Preface

About SunFounder

SunFounder is a technology company focused on Raspberry Pi and Arduino open source community development. Committed to the promotion of open source culture, we strive to bring the fun of electronics making to people all around the world and enable everyone to be a maker. Our products include learning kits, development boards, robots, sensor modules and development tools. In addition to high quality products, SunFounder also offers video tutorials to help your own project. If you have interest in open source or making something cool, welcome to join us! Visit **www.sunfounder.com** for more!

About the PiCar-S

The **PiCar-S** is a cool smart car that can work with Raspberry Pi 3 model B, 3 model B+, and 4 model B. Equipped with three sensor modules including ultrasonic obstacle avoidance, light follower, and line follower, you can better learn the programming on how to control the car.

In this manual, we will show you how to build the PiCar-S via description, illustrations of physical components, in both hardware and software respects. You will enjoy learning how all this work. You may visit our website www.sunfounder.com to download the related code and view the user manual on **LEARN -> Get Tutorials** and watch related videos under **VIDEO**, or clone the code on our page of github.com at

https://github.com/sunfounder/SunFounder_PiCar-S

You are welcome to pull requests and issue posts on our page on Github.

Free Support

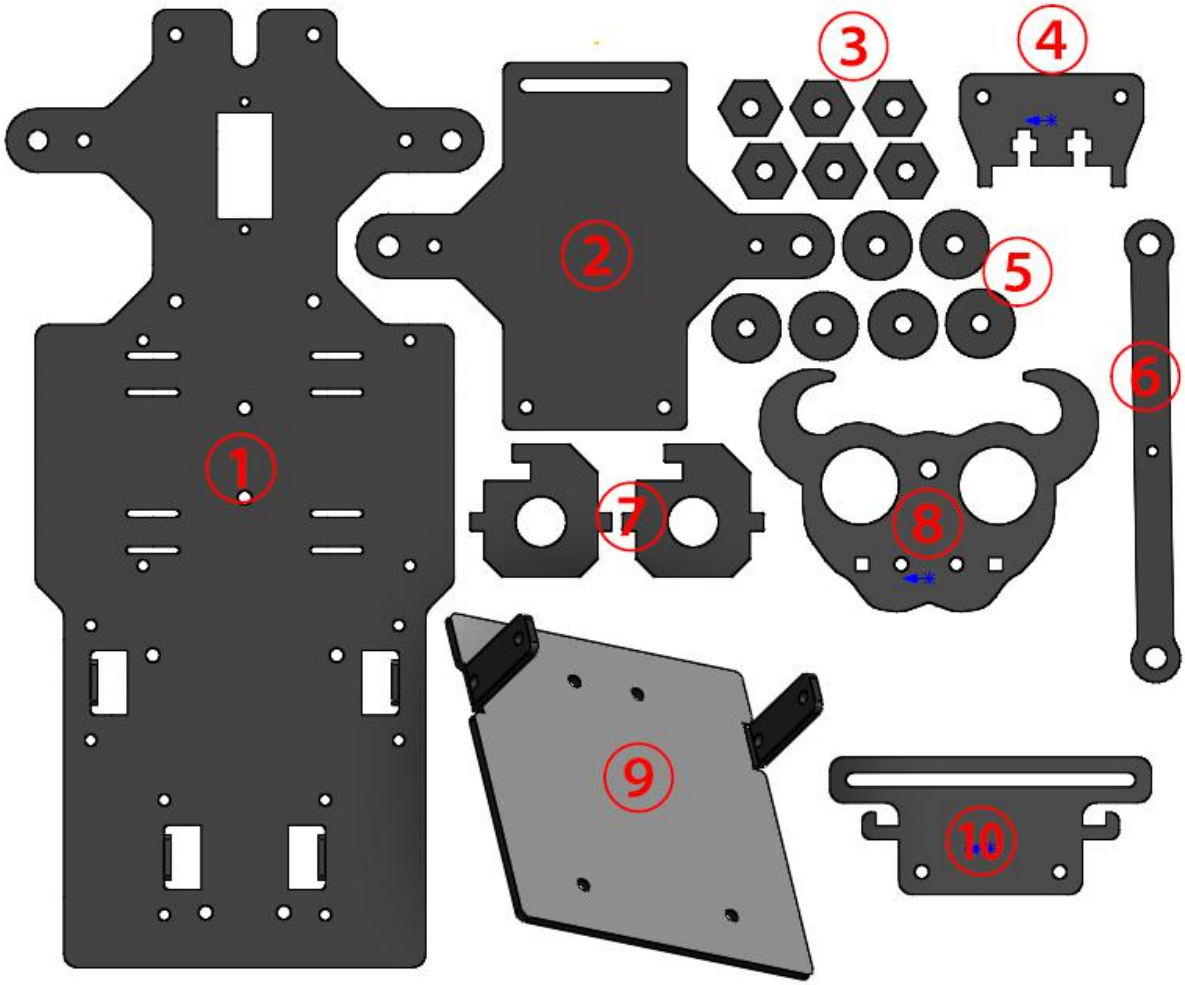
If you have any **TECHNICAL questions**, add a topic under **FORUM** section on our website and we'll reply as soon as possible.

For **NON-TECH questions** like order and shipment issues, please **send an email to service@sunfounder.com**. You're also welcomed to share your projects on FORUM.

Contents

Components List

Structure Plates



1. Upper Plate x 1
2. Front Half Chassis x 1
3. Hex Front Wheel Fixing Plate x 6
4. Ultrasonic Support x 1
5. Bearing Shield x 6
6. Steering Linkage x 1
7. Steering Connector x 2
8. Ultrasonic Connector x 1
9. Back Half Chassis x 1
10. Sensor Connector x 1

SunFounder SF006C Servo x 1

1. Servo x 1
2. 1-arm Rocker Arm x 1
3. arm Rocker Arm x 1
4. 4-arm Rocker Arm x 1
5. Rocker Arm Fixing Screw x 1
6. Rocker Arm Screw

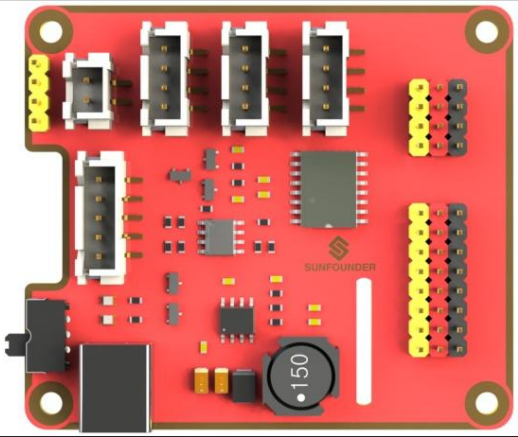
Mechanical Fasteners

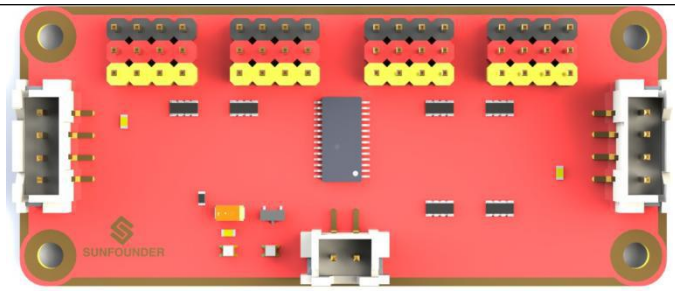
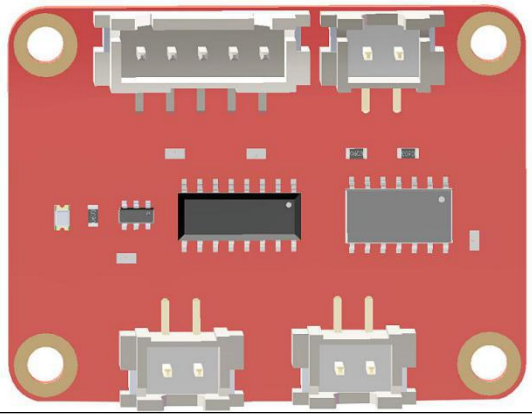
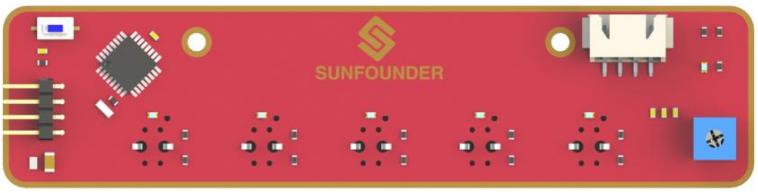
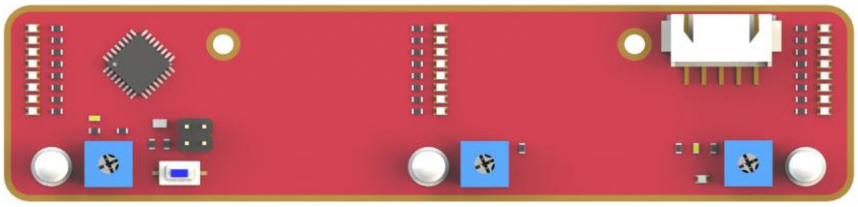
Name	Component	Qty.
M2x8 Screw		2
M2.5x6 Screw		4
M2.5x12 Screw		8
M3x8 Screw		8
M3x8 Countersunk Screw		2
M3x10 Screw		9
M3x25 Screw		4
M4x25 Screw		2
M2 Nut		2
M2.5 Nut		12
M3 Nut		23
M4 Self-locking Nut		2
M2.5x8 Copper Standoff		8
M3x25 Copper Standoff		8
4x11x4 F694ZZ Flange Bearing		2

Wires

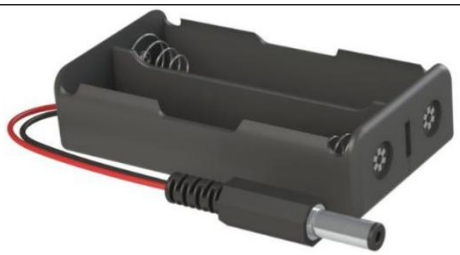
100mm 5-Pin Jumper Wire		1
50mm 4-Pin Jumper Wire		1
50mm 2-Pin Jumper Wire		1
100mm 2-Pin Jumper Wire		1
200mm 5-Pin Jumper Wire		1
200mm 4-Pin Jumper Wire		1
200mm 3-Pin Jumper Wire		1

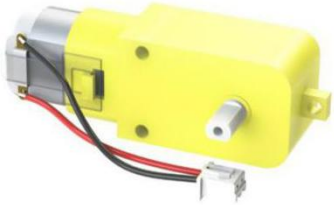
PCB

Robot HATS		1

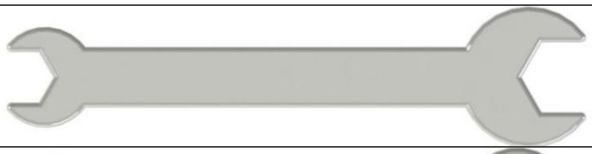
PCA9685 PWM Driver		1
Motor Driver Module		1
5-CH Line Follower Module		1
Ultrasonic Obstacle Avoidance Module		1
Light Follower Module		1

