1] == 0x00:
            cfg.LIGHT_STATUS = cfg.STOP
            go.stop()  # stop

        else:
            go.stop()
```

We can see that when we control the direction, when we receive the commands front, left, and right, it is a priority to determine whether the AVOID_CHANGER and AVOIDDROP_CHANGER flag bits are equal to 1, again using the and keyword; we can ignore the AVOID_CHANGER flag bits, this is the mark bit when ultrasonic obstacle avoidance, the principle is the same; if judgment is true when AVOIDDROP_CHANGER is equal to 1, the program can call before, left, and right class methods, when any infrared sensor of the car returns to low level, it will cause AVOIDDROP_CHANGER = 0, equal to 0, if judgment condition is false, and if judgment condition is false after receiving the front, left and right instructions, the result can not be invoked either. Careful you must find that the back command is not added if judgment conditions, because no matter when, the car should be able to back up.

5.3.4 Infrared following function

The infrared following function is only available in Gfs-x and TH-X because of the size of the car body. DS-X does not include this function. When reading the code follow the function the left infrared sensor definition pin is 4, followed by the right infrared sensor definition pin is 9.From our hardware profile design:

					Raspberry Pi
		ECH0	PB1	2	17
	Buzzer	0UT	PA7	3	10
	Expansion IO	IO_1	PB0	4	11
		IO_2	PA2	9	8
		IO_3	PA3	10	7
		IO_4	PA6	11	5

5.4 Ultrasonic wave

The ultrasonic module can measure the distance of 3cm-4m with an accuracy of 3mm. This module includes three parts: ultrasonic transmitter, ultrasonic receiver and control circuit. It have four pins.

5.4.1 Principle of ultrasonic distance measurement

The ultrasonic module is HC-SRO4 and the sensor has four pins:

1. VCC， The ultrasonic module power supply pin is connected with 5V power supply；

2. Trig， Ultrasonic sending foot, high level of 40KHZ ultrasonic；

3. Echo， Ultrasonic receiving foot,When the return ultrasonic wave is received, set to a low level；

4. GND， Ultrasound module, GND， Earthing。

How the ultrasound module works:

1. Using IO port TRIG trigger ranging, to at least 10 US high-level signal;

2. The HC-SR04 receives the signal, begins to send the ultrasound, and sets the Echo to a high level, then prepares to receive the returned ultrasound;

3. The HC-SR04 receives the returned ultrasound and sets the Echo to a low level;

4. The duration of the Echo high level is the interval between the time the ultrasound is

emitted and the time it is returned;

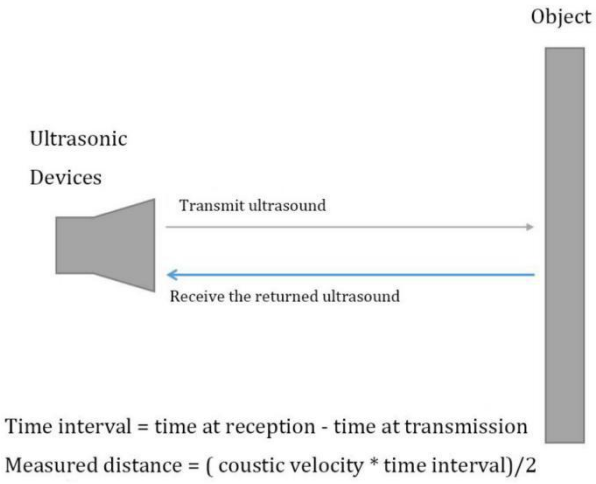
5. Calculated distance:

Distance(Unit: cm) = (Start Time - End Time) * Acoustic velocity / 2

Sonic velocity: 340m/s, The final measured distance needs to be converted to cm

So the final formula is:

Distance(Unit: cm) = Time difference * 340 / 2 * 100



5.4.2 Ultrasonic Test Code

We use the ultrasonic module alone with the test code, you can use this code to test whether the ultrasonic is working properly. This code will be in the terminal command line has been output ultrasonic measured distance.

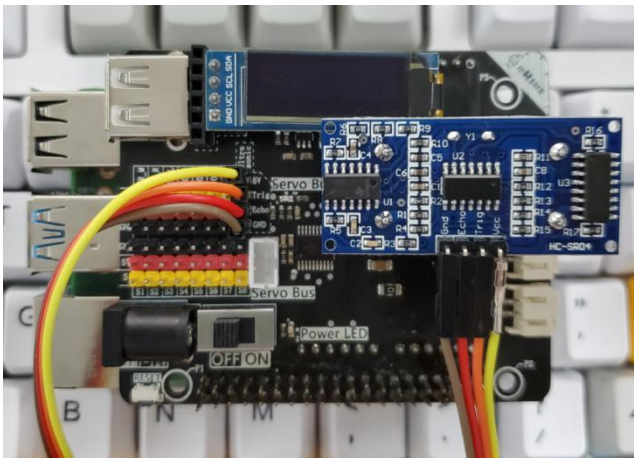
First of all, the ultrasonic wiring way is one-to-one correspondence, does not need the reverse connection, as follows chart:

1.Ultrasonic VCC—Driver plate 5V

2. Ultrasonic GND—Driver plate GND

3. Ultrasonic Trig—Driver plate Trig

4. Ultrasonic Echo—Driver plate Echo



Do not reverse connect the GND to the VCC. The reverse connection will fry the ultrasound.

The code works like this:

Shell

```
<class 'float'>
41.71cm

<class 'float'>
23.52cm

<class 'float'>
35.13cm

<class 'float'>
15.03cm

<class 'float'>
16.84cm

<class 'float'>
18.48cm
```

The ranging code is as follows, the code needs to run in the Raspberry Pi:

```
# !/usr/bin/env python
# encoding: utf-8
import RPi.GPIO as GPIO
import time

# Defines the position to which the module's pins are connected
ECHO = 4  # Ultrasonic receiving foot
TRIG = 17  # Ultrasonic leg position


# Define the initialization function
def setup():
    # Remove warning messages
    GPIO.setwarnings(False)
  # Set the pin mode to BCM mode
  GPIO.setmode(GPIO.BCM)
  # TRIG is the output
  GPIO.setup(TRIG, GPIO.OUT)
```

```
    # ECHO is the input
    GPIO.setup(ECHO, GPIO.IN)


# Ultrasonic ranging function
def distance():
    # Set to low
    GPIO.output(TRIG, 0)
    time.sleep(0.000002)

    # Pull up the Trig pin level
    GPIO.output(TRIG, 1)
    # Send high level above 10us
        time.sleep(0.000015)    # This is 15 us
        # Pull down the level
    GPIO.output(TRIG, 0)
    # Echo at low level is always empty cycle, when 1 is the beginning of the distance measurement
    while GPIO.input(ECHO) == 0:
        pass
    # Record the start time of Echo pin high level
    time1 = time.time()
    # No return of the received ultrasound has been output high level 1, also into the empty cycle
    while GPIO.input(ECHO) == 1:
        a = 1
    # Jump Out of the empty loop is the end of the distance measurement, recording the time at
this low level
    time2 = time.time()
    # Time difference
    during = time2 - time1
    # Apply the formula and keep the two decimal places
    return round(during * 340 / 2 * 100, 2)
    # The round () method returns the rounded value of the floating-point Number X


def loop():
    while True:
        # Call the distance function and assign the return value to the DIS
        dis = distance()
        print(type(dis))
        print(str(dis) + 'cm' + '\n')
        time.sleep(0.3)

# Program entry
if __name__ == "__main__":
        # Call the initialization function
            setup()
    try:
        loop()
    except KeyboardInterrupt:
        # Capture Keyboard ^ C exit program
```

```
    print("You have terminated the program and are about to exit！ ")
    # Release resources
    GPIO.cleanup()
```

5.4.3 Raspberry-X ultrasonic obstacle avoidance

Class： Ultrasonic File： xr_ultrasonic

In our code for ultrasonic obstacle avoidance, we change the value of the Peugeot bit when the ultrasonic wave reaches the value we need, and then we decide whether to issue the forward command by judging the value.

First, we define an ultrasonic avoidance start flag bit AVOIDCHANGER in xrconfig.

Next we change the mark position by judging the distance between the ultrasonic wave and the obstacle in the ultrasonic obstacle avoidance function.

```
  def avoidbyragar(self):
    """
    Ultrasonic obstacle avoidance function
    """
    dis = self.get_distance()
    if 25 < dis < 300:  # The range changes in this interval
       cfg.AVOID_CHANGER = 1
    else:
       if cfg.AVOID_CHANGER == 1:  # The goal is to make the stop only once, or else the car will
get stuck and get out of control
          go.stop()          # Call the trolley stop function
          cfg.AVOID_CHANGER = 0   # Position 0, which will not be called the second time the loop
enters
go.stop
```

As you can see here, after we added a flag bit, the ultrasonic obstacle avoidance function is more flexible, and it can be avoided in the course of our control. Its principle is, we through ultrasonic ranging to change the sign bit, and then call in the socket direction, left, right to determine the sign bit, whether to call the car direction function. In the else clause above, if CFG. Avoidchanger = = 1: Judgment is necessary because the car can not be called back without this judgment. The purpose of setting AVOIDCHANGER to 0 is to let go. The Stop () function is executed only once. Here's the way the sign bit and the front of the infrared cliff fall prevention method is the same, you can combine to understand.

In the command analysis instruction parsing function, when executing the cart direction instruction, we first judge the value of Avoidchanger, only when the value is 1 is, it wil call the motor function , when the value is 0, we can not enter the if statement; similarly, we can not add the judgment to the back up, because no matter when the state, cars are allowed to back up.

```
if buffer[0] == 0x00:  # Buffer [0] means type bit, equal to 0x00 means that this packet is motor
control instruction packet
       if buffer[1] == 0x01:  # Buffer [1] is the control bit, equal to 0x01 is the forward packet
```

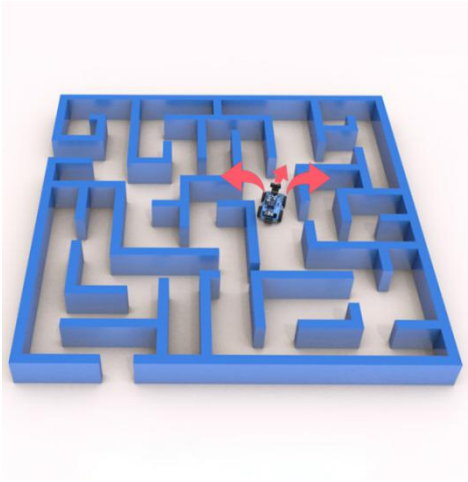
```
            if cfg.AVOID_CHANGER == 1:  # Ultrasonic flag bit, by judging the flag bit to determine
whether to call the following function
                go.forward()  # Forward

            elif buffer[1] == 0x02:
                go.back()  # Step back

            elif buffer[1] == 0x03:
                if cfg.AVOID_CHANGER == 1:  # Ultrasonic flag bit, by judging the flag bit to determine
whether to call the following function
                    cfg.LIGHT_STATUS = cfg.TURN_LEFT
                    go.left()  # Turn left

            elif buffer[1] == 0x04:
                if cfg.AVOID_CHANGER == 1:  # Ultrasonic flag bit, by judging the flag bit to determine
whether to call the following function
                    cfg.LIGHT_STATUS = cfg.TURN_RIGHT
                    go.right()  # Turn right
```

5.4.4 The Raspberry-X ultrasonic maze

Class：Ultrasonic File：xr_ultrasonic

First, let's talk about the idea.

In the ultrasonic maze, our car do not know where they are, it can only use the obstacle avoidance function to avoid obstacles, at the same time it need to keep moving. Therefore, we need to read the value of the obstacle from the three directions of the car in a loop. Then compare the size of the three values, determine which direction the car should go.



First, we'll go through the parameters we've defined. In Ultrasonic, we'll initialize the following variables:

1. MAZE_ABLE，Determine whether the front is a dead-end sign bit;；

2. MAZE_CNT，A counter to detect obstacles in front of you；

3. MAZE_TURN_TIME，The steering time used to control steering；

4. s_L，s_R Store range values on the left and right sides；

Moving on to the previous section of the maze function, the Code looks like this:

```
def maze(self):
    """
    Ultrasonic Maze Function
    """
    # First adjust the speed, to prevent too quickly and too late to react on the collision of
obstacles
    # When you need to adjust the ultrasonic speed, you can change the value here
    cfg.LEFT_SPEED = 35
    cfg.RIGHT_SPEED = 35
    # print("Ultrasonic Maze Function ")
    # Call the get () function to get the ranging value and assign it to the dis
    self.dis = self.get_distance()
    # When there are no obstacles ahead and it's not a dead end
    if self. MAZE_ABLE == 0 and ((self.dis > 25) or self.dis == 0):
        # The forward range value is greater than 25(0 or 0 is an obstacle not detected, out of
range)
        while ((self.dis > 25) or self.dis == 0) and cfg.CRUISING_FLAG:
            self.dis = self.get_distance()
            go.forward()
        # The if here is the same level as the while before
        if cfg.CRUISING_FLAG:
            self.MAZE_CNT = self.MAZE_CNT+1
            print(self.MAZE_CNT)
            go.stop()
            time.sleep(0.05)
            go.back()      # Back up a little bit
            time.sleep(0.15)
            go.stop()
            time.sleep(0.05)
            if self.MAZE_CNT > 3:      # Check whether the front is an obstacle several times to avoid
false detection
                self.MAZE_CNT = 0
                self.MAZE_ABLE = 1     # If there's an alley up ahead
```

Into the maze function, we first reduce the speed, if we go too fast in the ultrasonic maze mode, because of the inertia of the vehicle and the procedural delay, it can lead to ultrasonic detection of obstacles before the procedure has hit obstacles; therefore, the customer can use their actual environment to change the right and left motor speed value.

Then we judge if MAZE_ABLE is equal to 0 and dis detection distance is greater than 25. Here is a characteristic: dis is equal to 0. When the distance of obstacle exceeds the detection range of ultrasonic, the return value of ultrasonic will be 0 .When you run the maze function for the first time, the default value for the maze is 0, so if the distance from the obstacle ahead is greater than 25, you must enter the if statement, otherwise you will enter the else statement below. Let's start with the meaning of the if statement.

After entering the if statement, we use the while loop statement. If the range is greater than 25 and we don't switch modes, we go into the while loop. If we go into the while loop, we update the

dis value first to see if the while loop jumps out, the main function of the while loop statement is to drive the cart forward without obstacles in front. The loop condition for jumping out of while is dis < 25 or CRUISING_FLA=0, which makes CRUISING_FLA=0 when the user switches manual mode, and when changing other modes because of the CRUISING_FLA thread, it will enter different mode functions based on the values of CRUISING, we'll come back to that.

After jumping out of the while loop, you go into the second if to determine if cfg. Cruising_FLAG. The purpose of CRUISING_FLAG is to immediately exit the maze when the user switches mode, because if the process does not include this judgment, then when the user switches to normal mode there is no interruption in the judgment, the program will continue to run until the end of the process, creating a bad experience. Jumping out of the while loop means that an obstacle is detected ahead, that the MAZE_CNT counter adds one and then retreats. If the car is moving forward, it will jump out of the while loop at a distance of less than 25. This will cause the car to check several times to make sure that there is an obstacle in front of it. The MAZE_CNT>3 statement, after entering, first sets the MAZE_CNT to 0, then sets the MAZE_ABLE to 1, then the first if result must be false, hence the else statement, so let's move on to the else statement.

```
else:
    go.stop()
    self.s_L = 0
    self.s_R = 0
    time.sleep(0.1)
    # Turn the ultrasonic steering gear to the right
    servo.set(7, 5)
    # Judge the mode switch flag in order to be able to immediately interrupt maze mode
    if cfg.CRUISING_FLAG:
        time.sleep(0.25)
    # Store the right ranging value
    self.s_R = self.get_distance()
    if cfg.CRUISING_FLAG:
        time.sleep(0.2)

    # And turn the steering wheel to the left
    servo.set(7, 175)
    if cfg.CRUISING_FLAG:
        time.sleep(0.3)
    # Store the left ranging value
    self.s_L = self.get_distance()
    if cfg.CRUISING_FLAG:
        time.sleep(0.2)
        # And put the steering wheel in the middle
            servo.set(7, 80)
    time.sleep(0.1)
    # There is no obstruction or on the right (right distance is greater than left distance and right
distance is greater than 20)
    if (self.s_R == 0) or (self.s_R > self.s_L and self.s_R > 20):
        self.MAZE_ABLE = 0
        cfg.LEFT_SPEED = 99  # Steering Speed, if you need to manually adjust its speed to meet the
steering force in different ground, here is the speed on the carpet needs to be highe
```

```
        cfg.RIGHT_SPEED = 99
        go.right()
        if cfg.CRUISING_FLAG:
            time.sleep(cfg.MAZE_TURN_TIME / 1000)  # The time of turning, according to the above
speed adjustment, measured to about 90 degrees can be       cfg.LEFT_SPEED = 45
        cfg.RIGHT_SPEED = 45
    # No obstruction or on the left side (left distance greater than right distance and left distance
greater than 20)
        elif (self.s_L == 0) or (self.s_R < self.s_L and self.s_L > 20):
            # Set it 0, you can go back to the topmost if statement
        self.MAZE_ABLE = 0
        cfg.LEFT_SPEED = 99  # Steering Speed, if you need to manually adjust its speed to meet the
steering force in different ground, here is the speed on the carpet needs to be higher
        cfg.RIGHT_SPEED = 99
        go.left()
        if cfg.CRUISING_FLAG:
            time.sleep(cfg.MAZE_TURN_TIME / 1000)  # The time of turning, according to the above
speed adjustment, measured to about 90 degrees can be       cfg.LEFT_SPEED = 45
        cfg.RIGHT_SPEED = 45

        else:  # The road ahead is impassable, left and right are impassable, it is a dead end, can only
take the same road to go back
            # Put the sign 1, to avoid repeated into a dead end, only a little bit back then left and
right to detect whether there are other channels, when the left and right channels either side of
the Sign Position 0, can go forward       self.MAZE_ABLE = 1
        go.back()
        if cfg.CRUISING_FLAG:
            time.sleep(0.3)

    go.stop()
    time.sleep(0.1)
```

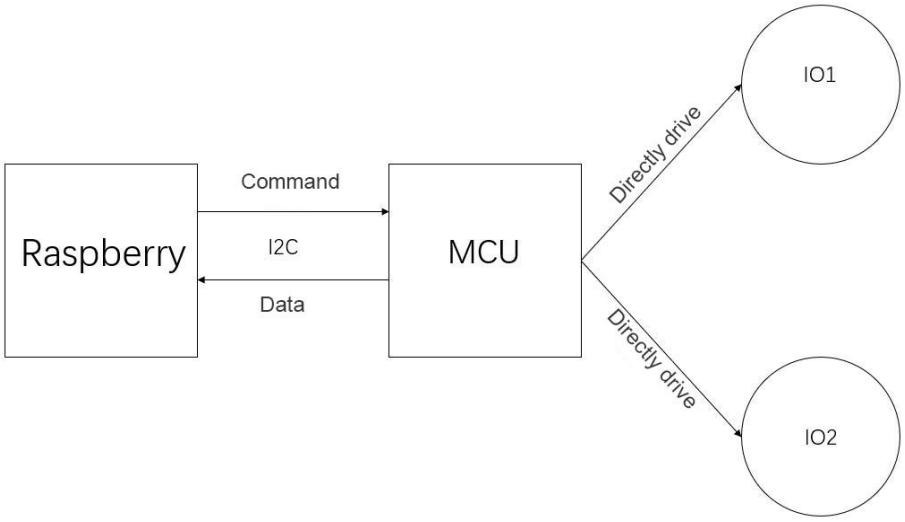
In the else statement, we first get the ultrasonic on the left and right side of the distance measurement value, by comparing the distance measurement value to determine how the car moves. Call Servo. The set function turns the actuator, and then the get_distance function is used to get the ranging value and assign it to s_L and s_R. Then compare the size of s_L and s_R to determine how to proceed to the car.

5.5 MCU-Co-processors

MCU (Microcontroller Unit) is also known as a microcontroller or microcomputer, on the DS-X we can call it a co-processor.

The main part of the DS-X is developed by the Raspberry Pi, but as the Raspberry Pi hardware itself cannot achieve a stable PWM signal, we have added an MCU to the upper driver board to assist the Raspberry Pi in controlling the car.

The MCU is always in working condition, reading certain sensor values while waiting for the CPU to give it instructions. If the CPU sends a pre-defined instruction, then it will return the result according to the pre-defined action of the program or directly drive the IO and other trigger events.



5.5.1 Communication Protocol between Raspberry Pi and MCU

The communication protocol between the Raspberry Pi and the MCU uses I2C communication, the communication protocol is as follows.

I2C communication protocol design

Function		Packet header	Type	Control	Data	End of header	Remarks (NC means arbitrary value, XX means set value)
8-way servo interface	SERV01	FF	0*01	0*01	XX	FF	Set the No. 1 angle of the servo, such as 30 degrees (FF01011EFF)
	SERV02	FF		0*02	XX	FF	
	SERV03	FF		0*03	XX	FF	
	SERV04	FF		0*04	XX	FF	
	SERV05	FF		0*05	XX	FF	
	SERV06	FF		0*06	XX	FF	
	SERV07	FF		0*07	XX	FF	
	SERV08	FF		0*08	XX	FF	
Servo function	Servo reset	FF	0*32	XX	XX	FF	Servo angle homing
	Servo memory	FF	0*33	XX	XX	FF	Servo angle memory
RGB light bar	Set the number of light groups 1RGB	FF	0*02	XX	CC	FF	Data bit XX represents the number of RGB (1-8) CC color value 0-8 (black, red orange yellow green cyan blue purple, white)
	Set the 2RGB quantity of the lamp group	FF	0*03	XX	CC	FF	Data bit XX represents the number of RGB (1-8) CC color value 0-8 (black, red orange yellow green cyan blue purple, white)
	Set light group 1 single RGB display color	FF	0*03	XX	CC	FF	The control bit XX indicates the position of the RGB color light (1-8), and the CC color value is 0-8 (black, red orange, yellow, green, blue, purple, and white)
	Set light group 2 single RGB display color	FF	0*04	XX	CC	FF	The control bit XX indicates the position of the RGB color light (1-8), and the CC color value is 0-8 (black, red orange, yellow, green, blue, purple, and white)
Battery detection	Read	address	0-05				If the reading voltage is 77 (decimal), it means the battery voltage is 7.7V

5.5.2 I2C communication: Python smbus function description

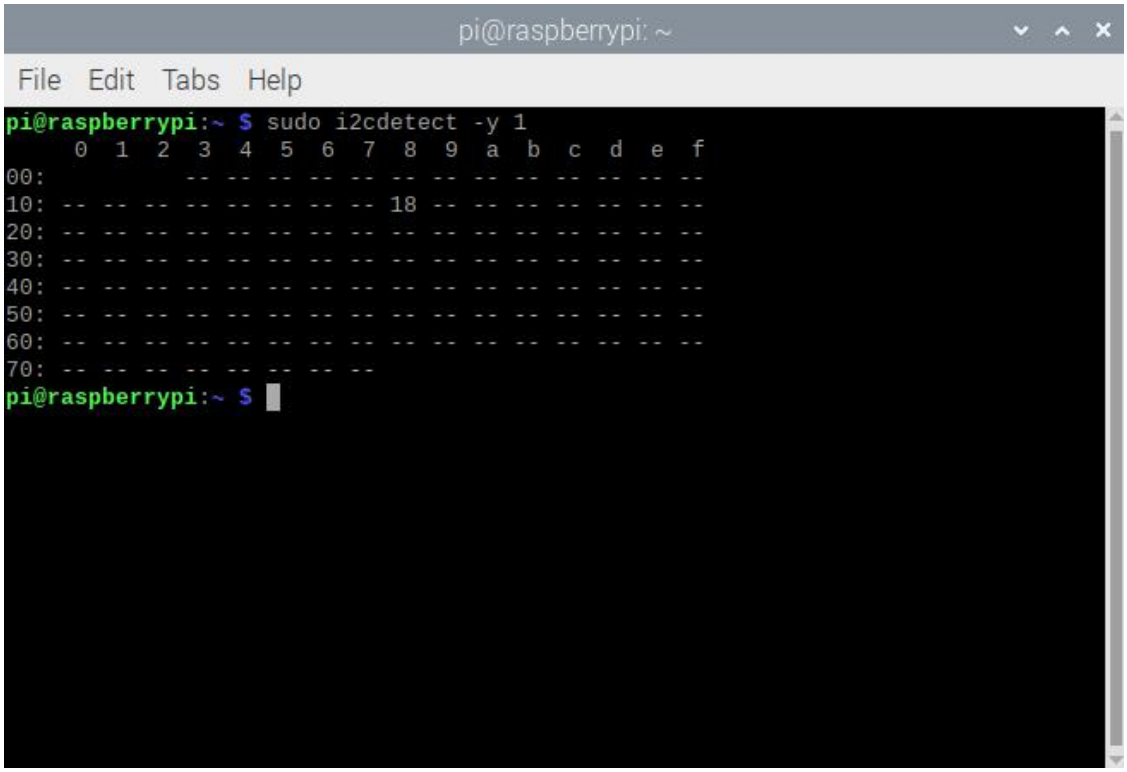
Before I go into the main text, I would like to remind you that if you are using our factory firmware, the smbus library and I2C tools are already configured, so there is no need to reinstall them, just do the following code test.

With the Raspberry Pi online, install the i2c-tools tool and python-smbus instantly by typing the following command into the command terminal line.

```
1. sudo apt-get update
2. sudo apt-get install i2c-tools python-smbus
```

After inserting the I2C device we can check whether the I2C address has been successfully read at the terminal command line with the following command:

```
sudo i2cdetect -y 1
```



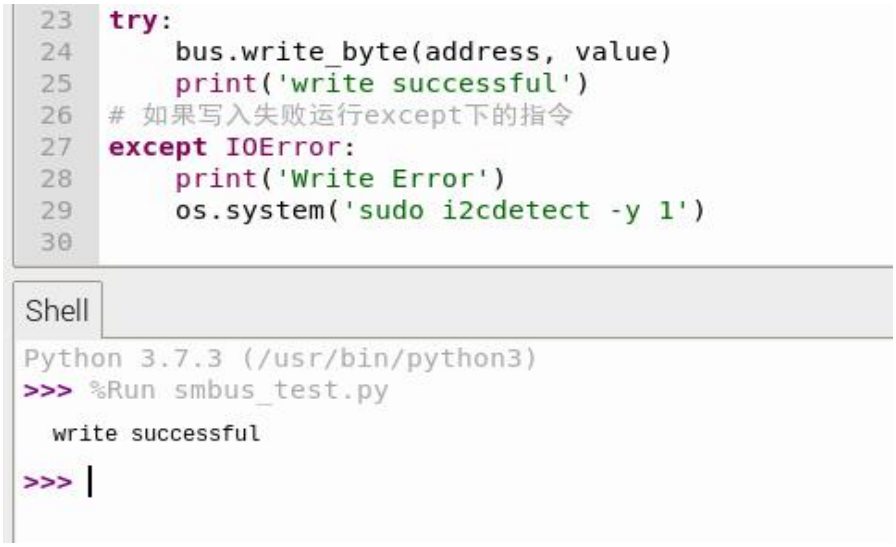
Here we can see that our I2C device address is 0x18, here I write a command to the I2C device to see if it can be written in to illustrate the use of I2C communication smbus

Create the following code and RUN in Thonny:

```
# !/usr/bin/env python
# encoding: utf-8

# Packet guide operation
import smbus
import os
# Creating smbus instances
bus = smbus.SMBus(1)  # 0 represent/dev/i2c0  1 represent/dev/i2c1
# I2C communication address
address = 0x18
#The value that need to write in
value = 123
#  try to write in first
```

```
try:
  bus.write_byte(address, value)
  print('write successful')
# If write error then run the command which gave by except
except IOError:
  print('Write Error')
  os.system('sudo i2cdetect -y 1')
```

You can see that the writing was successful:



The smbus communication is as mentioned above. As for the communication protocol, it is drawn up by ourselves. We only need to draw it up in the MCU in advance, if the upper layer sends a command and the MCU receives this command to do something, it is considered a successful communication protocol. Of course, the command '123' in the experiment is only used for writing test.

For detailed smbus usage instructions, please refer to the following link:

https://blog.csdn.net/Jacob_zhj/article/details/90518363

5.6 Passive buzzer

The buzzer is divided into active and passive; the active buzzer has an oscillating circuit inside, which buzzes as soon as the power is turned on, so it can be sounded at a high level just like a diode LED, and programming is more convenient than passive; There is no oscillating source inside the passive buzzer, and the DC signal cannot make it sound. It must be driven by an oscillating current, and the sound frequency is controllable, and different sound effects can be made.

5.6.1 Passive buzzer music production

The principle of music production is to control the sound frequency of the passive buzzer and the sound time of each frequency according to the music notation and beat time. First we simulate the seven note frequencies of music:

```
#    0  1  2  3   4  5  6  7
```

```
test = [1, 248, 294, 330, 349, 392, 440, 494]
```

Create a frequency list of songs according to the numbered musical notation, and let the buzzer sound according to the frequency in the list through the pause time of the beat:

```
# Song little star
song_0 = [test[1], test[1], test[5], test[5], test[6], test[6], test[5], test[0],
      test[4], test[4], test[3], test[3], test[2], test[2], test[1], test[0],
      test[5], test[5], test[4], test[4], test[3], test[3], test[2], test[0],
      test[5], test[5], test[4], test[4], test[3], test[3], test[2], test[0]]
```

The next step is to control the beat, control the duration of each audio sound through the beat, and then switch to the next audio sound; the last digital sound beat is (2/8) seconds, and the sound frequency is 1, It's equivalent to the pause time after singing a lyrics.

```
beta_0 = [4, 4, 4, 4, 4, 4, 4, 8,  # Song little star beat
      4, 4, 4, 4, 4, 4, 4, 8,
      4, 4, 4, 4, 4, 4, 4, 8,
      4, 4, 4, 4, 4, 4, 4, 8]
```

Cause I don't know much about music, so I don't understand. The general idea is to modify the frequency of the passive buzzer to achieve 7 audio frequencies for bass, midrange, and treble, for a total of 21 audio (of course, a song may be not use all audio), I stole a lazy here, and only used one tune frequency, but the idea is like this, you can try it by yourself to know the principle.

Next we look at the complete code:

```
# !/usr/bin/env python
# encoding: utf-8


import RPi.GPIO as GPIO
import time

Buzzer = 10
# 1155665
# 4433221
# 5544332
# 5544332
#     0   1   2  3    4   5   6   7
test = [1, 248, 294, 330, 349, 392, 440, 494]

song_0 = [test[1], test[1], test[5], test[5], test[6], test[6], test[5], test[0],  # song little star
      test[4], test[4], test[3], test[3], test[2], test[2], test[1], test[0],
      test[5], test[5], test[4], test[4], test[3], test[3], test[2], test[0],
      test[5], test[5], test[4], test[4], test[3], test[3], test[2], test[0]]

beta_0 = [4, 4, 4, 4, 4, 4, 4, 8,  # song little star beat
      4, 4, 4, 4, 4, 4, 4, 8,
      4, 4, 4, 4, 4, 4, 4, 8,
      4, 4, 4, 4, 4, 4, 4, 8]

song_1 = [test[1], test[2], test[3], test[1], test[0],  # Song two tiger
      test[1], test[2], test[3], test[1], test[0],
```

```
        test[3], test[4], test[5], test[0],
        test[3], test[4], test[5], test[0],
        test[5], test[6], test[5], test[4], test[3], test[1], test[0],
        test[5], test[6], test[5], test[4], test[3], test[1], test[0],
        test[1], test[5], test[1], test[0],
        test[1], test[5], test[1], test[0]]

beta_1 = [4, 4, 4, 4, 8,  # # song two tiger
        4, 4, 4, 4, 8,
        4, 4, 4, 8,
        4, 4, 4, 8,
        6, 6, 6, 6, 4, 4, 8,
        6, 6, 6, 6, 4, 4, 8,
        4, 4, 4, 8,
        4, 4, 4, 8]


def setup():
    # remove warning message
    GPIO.setwarnings(False)
    # set the pin mode as BCM mode
    GPIO.setmode(GPIO.BCM)
    # buzzer is output mode
    GPIO.setup(Buzzer, GPIO.OUT)
    # Specify a global variable to replace GPIO.PWM
    global Buzz
    # Initial frequency 440
    Buzz = GPIO.PWM(Buzzer, 440)
    # Start the buzzer pin, the duty cycle is 50%
    Buzz.start(50)


def loop():
    while True:
        print('\n    Play the first song...')
        for i in range(0, len(song_0)):
            # Change the frequency follow the notes of the song
            Buzz.ChangeFrequency(song_0[i])
            # Delay beat
            time.sleep(2 / beta_0[i])
        time.sleep(2)

        print('\n    Play the second song...')
        for i in range(1, len(song_1)):
            #Change the frequency follow the notes of the song
            Buzz.ChangeFrequency(song_1[i])
            # Delay beat
            time.sleep(2 / beta_1[i])
        # Wait four seconds to play the next song
        time.sleep(4)
```

```
if __name__ == '__main__':
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        # Capture keyboard^C to exit the program
        print('You terminated the program\nEnd of program！ ')
        GPIO.cleanup()
```

Run the above code in the Raspberry Pi, you can hear the buzzer sound!

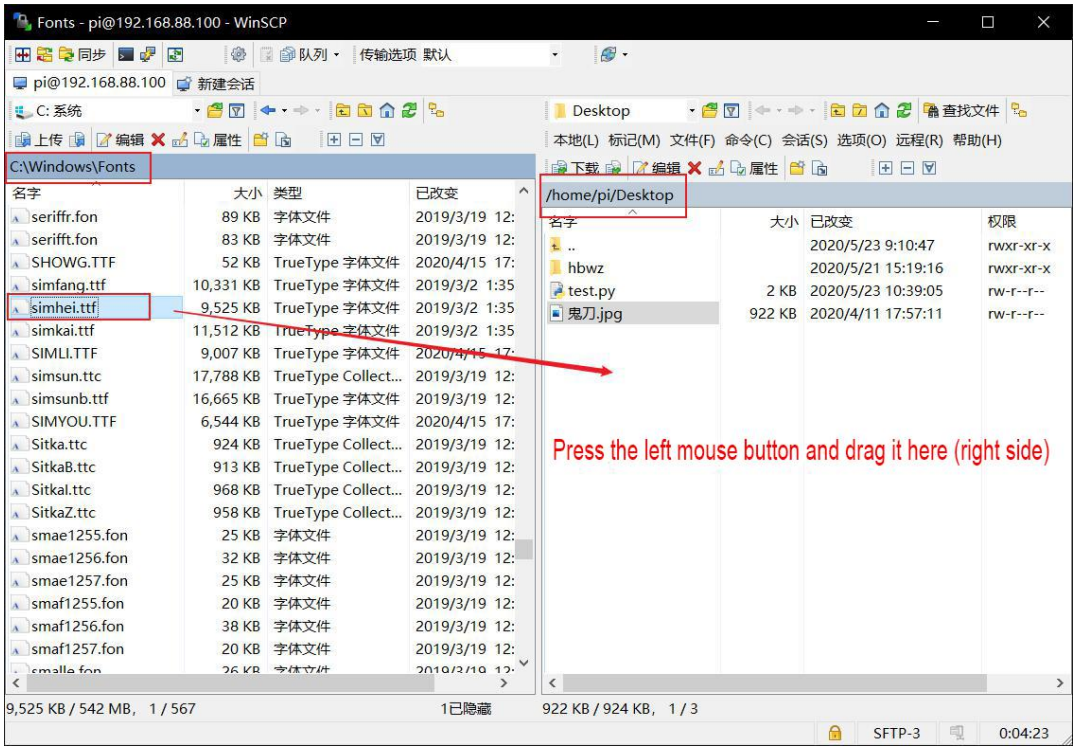
5.7 0.91 OLED display screen

The OLED screen will not light up after we connect the line. This is the difference from the LCD. It will only display when our code drives it to display; you need to remember, the screen will not light up after the line is connected, it is not the screen it's broken.

5.7.1 Control the OLED screen to display words

We make a small case to teach you how to use the OLED screen to display the words that you want; if you want to display Chinese on the screen, you need to prepare a Chinese font. I use Microsoft Yahei font here. First, find a copy of Microsoft Yahei. Black font, I directly use the Microsoft Yahei font that comes with Windows (path: simhei.ttf under C:\Windows\Fonts).

After finding this font, we use Winscp to drag it to the desktop of the Raspberry Pi (/home/pi/Desktop)



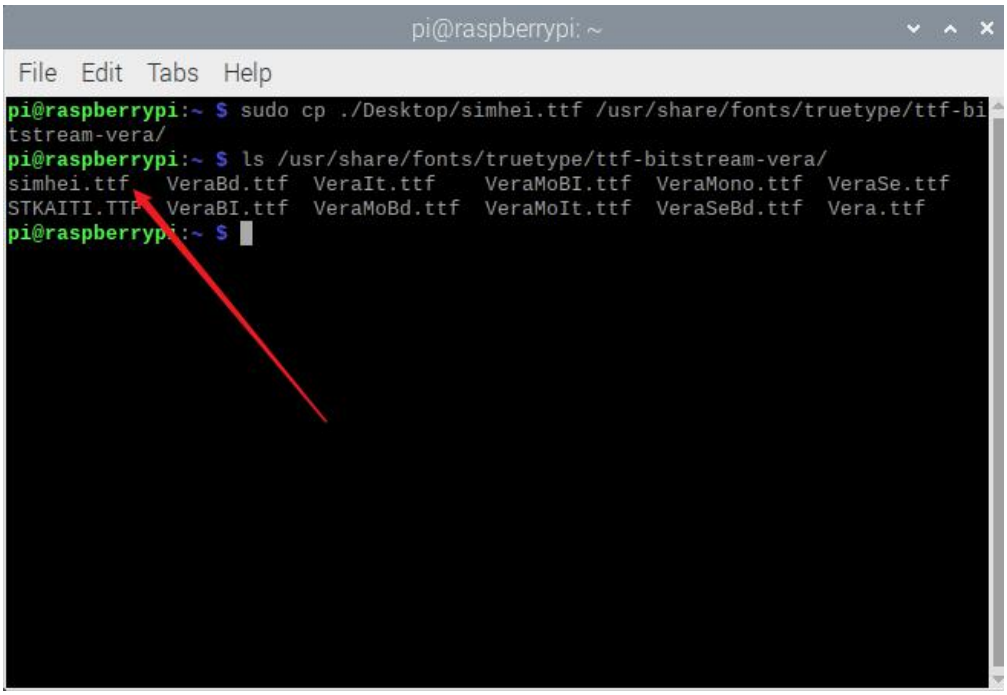
After dragging it to the desktop, we use the terminal command cp to copy to the /usr/share/fonts/truetype/ttf-bitstream-vera/ folder of the Raspberry Pi. you need to use sudo

permission, otherwise it will be rejected。

```
sudo cp ./Desktop/simhei.ttf /usr/share/fonts/truetype/ttf-bitstream-vera/
```

Then we can use the ls command to see that it is indeed copied.

```
ls /usr/share/fonts/truetype/ttf-bitstream-vera/
```



Experiment content: We let the OLED screen refresh and display the temperature of the Raspberry Pi in real time.

First look at the photos after success：

First， let's look at the initial use of the OLED screen:

In the first step, we need to introduce the modules that we need. Here we use the Adafruit_SSD1306 library file for the OLED screen. This library file needs to be installed first, and then

enter the following command in the Raspberry Pi terminal (Internet status is required)：

```
sudo pip install Adafruit-SSD1306
```

Operation code：

```
# Import related package files
import time
import os
import subprocess
import Adafruit_SSD1306

from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont
```

Step 2： Obtain an instance of OLED

```
disp = Adafruit_SSD1306.SSD1306_128_32(rst=None, i2c_bus=1, gpio=1)
```

Step 3： Initialize and clear the screen

```
disp.begin()
disp.clear()
disp.display()
```

Step 4th， Creating a new picture, the picture size is oled size

```
width = disp.width
height = disp.height
image = Image.new('1', (width, height))
```

Sep 5th， Loading the picture on the drawing object, which is equivalent to loading on the drawing board

```
draw = ImageDraw.Draw(image)
```

Step 6th : Select the font

```
# The default font of the library is in ImageFont
default_font = ImageFont.load_default()
# The font library in the Raspberry Pi, you can set the font size,
font = ImageFont.truetype('/usr/share/fonts/truetype/ttf-bitstream-vera/simhei.ttf', 35)
```

Write a function to get the Raspberry Pi CPU temperature

```
# Get the temperature of the Raspberry Pi
def cpu_temp():
  # Raspberry Pi CPU temperature is stored in this file, open the file   tempFile =
open('/sys/class/thermal/thermal_zone0/temp')
  # Read the file
  cputemp = tempFile.read()
  # Close the file
  tempFile.close()
  # Rounded to reserve integer
      temp = round(float(cputemp) / 1000, 1)
      # Function returns temperature value
  return str(temp)
```

Finally, we can refresh the display data in a loop. The first six steps are fixed; post the complete code, you can take a look:

```
import time
import os
import subprocess
import Adafruit_SSD1306

from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont


# Get an instance of OLED
disp = Adafruit_SSD1306.SSD1306_128_32(rst=None, i2c_bus=1, gpio=1)

# Initialize, clear screen
disp.begin()
disp.clear()
disp.display()

# Create a new picture, the picture size is oled size
width = disp.width
height = disp.height
image = Image.new('1', (width, height))

# Loading the picture on the drawing object, which is equivalent to loading on the drawing board

draw = ImageDraw.Draw(image)

# Draw a black filled box to clear the image
draw.rectangle((0, 0, width, height), outline=0, fill=0)

# font select
# The default font of the library is in ImageFont
default_font = ImageFont.load_default()
# you can set the font size in the font library of the Raspberry Pi
font = ImageFont.truetype('/usr/share/fonts/truetype/ttf-bitstream-vera/simhei.ttf', 15)


# Get the temperature of the Raspberry Pi
def cpu_temp():
   # Raspberry Pi CPU temperature is stored in this file, open the file
   tempFile = open('/sys/class/thermal/thermal_zone0/temp')
   # Read the file
   cputemp = tempFile.read()
   # Close the file
   tempFile.close()
   # Rounded to reserve integer
   tem = round(float(cputemp) / 1000, 1)
   return str(tem)

while True:
```

```
    temp = cpu_temp()
    # Draw a black filled box to clear the image
    draw.rectangle((0, 0, width, height), outline=0, fill=0)
    # To draw text, the two parameters of text() are the distance between the X and Y axes of the
screen, and the upper left corner of the screen is the 0 point of the XY axis
    draw.text((5, 0), "Current Temperature: " + temp, font=font, fill=255)
    draw.text((50, 16), "by_XiaoR Geek", font=font, fill=255)

    # displayed on oled
    disp.image(image)
    time.sleep(1)
    disp.display()
```

Run the above example and you can see the temperature of the Raspberry Pi displayed on the OLED screen.

5.7.2 Raspberry-X series OLED application

Class：Oled File：xr_oled.py

In xr_oled.py, we created the Oled class, and created 5 functions in the class, the names and functions of which are as follows:

Name	Function	Return value
get_network_interface_state	Get the network state of the network port, internal call	up/down
get_ip_address	Get network IP address, internal call	IP
draw_row_column	Display position, internal call	nothing
disp_default	Default display information	nothing
disp_crusing_mode	Information that needs to be displayed when entering a certain control mode	nothing

Let's look at the specific code:：

```
import time
import os
import subprocess
import Adafruit_SSD1306
import xr_config as cfg

from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont
from xr_ultrasonic import Ultrasonic
ultrasonic = Ultrasonic()


class Oled(object):
    # Each time Oled is called, __init__ will be initialized
    def __init__(self):
```

```
        # Get an instance of OLED
        self.disp = Adafruit_SSD1306.SSD1306_128_32(rst=None, i2c_bus=1, gpio=1)
        # Initialize, clear screen
        self.disp.begin()
        self.disp.clear()
        self.disp.display()

        # Create a new picture, the picture size is oled size       self.width = self.disp.width
        self.height = self.disp.height
        self.image = Image.new('1', (self.width, self.height))

        # Loading the picture on the drawing object, which is equivalent to loading on the drawing
board


        self.draw = ImageDraw.Draw(self.image)

        # Draw a black filled box to clear the image
        self.draw.rectangle((0, 0, self.width, self.height), outline=0, fill=0)

        # Draw a black filled box to clear the image.
        self.draw.rectangle((0, 0, self.width, self.height), outline=0, fill=0
        # font select
                # The default font of the library is in ImageFont
        self.font = ImageFont.load_default()
        # you can set the font size in the font library of the Raspberry Pi
        # font = ImageFont.truetype('/usr/share/fonts/truetype/ttf-bitstream-vera/STSONG.TTF', 14)
        self.font1 = ImageFont.truetype('/usr/share/fonts/truetype/ttf-bitstream-vera/simhei.ttf', 14)
        pass

    def get_network_interface_state(self, interface):
        """
        Get the network status of the network port, if it is the network status, it returns up, otherwise
it returns down
        """
        return subprocess.check_output('cat /sys/class/net/%s/operstate' % interface,
shell=True).decode('ascii')[:-1]

    def get_ip_address(self, interface):
        """
        Get the internet IP address
        """
        if self.get_network_interface_state(interface) == 'down':  #Determine whether to connect to
the Internet
            return None
        # Defines a cmd command
        cmd = "ifconfig %s | grep -Eo 'inet (addr:)?([0-9]*\.){3}[0-9]*' | grep -Eo '([0-9]*\.){3}[0-9]*' |
grep -v '127.0.0.1'" % interface  # Match the ip information output by the corresponding network
card
        return subprocess.check_output(cmd, shell=True).decode('ascii')[:-1]
```