Other Components

2x18650 Battery Holder		1

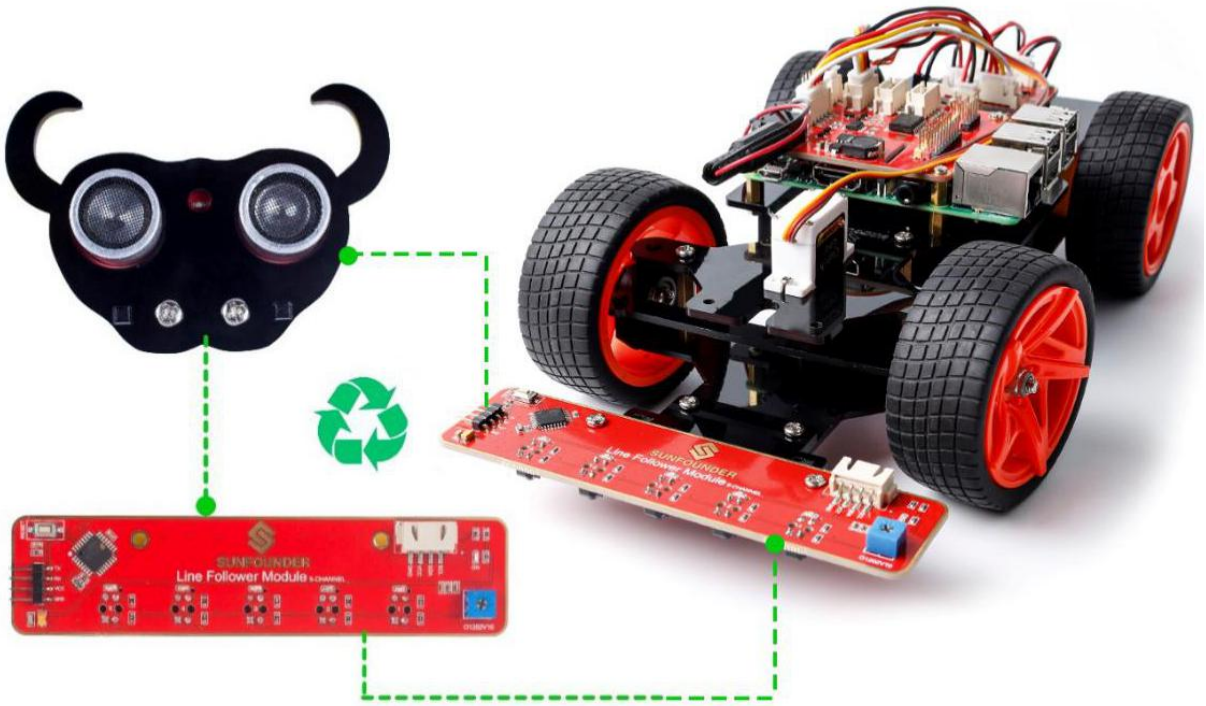
DC Gear Motor		2
Rear Wheel		2
Front Wheel		2
Ribbon (30cm)		1

Tools

Cross Screwdriver		1
Cross Socket Wrench		1
M2.5/M4 Small Wrench		1
M2/M3 Small Wrench		1

Introduction

The PiCar-S is a **SMART SENSOR** car robot based on Raspberry Pi, which comes with three sensor modules, including the light follower, line follower and ultrasonic obstacle avoidance. With these modules, this smart car is capable of some simple automatic actions. Thus, you can learn some basics of programming in Python to control the car with these sensors. Let's start with building this smart car!

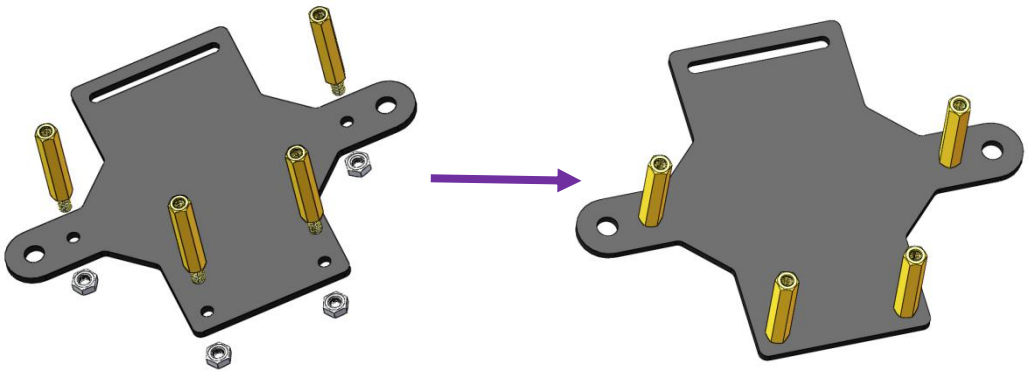


Building the Car

Extremely excited when opening the box and checking so many components? Keep your patience and take it easy. Please note that some details in the following steps need **CAREFUL** observation. You should double-check your work based on the figures in the manual after finishing each step. Don't worry! Kindly reminders will be given in some particular steps. Just follow the tutorial step by step. Okay, with no further ado, now let's start!

Front Half Chassis

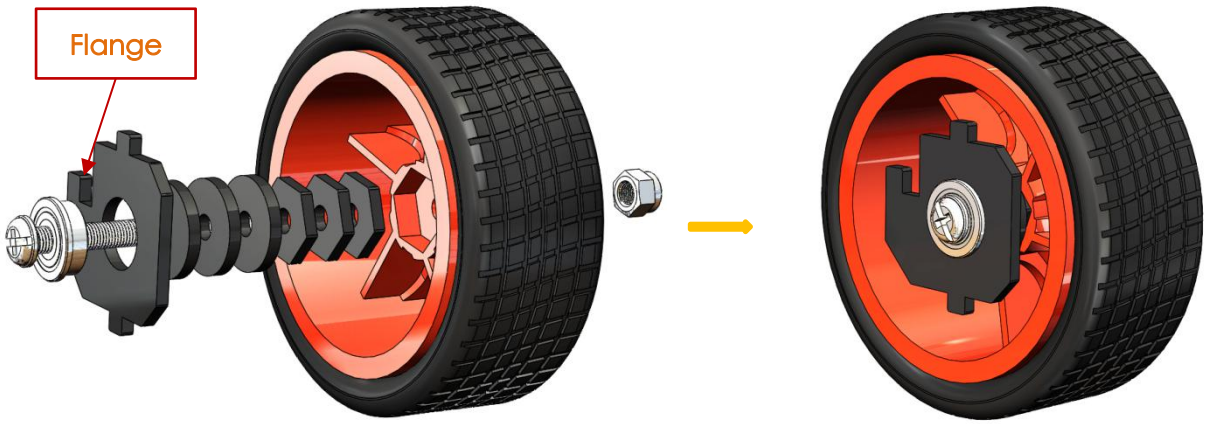
Assemble the **Front Half Chassis** with four **M3x25 copper standoffs** and four **M3 nuts** as shown below:



Front Wheels

Note: Please pay attention to the direction of Steering Connector before assembling.

Insert an **M4x25 screw** through a **Flange Bearing** (pay attention to the direction – the flange near the cap of the screw), a **Steering Connector**, 3 **Bearing Shields**, 3 **Hex Front Wheel Fixing Plates**, and a **front wheel**, into an **M4 Self-locking Nut** (note the direction) as shown below:

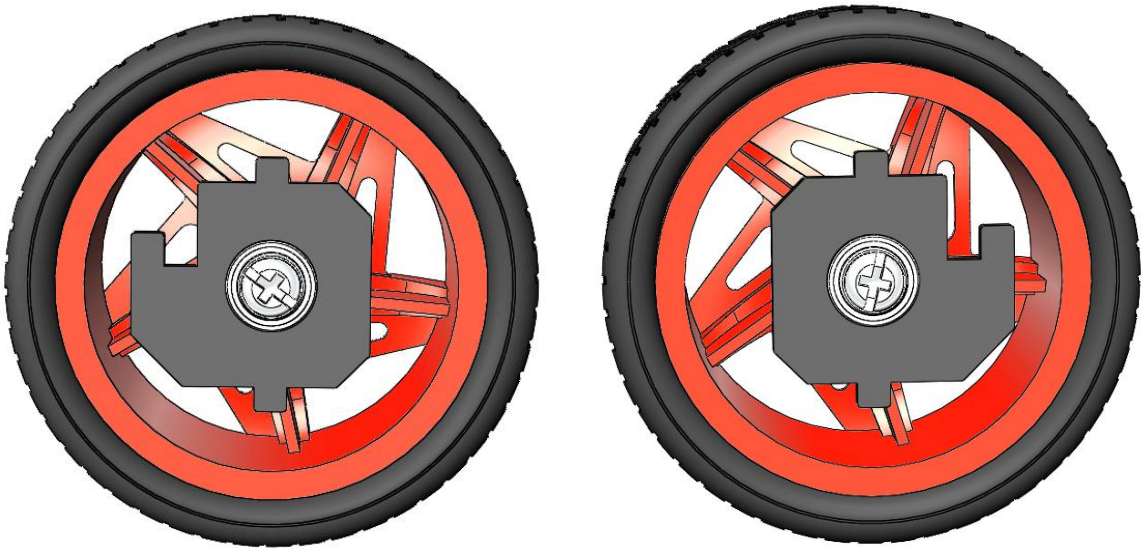


You can use the Cross Socket Wrench to secure the **M4 Self-locking Nut**, then use the screwdriver to tighten the **M4x25 screw.**

Note:

The Self-locking Nut should be screwed tight enough. It would be better to tighten the screw until the wheel and Steering Connector cannot move first, then loosen the screw a little, so that the Steering Plate can just move. Thus, the wheel can turn flexibly when the connection would not be too loose.

Assemble the other front wheel in the same way, but bear in mind the Steering Connector on the wheel should be symmetric with the previous one:

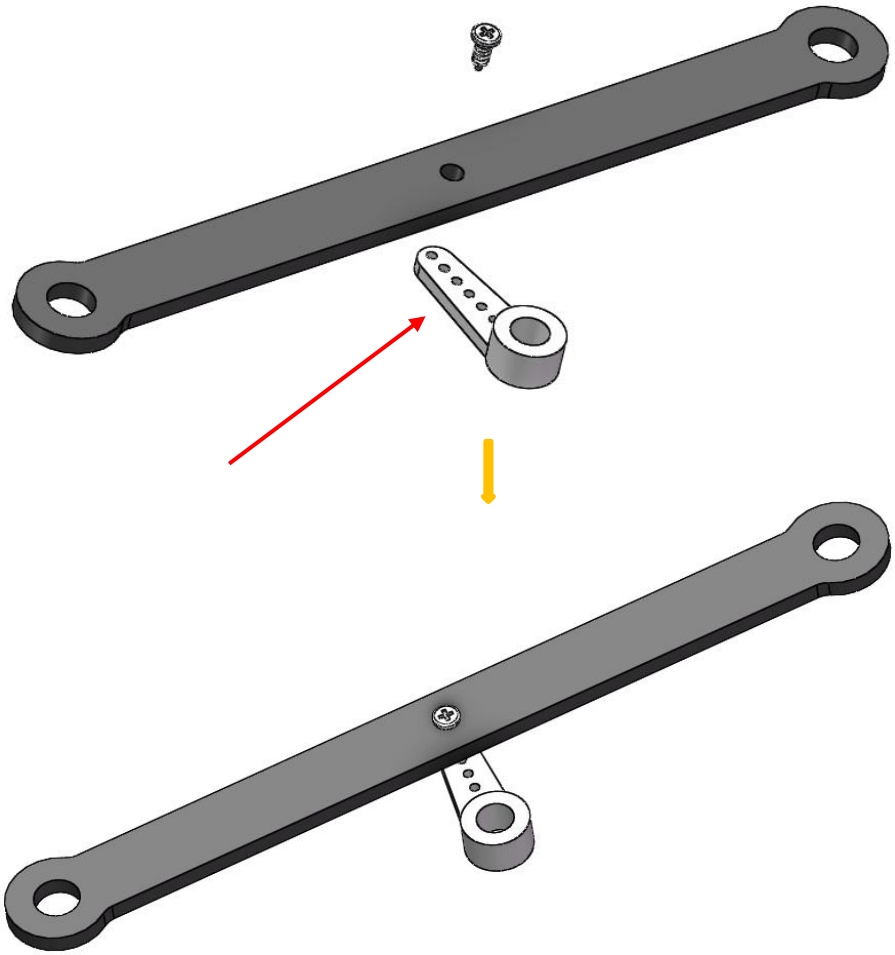


Now two front wheels have finished assembly.

Steering Part

Connect the **Steering Linkage** and the **rocker arm** with the **Rocker Arm Screw** (a longer one in the servo package).

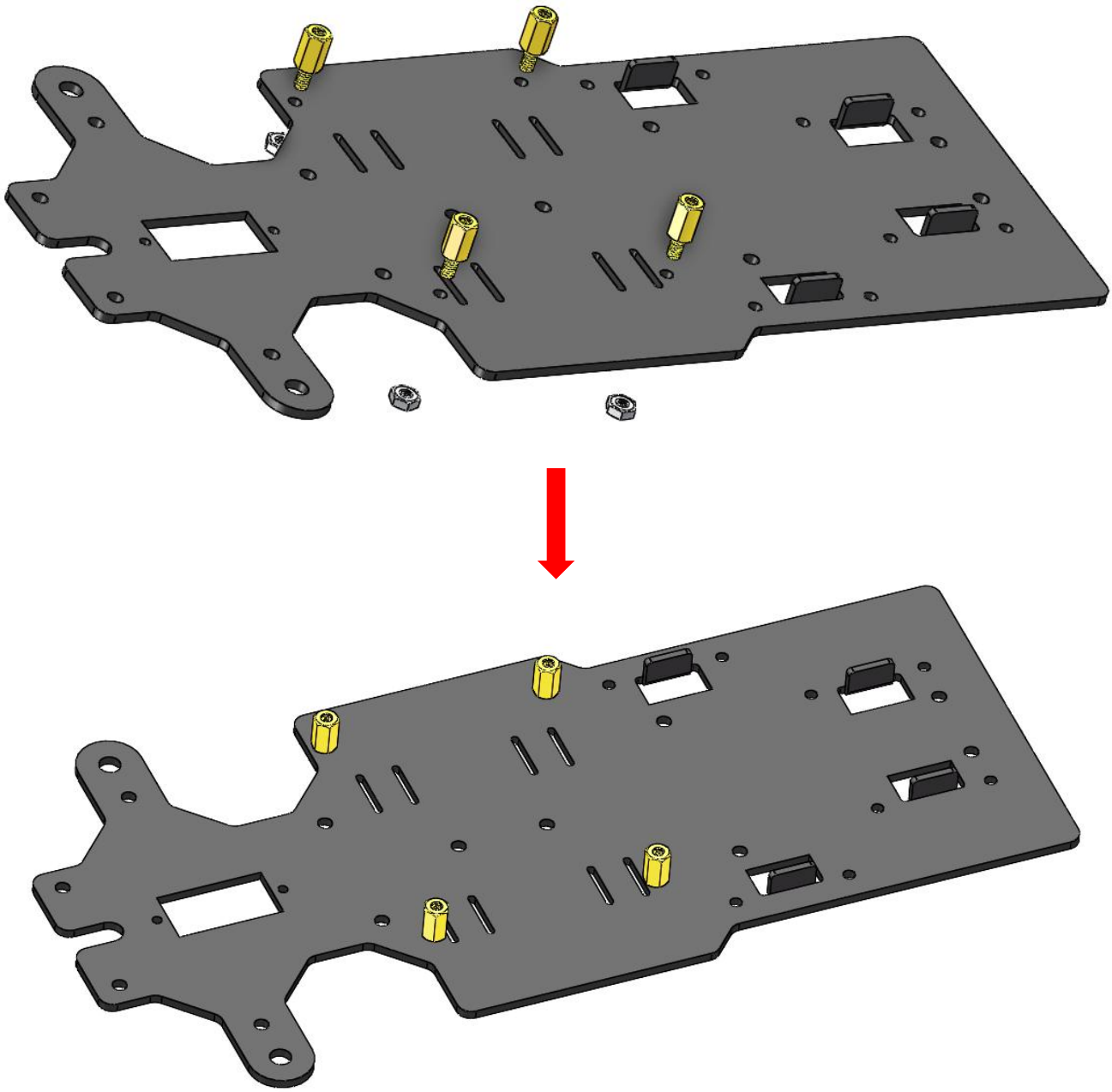
Note: Insert it into the first hole of the arm (as indicated by the arrow below) which is the farthest from the gears. Since the screw is larger than the hole, you should try to screw it hardly so as tight to the arm. Don't worry of the arm which is soft.



Note: Fasten them as tightly as possible, and then loosen the screw a little so the Steering Linkage can move flexibly. Besides, the screw is quite sharp at the end, so be careful to assemble in case of getting hurt.

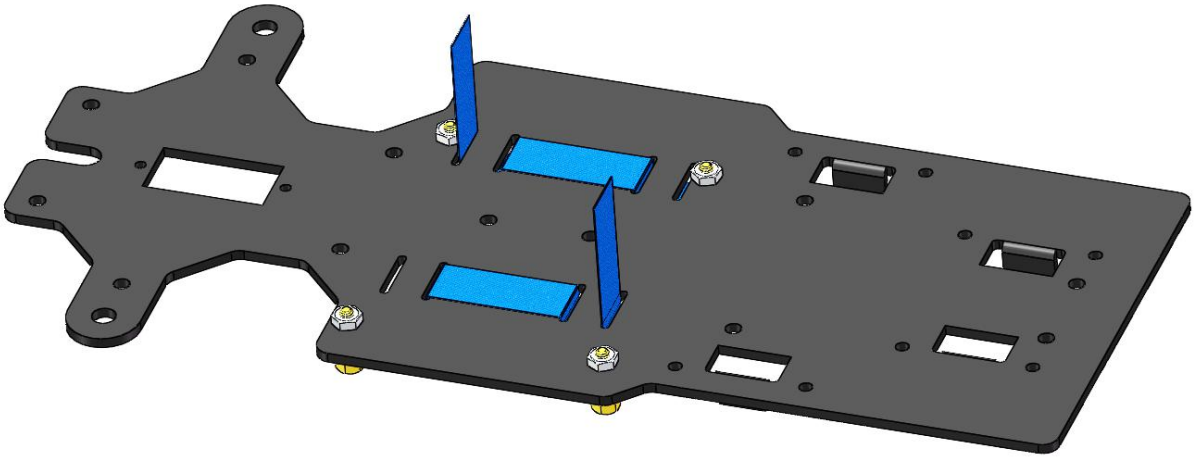
Upper Plate

Mount the **M2.5x8 copper standoffs** and **M2.5 nuts** into the **upper plate** first. Pay attention that the side the protruding prop should face up.

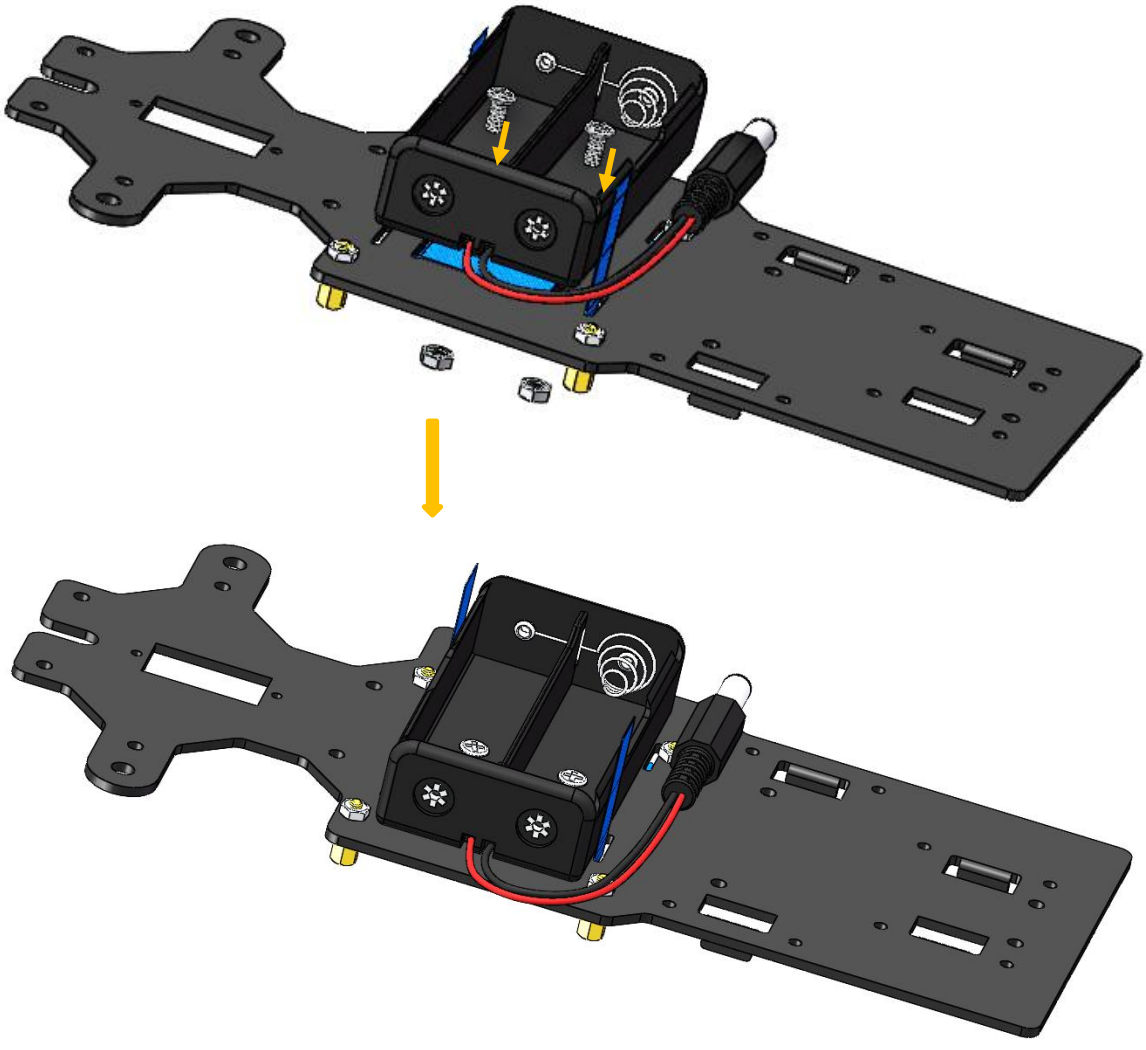


Battery Holder

Turn the Upper Plate upside down. Cut the **ribbon** into two halves. Thread them through the holes on the plate. Pay attention to the direction and leave one end longer out of the plate for each to remove the battery easily later.