```
    def draw_row_column(self, row, column, strs):
        """
        Row display, row represents the row number, column represents the column number, strs
represents the string to be displayed
        """
        if row == 1:
            self.draw.text((column, self.top), strs, font=self.font, fill=255)
        elif row == 2:
            self.draw.text((column, self.top + 8), strs, font=self.font, fill=255)
        elif row == 3:
            self.draw.text((column, self.top + 16), strs, font=self.font, fill=255)
        elif row == 4:
            self.draw.text((column, self.top + 25), strs, font=self.font, fill=255)

    def disp_default(self):
        """
        Basic information displayed after startup,
         The first line shows the ip of the wired network port
         The second line shows the wireless network port ip
         The third line shows memory usage information and usage rate
         The fourth line shows the SD card storage information and usage rate
        """
        # Draw a black filled box to clear the image
        self.draw.rectangle((0, 0, self.width, self.height), outline=0, fill=0)
        # Defines a cmd command
        cmd = "free -m | awk 'NR==2{printf \"Mem: %s/%sMB %.1f%%\", $3,$2,$3*100/$2 }'"
        MemUsage = subprocess.check_output(cmd, shell=True)
        # Defines a cmd command
        cmd = "df -h | awk '$NF==\"/\"{printf \"Disk: %d/%dGB %s\", $3,$2,$5}'"
        Disk = subprocess.check_output(cmd, shell=True)

        # display the following information
        self.draw_row_column(1, 0, "eth0: " + str(self.get_ip_address('eth0')))  # Wired network card
IP
        self.draw_row_column(2, 0, "wlan0: " + str(self.get_ip_address('wlan0')))  # Wireless network
card IP
        self.draw_row_column(3, 0, str(MemUsage.decode('utf-8')))  #  memory usage
        self.draw_row_column(4, 0, str(Disk.decode('utf-8')))  # Disk information

        # display image
        self.disp.image(self.image)
        self.disp.display()
        time.sleep(0.1)

    def disp_cruising_mode(self):
        """
        Display mode after entering the control function
        :return:none
        """
        self.draw.rectangle((0, 0, self.width, self.height), outline=0, fill=0)
```

```
    dispmod = cfg.OLED_DISP_MOD[cfg.CRUISING_FLAG]  # Display the corresponding mode according to the selected mode
    dispmodlength = len(dispmod) * cfg.OLED_DISP_MOD_SIZE  # Gets the string length
    positionmod = (128 - dispmodlength) / 2 - 1  # The starting position of the character display
    # print(cfg.CRUISING_FLAG -1)
    # print(cfg.OLED_DISP_MOD[cfg.CRUISING_FLAG -1])
    self.draw.text((0, -2), cfg.LOGO, font=self.font, fill=255)  # Display LOGO information
    self.draw.text((0, 8), "Dis:" + str(cfg.DISTANCE) + "cm", font=self.font, fill=255)  # Display distance value
    self.draw.text((positionmod, 17), dispmod, font=self.font1, fill=255)  # display mode

    self.draw.line((0, 8, 128, 8), fill=255)  # Horizontal line

    # draw battery frame

    m = 3  # Battery power level
    n = 3  # Pixel unit occupied by each gear of the battery
    batlength = m * n + 2 + 2 + 2  # m*n represents the pixels occupied by the battery cell, the first 2 represents the sum value of the battery frame and the end of the battery cell, the second 2 represents the sum value of the pixels at the beginning and end of the battery frame, and the third 2 represents the pixel value of the battery head
    x = 128 - batlength - 1  # Starting position on the left side of the battery box
    y = 0  # The top starting position of the battery frame

    # self.draw.text((x+23, -2), str(cfg.POWER*10) +"%", font=self.font, fill=255)
    # Draw battery frames
    self.draw.line((x, y + 2, x + 2, y + 2), fill=255)
    self.draw.line((x + 2, y + 2, x + 2, y), fill=255)
    self.draw.line((x + 2, y, x + batlength, y), fill=255)
    self.draw.line((x + batlength, y, x + batlength, y + 5), fill=255)
    self.draw.line((x + batlength, y + 5, x + 2, y + 5), fill=255)
    self.draw.line((x + 2, y + 5, x + 2, y + 3), fill=255)
    self.draw.line((x + 2, y + 3, x, y + 3), fill=255)
    self.draw.line((x, y + 3, x, y + 2), fill=255)
    # Calculate battery power level
    level = cfg.POWER
    #  clean battery power
    self.draw.line((x + 3, y + 2, x + batlength - 2, y + 2), fill=0)
    self.draw.line((x + 3, y + 3, x + batlength - 2, y + 3), fill=0)
    # re-draw battery power
    self.draw.line((x + batlength - 2 - level * n, y + 2, x + batlength - 2, y + 2), fill=255)
    self.draw.line((x + batlength - 2 - level * n, y + 3, x + batlength - 2, y + 3), fill=255)
    # Display image information on oled
    self.disp.image(self.image)
    self.disp.display()
    time.sleep(0.05)
```

The comments are quite detailed. The process is the same as the above to get the CPU temperature. The main thing is what you need to display on the screen and how to reflect it needs to be written by yourself.

Cmd explain again here:

```
import subprocess
def get_ip_address(interface):
    """
    Get network ip address
    """
    # Define a cmd command
    cmd = "ifconfig %s | grep -Eo 'inet (addr:)?([0-9]*\.){3}[0-9]*' | grep -Eo '([0-9]*\.){3}[0-9]*' | grep -v '127.0.0.1'" % interface  # Match the ip information output by the corresponding network card
    # The parameter cmd has a return value
    return subprocess.check_output(cmd, shell=True).decode('ascii')[:-1]

print(get_ip_address('wlan0'))
print(get_ip_address('eth0'))
```

cmd is actually equivalent to a command string, similar to the cmd command on our computer to enter a specific command, the window will return a specific value.

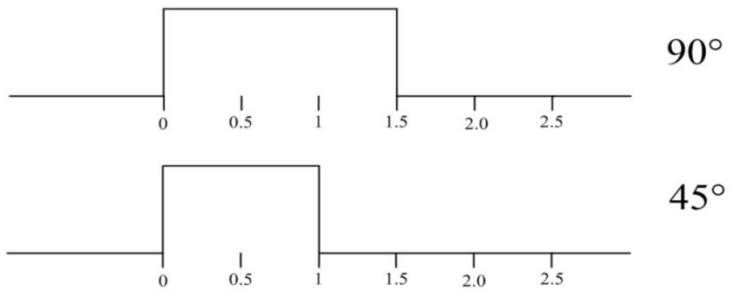
5.8 Servo

The servo motor is usually called servo, which is a small device with an output shaft. When we send a control signal to the server, the output shaft can be turned to a specific position. As long as the control signal remains unchanged, the servo mechanism will keep the angular position of the shaft unchanged. If the control signal changes, the position of the output shaft will change accordingly. In daily life, servo are often used in remote control aircraft, remote control cars, robots and other fields.

The servo motor (steering gear) has 3 input pins, GND, VCC and Signal; Signal is used to transmit angle control signals. This angle is determined by the duration of the control signal pulse, which is called pulse code modulation (PCM).

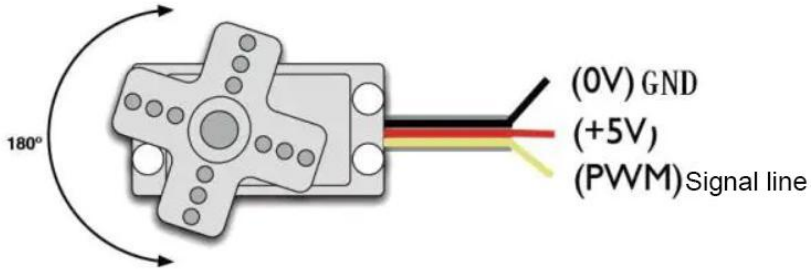
The control of the servo generally requires a time base pulse of about 20ms. The high-level part of the pulse is generally in the range of 0.5ms-2.5ms, with a total interval of 2ms; the pulse width will determine the distance of the motor rotation; for example: 1.5 milliseconds Pulse, the motor will turn to a position of 90 degrees (usually called the neutral position, for a 180° servo, it is the 90° position). If the pulse width is less than 1.5 milliseconds, then the motor axis faces 0 degrees. If the pulse width is greater than 1.5 milliseconds, the axial direction is 180 degrees. Taking the 180 degree servo as an example, the corresponding control relationship is as follows:

High level duration	Angle
0.5ms	0°
1.0ms	45°
1.5ms	90°
2.0ms	135°
2.5ms	180°



The duty cycle refers to the ratio of the power-on time to the total time in a pulse cycle.

External wiring: the servo generally has three external wires, which are distinguished by three colors of brown, red, and orange. The colors will be different due to different brands. The brown is the ground wire, the red is the power positive wire, and the yellow is the signal wire.



5.8.1 Raspberry Pi GPIO control servo

We have already talked about the principle above, the actual operation code, the code is as follows, the servo signal line is connected to pin 7 of Raspberry Pi BOARD mode

```
import RPi.GPIO as GPIO
import time

# GPIO port number, modify according to actual situation
servo = 7
a = 10
b = 2


def setup():
    global pwm
    # Remove warning message
    GPIO.setwarnings(False)
    # Set the pin mode to BOARD mode
    GPIO.setmode(GPIO.BOARD)
    # servo is output mode
    GPIO.setup(servo, GPIO.OUT)

setup()
while True:
```

```
for i in range(500, 2501, 10):
    GPIO.output(servo, 1)
    time.sleep(i/1000000)
    GPIO.output(servo, 0)
    time.sleep((20000 - i)/1000000)
```

Run the above code, the servo will turn up, try it.

5.8.2 Raspberry Pi drives MCU to realize PWM servo control

Similarly, in raspberry-x series, we don't need to use raspberry pi to produce PWM to drive the servo. We only need to send the servo number and angle to MCU to operate the servo.

We can know the communication protocol of control servo from MCU communication protocol which is in section 5.4.2. The protocol is as follows:

```
#packet header   function   servo number  Servo angle  end of packet
[0Xff, 0x01, 0x01, 0x1e, 0xff]
```

The above example protocol shows that except for the header and the end of the packet:

The first data bit: represents the function, and the servo function is 01, which is different from other functions (such as RGB);

The second data bit: represents the servo number. Raspberry-x series totally use 8 servo, so the value is 01-08;

The third data bit: represents the servo angle, for example: I2C writes the list value as [0xff, 1, 2, 45, 0xff], which represents the position that I need to control the servo 2 to rotate to 45 ° angle, and 1 represents the servo function.

After we know the communication protocol, let's actually operate how to control the servo through I2C. We use thorny to create the following code:

```
# !/usr/bin/env python
# encoding: utf-8

# import package operation
import smbus
import time
import os

# i2c communication address
address = 0x18
# Create instance
bus = smbus.SMBus(1)

# values = [packet header   function   servo number  Servo angle  end of packet]
values = [0xff, 1, 1, 0, 0xff]
# Gets the list length, used to slice the following list
length = len(values)
try:
```

```
    while 1:
        print('turn anticlockwise')
        # get servo value
        for i in range(0,150):
            # Change the value of the list index to 3(the value of the servo angle)
            values[3] = i
            # i2c write in, which is involved in indexing and slicing the list, indexing start from 0
            bus.write_i2c_block_data(address, values[0], values[1:length])
            time.sleep(0.05)
        print('turn clockwise')
        for i in range(0,150):
            x = 180 - i
            # Change the value of the list index to 3
            values[3] = x
            bus.write_i2c_block_data(address, values[0], values[1:length])
            time.sleep(0.05)
except IOError:
    print('Write Error')
    os.system('sudo i2cdetect -y 1')
```

Running the above code, we can see the servo cyclic rotate from 0-149 to 149-0.

5.8.3 X series servo steering gear application

From the previous section, we already know how to drive a servo , and understand that we only need to get the servo number and servo angle, then we can write to control the servo through I2C.

The steering gear below the gimbal is S7, and the servo above is S8.

Therefore, we can obtain the servo number and servo angle from the buffer list after the socket communication is process.

```
elif buffer[0] == 0x01:  # Control servo command
    servernum = buffer[1]  # get servo number
    angle = buffer[2]  # getservo angle
    if abs(cfg.BERFORE_ANGLE[servernum - 1] - angle) > 2:  # Restrict the angle of the servo to be issued repeatedly
        servo.set(servernum, angle) #Call the set function which is in the servo module, and pass in the parameters: servo number, angle
```

Then let's take a look at how the servo number and the angle of the servo are written into the MCU. In the servo module, the following angle setting function is defined in the Servo class：

```
def set(self, servonum, servoangle):
    """
    Set servo angle
    :param servonum:servo number
    :param servoangle:servo angle
    :return:
    """
    angle = self.angle_limit(servoangle)  # Angle limit
    buf = [0xff, 0x01, servonum, angle, 0xff]
```

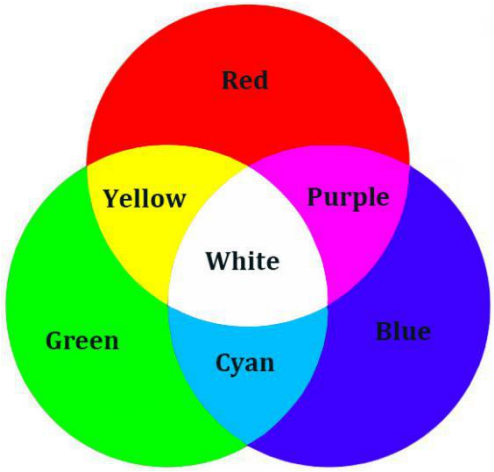
```
    try:
      i2c.writedata(buf)  # ic data write
    except Exception as e:
      print('servo write error:', e)
```

You can see that it is the same as our example in the previous section.

5.9 RGB light bar

RGB color lights are light bulbs that we usually see that can emit a variety of different colors. The principle is relatively simple. As the name suggests, RGB represents the three primary colors Red, Green, Blue, if red, green, and blue LED are light up at the same time, it will produce white light. When the two LED are light up separately, it can emit yellow, purple, and cyan (for example, red and blue LEDs emit purple light when they are light up), so colorful LED lights appear this phenomenon.

Seven colors: red, green, blue, yellow, purple, blue, white



5.9.1 Raspberry Pi drives MCU to control RGB lights

From Chapter 4.6, we can see the communication protocol of RGB colored lights. This section will teach you how to control RGB colored lights through a separate experiment. The hardware RGB colored lights that we provide are divided into two parts, one of which is the power indicator light. The other part is RGB car lights.

The communication protocol of RGB color lights is as follows, the specific protocol can write parameters, you can see the communication protocol in chapter 5.4.1.

```
#Packet header, first data bit, second data bit, third data bit, end of packet]
[0xff, 2, 8, 5, 0xff]
```

The first data bit: the choice of RGB lamp group, the optional value is 2 or 3, and the lamp group closest to the driver board is 2;

The second data bit: the number of lights up the RGB lamp group, one lamp group has 8 LED lamp beads, so the optional value is 0-8, and 0 is all off;

The third data bit: the light color of the RGB lamp group, the optional value is 0-8, and 0 is black (light off).

Let's use the following code to test the RGB light bar, here only one light bar is used for the experiment, regardless of the type of light bar:

```
# import package operation
```

```
import smbus
import time
import os

# i2 ccommunication address
address = 0x18
# Create instance
bus = smbus.SMBus(1)

# values = [packet header, which light bar is controlled, the number of light up, color, end of
packet]
values = [0xff, 2, 8, 0, 0xff]
# Get the length of the list, and use it in the following list slice
length = len(values)
try:
  while 1:
    # for nesting (double for loop)
     # j Control color
    for j in range(8):
      # i Control the number of lights
      for i in range(8):
        values[2] = i + 1
        values[3] = j + 1
        # Write delay to prevent IO Error due to fast writing
        time.sleep(0.05)
        # This involves the index and slice of the list, when the index starts from 0
        bus.write_i2c_block_data(address, values[0], values[1:length])

except IOError:
  print('Write Error')
  os.system('sudo i2cdetect -y 1')
```

5.9.2 Python multi-threading synchronous marquee

In fact, the main purpose here that is to tell you about the use of multi-threading. Multi-threading is used in our source code, so here I also use the marquee experiment to introduce you the use of multi-threading. Before that, I will also popularize the basic knowledge with you:

Python multi-threading

) Before introducing threads in Python, let me clarify a problem. Multi-threading in Python is fake multi-threading!

Why do I say that, we first clarify a concept, the global interpreter lock (GIL)

What is GIL

The execution of Python code is controlled by the Python virtual machine (interpreter), and only one thread is executing at the same time. Access to the Python virtual machine is controlled by the Global Interpreter Lock (GIL), which ensures that only one thread is running at the same time.

Why GIL

At the beginning of the design of Python, the decision was made for data safety; for the consistency of data between threads and the integrity of state synchronization.

The impact of GIL

Only one thread is running, and multiple cores cannot be used.

In a multithreading environment, the Python virtual machine executes in the following manner.

1. Set the GIL.
2. Switch to a thread to execute.
3. Run.
4. Set the thread to sleep.
5. Unlock the GIL.
6. Repeat the above steps again.

For example, I have a 4-core CPU, so in this way, each core can only run one thread per unit time, and then the time slices are switched in rotation. But Python is different. It doesn't matter how many cores you have, multiple cores can only run one thread per unit time, and then the time slice rotates. Give up after a period of execution, multi-threading can only be alternately executed in Python, and 10 cores can only use 1 core.

There are many ways to use multithreading. There are many ways to use threads. Only one is selected here. As everyone knows, if you are interested, you can directly search for relevant knowledge points on Google.

```
# using thread, import relevant packages
from threading import Thread
def loop():
  while True:
    print("Thread start")

if __name__ == '__main__':
  for i in range(3):
    t = Thread(target=loop)
    t.start()
  # Empty loop，
  while True:
    pass
```

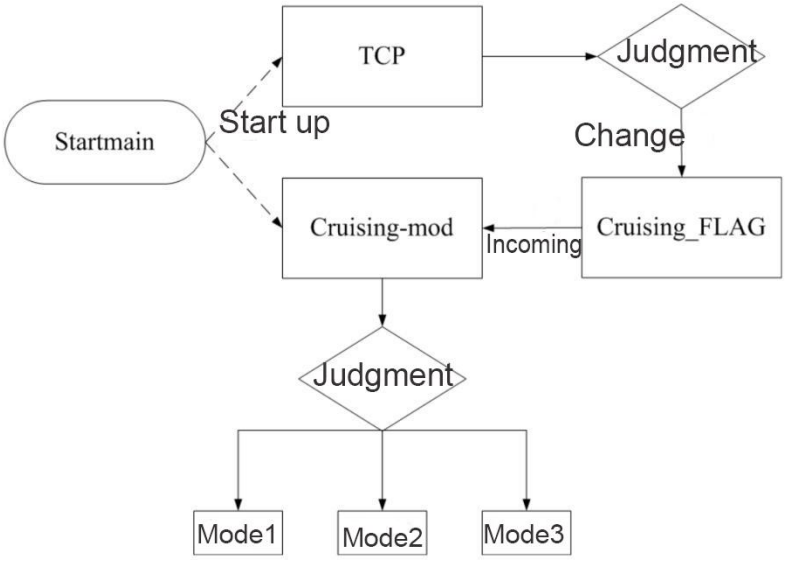
Use multi-threading to control two sets of RGB light bars at the same time：

```
# !/usr/bin/env python
# encoding: utf-8

# import package operation
import smbus
import time
from threading import Thread

# I2C ccommunication address

address = 0x18
# Create instance
bus = smbus.SMBus(1)
```

```
# A function that needs to be started by multi-threading. The function accepts a parameter. The parameter is the lamp group 2 or 3 that needs to be controlled.
def rgb_led(group):
    # values = [packet header, which light bar is controlled, the number of light up, color, end of packet]
    values = [0xff, group, 8, 0, 0xff]
    length = len(values)
    while True:
        # j control color
        for j in range(8):
            # i control the number of lights
            for i in range(8):
                values[2] = i + 1
                values[3] = j + 1
                # Write delay to prevent IOError due to fast writing
                time.sleep(0.05)
                # This involves the index and slice of the list, when the index starts from 0
                bus.write_i2c_block_data(address, values[0], values[1:length])


#  program entry
if __name__ == '__main__':
    # Create a thread empty list
    t_list = []
    for i in range(2, 4):
        # Create a thread, the thread enables the rgb_led function, and pass the parameter i, the comma must be present, otherwise an error will be reported
        t = Thread(target=rgb_led, args=(i,))
        # add threads to the list
        t_list.append(t)

    for t in  t_list:
        #enable threads
        t.start()
```

When we run the above code, we can see that the two sets of RGB lights change colors almost at the same time. In fact, this is a multi-threaded use in Python (although it is a fake multi-thread). The human eye can hardly judge whether they are started at the same time. , Because the program runs very fast.

Multi-threading is mainly reflected in the mode switching in the lower computer. The communication thread (Bluetooth and WiFi) has been running in the background and never ended. Crusingmody also has never ended in the main loop. The two threads have been running in coordination with each other; The TCP thread has been monitoring the analysis and judgment of the data sent by the host computer,. If the function function is monitored, the value of the global static file CruisingFLAG will be changed, and the Cruisingmod function will always monitor to determine whether the value of CruisingFLAG is the value of the preset mode; thus enter the corresponding function mode, After entering the functional mode, the two threads are still working in coordination, so that one mode can jump out of another mode.



```python
time.sleep(0.1)
os.system("sudo hciconfig hci0 reset")  # 重启蓝牙
time.sleep(0.3)
os.system("sudo hciconfig hci0 piscan")  # 恢复蓝牙扫描功能
time.sleep(0.2)
print("now bluetooth discoverable")

servo.restore()          # 复位舵机
oled.disp_default()      # oled显示初始化信息
car_light.init_led()     # 车灯秀
time.sleep(0.2)

threads = []  # 创建一个线程序列
t1 = threading.Thread(target=camera.run, args=())  # 摄像头数据收集处理线程
threads.append(t1)  # 将线程添加到线程队列中
t2 = threading.Thread(target=socket.bluetooth_server, args=())  # 新建蓝牙线程
threads.append(t2)
t3 = threading.Thread(target=socket.tcp_server, args=())  # 新建Wifi tcp通信线程
threads.append(t3)

ti = threading.Timer(0.1, status)  # 新建一个定时器
ti.start()  # 开启定时器

path_sh = 'sh ' + os.path.split(os.path.abspath(__file__))[0] + '/start_mjpg_streamer.sh &'  # start_mjpg
call("%s" % path_sh, shell=True)  # 新开一个进程运行start_mjpg_streamer
time.sleep(1)
```

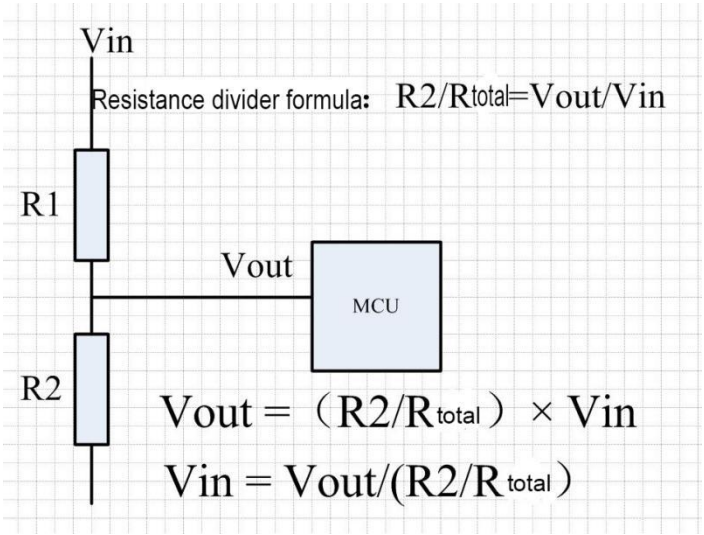
Multi-threading is an important part of Python. The use of threads is not limited to the above method. If you are interested, you can search for relevant information and learn.

5.10 Voltage detection

If you are just learning the code and don't need to know how to obtain the internal voltage value, then you can skip the explanation of physics here and go directly to the code behind.

First, let's take a look at how to calculate the current total voltage. We will talk about it in three parts.

In the first step, according to the circuit diagram, we can calculate the total voltage according to the resistor divider formula.



The resistance value has been provided on the schematic, here we just show you the calculation method.

In the second step, the value of Vout is directly calculated by the microcontroller internal. The chip

has integrated a 12-bit AD converter and the voltage that the chip can read is 0-5V; therefore, we know the value of Vout and the values of R1 and R2. You can directly calculate the value of Vin.

In the third step, from the circuit diagram that provided by us, we can see that the voltage we actually supply to the chip is 5.12V, so the code on the Raspberry Pi is reflected as:

Vin =MCU return value*5/5.12

The above are all theoretical knowledge, which is what the MCU needs to do; through the communication protocol, we can know that we only need to read the specified address 0x05 to get the voltage return value, so we can directly write the following code:

```
# !/usr/bin/env python
# encoding: utf-8

# import package operation
import smbus
import time
import os

# i2c communication address
address = 0x18
# create instance
bus = smbus.SMBus(1)

try:
    # Reading the specified address will return a value and assign it to value
    value = bus.read_byte_data(address, 0x05)
    time.sleep(0.05)
    # Vin = MCU return value*5/5.12
    value = value*5/5.12/10
    # Print value on console
    print(value)
except IOError:
    print('Write Error')
    os.system('sudo i2cdetect -y 1')
```

After that, we need to reflect the power, so we use a set of light bars to reflect the design. We divide the battery into 3 gears; each is represented by three colors of green/yellow/red.

```
# !/usr/bin/env python
# encoding: utf-8

# import package operation
import smbus
import time
import os

# i2c communication address
address = 0x18
# create instance
bus = smbus.SMBus(1)
```

```
# Get the voltage value function
def get_voltage():
   value = bus.read_byte_data(address, 0x05)
   time.sleep(0.05)
      # Vin = MCU return value *5/5.12
   value = value * 5 / 5.12 / 10
   return value


try:
   while True:
      voltage = get_voltage()
      print(voltage)
      if voltage > 11:
         print('RGB：green')
      elif 9 < voltage < 11:
         print('RGB：yellow')
      elif voltage < 9:
         print('RGB:red')
except IOError:
   print('Write Error')
   os.system('sudo i2cdetect -y 1')
```

5.11 Socket communication

Before any type of communication starts, the network application must create a socket (Socket)

Socket address: host-port pair. If a socket is like a telephone jack—some infrastructure that allows communication, then the host name and port pair are like the area code and phone number in the phone. If you have telephone facilities, but don't have a phone number to dial, does it have no effect in itself?

In addition, it must also be stated that someone is answering on the other end, otherwise it will appear: the number you dialed is empty, please check before dialing, just like we usually browse the web and see that the server does not respond or 404.

Let's not talk about conceptual things, just look at how to use them.

The socket module for sockets is used in Python. After importing the module, you can use the socket.socket() function. Its general usage is as follows:

```
socket(socket_family, socket_type, protocol=0)
```

Where socketfamily is AFUNIX or AFINET, sockettype is SOCKSTREAM or SOCKDGRAM, protocol is usually omitted, and is generally 0.

AFUNIX is a file-based socket, AFUNIX is not very commonly used in programming. AF_INET is used for network-based Socket. There is also an address family AF_INET6, which is used for IPv6. In fact, there are some other address families, those are either professional, outdated, rarely used, or still unimplemented. Among all address families, AF_INET is the most widely used.

The following table is the common built-in methods in the Python socket module:

Name	Description
Server socket method	
s.bind()	Bind the address (host name, port number is right) to the socket
s.listen()	Set up and start the TCP listener
s.accept()	Passively receive the TCP client listener and wait until the connection is reached (blocked)
Client socket method	
s.connect	Actively initiate a TCP server connection
s.connect_ex()	The extended version of connect() will return the problem in the form of an error code instead of throwing an exception
Ordinary socket method	
s.recv()	Receive TCP messages
s.recv_into()	Receive TCP message to the specified buffer
s.send()	Send TCP message
s.sendall()	Complete sending of TCP messages
s.recvform()	Receive UDP messages
s.recvfrom_into()	Receive UDP message to the specified buffer
s.sendto()	Send UDP message
s.getpeername()	Connect to the remote address of the socket (TCP)
s.getsockname()	Current socket address
s.getsockopt()	Returns the value of the given socket option
s.setsockopt	Set the value of the given socket self option
s.shutdown()	Close the connection
s.close()	Close socket
s.detach()	Close the socket while the file descriptor unclose and return the file descriptor
s.ioct()	Control the mode of the socket (only supports Windows)
Block-oriented socket method	
s.setblocking()	Set the blocking or non-blocking mode of the socket
s.settimeout()	Set the timeout period for blocking socket operations
s.gettimeout()	Get the timeout period for blocking socket operations
File-oriented socket method	
s.fileno()	File descriptor of the socket
s.makefile()	Create a file object associated with the socket
Digital attributes	
s.family	Socket family
s.type	Socket type
s.proto	Socket protocol

The following code example, each sentence has a comment, you can look at the comment of each sentence in detail.

Server code:

```
from socket import *

# The host address to be bound to the server, the server side can be empty
HOST = ''
PORT = 9999  # port number which is monitored by the server
ADDR = (HOST, PORT)
BUFSIZE = 1024  # set the buffer size of the server

while True:
    tcpSevSock = socket(AF_INET, SOCK_STREAM)  # create socket object
    tcpSevSock.bind(ADDR)  # bind the socket to the specified address and port
    tcpSevSock.listen(5)  # open server listener
    print('waiting for Client Connection...')
```

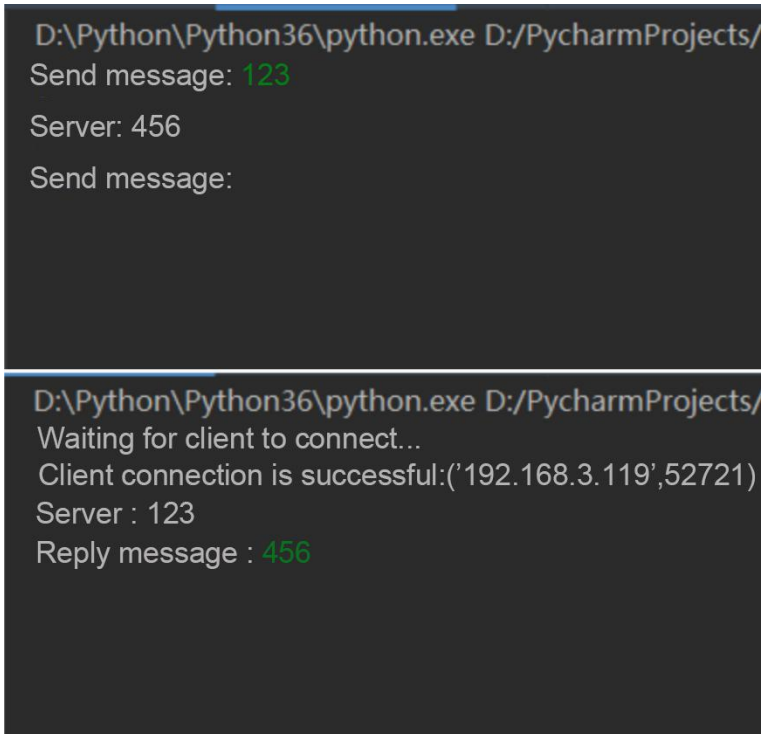
```
tcpCliSock, cliAddress = tcpSevSock.accept()  # accept client connections
print('client connection successful：', cliAddress)

while True:
    try:
        data = tcpCliSock.recv(BUFSIZE)  # accept the data sent by the client
    except:
        print("Receiving error")
        break
    # If there's no data
    if not data:
        # Jump out of the loop
        break
    # Decode the received byte data
    data = data.decode('utf-8')
    print('received massage：', data)
    senddata = input('reply massage：')
    # send coding
    tcpCliSock.sendall(senddata.encode('utf-8'))
# close client
tcpCliSock.close()
print('One client is offline...')
# close server
tcpSevSock.close()
```

Client code：

```
from socket import *

# The address of the server to which the client is going to connect
HOST = '192.168.3.128'
# The port number of the server should be keep the same
PORT = 9999
# address of the server
ADDR = (HOST, PORT)
# The size of the client buffer (in bytes)
BUFSIZE = 1024
# All sockets are created by using socket function
tcpCliSock = socket(AF_INET, SOCK_STREAM)
# The client connects to the server
tcpCliSock.connect(ADDR)

while True:
    # Reading data from the keyboard
    data = input(' send message：')
    if not data:
        break
    # Send a message to the server. Because send can only send byte data, so encode the string
before sending it
    tcpCliSock.send(data.encode('utf-8'))
```

```
    # accept the information which is sent by the server
    data = tcpCliSock.recv(BUFSIZE)
    if not data:
        break
    # Since the data passed through the network is actually byte data, it will be output after
decoding
    print(data.decode('utf-8'))
tcpCliSock.close()
```

When running the above case, you need to start the server program first, and then start the client program; the effect is as follows:



5.11.1 Raspberry-X series serial port data analysis

We learned from the above case to know how to use sockets for network communication in Python, then let's take a look at the socket communication source code in DS-X:

```
class Socket:

    def __init__(self):
        self.clent = None

    def load_server(self, server, servername):
        """
        Socket Service function
        Parameters: self instance class, the service to be started by the parameter server, the data
received by buf, the name of the service type to be started by server name
        """
        while True:
            print("waitting for %s connection..." % servername, "\r")
```

```
        if (servername == 'bluetooth'):  # If you choose bluetooth communication
            cfg.BT_CLIENT = False
            cfg.BT_CLIENT, socket_address = server.accept()  # Initialize the socket, and create a
client and address
                            self.clent = cfg.BT_CLIENT
            print(str(socket_address[0]) + " %s connected!" % servername + "\r")  # Print client and
address
        elif (servername == 'tcp'):  #If you choose wifi communication
            cfg.TCP_CLIENT = False
            cfg.TCP_CLIENT, socket_address = server.accept() # Initialize the socket, and create a
client and address
            self.clent = cfg.TCP_CLIENT
            print(str(socket_address[0]) + "%s connected!" % servername + "\r")  # Print client and
address

        while True:
            try:
                data = self.clent.recv(cfg.RECV_LEN)  # cfg.RECV_LEN The length of characters received
at one time
                if len(data) < cfg.RECV_LEN:  # non-conformity the receiving length standard
                    break
                if data[0] == 0xff and data[len(data) - 1] == 0xff:  # If the header and the end of the
packet are 0xff, it conforms to the xiaoR technology communication protocol
                    buf = []  # Define a list
                    for i in range(1, 4):  # Gets 3-bit data in the middle of the protocol package
                        buf.append(data[i])  # Add data to the buf
                    self.communication_decode(buf)  # Call serial port analysis function
            except Exception as e:  # Receiving error
                print('socket received error:', e)  # Printing Error message

        self.clent.close()
        cfg.SOCKET_RUN = False
    go.stop()
    cfg.SOCKET_RUN = False
    server.close()
```

In the socket instance, we first initialize a client in the __init__ method, and its default value is empty.

The loadserver() function receives two parameters, servername is used to determine whether to start Bluetooth or wireless WiFi communication, so when we call the loadserver function, we need to pass in parameters to determine which communication function to open, but we continue to define bluetoothserver and tcpserver these two functions under the Socket class, so if we directly call these two functions, the Bluetooth and TCP communication will be turned on. We also operate in the same way, (using multithreading to start the two communication functions).

```
class Socket:
  def __init__(self):
    self.clent = None
```

```
    def load_server(self, server, servername):
        # ···omit
    def bluetooth_server(self):
        """
        start Bluetooth receiving service
         Parameters: The first parameter indicates the service to be started, and the second
parameter indicates the name of the service to be started
        """
        print("load bluetooth_server")
        # Pass parameters and call load_server function
        self.load_server(cfg.BT_SERVER, 'bluetooth')

    def tcp_server(self):
        """
        Start tcp receiving service
         Parameters: The first parameter indicates the service to be started, and the second
parameter indicates the name of the service to be started
        """
        print("load tcp_server")
        # Pass parameters and call load_server function
        self.load_server(cfg.TCP_SERVER, 'tcp')
```

The parameter server is an instantiated object and is configured in xr_config.

The next step is the serial port analysis. The serial port analysis function is relatively simple. We just need to analyze the data according to the DS-X communication protocol, and then call the corresponding function function according to the corresponding protocol.

The communication protocol between the host computer and the Raspberry Pi is as follows:

Type	Features	Packet header	Type bit	Control bit	Data bit	End of packet	
Motor	Stop	0xff	00	00	00	0xff	
	Go forward			01			
	Go back			02			
	Turn left			03			
	Turn right			04			
Servo	Rotate		01	01-08	Angle		7-8 are gimbal servos, 1-6 are robotic arm servos
Motor	Left side speed		02	01	Gear position		Gear position: 0-100
	Right side speed			02	Gear position		
Mode switch	Normal mode		13	02	xx		
	Infrared line patrol			03			
	Infrared drop mode			04			
	Ultrasonic obstacle			05			
	Ultrasonic maze			06			
	Buzzer music			07			
Advanced Featur	Camera line			08			Only supports Raspberry Pi
	QR code			09			
	Color recognition			0A			
RGB	Multiple lights are on at the same time		40	X	Y		X represents the number of lamp beads, the value range (0x02-0x09) corresponds to the number of lamps 08 Y represents the display color, the value range (0-8), corresponding to black, orange, red, green, yellow and cyan
	Single light is on			M	N		
	Turn on the lights			00	00		All of lights are lit then the color is white
	Turn off the lights			01			All of lights are all lit then the color is black
Other	Get battery		31	xx	xx		
	Servo reset		32	xx	xx		
	Servo memory		33	xx	xx		

Serial data analysis code:

```
def communication_decode(self, buffer):
    """
    Data analysis function, according to the data filtered by the socket, that is, the small R technology communication protocol is parsed into the corresponding function and action according to the bit
     Protocol format----------0xff 0xXX 0xXX 0xXX 0xff
     Meaning ---------- Package header  Type bit  Control bit  Data bit  End of packet
     Socket filtered data buffer[]--- buffer[0] buffer[1] buffer[2]
    """
    print(buffer)
        if buffer[0] == 0x00:  # buffer[0] indicates the type bit, equal to 0x00 indicates that this data packet is the data packet of the motor control instruction
        if buffer[1] == 0x01:  # buffer[1] represents the control bit, equal to 0x01 represents the forward data packet
            go.forward()  # go forward
        elif buffer[1] == 0x02:
            go.back()  # go back
        elif buffer[1] == 0x03:
            cfg.LIGHT_STATUS = cfg.TURN_LEFT
            go.left()  # turn letf
        elif buffer[1] == 0x04:
            cfg.LIGHT_STATUS = cfg.TURN_RIGHT
            go.right()  # turn right
        elif buffer[1] == 0x00:
            cfg.LIGHT_STATUS = cfg.STOP
            go.stop()  # stop
        else:
            go.stop()

    elif buffer[0] == 0x01:  # Control servo command
        servernum = buffer[1]  # get servo number
        angle = buffer[2]  # get servo angle


        if abs(cfg.BERFORE_ANGLE[servernum - 1] - angle) > 2:  # Restrict the angle of the servo to be issued repeatedly


            servo.set(servernum, angle)

        elif buffer[0] == 0x02:  # Adjust motor speed
            if buffer[1] == 0x01:  # Adjust the left motor speed
            cfg.LEFT_SPEED = buffer[2]
            go.set_speed(1, cfg.LEFT_SPEED)

        elif buffer[1] == 0x02:  # Adjust the right motor speed
            cfg.RIGHT_SPEED = buffer[2]
            go.set_speed(2, cfg.RIGHT_SPEED)
```

```
    elif buffer[0] == 0x13:
        if buffer[1] == 0x01:
            cfg.CRUISING_FLAG = 1  # Enter infrared following mode

        elif buffer[1] == 0x02:  # Enter Infrared Patrol Mode
            cfg.CRUISING_FLAG = 2

        elif buffer[1] == 0x03:  # Enter Infrared anti-fall mode
            cfg.CRUISING_FLAG = 3

        elif buffer[1] == 0x04:  # Enter ultrasonic avoidance mode
            cfg.CRUISING_FLAG = 4

        elif buffer[1] == 0x05:  # Enter ultrasonic distance PC display
            cfg.CRUISING_FLAG = 5

        elif buffer[1] == 0x06:
            cfg.CRUISING_FLAG = 6

        elif buffer[1] == 0x07:
            cfg.CRUISING_FLAG = 7
        # buf = bytes([0xff, 0xa8, 0x00, 0x00, 0xff])
        # self.clent.send(buf)

        elif buffer[1] == 0x08:
            if buffer[2] == 0x00:  # Path_Dect  debug mode
                cfg.Path_Dect_on = 0
                cfg.CRUISING_FLAG = 8

            elif buffer[2] == 0x01:  # Path_Dect Track mode
                path_sh = 'sh ' + os.path.split(os.path.abspath(__file__))[0] + '/stop_mjpg_streamer.sh &'
                call("%s" % path_sh, shell=True)
                time.sleep(2)
                cfg.Path_Dect_on = 1
                cfg.CRUISING_FLAG = 9

        elif buffer[1] == 0x09:  # Camera QR Code Recognition          cfg.CRUISING_FLAG = 10
            cfg.CAMERA_MOD = 2
        elif buffer[1] == 0x0a:  # Camera face recognition
            cfg.CRUISING_FLAG = 11
            cfg.CAMERA_MOD = 3
        elif buffer[1] == 0x0b:  # Camera color recognition
            cfg.CRUISING_FLAG = 12
            cfg.CAMERA_MOD = 4
        elif buffer[1] == 0x0c:  # Camera color recognition
            cfg.CRUISING_FLAG = 13

        elif buffer[1] == 0x00:
            cfg.RevStatus = 0
            cfg.CRUISING_FLAG = 0
```

```
        print("CRUISING_FLAG normal mode  %d " % cfg.CRUISING_FLAG)

    elif buffer[0] == 0xa0:
        cfg.TurnAngle = buffer[1]
        cfg.Golength = buffer[2]
        cfg.RevStatus = 2

    elif buffer[0] == 0xa1:
        cfg.TurnAngle = buffer[1]
        cfg.Golength = buffer[2]
        cfg.RevStatus = 1

    elif buffer == [0x31, 0x00, 0x00]:  # Query battery power information
        buf = bytes([0xff, 0x31, 0x01, cfg.POWER, 0xff])
        self.clent.send(buf)

    elif buffer[0] == 0x32:  # store angle
        servo.store()

    elif buffer[0] == 0x33:  # Read angle
        servo.restore()

        elif buffer[0] == 0x40:  # Switch light mode FF040000FF on FF040100FF off lights
        if buffer[1] == 0x00:
            car_light.open_light()  # All lights on, white
        elif buffer[1] == 0x01:
            car_light.close_light()  # All lights off, black
        else:
            lednum = buffer[1]  # Get light quantity command massage
            ledcolor = buffer[2]  # Get light color command massage

            if lednum < 10:  # Multi-lamp mode light on
                car_light.set_ledgroup(cfg.CAR_LIGHT, lednum - 1, ledcolor)
            elif 9 < lednum < 18:  # Single light mode ligh on
                car_light.set_led(cfg.CAR_LIGHT, lednum - 10, ledcolor)

    elif buffer[0] == 0x41:
        if buffer[1] == 0x00:
            tune = buffer[2]
            cfg.TUNE = tune
        # beep.tone(beep.tone_all)
        elif buffer[1] == 0x01:  # Received bass
            beet1 = buffer[2]
            beep.tone(beep.tone_all[cfg.TUNE][beet1 + 14], 0.5)
        elif buffer[1] == 0x02:  # Received mid-tone
            beet2 = buffer[2]
            beep.tone(beep.tone_all[cfg.TUNE][beet2], 0.5)
        elif buffer[1] == 0x03:  # Received high pitch
                    beet3 = buffer[2]
            beep.tone(beep.tone_all[cfg.TUNE][beet3 + 7], 0.5)
```

```
# Heartbeat packet
elif buffer == [0xef, 0xef, 0xee]:
    print("Heartbeat Packet!")
# Raspberry shutdown
elif buffer[0] == 0xfc:  # FFFC0000FF  shutdown
    os.system("sudo shutdown -h now")

else:
    print("error command!")
```

The buffer has been converted to a list in the socket communication:

```
if data[0] == 0xff and data[len(data) - 1] == 0xff:

    buf = []     # Define a list
    for i in range(1, 4):# Get the data of the middle 3 bits of the protocol report
        buf.append(data[i]) # Add data to buf
    self.communication_decode(buf)# Call serial port analysis function
```

Therefore, in the serial port analysis, we only need to judge the list data to parse the serial port data and execute the corresponding function call.

5.12 PS2 control function

Everyone must have played a remote control car. Use the wireless handle to control the car to run on the ground. Most of boys want to play the toy when they were young. Now we will use technology to realize it and make a handle remote control car.

First, the PS2 handle is divided into a handle and a receiver. The receiver is connected to the Arduino microcontroller. The PS2 uses the SPI communication protocol. SPI is the abbreviation of Serial Peripheral Interface, which is a high-speed, full-duplex, and synchronous communication bus. The communication principle is similar. The handle and the receiver communicate with each other. The receiver is connected to the four pins of the chip DI, DO, CS, and CLK. Obviously, the receiver (chip) can directly analyze the communication protocol after receiving the data. Of course, this is due to the cracking of PS2; the PS2 controller is the remote controller of the PlayStation2 game console of Japan's SONY company. Sony's PSX series of game consoles are very popular all over the world. I don’t know when someone came up with the idea of a PS2 controller and cracked the communication protocol so that the controller can be connected to other devices for remote control, such as remote control of our car.

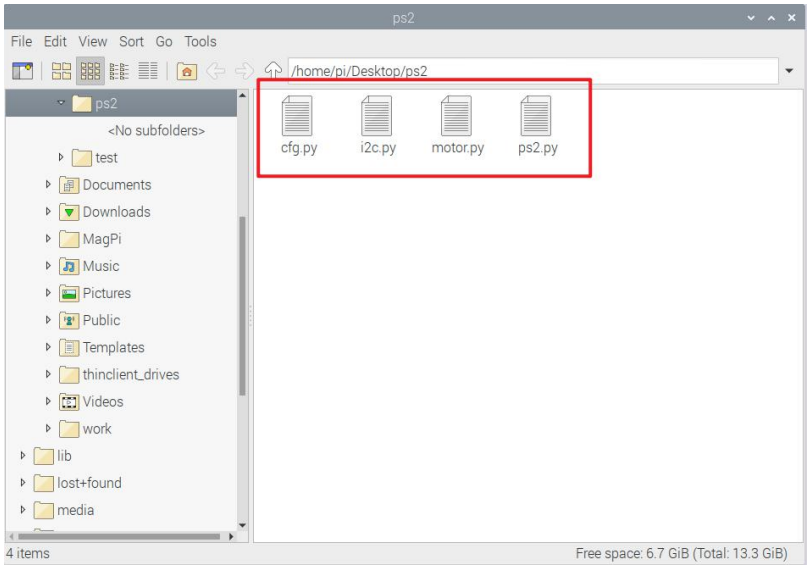
The PS2 handle is connected to the I2C interface on the back of the drive board, which means that PS2 communicates with the microcontroller in the drive board through I2C (not directly with the Raspberry Pi). The corresponding protocol has been defined, and we only need to perform related operations according to the protocol.

The following table is the communication protocol between X series development car and PS2 handle.

I2C PS2-Green light mode Slave address 0x19 (I2C and serial port data bits are the same)									
Address	0	1	2	3	4	5	6	7	8
Direction left-forward	0XFF	0X41	0X5A	0XEF	0XFF	0XFF	0XFF	0XFF	0XFF
Direction left-back	0XFF	0X41	0X5A	0XBF	0XFF	0XFF	0XFF	0XFF	0XFF
Direction left-right	0XFF	0X41	0X5A	0XCF	0XFF	0XFF	0XFF	0XFF	0XFF
Direction left-left	0XFF	0X41	0X5A	0XDF	0XFF	0XFF	0XFF	0XFF	0XFF
Direction right-forward	0XFF	0X41	0X5A	0XFF	0XEF	0XFF	0XFF	0XFF	0XFF
Direction right-back	0XFF	0X41	0X5A	0XFF	0XBF	0XFF	0XFF	0XFF	0XFF
Direction right-right	0XFF	0X41	0X5A	0XFF	0XCF	0XFF	0XFF	0XFF	0XFF
Direction right-left	0XFF	0X41	0X5A	0XFF	0XDF	0XFF	0XFF	0XFF	0XFF
L1	0XFF	0X41	0X5A	0XFF	0XFB	0XFF	0XFF	0XFF	0XFF
L2	0XFF	0X41	0X5A	0XFF	0XFE	0XFF	0XFF	0XFF	0XFF
R1	0XFF	0X41	0X5A	0XFF	0XF7	0XFF	0XFF	0XFF	0XFF
R2	0XFF	0X41	0X5A	0XFF	0XFD	0XFF	0XFF	0XFF	0XFF
SELECT	0XFF	0X41	0X5A	0XFE	0XFF	0XFF	0XFF	0XFF	0XFF
START	0XFF	0X41	0X5A	0XF7	0XFF	0XFF	0XFF	0XFF	0XFF
Red light mode									
Address	0	1	2	3	4	5	6	7	8
Left-joystick Y axis	0XFF	0X73	0X5A	0XFF	0XFF	0X80	0X80	0X80	0-FF
Left-joystick X axis	0XFF	0X73	0X5A	0XFF	0XFF	0X80	0X80	0-FF	0X80
Right-joystick Y axis	0XFF	0X73	0X5A	0XFF	0XFF	0X80	0-FF	0X80	0X80
Right-joystick X axis	0XFF	0X73	0X5A	0XFF	0XFF	0-FF	0X80	0X80	0X80

In our experiment, only the 4 buttons(up,down,left and right) on the left side of the handle are used, which is the direction left.

First, we create the following 4 .py files in the same folder, as shown in the figure:



cfg.py is the configuration file; i2c is the read and write operation function of i2c, in this experiment we only need to use the read operation function; motor.py is the motor-related function; ps2.py is the PS2 handle-related control function, program The main loop is also in this file; next we will interpret the code in detail.

First, let's look at the code in the cfg.py file, the code is as follows:

```
# !/usr/bin/env python
# encoding: utf-8

import RPi.GPIO as GPIO # Import GPIO related operating packages

# PS2 handle related buttons definition
PS2_ABLE = False  # Whether the PS2 handle is connected normally
PS2_READ_KEY = 0  # the read value of PS2 handle
PS2_LASTKEY = 0  # The last read value of the PS2 handle
PS2_KEY = {'PSB_PAD_UP': 1, 'PSB_PAD_DOWN': 2, 'PSB_PAD_LEFT': 3, 'PSB_PAD_RIGHT': 4} #
Up, down, left, and right buttons on the left side of the handle

# Motor drive interface definition#
ENA = 13  # L298 enable A
ENB = 20  # L298enableB
IN1 = 16  # motor interface 1
IN2 = 19  # motor interface2
IN3 = 26  # motor interface3
IN4 = 21  # motor interface4

GPIO.setwarnings(False) # Remove warning information
GPIO.setmode(GPIO.BCM) # set raspberry pi GOIP pin mode

# Motor Initialization Settings
GPIO.setup(ENA, GPIO.OUT, initial=GPIO.LOW)
ENA_pwm = GPIO.PWM(ENA, 1000)
ENA_pwm.start(0)
```

```
ENA_pwm.ChangeDutyCycle(50)
GPIO.setup(IN1, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(IN2, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(ENB, GPIO.OUT, initial=GPIO.LOW)
ENB_pwm = GPIO.PWM(ENB, 1000)
ENB_pwm.start(0)
ENB_pwm.ChangeDutyCycle(50)
GPIO.setup(IN3, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(IN4, GPIO.OUT, initial=GPIO.LOW)


# GPIO port output setting, here is used for motor port setting, called in motor.py
def digital_write(gpio, status):
    """
    Set gpio port to level
     Parameters: gpio is the set port, status is the status value can only be True (high level), False
(low level)
    """
    GPIO.output(gpio, status)
```

In cfg.py are the basic configuration parameters, in which including the parameter definitions related to the PS2 handle, as well as the definition of the motor drive interface and the initialization of the motor. Both are relatively simple initialization parameter settings. A digital_write() function is defined. The function receives two parameters: GPIO pin number and status. The function internal the function actually calls the output function in RPi.GPIO to set the GPIO status.

Next we look at the code in the i2c.py file, the code is as follows:

```
# Import related libraries
import os
import time
import smbus


class I2c:
    def __init__(self):
        self.mcu_address = 0x18
        self.ps2_address = 0x19
        self.device = smbus.SMBus(1)
        pass

    # Omitting the write function

    def readdata(self, address, index):
        """
        #Read one byte of data from I2C
        """
        try:
            value = self.device.read_byte_data(address, index)  # read a byte index from the device
offset
            time.sleep(0.08)
            return value  # return read date
```

```
    except Exception as e:  # read error
        print('i2c read error:', e)
        os.system('sudo i2cdetect -y 1')
```

In i2c communication, we only use the readdata function. Readdata is an I2C read operation. The function accepts two parameters: the address address and the value index to be written. According to the communication protocol, writing the specified value to the address 0x19 will get the specified return value

Then we continue to check the code in the motor.py file, the code is as follows:：

```
import cfg


class RobotDirection:
    def __init__(self):
        pass

    def m1m2_forward(self):
        # Set the motor group M1 and M2 to rotate forward
        cfg.digital_write(cfg.IN1, True)
        cfg.digital_write(cfg.IN2, False)

    def m1m2_reverse(self):
        # Set the motor group M1 and M2 to rotate reverse
        cfg.digital_write(cfg.IN1, False)
        cfg.digital_write(cfg.IN2, True)

    def m1m2_stop(self):
        # Set the motor group M1 and M2 to stop
        cfg.digital_write(cfg.IN1, False)
        cfg.digital_write(cfg.IN2, False)

    def m3m4_forward(self):
        # Set the motor group M3 and M4 to rotate forward
        cfg.digital_write(cfg.IN3, True)
        cfg.digital_write(cfg.IN4, False)

    def m3m4__reverse(self):
        # Set the motor group M3 and M4 to rotate reverse
        cfg.digital_write(cfg.IN3, False)
        cfg.digital_write(cfg.IN4, True)

    def m3m4__stop(self):
        # Set the motor group M3 and M4 to stop
        cfg.digital_write(cfg.IN3, False)
        cfg.digital_write(cfg.IN4, False)

    def forward(self):
        """
        Set the robot movement direction to forward
```

```
        """
        self.m1m2_forward()
        self.m3m4_forward()

    def back(self):
        """
        #Set the robot movement direction to back
        """
        self.m1m2_reverse()
        self.m3m4__reverse()

    def left(self):
        """
        #Set the robot movement direction to turn left
        """
        self.m1m2_reverse()
        self.m3m4_forward()

    def right(self):
        """
        #Set the robot movement direction to turn right
        """
        self.m1m2_forward()
        self.m3m4__reverse()

    def stop(self):
        """
        #Set the robot movement direction to stop
        """
        self.m1m2_stop()
        self.m3m4__stop()
```

The motor-related procedures have been explained in chapter 5.1, so I won't repeat the introduction here. If you don't understand, you can go back to chapter 5.1 for reference.

Finally, let's take a look at the code in ps2.py：

```
# !/usr/bin/env python
# encoding: utf-8

# Import related libraries and classes
import time
import cfg
from i2c import I2c
from motor import RobotDirection

# Create object
i2c = I2c()
go = RobotDirection()


class PS2:
```

```
    def __init__(self):
        pass

    def ps2_able(self):
        """
        Determine whether PS2 is connected successfully
        """
        dat = i2c.readdata(i2c.ps2_address, 0x01)
        if dat == 0x41 or dat == 0xC1 or dat == 0x73 or dat == 0xF3:  # When the return value is one of
them, it means that the PS2 handle is connected normally
            cfg.PS2_ABLE = True
        else:
            cfg.PS2_ABLE = False

    def ps2_button(self):
        """
        Get the key value of the PS2 controller
         :return:cfg.PS2_READ_KEY parsed key value
        """
        ps2check = i2c.readdata(i2c.ps2_address, 0x01)  # Get the mode value returned by PS2
        read_key = i2c.readdata(i2c.ps2_address, 0x03)  # Get the key value returned by PS2
        cfg.PS2_READ_KEY = 0                    # When the button is not pressed, directly return to 0
        if ps2check == 0x41 or ps2check == 0xC1:  # PS2 common model
            if read_key == 0xef:
                #PS2_KEY is the dictionary, PSB_PAD_xxx is the key of the dictionary, and the
corresponding values are 1234, which are defined in cfg.py            cfg.PS2_READ_KEY =
cfg.PS2_KEY['PSB_PAD_UP']
            elif read_key == 0xbf:
                cfg.PS2_READ_KEY = cfg.PS2_KEY['PSB_PAD_DOWN']
            elif read_key == 0xcf:
                cfg.PS2_READ_KEY = cfg.PS2_KEY['PSB_PAD_LEFT']
            elif read_key == 0xdf:
                cfg.PS2_READ_KEY = cfg.PS2_KEY['PSB_PAD_RIGHT']
        else:
            print('PS2 Read Error!')
        return cfg.PS2_READ_KEY

    def control(self):
        """
        PS2 handle control function
        """
        if cfg.PS2_ABLE:
            print(1)
            read_ps2 = self.ps2_button()  # Get the key value
            if cfg.PS2_LASTKEY != read_ps2 and cfg.PS2_LASTKEY != 0:  # If the last value is not 0 (the
default value is 0) and is not equal to the value of this time, it means that the key value has
changed and is not 0
                go.stop()  # Execute once and stop once when the key value changes
                print('Car stop')
                cfg.PS2_LASTKEY = 0  # Assign a value of 0 to the last state to avoid re-entering and
```

```
stopping

        else:
            if read_ps2 == cfg.PS2_KEY['PSB_PAD_UP']:  # Equal to the up button of the left button
                print('car go forward')
                go.forward()
                time.sleep(0.02)
                cfg.PS2_LASTKEY = read_ps2  # Update last value

            elif read_ps2 == cfg.PS2_KEY['PSB_PAD_DOWN']:  # Equal to the down button of the left
button
                print('car go back')
                go.back()
                time.sleep(0.02)
                cfg.PS2_LASTKEY = read_ps2

            elif read_ps2 == cfg.PS2_KEY['PSB_PAD_LEFT']:  # Equal to the left button of the left
button
                print('car turn left')
                go.left()
                time.sleep(0.02)
                cfg.PS2_LASTKEY = read_ps2

            elif read_ps2 == cfg.PS2_KEY['PSB_PAD_RIGHT']:  # Equal to the right button of the left
button
                print('car turn right')
                go.right()
                time.sleep(0.02)
                cfg.PS2_LASTKEY = read_ps2


ps2 = PS2()    # create object
ps2.ps2_able()  # Calling ps2_able once to change the value of PS2_ABLE is essential
while True:
    try:
        # Class object calls class method
        ps2.control()
    except Exception as e:  # Program running error
        print(e)
```

In this file code, first we imported related libraries and classes, created i2c and go objects, the corresponding classes are I2c() and RobotDirectio(); then we created the class PS2, which contains ps2able, ps2button and control these three class methods, ps2able is used to determine whether the handle is successfully connected, and if the connection is successful, the value of PS2ABLE is changed to True, otherwise it is False.

After the connection is successful, we need to get the key value of the PS2 handle. Also through the I2C read function, write 0x03 to the specified address 0X19. We can get the key value of the handle, and then compare and judge which key was pressed according to the PS2 communication protocol. .

After obtaining the button value, call the corresponding motor program according to the button value to complete the PS2 control of the motor.

When entering the main loop while True, we first call ps2.ps2able() once. This step is necessary because the initial value of PS2ABLE is False. After entering the loop, we will call the control method in PS2. In the control method, we will first judge PS2ABLE. Only after the handle is successfully connected can we enter the internal motor control program. Otherwise, if the PS2ABLE value is False, it will remain stuck in "if cfg .PS2_ABLE: ".

Chapter 6 Detailed Explanation of Advanced Functions

6.1 Color detection

color model

The commonly used models in digital image processing are RGB (red, green, blue) model and HSV (hue, saturation, brightness). RGB is widely used in color monitors and color video cameras. Our usual pictures are generally RGB models. . The HSV model is more in line with the way people describe and interpret colors. The color description of HSV is natural and very intuitive to people.

HSV model

The color parameters in the HSV model are: hue (H: hue), saturation (S: saturation), and brightness (V: value). A color space created by A. R. Smith in 1978, also known as the Hexcone Model.

(1) Hue (H: hue): Measured by angle, the value range is 0° ~ 360°, starting from red and counting in a counterclockwise direction, red is 0°, green is 120°, and blue is 240°. Their complementary colors are: 60° for yellow, 180° for cyan, and 300° for magenta;

(2) Saturation (S: saturation): the value range is 0.0 to 1.0, the larger the value, the more saturated the color.

(3) Brightness (V: value): the value range is 0 (black) ~ 255 (white)

RGB to HSV

Let (r, g, b) be the red, green, and blue coordinates of a color, and their values are real numbers between 0 and 1. Let max be equivalent to the largest of r, g, and b. Let min be equal to the smallest of these values. To find the (h, s, v) value in the HSV space, where h ∈ [0, 360) is the hue angle of the angle, and s, v ∈ [0,1] is the saturation and brightness, the method is as follows:

```
max=max(R,G,B)
min=min(R,G,B)
if R = max, H = (G-B)/(max-min)
if G = max, H = 2 + (B-R)/(max-min)
if B = max, H = 4 + (R-G)/(max-min)

H = H * 60
if H < 0, H = H + 360
V=max(R,G,B)
S=(max-min)/max
```

There is a function under OpenCV that can directly convert RGB model to HSV model. In OpenCV, H ∈ [0, 180), S ∈ [0, 255], V ∈ [0, 255]. We know that the H component can basically represent the color of an object, but the values of S and V must also be within a certain range, because S represents the degree of mixing of the color represented by H and white, that is, the smaller S is, The whiter the color, the lighter the color; V represents the mixing degree of the color represented by H and black, that is to say, the smaller the V, the darker the color. After experiments, the value of identifying blue is H from 100 to 140, and both S and V are between 90 and 255. The values of some basic colors H can be set as follows:：

```
Orange 0-22， Yellow 22- 38， Green 38-75， Blue 75-130， Violet 130-160， Red 160-179
```

Generally, the effective processing of the color space image is performed in the HSV space, and

then a strict range needs to be given for the corresponding HSV component in the basic color. The following is the fuzzy range calculated by experiment

H: 0 — 180

S: 0 — 255

V: 0 — 255

This part of the red is classified as the purple range here：

	Black	Grey	White	Red		Orange	Yellow	Green	Cyan	Blue	Purple
hmin	0	0	0	0	156	11	26	35	78	100	125
hmax	180	180	180	10	180	25	34；	77	99	124	155
smin	0	0	0	43		43	43	43	43	43	43
smax	255	43	30	255		255	255	255	255	255	255
vmin	0	46	221	46		46	46	46	46	46	46
vmax	46	220	255	255		255	255	255	255	255	255

Color	R	G	B	H	H_2	C	C_2	V	L	I	Y'_{601}	S_{HSV}	S_{HSL}	S_{HSI}
	1.000	1.000	1.000	n/a	n/a	0.000	0.000	1.000	1.000	1.000	1.000	0.000	0.000	0.000
	0.500	0.500	0.500	n/a	n/a	0.000	0.000	0.500	0.500	0.500	0.500	0.000	0.000	0.000
	0.000	0.000	0.000	n/a	n/a	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	1.000	0.000	0.000	0.0°	0.0°	1.000	1.000	1.000	0.500	0.333	0.299	1.000	1.000	1.000
	0.750	0.750	0.000	60.0°	60.0°	0.750	0.750	0.750	0.375	0.500	0.664	1.000	1.000	1.000
	0.000	0.500	0.000	120.0°	120.0°	0.500	0.500	0.500	0.250	0.167	0.293	1.000	1.000	1.000
	0.500	1.000	1.000	180.0°	180.0°	0.500	0.500	1.000	0.750	0.833	0.850	0.500	1.000	0.400
	0.500	0.500	1.000	240.0°	240.0°	0.500	0.500	1.000	0.750	0.667	0.557	0.500	1.000	0.250
	0.750	0.250	0.750	300.0°	300.0°	0.500	0.500	0.750	0.500	0.583	0.457	0.667	0.500	0.571
	0.628	0.643	0.142	61.8°	61.5°	0.501	0.494	0.643	0.393	0.471	0.581	0.779	0.638	0.699
	0.255	0.104	0.918	251.1°	250.0°	0.814	0.750	0.918	0.511	0.426	0.242	0.887	0.832	0.756
	0.116	0.675	0.255	134.9°	133.8°	0.559	0.504	0.675	0.396	0.349	0.460	0.828	0.707	0.667
	0.941	0.785	0.053	49.5°	50.5°	0.888	0.821	0.941	0.497	0.593	0.748	0.944	0.893	0.911
	0.704	0.187	0.897	283.7°	284.8°	0.710	0.636	0.897	0.542	0.596	0.423	0.792	0.775	0.686
	0.931	0.463	0.316	14.3°	13.2°	0.615	0.556	0.931	0.624	0.570	0.586	0.661	0.817	0.446
	0.998	0.974	0.532	56.9°	57.4°	0.466	0.454	0.998	0.765	0.835	0.931	0.467	0.991	0.363
	0.099	0.795	0.591	162.4°	163.4°	0.696	0.620	0.795	0.447	0.495	0.564	0.875	0.779	0.800
	0.211	0.149	0.597	248.3°	247.3°	0.448	0.420	0.597	0.373	0.319	0.219	0.750	0.601	0.533
	0.495	0.493	0.721	240.5°	240.4°	0.228	0.227	0.721	0.607	0.570	0.520	0.316	0.290	0.135

Next, we use Python-Opencv to actually operate, we edit the following code：

```
import cv2
import numpy as np  # Import library

font = cv2.FONT_HERSHEY_SIMPLEX # Font file

# Set the color interval
color_lower = np.array([100, 43, 46])  # The lower limit of the blue interval min
color_upper = np.array([124, 255, 255])  # Upper limit of blue interval max

cap = cv2.VideoCapture(0)  # Use opencv to get the camera
```

```
cap.set(3, 780)  # Set the width of the image (window) to 320 pixels
cap.set(4, 780)  # Set the height of the image (window) to 320 pixels
while 1:  # Enter the wireless loop
   ret, frame = cap.read()  # Take the picture captured by the camera as the value of the frame
cap.read() returns two values
   # print(ret)
   if ret:  # # The return value of ret is True or False, indicating whether the picture has been read
      frame = cv2.GaussianBlur(frame, (5, 5), 0)  # Gaussian filter GaussianBlur() makes the picture
blur
      hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)  # Convert the color gamut of the picture to
HSV style for detection
      # Use the inRange() function and the upper and lower bounds of the blue range in the HSV
model to obtain the mask. The blue part of the original video in the mask will be made white, and
the other parts are black.
      mask = cv2.inRange(hsv, color_lower, color_upper)  # Set the threshold, remove the
background and keep the set color
      mask = cv2.erode(mask, None, iterations=2)  # Display the image after corrosion
      mask = cv2.GaussianBlur(mask, (3, 3), 0)  # Gaussian Blur
      # Perform a bitwise and operation on the mask on the original video frame, and the white in
the mask will be replaced with a real image
      res = cv2.bitwise_and(frame, frame, mask=mask)  # Image merging

      cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
# Edge detection

      if len(cnts) > 0:  # Determine the position information of the identified object through edge
detection to obtain relative coordinates
         cnt = max(cnts, key=cv2.contourArea)
         # min Enclosing Circle
         (x, y), radius = cv2.minEnclosingCircle(cnt)
         cv2.circle(frame, (int(x), int(y)), int(radius), (255, 0, 255), 2)  # draw a circle
         # Use putText function to mark colors
         cv2.putText(frame, "blue", (int(x)+20, int(y)), font, 0.5, (0, 255, 0), 1)
         print(int(x), int(y))
      else:
         pass
      cv2.imshow('frame', frame)  # Show specific test results
      # cv2.imshow('mask', mask)
      # cv2.imshow('res', res)
      if cv2.waitKey(1) == ord('q'):  # Exit when the q key is pressed          break
# The next two sentences are normal operations. You need to set up like this every time when you
use the camera.
cap.release()
cv2.destroyAllWindows()  # it can easily delete any (all) windows that we create
```

If you don't understand the above program, it doesn't matter, we will show you step by step. Before the program starts, we first import the relevant library, and then start the color threshold setting. The threshold comes from the color classification in the above table.

Capture camera video

Before performing color detection, first we need to read the image of the camera; OpenCV provides a very simple interface for this application. Let's use the camera to capture a video. Let's start with this simple task. In order to get the video, you should create a VideoCapture object. Its parameter can be the index number of the device, or a video file. The device index number is to specify the camera to be used. The general laptop computer has a built-in camera. So the parameter is 0. You can select other cameras by setting it to 1 or others. After that, you can capture the video frame by frame. But in the end, don't forget to stop capturing the video.