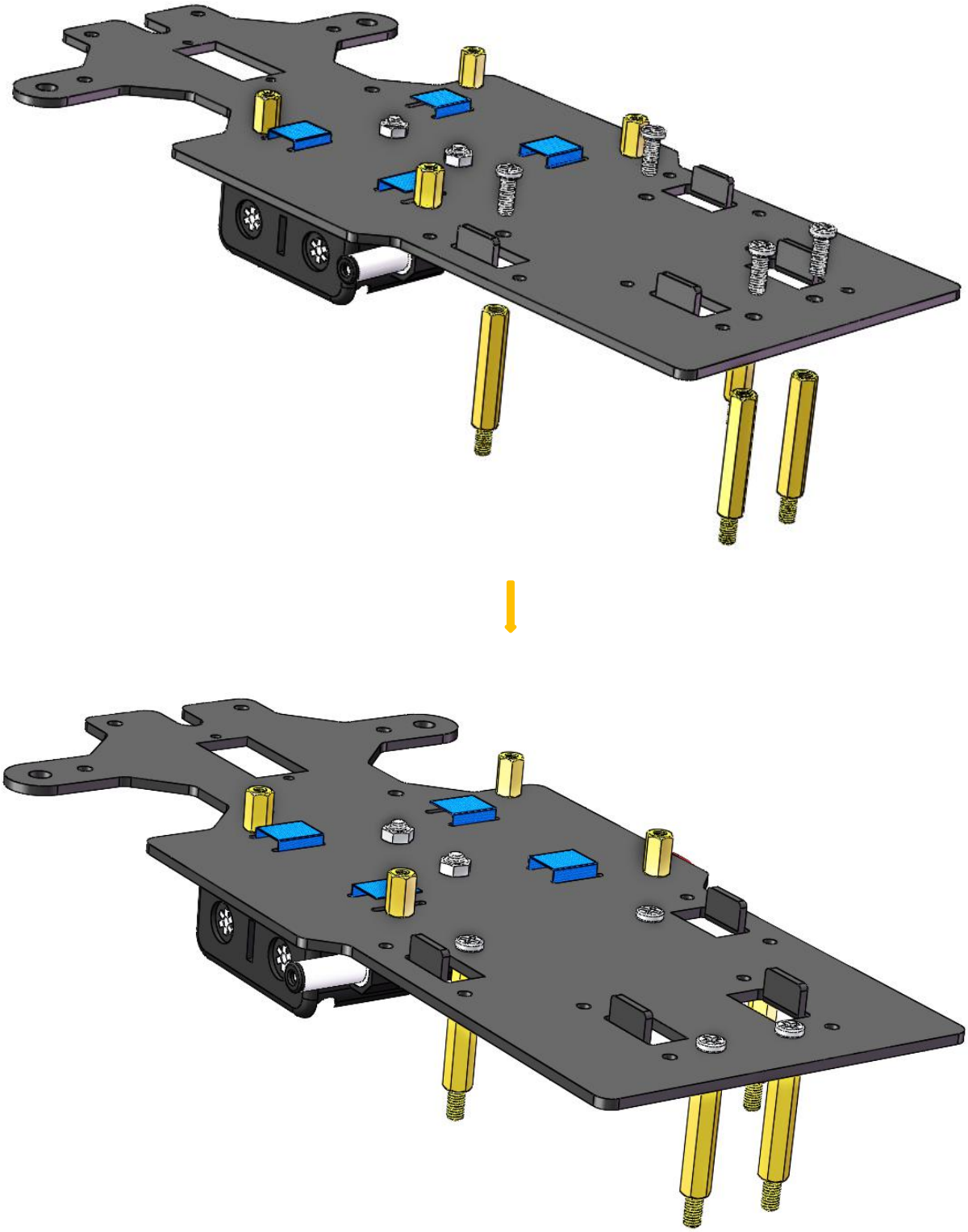


Fasten the battery holder with two **M3x8 countersunk screws** and two **M3 nuts**: pay attention to the direction of battery holder's wire.



Rear Wheels (Screws)

Insert four **M3x8 screws** with four **M3x25 copper standoffs**:

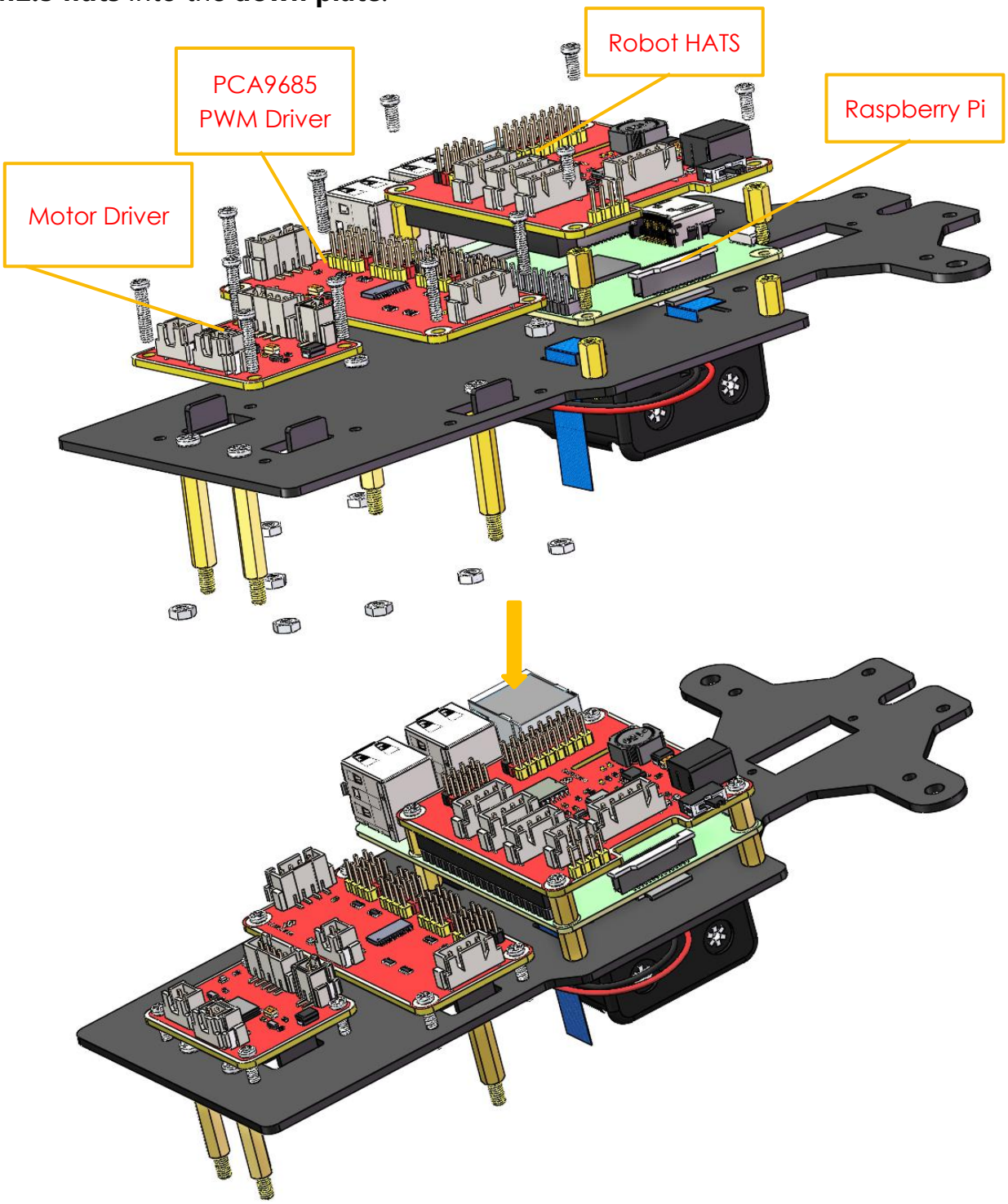


PCB Assembly

1) Assemble the **Raspberry Pi** (TF Card inserted) with eight **M2.5x8 copper standoffs**, then plug the **Robot HATS** onto it.

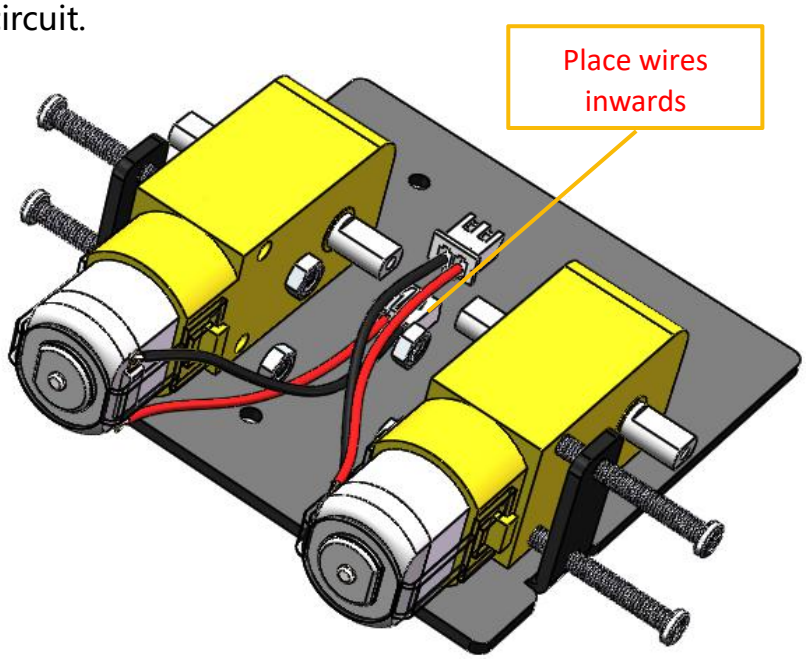
2) Fix the **Robot HATS** with four **M2.5x6 screws**.

3) Fix The **PCA9685 PWM Driver**, the **Motor Driver** with eight **M2.5x12 screws and M2.5 nuts** into the **down plate**.

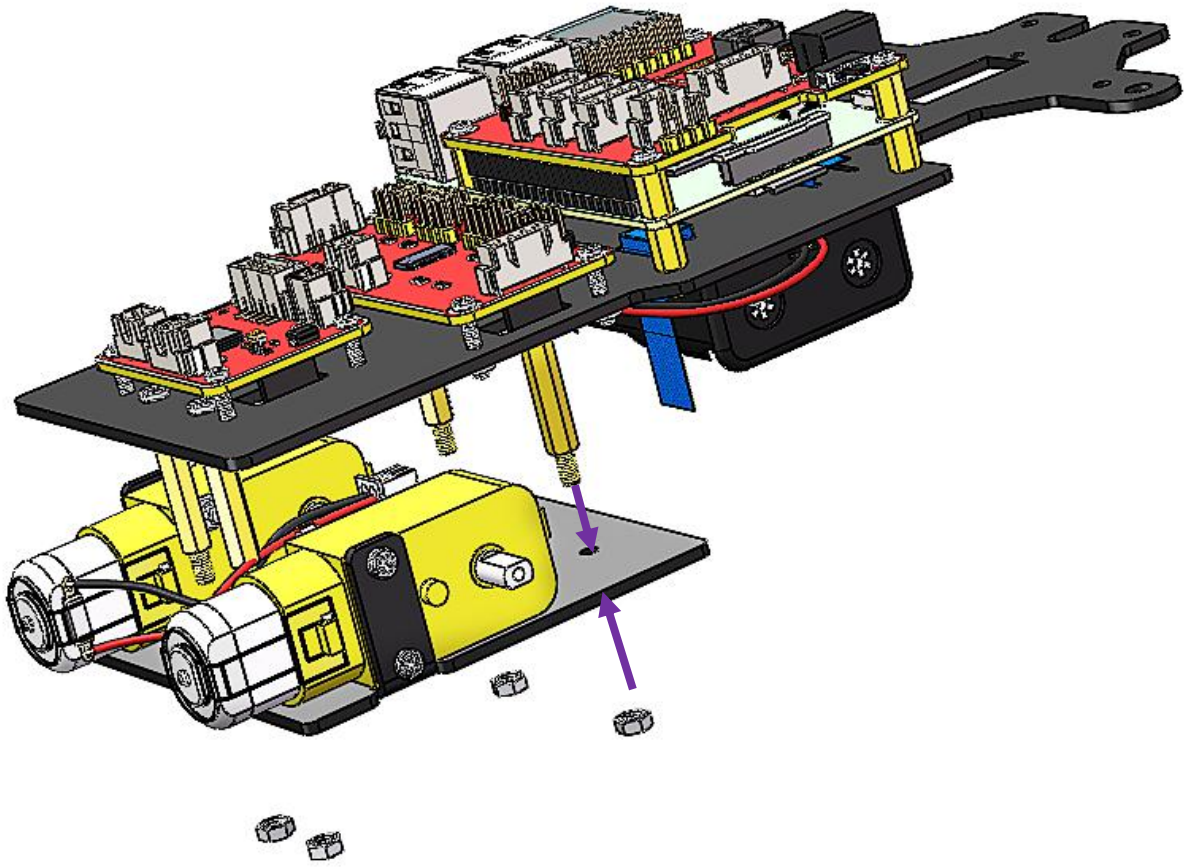


Rear Wheels (Driving)