```
import cv2
# 0 is the camera device number
cap = cv2.VideoCapture(0)

while True:
   # Capture frame by frame
   ret, frame = cap.read()
   # Display result frame
   cv2.imshow('frame', frame)
   # cv2.waitKey() is a keyboard binding function
   if cv2.waitKey(1) & 0xFF == ord('q'):
      break

# Release resources when everything is done
cap.release() # Release camera
cv2.destroyAllWindows()  # it can easily delete any window that we create
```

The function cap.read() returns a Boolean value (True/False). If the frame read is correct, it is True. So in the end, you can check whether the video file has reached the end by checking its return value. Sometimes cap may not be able to successfully initialize the camera device. In this case, the above code will report an error. You can use cap.isOpened() to check whether the initialization is successful. If the return value is True, there is no problem. Otherwise, use the function cap.open().

The function cv2.imshow() displays the image. The window will automatically adjust to the image size. The first parameter is the name of the window, and the second is our image. You can create as many windows as you like, but you must give them different names.

The function cv2.waitKey() is a keyboard binding function. It needs to be pointed out that its time scale is milliseconds. The function waits for a certain number of milliseconds to see if there is keyboard input. Within a few milliseconds, if any key is pressed, this function will return the ASCII code value of the key, and the program will continue to run. If there is no keyboard input, the return value is -1. If we set the parameter of this function to 0, it will wait for keyboard input indefinitely. It can also be used to detect whether a specific key is pressed, such as whether the key q is pressed.

Gaussian Blur

Next, let’s talk about the Gaussian blur, which is implemented by using the function cv2.GaussianBlur(). We need to specify the width and height of the Gaussian kernel (must be an odd number). And the standard deviation of the Gaussian function along the X and Y directions. If we only specify the standard deviation in the X direction, the Y direction will also take the same value. If both standard deviations are 0, then the function will calculate itself based on the size of the kernel function. Gaussian filtering can effectively remove Gaussian noise from the image.

```
# !/usr/bin/env python
# encoding: utf-8

import numpy as np
import cv2

cap = cv2.VideoCapture(0)

while True:
    # Capture frame by frame
    ret, frame = cap.read()
    # Display result frame
    cv2.imshow('frame', frame)
    # Use Gaussian blur function to process frame images
    new_frame = cv2.GaussianBlur(frame, (5, 5), 0)
    # Display blurred images
    cv2.imshow('GaussianBlur', new_frame)
    # cv2.waitKey() is a keyboard binding function
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release resources when everything is done
cap.release()
cv2.destroyAllWindows()  # it can easily delete any window we create
```

Color space conversion

There are more than 150 methods for color space conversion in OpenCV. But later you will discover that there are only two types that we often use: BGR⇄ Gray and BGR⇄ HSV. The function we will use is: cv2.cvtColor(inputimage, flag), where flag is the conversion type. For the conversion of BGR⇄ Gray, the flag we need to use is cv2.COLORBGR2GRAY. Also for the conversion of BGR⇄ HSV, the flag we use is cv2.COLOR_BGR2HSV.

Note: In the OpenCV HSV format, the value range of H (color/chroma) is [0, 179], the value range of S (saturation) is [0, 255], and the value range of V (brightness) [0,255]. But different software may use different values. So when you need to compare the HSV value of OpenCV with the HSV value of other software, you must remember to normalize.

```
# Convert the color gamut of the picture to HSV style for detection
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

Object tracking

Now that we know how to convert an image from BGR to HSV, we can use this to extract objects with a certain color. It is easier to represent a specific color in the HSV color space than in the BGR space. In our program, what we want to extract is a blue object. Here are the steps we need to do:

- Get each frame image from the video
- Convert images to HSV space
- Set HSV threshold to blue range
- Get the blue object, of course we can also do anything else what we want to do,

such as: draw a circle around the blue object.

Here is our code：

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while (1):

    # Capture frame by frame
    ret, frame = cap.read()
    # Color space conversion BGR→HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # Define the range of blue in the HSV
    lower_blue = np.array([110, 50, 50])
    upper_blue = np.array([130, 255, 255])
    # Threshold processing HSV image, only get blue
    mask = cv2.inRange(hsv, lower_blue, upper_blue)
    # Image bitwise and operation
    res = cv2.bitwise_and(frame, frame, mask=mask)
    cv2.imshow('frame', frame)
    cv2.imshow('mask', mask)
    cv2.imshow('res', res)
    if cv2.waitKey(1) == ord('q'):  # Exit when q key is pressed
        break


cap.release()
cv2.destroyAllWindows()
```

Note: This is the simplest method in object tracking. After you learn the outline, you will learn more related knowledge, that is, you can find the center of gravity of the object, and track the object according to the center of gravity, just wave your hand in front of the camera to draw different graphics, or other More interesting things.

Contours in OpenCV

The contour can be simply thought of as a curve connecting continuous points (connected to the boundary) together, with the same color or grayscale. Contours are useful in shape analysis and object detection and recognition.

• To be more accurate, use a binary image. Before looking for the contour, thresholding or Canny boundary detection is required.

• The function for finding contours will modify the original image. If you want to use the original image or image after finding the contour, you should store the original image in other variables.

• In OpenCV, finding contours is like super white objects on a black background. You should remember that the object which you are looking for should be white and the background should be black. Let us see how to find contours in a binary image: the function cv2.findContours() has three

parameters, the first is the input image, the second is the contour retrieval mode, and the third is the contour approximation method. There are three return values, the first is the image, the second is the contour, and the third is the (contour) tomographic structure. The contour (the second return value) is a Python list that stores all the contours in this image. Each contour is a Numpy array containing the coordinates of the boundary point (x, y) of the object.

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while (1):

   # Capture frame by frame
   ret, frame = cap.read()
   # Color space conversion BGR→HSV
   hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
   # Define the range of blue in the HSV
   lower_blue = np.array([110, 50, 50])
   upper_blue = np.array([130, 255, 255])
   # Threshold processing HSV image, only get blue
   mask = cv2.inRange(hsv, lower_blue, upper_blue)
   # Image bitwise and operation
   res = cv2.bitwise_and(frame, frame, mask=mask)
   cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]  #
Find contour
   if len(cnts) > 0:  # cnts is the return value of the cv2.findContours function, which is 3 in the new
version of CV2
      # The area of the contour can be calculated using the function cv2.contourArea() and returns
the maximum value in cnt (provided that the maximum value after cv2.contourArea is processed)
      cnt = max(cnts, key=cv2.contourArea)
      # min Enclosing Circle
      (x, y), radius = cv2.minEnclosingCircle(cnt)
      cv2.circle(frame, (int(x), int(y)), int(radius), (255, 0, 255), 2)  # Draw a circle
      # cv2.circle(mask, (int(x), int(y)), int(radius), (255, 0, 255), 2)  # Draw a circle
      # cv2.circle(res, (int(x), int(y)), int(radius), (255, 0, 255), 2)  # Draw a circle
   cv2.imshow('frame', frame)
   # cv2.imshow('mask', mask)
   # cv2.imshow('res', res)
   if cv2.waitKey(1) == ord('q'):  # Exit when q key is pressed
      break

cap.release()
cv2.destroyAllWindows()
```

The function cv2.minEnclosingCircle gets the smallest circle that covers this contour, and its return value is the center (indicated by the coordinate axis) and the radius of the circle. Then we draw a circle. The cv2.circle function accepts 5 parameters, which are the image (Draw a circle on

that image), the coordinates of the center of the circle, the radius of the circle, the color of the line that used to draw the circle, and the thickness of the line.

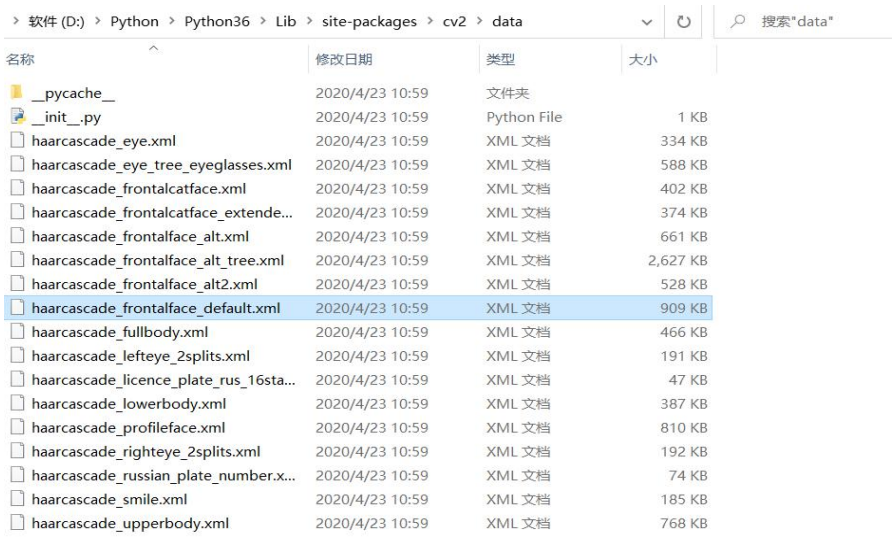
At this point, you can go back to Chapter 6.1 and start to look at the color detection code, which is relatively easy to read.

In fact, color detection is an extremely simple Opencv application. Everyone is unfamiliar with the code and can't understand the program because they don't know the function usage of Opencv. If you want to learn how to use Opencv, you still need to work hard to study related materials. . You can purchase related Python Opencv books. The explanation materials in this chapter are all from the Opencv-Python tutorial. The information can refer to the following address :https://www.cnblogs.com/Undo-self-blog/p/8423851.html

6.2 Face detection

Let me talk about the training data acquisition of face detection. In the experiment, we are going to use pre-trained XML files. These XML files are more difficult to train, but we don't need to worry, because OpenCV has already provided us with a lot of face detection related Classifier, to use these classifiers, we need to copy the classifier XML file haarcascade_frontalface_default.xml from the opencv folder /sources/data/haarcascades/ to our project directory, which is the directory where we will write the program. If there is no opencv folder /sources/data/haarcascades/, you can try to find the opencv folder /share/OpenCV/haarcascades/.

Here is an example of opencv installed on Windows, the path of haarcascadefrontalfacedefault.xml is as follows：



The next step is also to get the camera picture, which has been mentioned in Chapter 6.1, and pre-load the classifier

```
import cv2

cap = cv2.VideoCapture(0)
# Load face detection classifier
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml.xml')
while True:
   # Capture frame by frame
```

```
    ret, frame = cap.read()
    # Display result frame
    cv2.imshow('frame', frame)
    # cv2.waitKey() is a keyboard binding function
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release resources when everything is done
cap.release()  # Release camera
cv2.destroyAllWindows()  # it can easily delete any window we create
```

Next, we will first convert the picture to a grayscale picture, the purpose is to reduce the amount of computer calculations

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Then we can start face detection. The whole process is to recognize the new picture according to the training data.

```
faces = detector.detectMultiScale(gray, 1.3, 5)
```

In the above code, the value of faces will be a list, each of which has 4 variables x, y, height, and width. The list represents the detected faces, that is, the size of the list is the number of faces, and the position of each face in the picture is (x, y, height, width).

In order to let us see the detection results more intuitively, we frame these faces:

```
import cv2

cap = cv2.VideoCapture(0)
cap.set(3, 320)
cap.set(4, 320)

# Load face detection classifier
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml.xml')
while True:
    # Capture frame by frame
    ret, frame = cap.read()
    # Grayscale processing
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)
    # print(faces)
    for (x, y, w, h) in faces:
        # Draw a frame and circle it
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)

    # Display result frame
    cv2.imshow('frame', frame)
        # cv2.waitKey() is a keyboard binding function
            if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release resources when everything is done
```

```
cap.release()  # Release camera
cv2.destroyAllWindows()  # it can easily delete any window we create
```

This is a complete face detection case.

6.3 PID algorithm to control the servo

PID Control Principle and Characteristics

In engineering practice, the most widely used regulator control law is proportional, integral, and differential control, referred to as PID control, also known as PID regulation. The PID controller has a history of nearly 70 years since its inception. It has become one of the main technologies of industrial control due to its simple structure, good stability, reliable work, and convenient adjustment. When the structure and parameters of the controlled object cannot be fully grasped, or an accurate mathematical model cannot be obtained, and other techniques of control theory are difficult to adopt, the structure and parameters of the system controller must be determined by experience and on-site debugging. PID control technology is the most convenient. That is, when we do not fully understand a system and the controlled object, or cannot obtain system parameters through effective measurement methods, PID control technology is most suitable. PID control, PI and PD control are also in practice. PID controller is based on the error of the system, using proportional, integral, and derivative to calculate the control quantity for control.

Proportional (P) controls

Proportional control is the simplest control method. The output of the controller is proportional to the input error signal. When there is only proportional control, the system output has a steady-state error.

Integral (I) control

In integral control, the output of the controller is proportional to the integral of the input error signal. For an automatic control system, if there is a steady-state error after entering the steady state, it is said that the control system has a steady-state error or simply a system with a Steady-state Error. In order to eliminate the steady-state error, an "integral term" must be introduced in the controller. The error of the integral term depends on the integral of time. As time increases, the integral term will increase. In this way, even if the error is small, the integral term will increase with the increase of time, which pushes the controller's output to increase so that the steady-state error is further reduced until it equals zero. Therefore, the proportional + integral (PI) controller can make the system have no steady-state error after entering the steady state.

Differential (D) control

In derivative control, the output of the controller is proportional to the derivative of the input error signal (that is, the rate of change of error). The automatic control system may oscillate or even lose stability during the adjustment process to overcome the error. The reason is that due to the presence of large inertial components (links) or delay (delay) components, which have the effect of suppressing errors, their changes always lag behind the changes of errors. The solution is to make the change of the effect of suppressing the error "leading", that is, when the error is close to zero, the effect of suppressing the error should be zero. That is to say, it is not enough to introduce only the “proportional” term in the controller. The function of the proportional term is only to amplify the magnitude of the error. At present, the “differential term” needs to be added, which can predict the

trend of error changes. In this way, the controller with proportional + derivative can make the control effect of suppressing error equal to zero or even negative in advance, thereby avoiding serious overshoot of the controlled quantity. Therefore, for the controlled object with greater inertia or lag, the proportional + derivative (PD) controller can improve the dynamic characteristics of the system during the adjustment process.

When performing color detection or face detection, we can make the servo PTZ rotate with the color (or face) recognized in the camera screen rotate. Here we actually use the PID algorithm to follow, but we won't talk about PID in detail here. , If you are interested in PID algorithm, you can come down and learn about it yourself. In university courses, we will also learn PID algorithm.

PID commonly used formulas:

Find the best parameter setting, check in order from small to large, first proportional and then integrating, and finally adding the differential. The curve oscillates frequently, the proportional dial should be enlarged, the curve floats around the big bay, the proportional dial is pulled down, and the curve deviation is restored slow, the integration time decreases, the curve fluctuation period is longer, the integration time is longer, the curve oscillation frequency is fast, the differential is lowered first, the fluctuation is slow when the momentum is large, the differential time should be longer, the ideal curve has two waves,The front is high and the back is low 4 to 1, the first is look and the second is adjustment and more analysis, the adjustment quality will not be low.

Here we take the face camera follow as an example, you can see that our code contains the following paragraph:

```
# Endless loop
while True:
    ret, frame = cap.read()  # Get camera video stream
    if ret == 1:  # Determine whether the camera is working
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # First convert each frame into a grayscale image, and search in the grayscale image
        faces = face_cascade.detectMultiScale(gray)  # Find Face
        if len(faces) > 0:  # When someone's face in the video
            # print('face found!')
            for (x, y, w, h) in faces:
                # Parameters are "target frame "," rectangle "," rectangle size "," line color "," width"
                cv2.rectangle(frame, (x, y), (x + h, y + w), (0, 255, 0), 2)
                result = (x, y, w, h)
                x_middle = result[0] + w / 2  # x axis centre
                y_middle = result[1] + h / 2  # y axis centre

                X_pid.update(x_middle)  # Put the X axis data into the pid to calculate the output value
                Y_pid.update(y_middle)  # Put the Y axis data into the pid to calculate the output value
                # print("X_pid.output=%d"%X_pid.output)      #Print X output
                # print("Y_pid.output=%d"%Y_pid.output)      #Print Y output
                angle_X = math.ceil(angle_X + 1 * X_pid.output)  # Update the servo angle of the X axis, use the last servo angle plus a certain percentage of the incremental value to update the servo angle
                angle_Y = math.ceil(angle_Y + 0.8 * Y_pid.output)  # Update the servo angle of the Y axis, use the last servo angle plus a certain percentage of incremental value to update the servo angle
```

```
        # print("angle_X-----%d" % angle_X) #Print X axis servo angle
        # print("angle_Y-----%d" % angle_Y) #Print Y axis servo angle
        if angle_X > 180:  # Limit X axis maximum angle
            angle_X = 180
        if angle_X < 0:  # Limit X axis minimum angle
            angle_X = 0
        if angle_Y > 180:  # Limit Y axis maximum angle
            angle_Y = 180
        if angle_Y < 0:  # Limit Y axis minimum angle
            angle_Y = 0
        servo.set(servo_X, angle_X)  # set X axis servo
        servo.set(servo_Y, 180 - angle_Y)  # set Y axis servo
```

Let's talk about the cv2.rectangle function in Python opencv, which is a function to draw a rectangle on an image; to draw a rectangle, we need to know the upper left vertex and the lower right vertex of the rectangle, as shown in the following figure:：

```
import cv2
cv2.rectangle(img, (x1, y1), (x2, y2), (255,0,0), 2)

x1,y1 ------
|       |
|       |
|       |
--------x2,y2
```

This is an official example. We can see that x1, y1, x2, y2 represent the coordinates of the upper left corner and the lower right corner of the graph we draw. Knowing the coordinates of these two points, we can calculate the center point of X axis and Y axis what we want. . As our faces continue to move, this center point will continue to change. We only need to calculate this offset, plus the last coordinate value of the X-axis servo, then we can know how many degrees the gimbal should rotate on the X-axis, and Y-axis is the same.

Here we will no longer explain to you how to use the PID algorithm, we only teach you how to use the PID algorithm that we wrote. Everyone will learn the use of PID algorithm in detail in university courses.

The update method in PID receives a parameter value. Here we should pass in the value of the coordinate axis that needs to be calculated. After passing in, this method will calculate the value of self.output. This value represents the calculated gain value (It may be positive or negative), so when we use this method, we can call it directly through the object

```
X_pid.update(x_middle)  # Put the X axis data into the pid to calculate the output value
Y_pid.update(y_middle)  # Put the Y axis data into the pid to calculate the output value
```

The angle value of the previous time plus the calculated gain value * weight value is used as the angle value of the next servo. Repeated operation of this process will form a closed loop adjustment so that the gimbal will always follow the face. The value of the weight value can be adjusted and optimized manually. Here 1 and 0.8 are the best values obtained according to the adjustment of this program. The adjustment of the weight value is mainly due to the different movable ranges of the X-axis and Y-axis of the gimbal, and the second is that the pixel lengths of the X and Y-axis of the

camera output image are different (640*480)

6.4 QR code recognition

QR code recognition case

Here, we will first teach you how to complete the QR code recognition. In the next experiment, we will teach you how to drive the car through the QR code.

First of all, we have to install the library opencv and QR code parsing library pyzbar that we need for this development.

```
pip install opencv-python
pip install pyzbar
# If it under linux you need to install
sudo apt-get install libzbar0
```

Next, we will start to code, the code is relatively simple, let's do it step by step

1. Import related libraries

```
import cv2  # opencv library
from pyzbar import pyzbar   #QR code analysis library
```

2. Get camera video via opencv

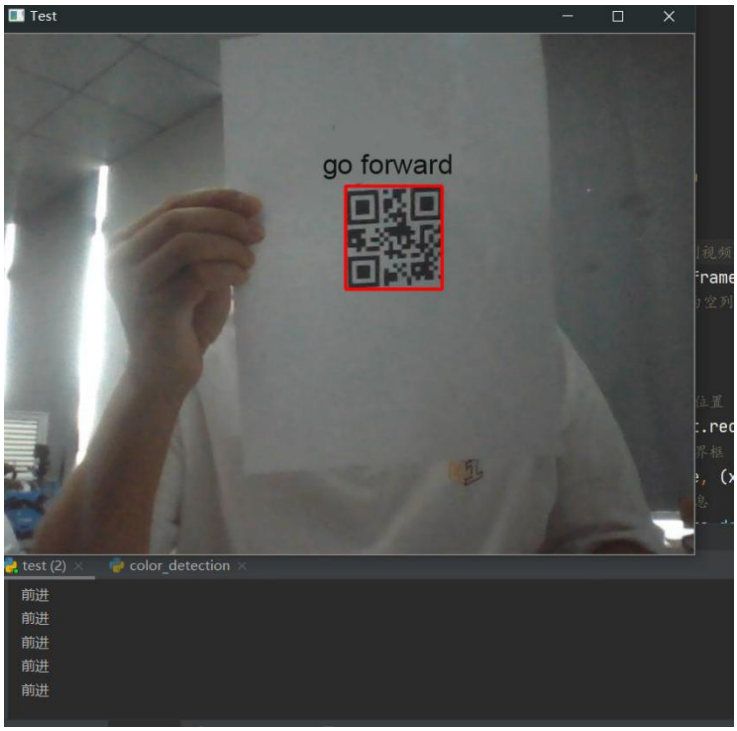
```
# Get the camera device number
capture = cv2.VideoCapture(0)
```

3.Next, we need to always read the QR code information in the picture which is taken by the camera, so we first need to enter the wireless loop, and then use the camera that just instantiated to collect real-time photos, and then use the pyzbar function to analyze whether The iQR code is the in the picture . If there is no QR code in the screen, the tests will be an empty list, so the for will not be executed.

The above is a program for individual QR code recognition, you can look at the test of the running result:

```
import cv2
from pyzbar import pyzbar

# Get the camera device number
capture = cv2.VideoCapture(0)
# The wireless loop keeps reading the images of the camera screen
while True:
    # read frame by frame
    ret, frame = capture.read()
    # Whether the camera video is successfully obtained
    if ret:
        # pyzbar.decode can get the QR code in the video
        tests = pyzbar.decode(frame)
        # If there is no QR code, tests is an empty list
        # print(tests)
        # Empty lists will not execute for loop
        for test in tests:
            # Extract the position of the bounding box of the barcode
            (x, y, w, h) = test.rect
            # Draw the bounding box of the barcode in the image
```

```
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
        # Read the parsed QR code information
        testdate = test.data.decode('utf-8')
        print(testdate)
    cv2.imshow('Test', frame)
    if cv2.waitKey(1) == ord('q'):
        break
# If the camera information is not obtained, output an error message
else:
    print('Open Camera Error!')
```



Use a QR code to drive the motor

In the previous experiment, we already know that if the QR code information is recognized by the camera, then this section of the experiment will use the QR code information to drive the motor.