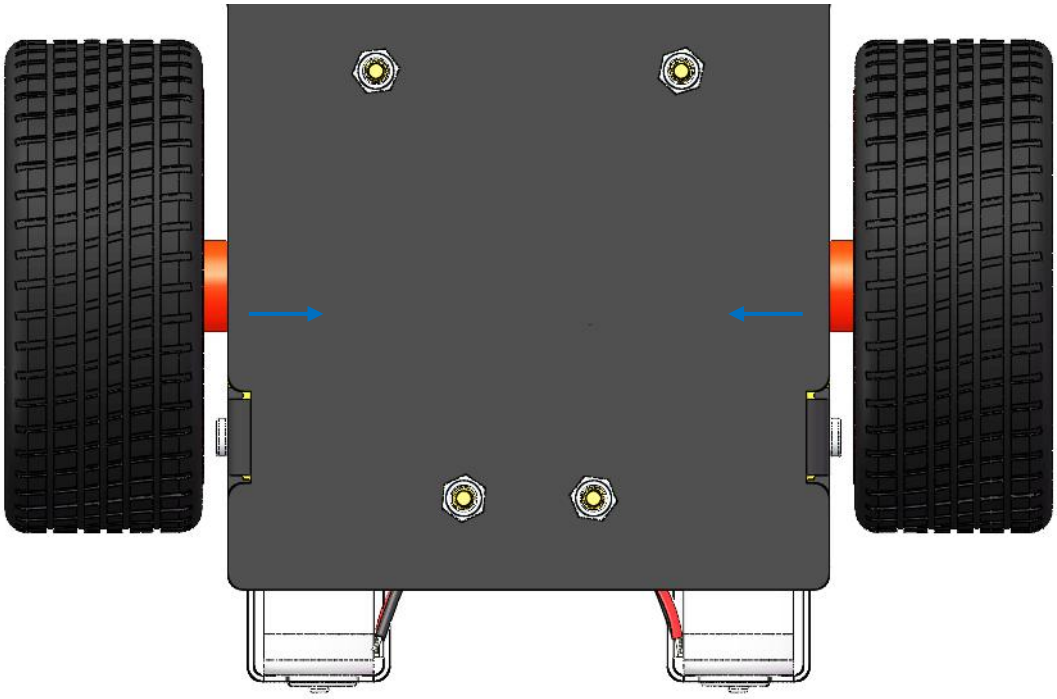
Assemble the two motors to the Back Half Chassis with four **M3x25 screws** and **M3 nuts**. Pay attention to place the motors with wires inward, providing convenience for connecting the circuit.



Assemble the rear wheels with 4 M3 nuts.

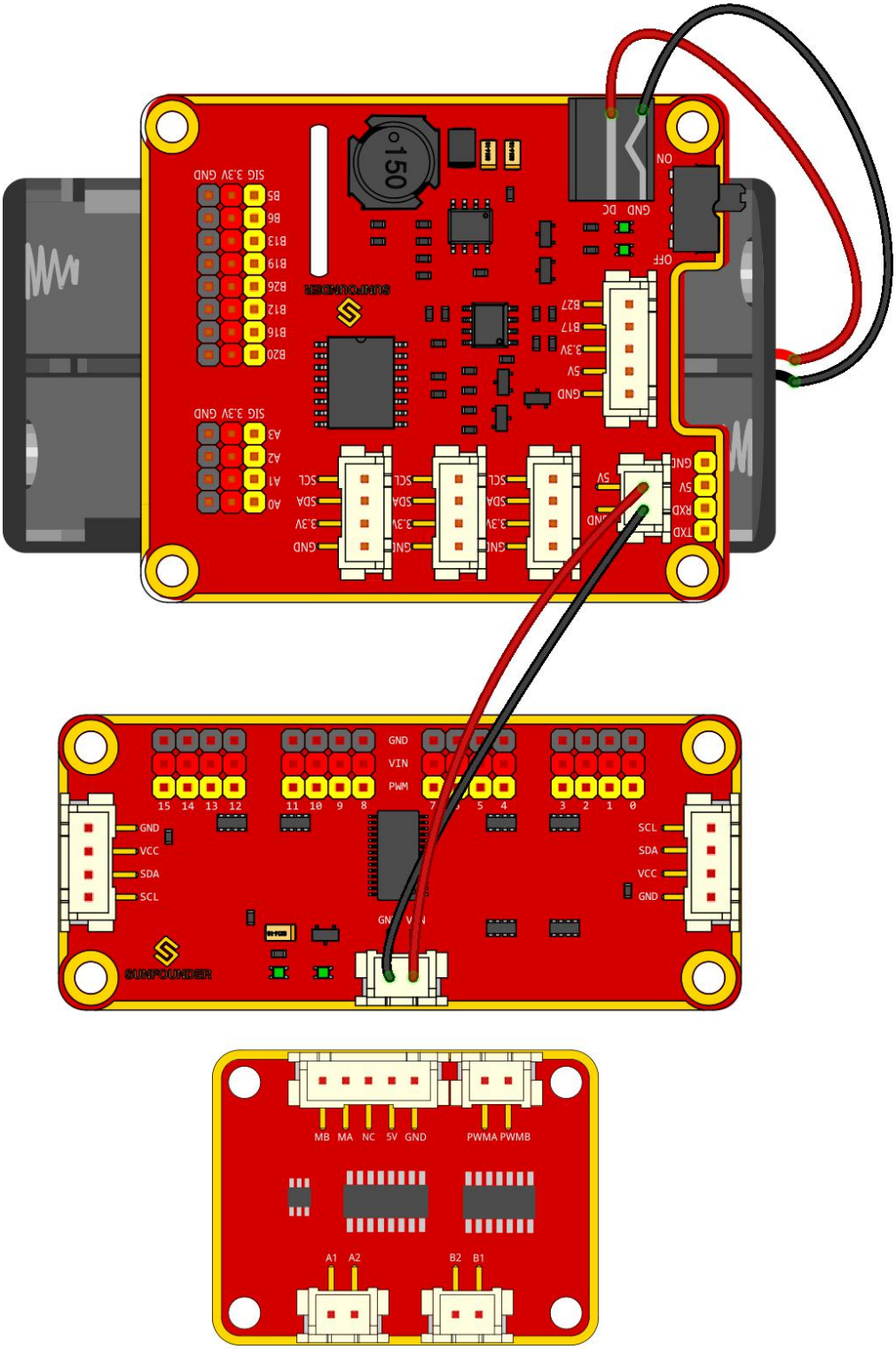


Align the **rear wheels** with the motor shaft, and rotate to insert them gently.

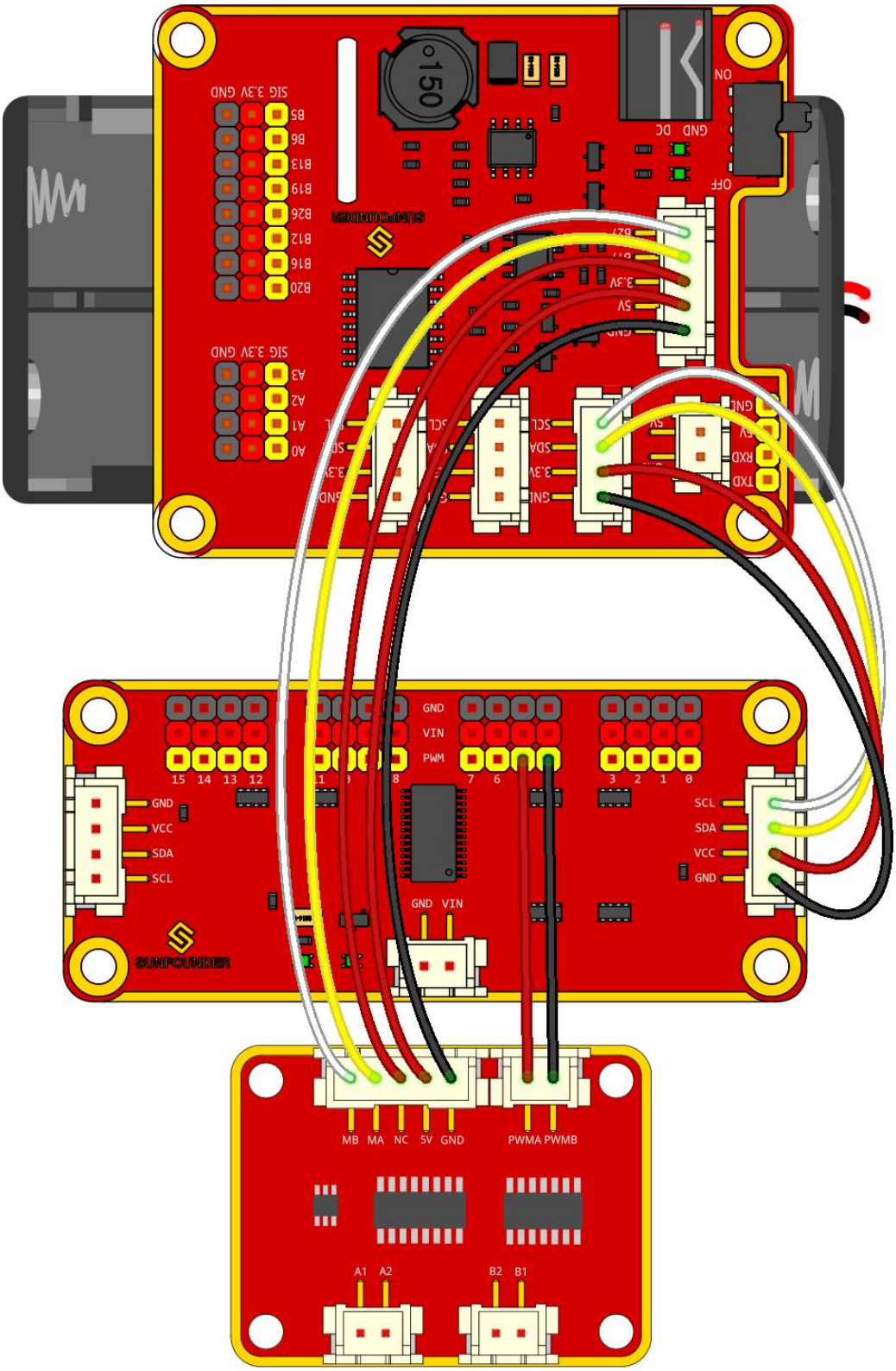


Circuits Building

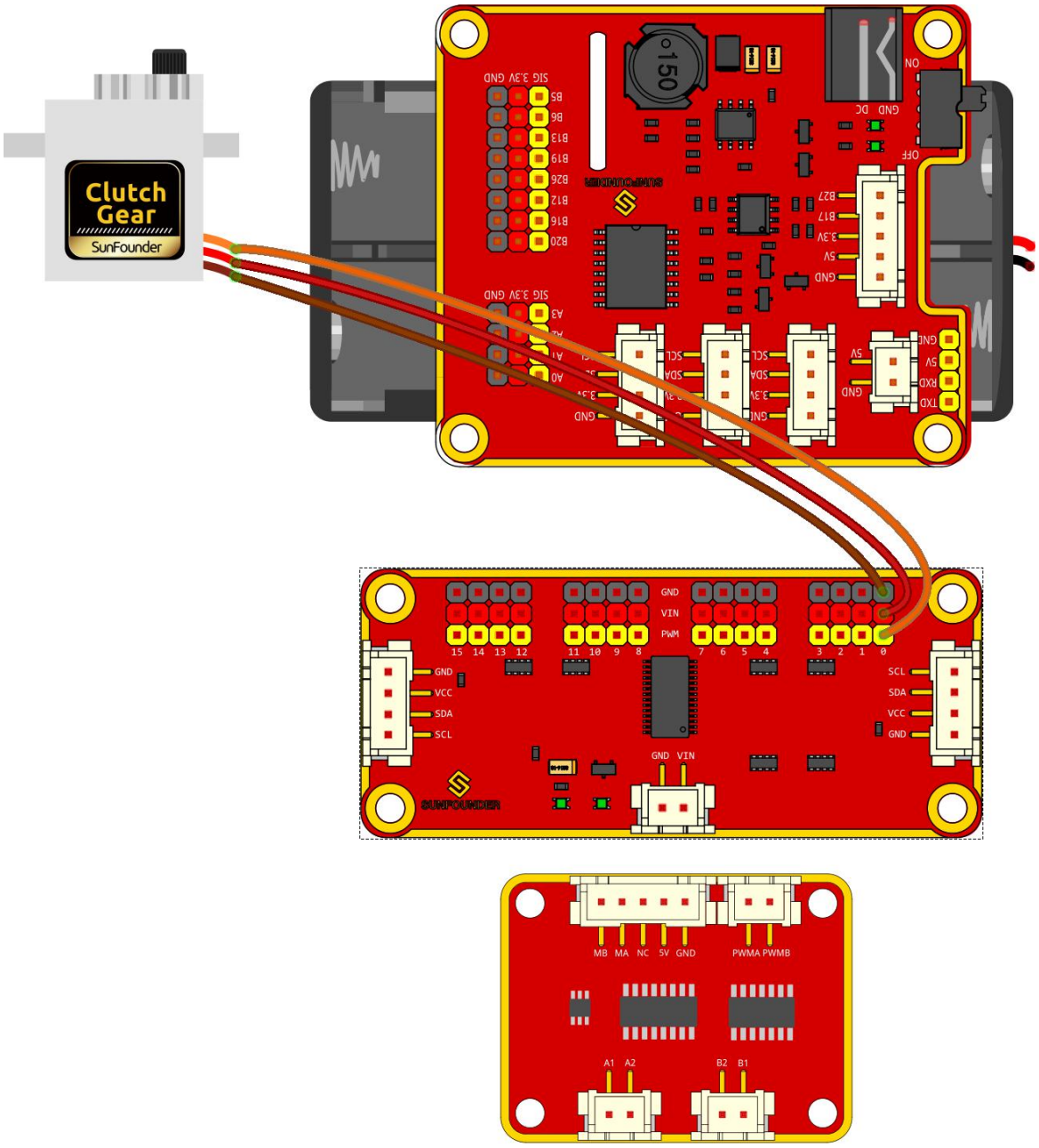
Connect the Power



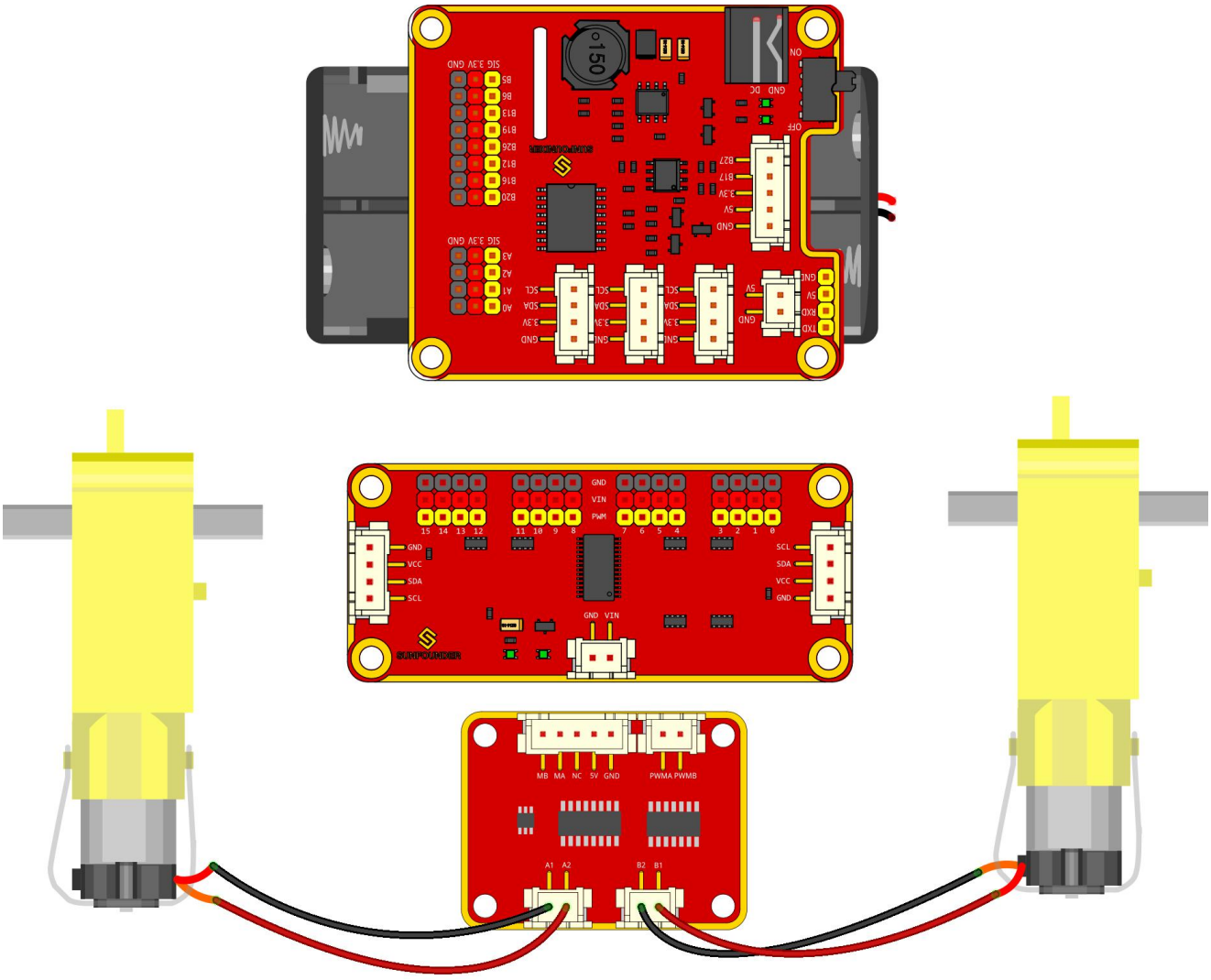
Connect the Modules



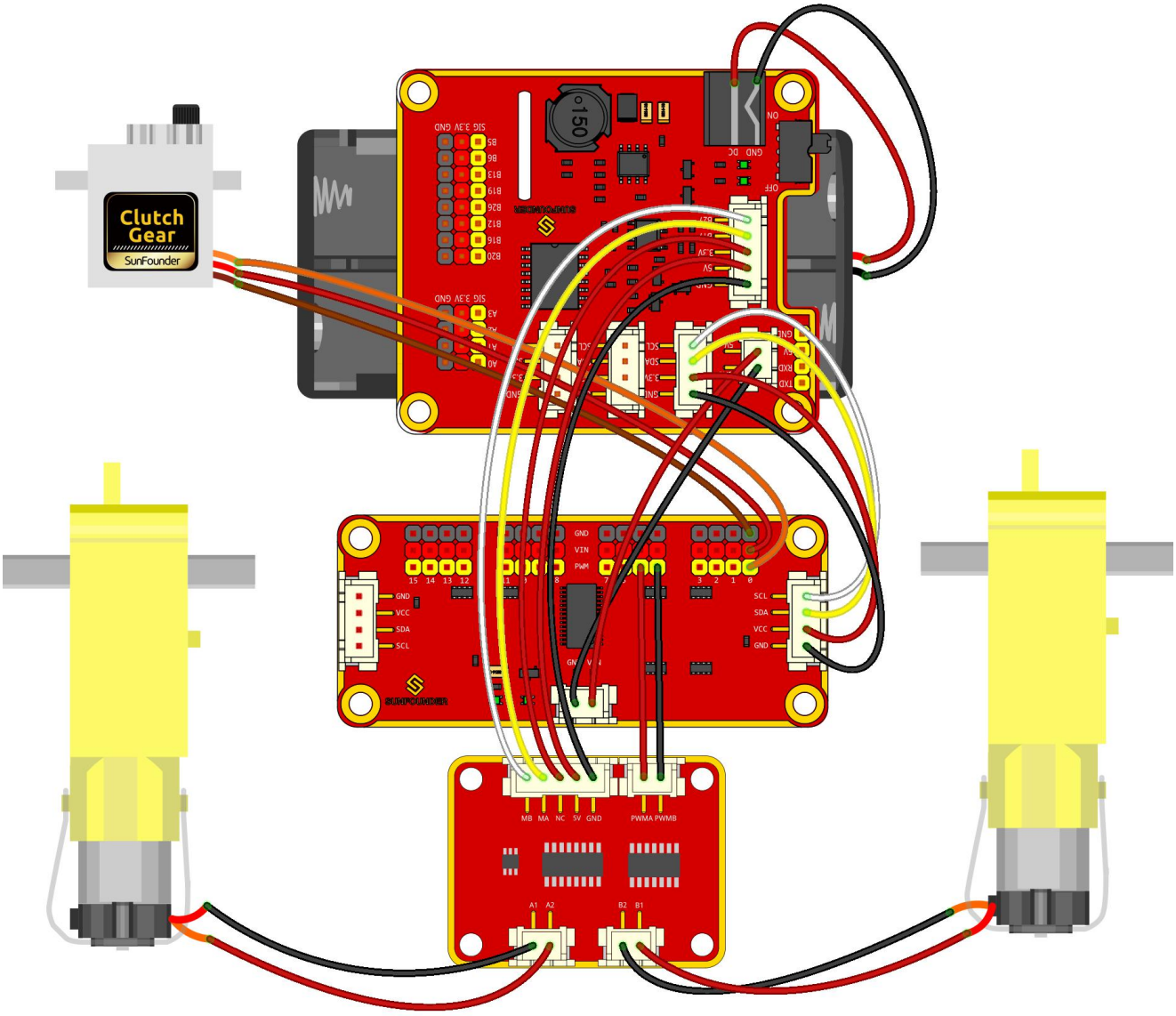
Connect the Servo



Connect the Motor



The complete connection is shown as follows.