Here, we provide ideas, the most important thing is to let everyone understand how to achieve the function.

We first write the following code:

```
import cv2
from pyzbar import pyzbar


def movement(status):
    if status == 'forward':
        # Call the corresponding motor to run function
        print('The car is going forward')
    elif status == 'back':
```

```
        print('The car is going back')
    elif status == 'turn left':
        print('The car is turning left')
    elif status == 'turn right':
        print('The car is turning right')
    else:
        print('stop')


def detect():
    # Get the camera device number
    camera = cv2.VideoCapture(0)

    while True:
        # Read current frame
        ret, frame = camera.read()
        if ret:
            # change to grayscale image
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            tests = pyzbar.decode(gray)
            # If there is no QR code, tests is an empty list
            # print(tests)
            if tests:
                # Empty lists will not execute for loop
                for test in tests:
                    (x, y, w, h) = test.rect
                    cv2.rectangle(gray, (x, y), (x + w, y + h), (0, 0, 255), 2)
                    # Read the parsed QR code information
                    testdate = test.data.decode('utf-8')
                    # passes the recognition result as a parameter to the movement function
                    movement(testdate)
            # When the QR code is not recognized
            else:
                # the car is stop
                movement('error')
            cv2.imshow("qrcode", gray)

            if cv2.waitKey(1) & 0XFF == ord('q'):  # Press q to exit is detected            break
        # If the camera information is not obtained, output an error message      else:
            print('Open Camera Error!')
    camera.release()
    cv2.destroyAllWindows()


detect()
```

In the above function, two functions are created, one is the motor call function movement and the QR code recognition function detect. In the QR code recognition function, we pass the recognition result as a parameter to the movement function; It can complete the QR code

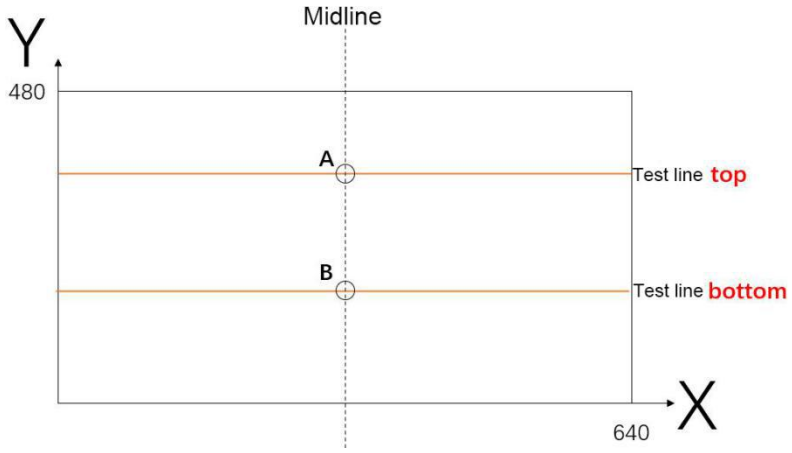
recognition call by using two functions, but the function called by the motor is not listed here, and usd print to instead it. You can try it yourself.

6.5 Visual inspection

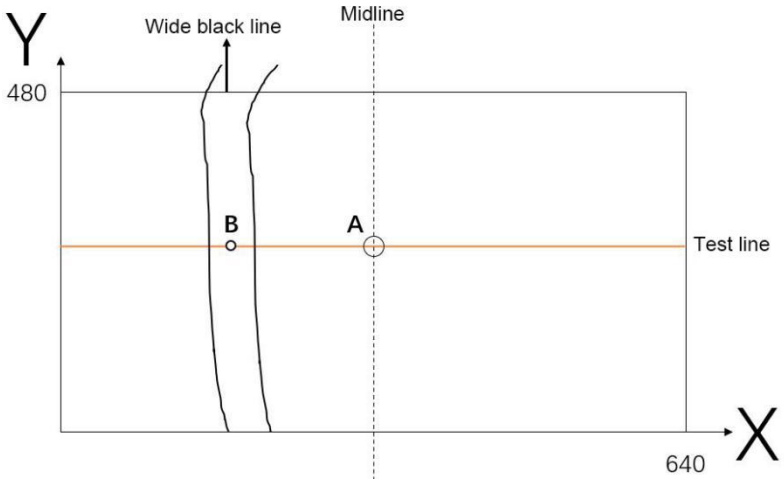
Principle of visual inspection

In this section we won't talk about the code, but let's talk about the principle of how visual walkthroughs work.

Known: The camera has a resolution of 640 x 480 pixels, so we can get the following information:



As can be seen from the diagram, the x-axis coordinates of point on top of the detection line are 320. Similarly, the x-axis coordinates of point B on bottom of the detection line are 320. Why do we need a detection line? We need to determine the actual line detection line, the camera screen black line deviation, we continue to use a picture to illustrate, as shown in the following figure:



We know that the x-axis coordinate of point a is 320, if we can calculate the x-axis coordinate of point B in real time (point B is the midpoint of the black line) , then compare the value of the x-axis of the two points to make the car turn left and right and execute; The condition for going straight is A = B on the X axis, if B > A, as shown in figure, the black line is on the left side of the frame. To make the black line move to the center, you need to move the car to the left (turn left). If B > A , you need to move the car to the right (turn right)

Problem solved

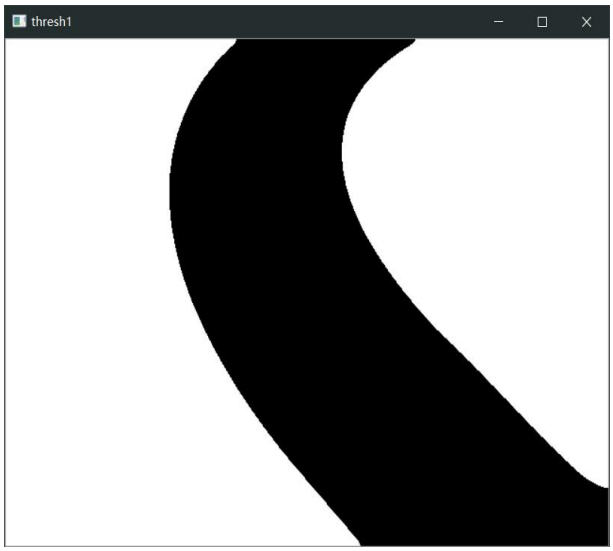
So the question is, how do we compute the coordinates of point B? If we can solve this problem, it seems to make it easier, doesn't it? Images can be binarized in opencv, which offers several forms of binarization:

cv2.THRESH_BINARY
cv2.THRESHBINARYINV
cv2.THRESH_TRUNC
cv2.THRESH_TOZERO
cv2.THRESHTOZEROINV

In grayscale images, 0-255 represents the brightness level, 0 for black and 255 for White. Standard binarization is based on the set brightness threshold, above this threshold will be assigned to the point of White 255, below the Threshold of black assigned to 0.

We do black line patrol, using the normal binary CV2. THRESHBINARY.If you want to train a white line, just change it to anti-color mode CV2. THRESHBINARY converts white lines into black dots without changing the program.

Here's the binary image:



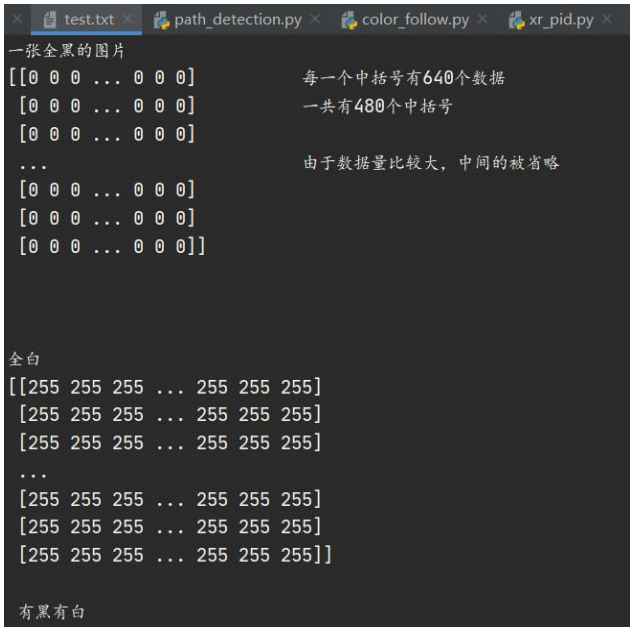
This is what it looked like:

Binary Processing Video Code:

```
import cv2

# 0 is the camera device number
cap = cv2.VideoCapture(0)

while True:
      # Frame by frame capture
        ret, frame = cap.read()
  if ret:
    # To get a grayscale image
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Gaussian blur
    Gauss_frame = cv2.GaussianBlur(gray, (19, 19), 0)
    # Binarization of gray-scale image
    ret, thresh1 = cv2.threshold(Gauss_frame, 120, 255, cv2.THRESH_BINARY)
    # print(type(thresh1))
    # X Axis is 5, Y is 300 pixels
    print(thresh1[300, 5])
    # The threshold 1 is 480
    # print(len(thresh1))
    # with open('./test.txt', 'a') as tmp:
    #     print(thresh1, end='\t', file=tmp)
    # Display result frame
    cv2.imshow('thresh1', thresh1)

  # CV2. waitKey () is a keyboard binding function
  if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# When it's done, release the resource
cap.release()  # Release the camera
```

```
cv2.destroyAllWindows()  # Any window we create can be easily deleted
```

After we got the image , we had to calculate the X axis value of point B, and we continued to test it with the code, and we knew that the image we saw on the PC was beautiful, but at the bottom of the computer, it's all numbers, and using binarization in the code, we also set brightness above 120 to 255 and Brightness Below 120 to 0. So we can see that we are printing the data threshold 1 with a length of 480 and a type of numpy. Ndarray, the 480 here is exactly 480 pixels across our entire image, and that's exactly what happened.

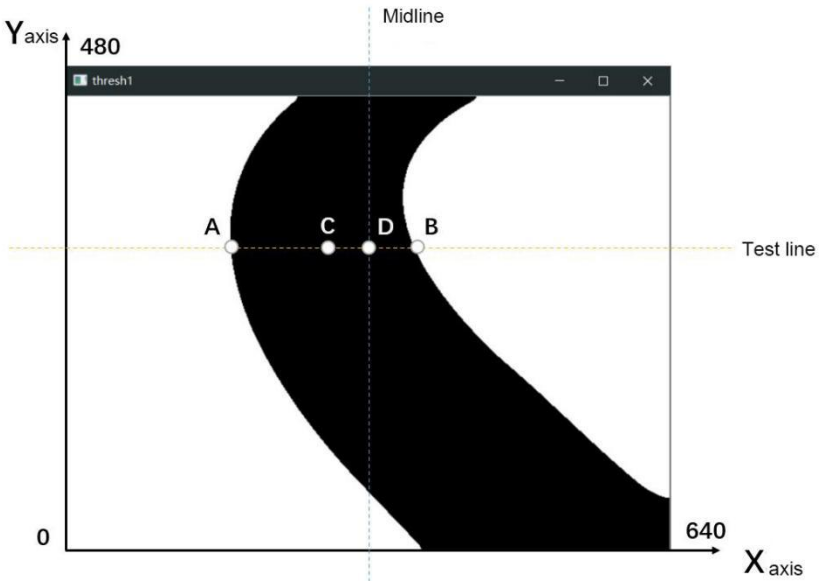


So when we go to the Y axis of 300 pixels, we should do something like this:

```
# X Axis is 5, Y is 300 pixels
print(thresh1[300, 5])
# X is the pixel point on the x axis, and the Y axis is fixed at 300
print(thresh1[300, x])
```

Tips: Numpy. Ndarray is an array, Python itself is not an array, here is the support of the NumPy library, if you are interested in it,you can learn NumPy.

As we continue our theoretical exploration, we get a binary image where the black wire has a width, as shown in the image below:



We know the x-axis coordinates of point a and point B, then we can find the x-axis coordinates of the midpoint C between AB and then compare it with the Midpoint D of the detection line, so it comes back to the above scenario: if C is to the left of D then let the picture shift to the left (cart left) , and if C is to the right of D then let the picture shift to the right (cart right) .

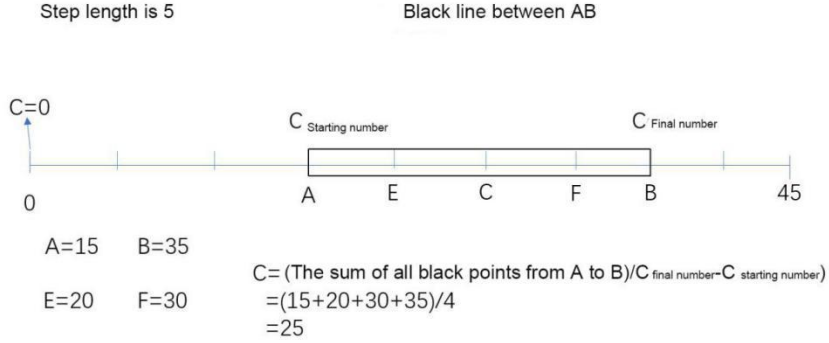
The next problem is the value of the x axis at AB. If we take the Y axis of the detection line to be 300, then there will be 640 pixels in the row of 300, so to determine if a dot is a black dot, let's say it's a black dot at 5 pixel intervals and code it like this:

```
# Sampling Pixels, 5 for the step, a total of 128 points, 640 for the maximum length of the image (640 x 480 pixels)
for j in range(0, 640, 5):
        # If the Pixel here is 0, it is black
        if thresh1[300, j] == 0:
           pass
```

We knew there would be a margin for error, but it was negligible. In this way we can get the approximate coordinates of A and B on the x axis, but how do we determine the midpoint between AB?

We can set up a counter C, the counter starts at 0, gets a black point once and then the counter adds itself to 1, then the midpoint of Ab can be calculated using the following formula:

(The sum of all X-axis points from Ax to Bx)/(C final number-C starting number)



However, there is a problem if we set the initial value of the counter to 0. If there is no black dot on the screen and it is all white, the divisor can not be 0 in Python or we will get an error, if we set the counter to 1, the code should look something like this:

```
# Counter (initially 0, Error Possible: 0 Can Not Be Divisor)
count = 1
    # Sample pixels, 5 for step, 128 total points
    for j in range(0, 640, 5):
      # The if judgment statement is entered only if the Pixel is black
      if thresh1[300, j] == 0:
        # Sum of coordinates of black pixels
        Path_Dect_px_sum = Path_Dect_px_sum + j
        # Sum The number of black pixels
        Path_Dect_fre_count = Path_Dect_fre_count + 1
    # The center point of the Black Pixel is the coordinates and the number divided by
    Path_Dect_px_top = Path_Dect_px_sum / Path_Dect_fre_count
```

In this way, we can roughly get the midpoint of the black line in the detection line, and finally compare with the midpoint of the detection line. This is the whole line of thought.

In actual coding, we take two detection lines to minimize the noise of error cases:

```
while True:
    ret, frame = cap.read()  # capture frame_by_frame
    if ret:
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # 获取灰度图像
        ret, thresh1 = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)  # 对灰度图像进行二值化
        Path_Dect_fre_count = 1
        for j in range(0, 640, 5):  # 采样像素点，5为步进，一共128个点
            if thresh1[300, j] == 0:
                Path_Dect_px_sum = Path_Dect_px_sum + j  # 黑色像素点坐标值求和
                Path_Dect_fre_count = Path_Dect_fre_count + 1  # 黑色像素点个数求和
        Path_Dect_px_top = Path_Dect_px_sum / Path_Dect_fre_count  # 黑色像素中心点为坐标和除以个数
        Path_Dect_px_sum = 0

        Path_Dect_fre_count = 1
        for j in range(0, 640, 5):  # 采样像素点，5为步进，一共128个点
            if thresh1[160, j] == 0:
                Path_Dect_px_sum = Path_Dect_px_sum + j  # 黑色像素点坐标值求和
                Path_Dect_fre_count = Path_Dect_fre_count + 1  # 黑色像素点个数求和
        Path_Dect_px_bottom = Path_Dect_px_sum / Path_Dect_fre_count  # 黑色像素中心点为坐标和除以个数
        Path_Dect_px_sum = 0

        cv2.imshow('BINARY', thresh1)  # 树莓派桌面显示二值化图像，比较占资源默认注释掉调试时可以打开
        if cv2.waitKey(1) & 0XFF == ord('q'):  # 检测到按键q退出
            go.stop()
            break
cap.release()
cv2.destroyAllWindows()
```

The Y-axis takes the test line of 300

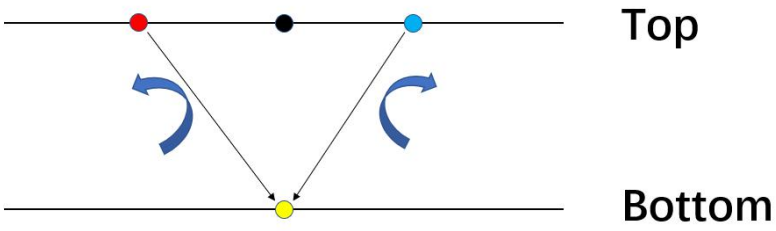
The Y axis takes the test line of 160

Here is the final call function:

```
def PathDect(func):
  # global Path_Dect_px  # The coordinate value of the center point of the survey line
  while True:
    # The difference between the black midpoint at the vertex and the black midpoint at the
bottom
    dx = Path_Dect_px_top - Path_Dect_px_bottom
    # Mean value of black line midpoint of two detection lines
    mid = int(Path_Dect_px_top + Path_Dect_px_bottom) / 2

    print("dx==%d" % dx)
```

```
    print("mid==%s" % mid)

    if (mid < 260) & (mid > 0):  # If the center point of the line is off to the left, you need to turn
left to correct it.
        print("turn left")
        go.left()
    elif mid > 420:  # If the center point of the line is off to the right, you need to turn right to
correct it.
        print("turn right")
        go.right()
    else:  # If the center point of the line is centered, you can go straight.
        if dx > 30:
            print("turn left")
            go.left()
        elif dx < -45:
            print("turn right")
            go.right()
        else:
            print("go stright")
            go.forward()
    time.sleep(0.007)
    go.stop()
    time.sleep(0.007)
```

In the function, mid is the average value of the midpoint of the black line pass, as mentioned above, and the average value is taken to reduce the error. Then the DX is the difference between the midpoint of the top and bottom detection lines. How do you understand that? Take a look at this image:

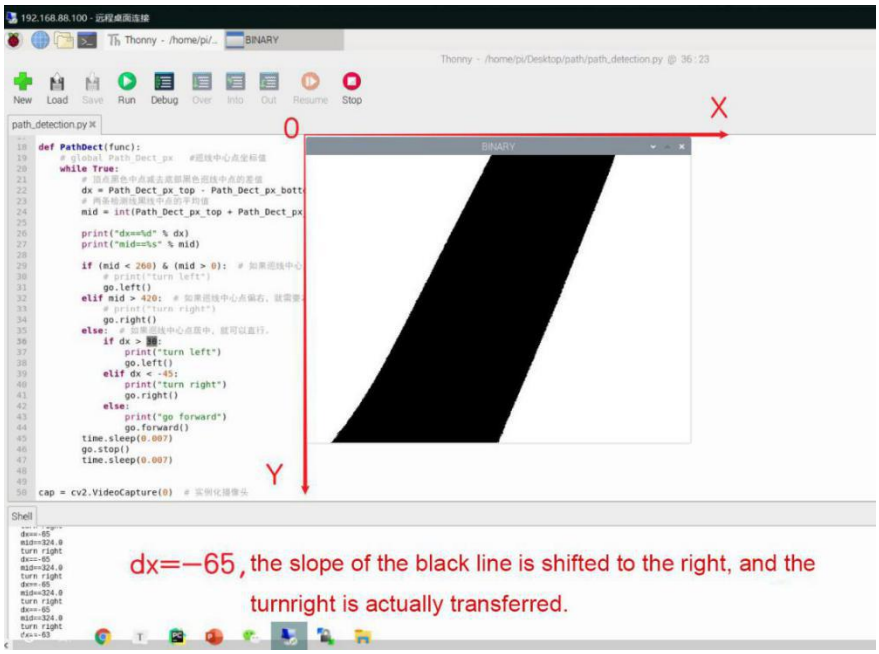


1. If the X-axis value of the red dot is minus the x-axis value of the yellow dot is negative, then the whole line (the black line) is moving to the left, so the left turn should be called.

2. The fact that the x-axis value of the blue dot minus the x-axis value of the yellow dot is positive indicates that the whole line (the black line) is moving to the right, so a right turn should be called.

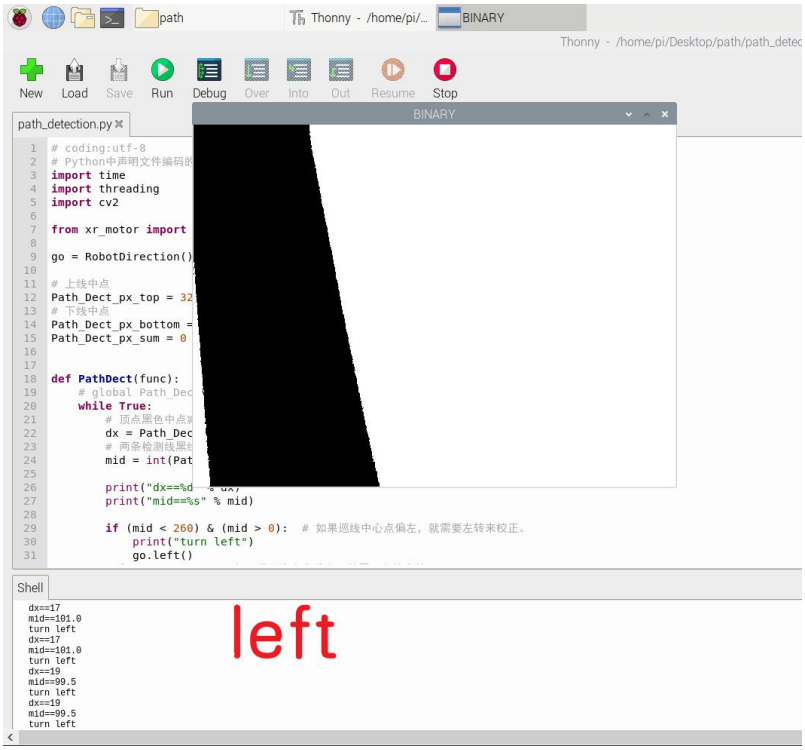
Someone saw this and couldn't wait to see the code ,but may see DX is negative case we call the right turn, this is why?