So now the circuit boards are all installed onto the car and the wiring is done. But still you're not ready to adjust the servo yet. First you need to complete some software installation.

Get Started with Raspberry Pi

So the back part of the car is completed. Next we'll move on to the front part. But before assembly, since servos are used in this part, they need some configuration for protection. We need to make the servo keep in 90 degrees when it's mounted.

In this chapter, we firstly learn to start up Raspberry Pi.

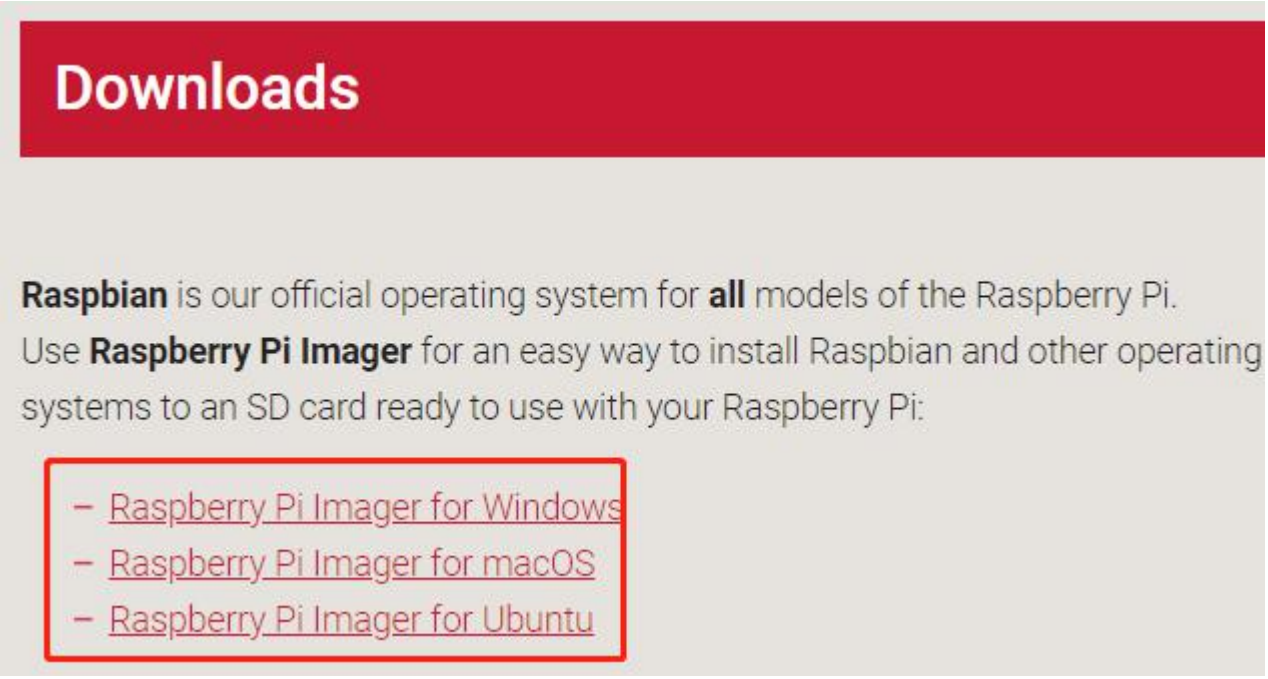
Depending on the different devices you use, you can start up the Raspberry Pi in different methods. We have two situations: having a screen or no screen, and you can refer to relevant tutorials respectively. If your Raspberry Pi is set up, you can skip the part: Get Started with Raspberry Pi and go into the next chapter Servo Configuration.

Note

Recently, RPi lauched a new method-Raspberry Pi Imager to install Raspbian and other operating systems to an SD card ready to be used with your Raspberry Pi.

Compared to the previous methods, it ‘s more time-saving for that it processes the flashing and download of the image at the same time.

https://www.raspberrypi.org/downloads/



There are Raspberry Pi Imagers of Windows, Mac OS and Ubuntu. Check the link to download, install and open Raspberry Pi Imager. Select Operating System and SD card then click WRITE. After flashing, you can start your Raspberry Pi and plug in a screen.

Without a screen, you can refer to Page 31 to configure Wi-Fi and start SSH after flashing.

If You Have A Screen

If you have a screen, you can use the NOOBS (New Out Of Box System) to install the Raspbian system.

- Required Components

Any Raspberry Pi	1 * Power Adapter
1 * Monitor	1 * Monitor Power Adapter
1 * HDMI cable	1 * Micro SD card
1 * Mouse	1 * Keyboard
1 * Personal Computer	

- Procedures

Step 1

To download NOOBS from your PC, you can choose **NOOBS** or **NOOBS LITE -** the only difference is that there is a built-in offline Raspbian installer in **NOOBS**, while the

NOOBS LITE can only be operated online. Here, you are suggested to use the former. Here is the download address of NOOBS:

https://www.raspberrypi.org/downloads/noobs/

Step 2

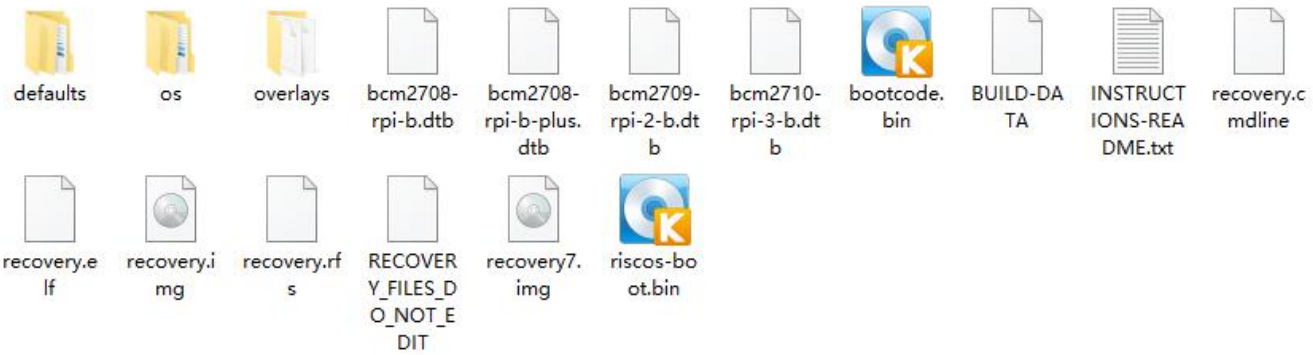
Plug in the Micro SD reader and format the Micro SD card with the **SD Formatter** (https://www.sdcard.org/downloads/formatter/index.html). If there are some important files in the Micro SD card, please backup them first.

Step 3

Next, you will need to extract the files from the NOOBS zip archive you downloaded from the Raspberry Pi website.

- Find the downloaded archive — by default, it should be in your Downloads folder.
- Double-click on it to extract the files, and keep the resulting Explorer/Finder window open.

Finally Select all the files in the NOOBS folder and copy them to the Micro SD card.



Step 4

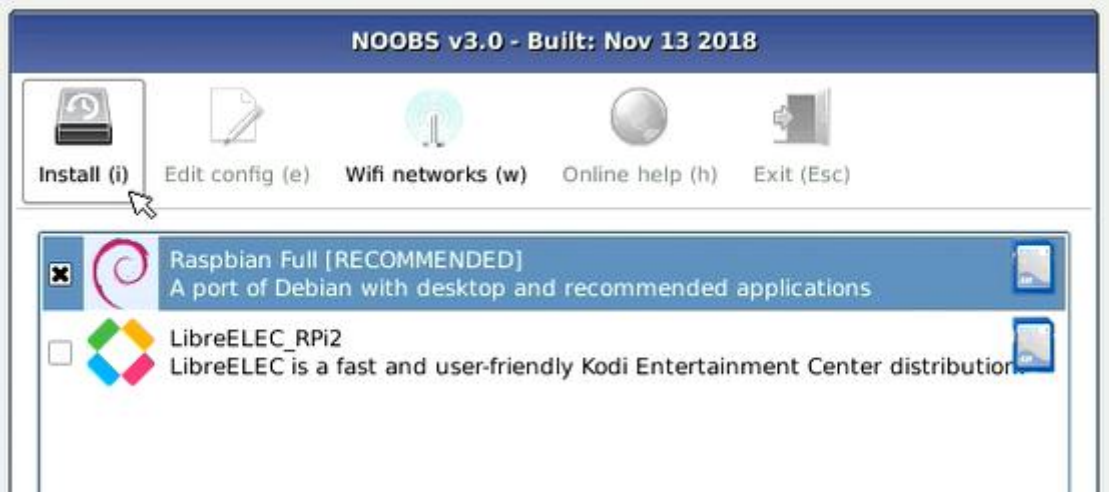
All the files transferred, the Micro SD card pops up.

Step 5

Insert the Micro SD card into the Raspberry Pi. In addition, connect the screen, keyboard and mouse to it. Finally you are also recommended to use the power adapter of Raspberry Pi to power your car for that the first test will take a long time.

Step 6

It will go to the NOOBS interface after starting up. If you use NOOBS LITE, you need to select Wi-Fi networks (w) first. Tick the checkbox of the Raspbian and click Install in the top left corner. The NOOBS will help to conduct the installation automatically. This process will take a few minutes.



Step 7

When the installation is done, the system will restart automatically and the desktop of the system will appear.

Step 8

If you run Raspberry Pi for the first time, the application of "Welcome to Raspberry Pi" pops up and guides you to perform the initial setup.

Step 9

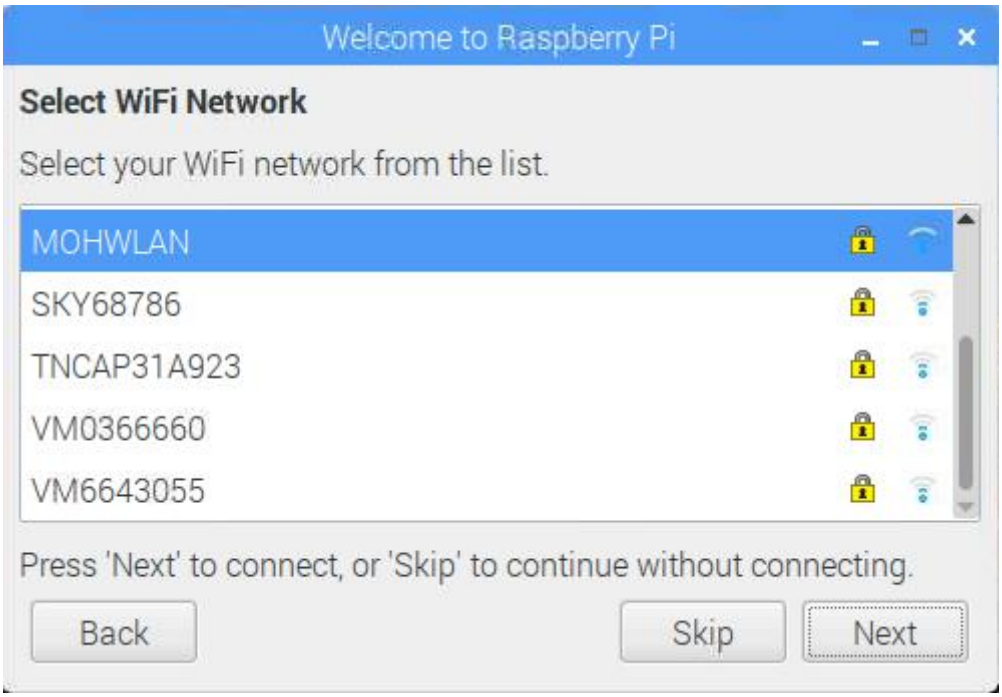
Set country/region, language and time zone, and then click "next" again.