Take a look at the following illustration:



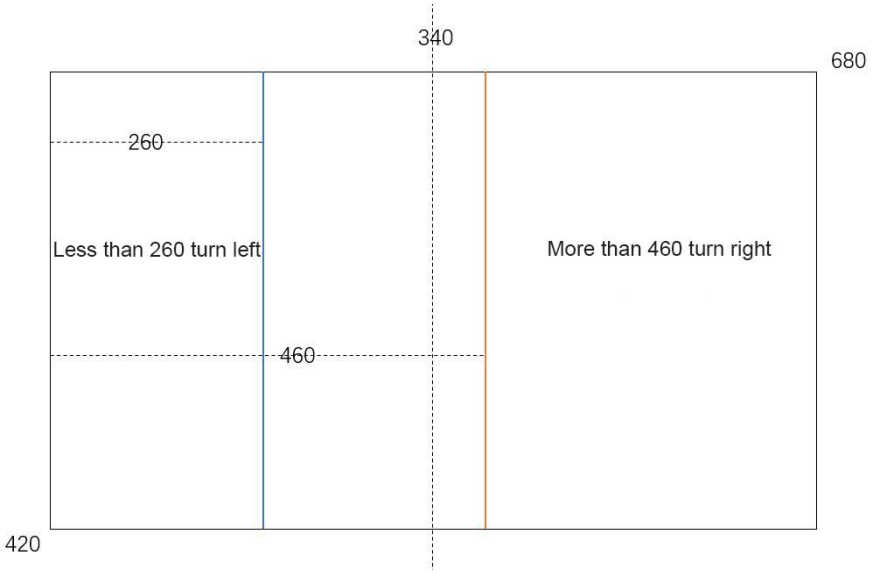
In figure we can see that the black line does shift to the right and the function calls turn right, but we can see that the actual result for dx is negative.

Keep looking at the image below:



So these two graphs confirm that the origin is indeed in the upper left-hand corner.

When we make the call to turn left and right, we divide the screen into three areas. The black lines all call to turn left within 260, the area greater than 460 calls to turn right, and then call the corresponding left and right turns in the case of judging the DX value in the middle.



So much for the overall visual walkthrough. In the experiment, calling the camera took up a lot of raspberry pi resources (and sometimes caused the program to die) , so we didn't show CV2 in the actual call.

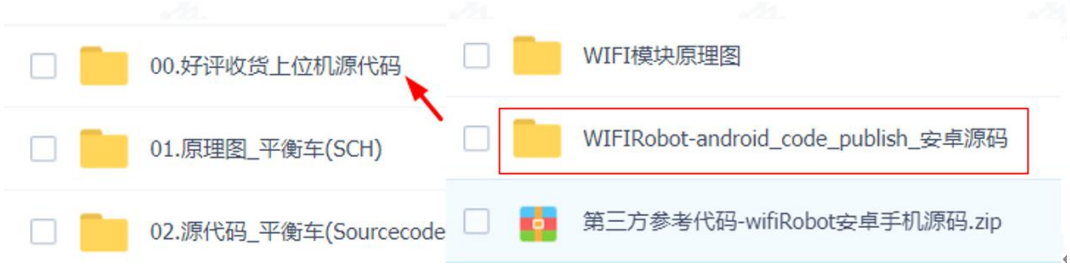
Chapter 7 Upper computer system learning

7.1 Analysis of Control Software Architecture for Android

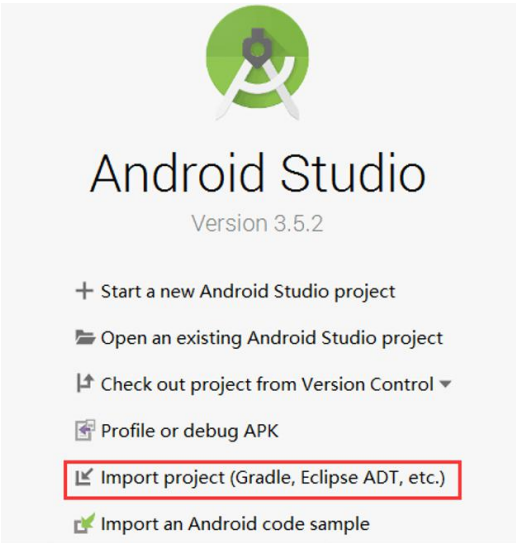
We provide the source code of the Android control software, is the basic version. It contains the basic car motion attitude control, video transmission and other functions. The development software we originally used was Eclipse, and the development language was Java. Since Google has officially released the Android Studio (AS) development tool, and it is much easier to develop on AS, we will translate the Eclipse project source code into AS for this chapter. How to install AS and configure Java development environment , please search it on Baidu.

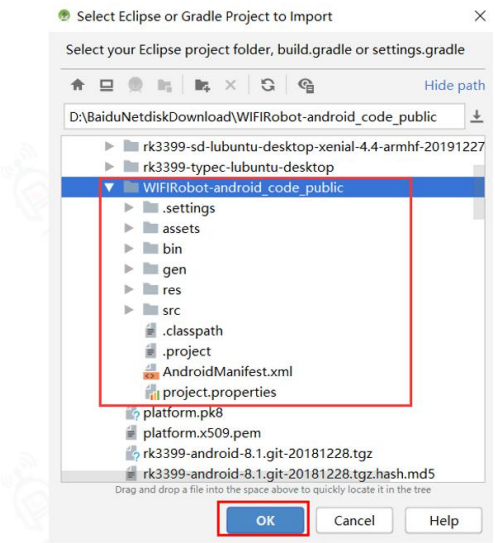
7.1.1 Eclipse to AS project

First in our network disk material, find the corresponding Android source code, and then download.



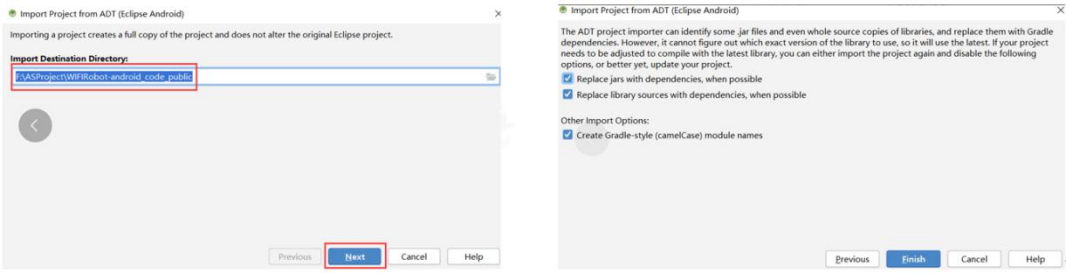
After downloading the source code, open AS, in the start interface to select the option of the red box below the figure, and then select the already downloaded Eclipse project source code, Click OK.





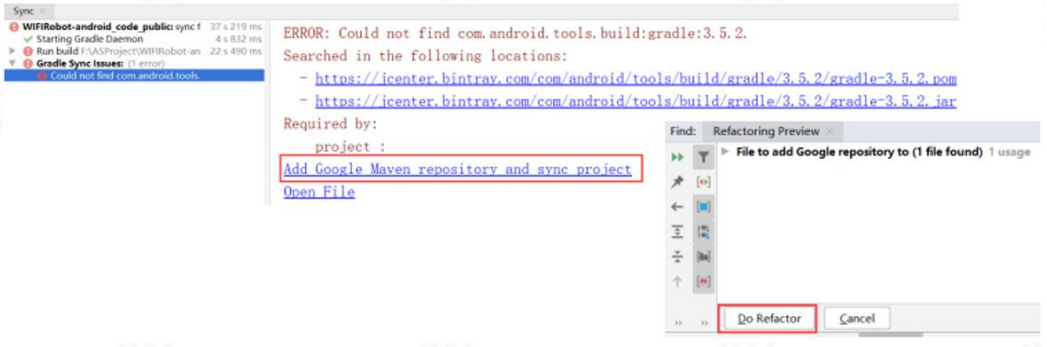
Import the Eclipse project

Next, select the default configuration for the import, such as the project folder path after the import and some of the recommended default configurations, and Click Finish to complete the import of the project.

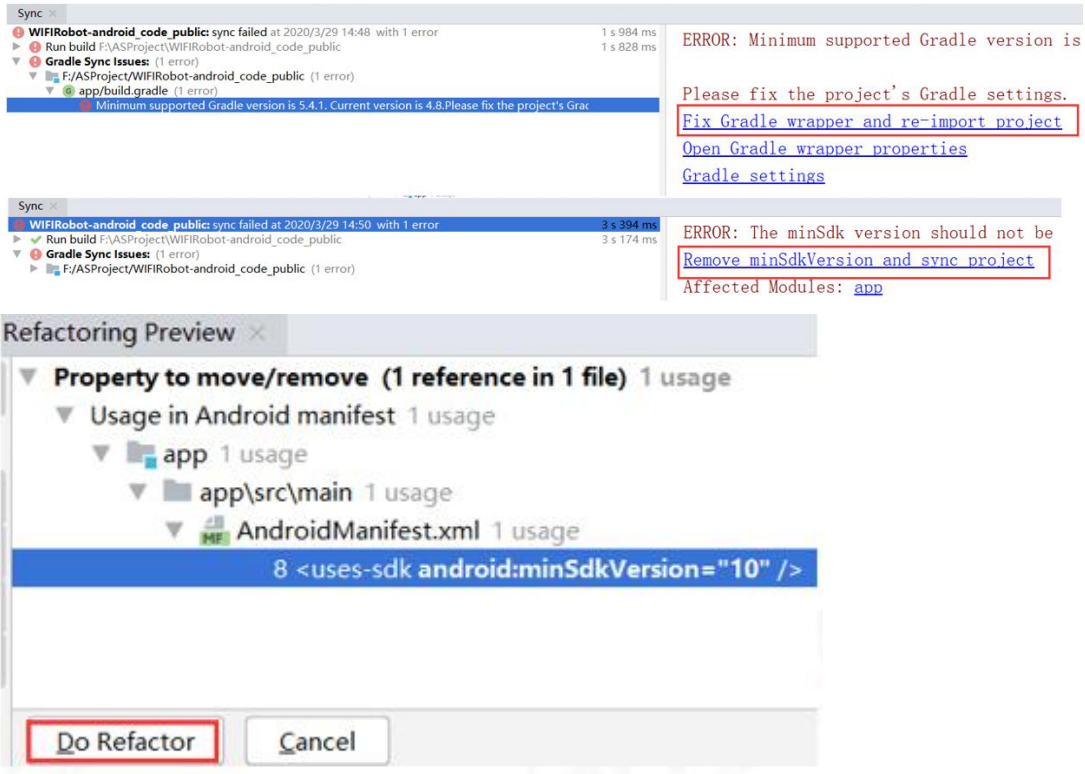


Import some default configurations for the Eclipse Project.

After completing the above steps, AS sync, the SYNC project, which may return some errors, we can follow the red box prompt in the figure below.



Add the Google Maven repository and sync the project .



Fix Gradle and Reimport the project as well as remove the minimum SDK version and synchronize the project.

enerally to this step, Eclipse project has been fully AS project, at this point, we will open any file, if there is the following garbled code, then we need to modify the project's coding, change to GBK coding, because the source Eclipse project uses GBK encoding.

Set Engineering Code

After setting the code above,open build. .gradle(Module:app) in Gradle Scripts,and add the following statement compileOptions.encoding = "GBK" to Android.

```
android {
   compileOptions.encoding = "GBK"
}
```

7.1.2 Analysis of the structure of the original engineering catalogue

```
main                          Source Location Directory
 ├── AndroidManifest.xml              Android project profile
 ├── java
 |   └── wificar                  Package name
 |       ├── BgPictureShowActivity.java     Class to view a picture window after taking a picture
 |       ├── Constant.java              Global variable store class|   ├── MyMainFrm.java
Main Window
 |       ├── MySurfaceView.java          Parse the video class, which is dependent on the following
class
 |       └── MyVideo.java              Visual Window class
 └── res                        Software resources folder
 ├── drawable-hdpi                  Image resources folder
 |   └── icon.png
 ├── drawable-ldpi
 |   └── icon.png
 ├── drawable-mdpi
 |   └── icon.png
 ├── layout                     Layout folder
 |   ├── mymainfrm.xml                Main window layout file
 |   ├── myvideo.xml                  Video Window layout file
 |   └── pictureshow.xml              Image viewer layout file
 └── values                     Value Resource Folder       ├── attrs.xml
     └── strings.xml              String store, APP name changed here
```

After looking at the original project's file directory structure, let's take a look at the Android manifest file, which is the Android project's configuration file, where permissions are applied and corresponding classes are registered. The file permission request configuration for our source project is as follows:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="adnroid.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!-- Create and delete file permissions in the sdcard -->
<uses-permission
  android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"
  tools:ignore="ProtectedPermissions" />
<!-- Write data permission to sdcard -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

The above permission request configuration has a serious problem that we would address in the next section.

7.1.3 Source Code Problems and correction methods

Since our open source code is based on Android version 4.4 and the SDK version used is older, So here's the problem with running on your current android phone:

1.APP takes pictures that don't work, often fail

2. View features, one click, and the APP is gone

The reason for this problem is that since Android 7, Google's official application for permissions has become more stringent, and if users need to use certain important permissions, the software will fail if they don't apply them dynamically, it could seriously crash the program.

We provide the source code even with read and write SD card permissions and the right to take pictures, both of which are sensitive after Android 7. The ANDROID system doesn't grant permissions if you don't ask for them, so the solution is to request them dynamically in the main class. That is, at Mymainfrm. Dynamic application in Java. As to how to apply, this section does not do too much elaboration, the user may by oneself Baidu.

7.2 Android video decoding principle

Video decoding operation mainly in mysurfacing. In Java, our video stream only needs to be in Mymainfrm. Java to pass video stream address to myvideo. Java interface, and then in the layout file of the MyVideo interface, we call the MySurfaceView control, which is one of our custom controls, where the decoding of the video is processed. Here's how it works:

MyVideo.xml introduces custom controls

```
<wificar.MySurfaceView
    android:id="@+id/mySurfaceViewVideo"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
```

MyVideo.org. Initializes the control in Java and gets the video stream IP address

```
MySurfaceView r = (MySurfaceView)findViewById(R.id.mySurfaceViewVideo);
// Get the value from the Intent by the key
CameraIp = intent.getStringExtra("CameraIp");
CtrlIp= intent.getStringExtra("ControlUrl");
CtrlPort=intent.getStringExtra("Port");
/Pass the video address to the MySurfaceView control
r.GetCameraIP(CameraIp);
```

In My SurfaceVieww.Java, we inherit the SurfaceView and implement Callback, and when the

SurfaceView is initialized, we open a thread to read and decode the data stream.

The Surface is created by initializing a thread class and starting.

```
public void surfaceCreated(SurfaceHolder holder) {
    isThreadRunning=true;
    new DrawVideo().start();
  }
```

Drawvideo () is the class that handles the data flow, inheriting the Thread Thread Class and overriding the run () method. Please check the source code.

```
public void run() {
     ....
    while (isThreadRunning) {
      try {
        urlConn = (HttpURLConnection) url.openConnection();
        ....
        while (true) {
          read = urlConn.getInputStream().read(buffer, 0,
              readSize);
          if (read > 0) {
// Start parsing protocol heads
switch (status) {
                    // Content-Length:
                    case 0:
                      if (buffer[i] == (byte) 'C')
                        status++;
                      else status = 0;
                      break;
                     .....
                    // Start Parsing the data frames
case 15:
                      if (buffer[i] == (byte) 0xFF)
                        status++;
                      jpg_count = 0;
                      jpg_buf[jpg_count++] = (byte) buffer[i];
                      break;
                     ....
          }
        }
      } catch (IOException ex) {  ...  }
    }
}
```

7.3 Android communication principle

The Roly Robot uses standard Socket communication and the default port is 2001. Our PC software can communicate with Roly Robot simply by binding its IP address and port number. Initialize method in Myvideo. In the InitSocket () of the Java file.

```
public void InitSocket()
{
  try {
   socket = new Socket(InetAddress.getByName(CtrlIp),Integer.parseInt(CtrlPort));
```

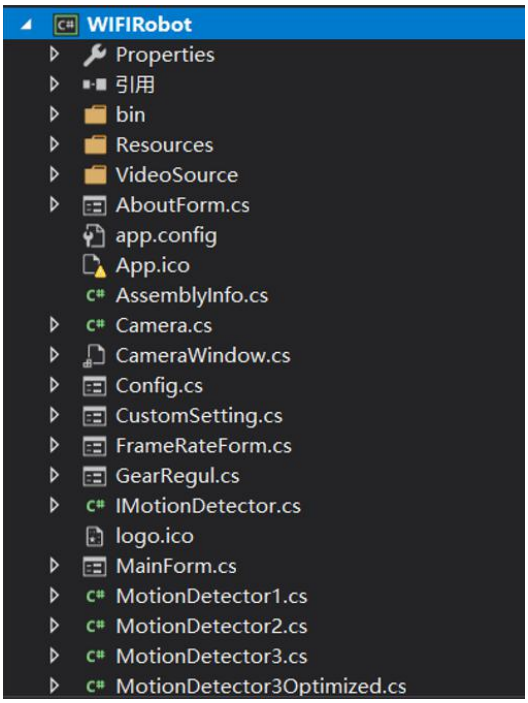
```
socketWriter = socket.getOutputStream();//Get the output stream
 } catch (Exception e) {
  e.printStackTrace();
 }
 }
```

After we initialize the socket, we get the output stream socketWriter. When we press the relevant button, push out the data we need to send. It is similar to the structure that sends bytes of data:

```
byte[] cmd = new byte[]{(byte)0xff,(byte)0x00,(byte)0x00,(byte)0x00,(byte)0xff};
socketWriter.write(cmd );
socketWriter.flush();
```

7.4 Analysis of PC control software architecture

PC software development language is c # , development environment is Net.Tool is VS2012. Its file directory structure is as follows:



File name	Attribute
bin	The directory in which the final generated executable is located
Resources	Resource folder
VideoSource	Video resource decoding folder
AboutForm	About the interface
app.config	Software configuration
logo.ico	Software logo
CameraWindow.cs	Video form interface

Config.cs	System settings form
CustomSettings.cs	Custom Command form
FrameRateForm.cs	Frame rate setting form to start recording evocations
GearRegul.cs	Mechanical Control Window , Mechanical arousal
MainForm.cs	The main form, the first interface to be shown to the user
IMotionDetector.cs	The interface for the MotionDetector
MotionDetector1~4.cs	Probe Pattern Implementation Class

7.5 Video decoding principle of PC terminal control software

PC side software and Android side of the decoding are similar to the decoding, they use Mjpeg video stream decoding method, Mjpeg frame data format as follows:

Frame header		Data	End of frame	
0xFF	0xD8	....	0xFF	0xD9

So if we want to decode MJPEG video stream, we need to capture the frame header and end of the video stream, and then convert the middle data into a bitmap image to be drawn on the video form. Parsing Code Within videosource--> MJPEGStream.cs. Its main code is as follows, see the source code for details:

```
while ((align == 2) && (todo >= boundaryLen))
{
  stop = ByteArrayUtils.Find(buffer, boundary, pos, todo);
  if (stop != -1)
  {
    pos = stop;
    todo = total - pos;

    // increment frames counter
    framesReceived++;

    // image at stop
    if (NewFrame != null)
    {
      Bitmap bmp = (Bitmap)Bitmap.FromStream(new MemoryStream(buffer, start, stop -
start));
      // notify client
      NewFrame(this, new CameraEventArgs(bmp));
      // release the image
      bmp.Dispose();
      bmp = null;
```

```
        }
        //System.Diagnostics.Debug.WriteLine("found image end, size = " + (stop - start));

        // shift array
        pos = stop + boundaryLen;
        todo = total - pos;
        Array.Copy(buffer, pos, buffer, 0, todo);

        total = todo;
        pos = 0;
        align = 1;
    }
    else
    {
        // delimiter not found
        todo = boundaryLen - 1;
        pos = total - todo;
    }
}
```

7.6 Communication principle of PC terminal control software

7.6.1 WiFi mode

PC side control software uses the same TCP communication, its initialization similar to Android ,no further explanation.Let's take a look at the basic socket initialization InitWIFISocket (String controlIp, String port) :

```
// Initialize wi-fi socket serial port
controlType= InitWIFISocket(ControlIp, Port)? 0:2;

/// <summary>
/// Initilize wi-fi socket connection
/// </summary>
/// <param name="controlIp"> Control address </param>
/// <param name="port"> Port number </param>
/// <returns> Whether the initialization was successful </returns>
private bool InitWIFISocket(String controlIp,String port)
{
    this.Systemstatus.Text = " Trying to access the wi-fi panel…";
    try
    {
        ips = IPAddress.Parse(controlIp.ToString());
        ipe = new IPEndPoint(ips, Convert.ToInt32(port.ToString()));
        socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
        RobotEngine2.SOCKET = socket;
        RobotEngine2.IPE = ipe;
        ret = RobotEngine2.SocketConnect();
    }
    catch (Exception e)
    {
        MessageBox.Show("WIFI initialization failed：" + e.Message, wi-fi initialization failure alert ",
```

```
MessageBoxButtons.OK, MessageBoxIcon.Error);
  }
  return ret;
}
```

To send data, call our library function robotengine 2. SENDCMD (), example:

```
SerialPort comm = new SerialPort();
...
// ControlType control mode, Wifi or Bluetooth
// The first parameter represents the steering gear, the second represents which steering gear,
and the third represents the rotation angle value
byte[] Speedright_data = RobotEngine2.CreateData(0X02,
0X01,Convert.ToByte(this.Speedright.Value));
RobotEngine2.SendCMD(controlType, Speedright_data, comm);
```

7.6.2 Bluetooth mode

Our car is Bluetooth module, the way we communicate with it is to transfer Bluetooth to serial port. Its initialization examples as follows, see the source code for more:

```
if (1 == controlType) return;
// Turn on Bluetooth mode
if (comm.IsOpen) { return;}
else
{
  if (btCom == null || btCom=="")
  {
    MessageBox.Show("Please set the Bluetooth port number in the system !","Bluetooth port
number settings prompt." ",MessageBoxButtons.OK,MessageBoxIcon.Error);
    return;
  }
  comm.PortName = btCom;
  comm.BaudRate = int.Parse(btBaudrate);
  comm.DataReceived += comm_DataReceived;  // Define the receive event
  try
  {
    comm.Open();
  }
  catch (Exception ex)
  {
    // Catch exception information, create a new comm object, the previous one is not working。
    comm = new SerialPort();
    MessageBox.Show("Error in Bluetooth mode！ "+ex.Message," Bluetooth mode setting
prompt ",MessageBoxButtons.OK,MessageBoxIcon.Error);
    return;
  }
  controlType = 1;
}
```

To Send an instance, refer to wifi mode.

Conclusion

All the examples in this book are the actual test cases of our company, only for our company car case explanation; if users use non-our company products, then all the code for reference only.

Users in use, there will be other questions not answered, if you have any questions in the use of our products, you can contact our technology in time,You can get the contact through your purchase order (online) or sales (offline), they will push the contact technology to you, in roder to answer your questions.

This book is for beginners, so the written expression is also used to describe the vernacular, but because everyone may understand the direction of bias, if you do not understand somewhere, you can also contact us for technical support.

Again, it's not uncommon for people to have different ideas about a function (solution) after reading this book because everyone has a different level of understanding of the code; as we said, there are multiple problem solving processes for the same outcome, and different people may have different ideas, so many features just lead to one solution when you have different solutions, you can try your own secondary development with your own ideas to solve the problem.