Step 10

Input the new password of Raspberry Pi and click “Next”.

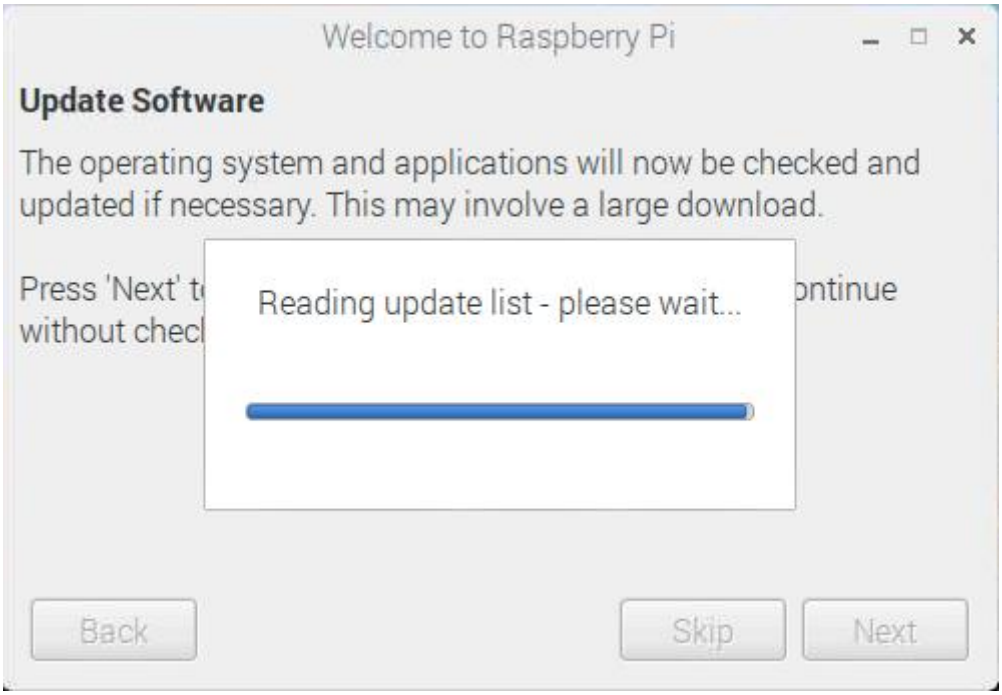
Step 11

Connect the Raspberry Pi to WIFI and click "Next".



Step 12

Retrieve update.



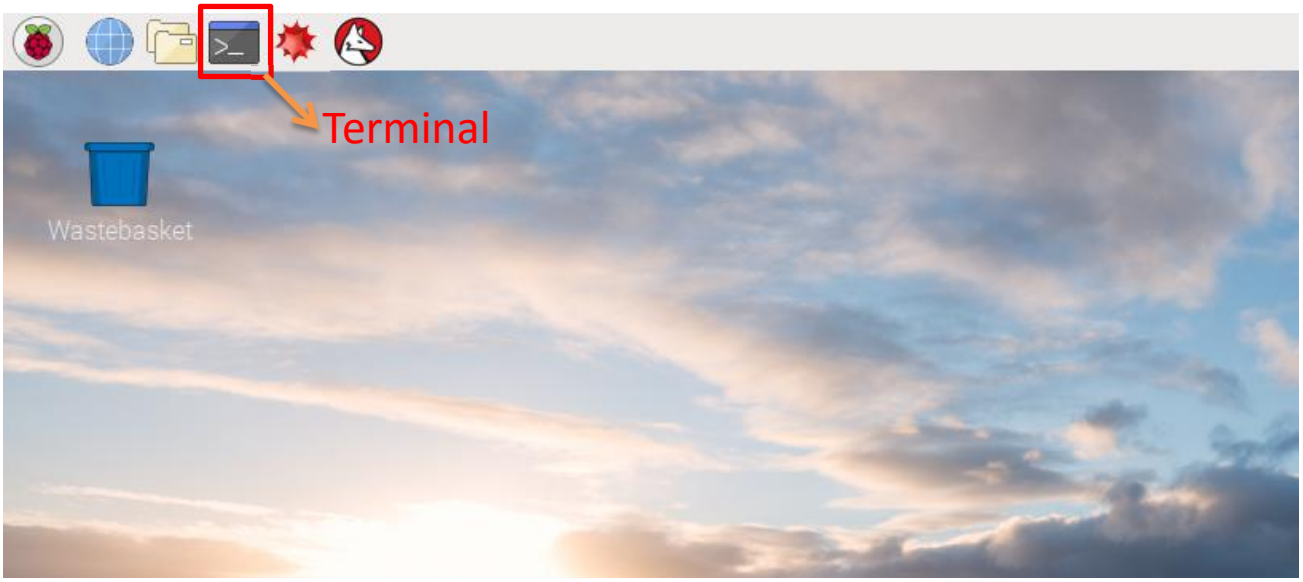
Step 13

Click "Done" to complete the Settings.

Now we can run the Raspberry Pi.

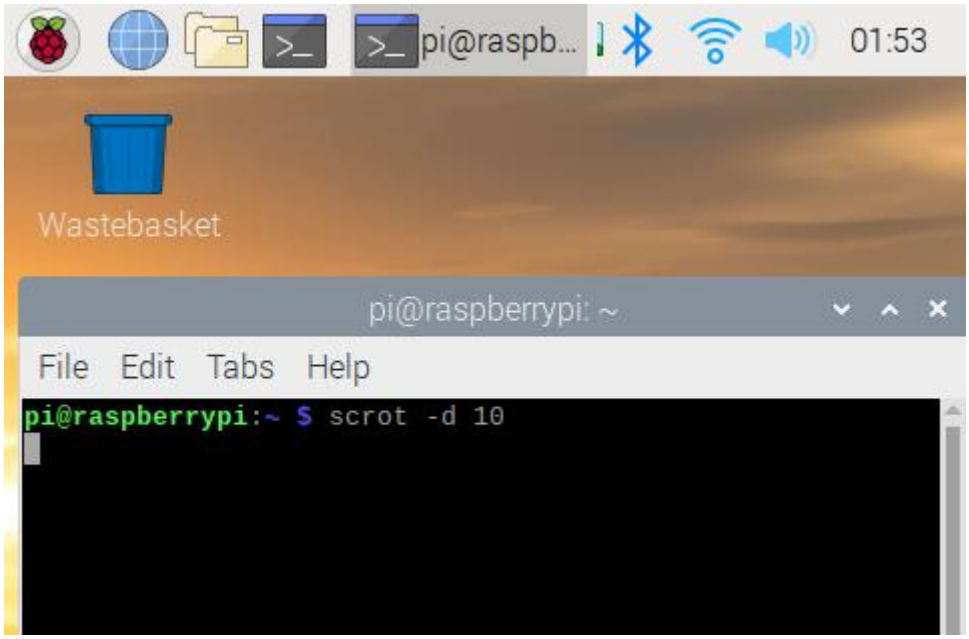
Step 14

Click the Terminal icon on the top left corner.



Step 15

Then you can input the commands on the Terminal.



Note: You can check the complete tutorial of NOOBS on the official website of the Raspberry Pi: https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up

If You Have No Screen

If we don't have a screen, we can directly write the Raspbian system to the Micro SD card and we can control the Raspberry Pi on PC remotely by directly modifying the configuration file of the network settings in the Micro SD card.

- Burn System

Step 1

Prepare the tool of image burning. Here we use the **balenaEtcher**. You can download the software from the link: *https://www.balena.io/etcher/*

Step 2

Download the complete image on the official website by clicking this link: *https://www.raspberrypi.org/downloads/raspbian/*. There are three different kinds of Raspbian system available, You are recommend to install the version: **Raspbian Buster with desktop and recommended software**.

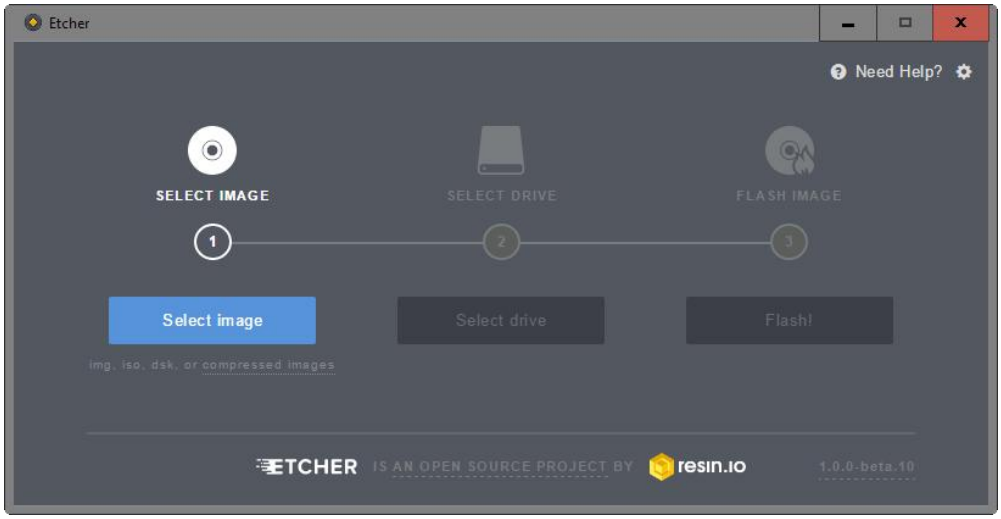
Step 3

Unzip the package downloaded and you will see the *.img* file inside.

> **Note**: DO NOT extract the file.

Step 4

Plug the USB Card Reader into the computer, then you can burn the image file with the Etcher.



Step 5

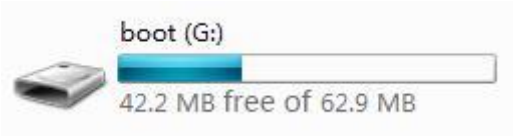
At this point, Raspbian is installed. **Keep the USB card reader plug in your computer**. If you want to apply it, next you need to complete the settings accordingly.

- Connect the Raspberry Pi to the Internet

There are two methods to help get the Raspberry Pi connected to the network: the first one is using a network cable, the other way is using WIFI. We will talk in detail about how to connect via WIFI as below.

If you want to use the WIFI function, you need to modify a WIFI configuration file wpa_supplicant.conf in the Micro SD card by your PC that is located in the directory **/etc/wpa_supplicant/**.

If your personal computer is working on a linux system, you can access the directory directly to modify the configuration file; however, if your PC use Windows system, then you can't access the directory and what you need next is to go to the directory, */boot/* to create a new file with the same name, wpa_supplicant. conf.



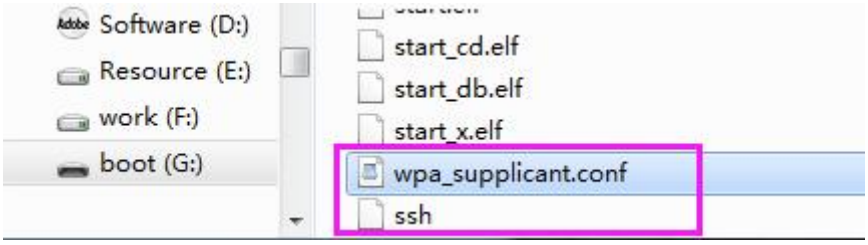
Input the following content in the file.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=GB
network={
ssid="WiFi-A"
psk="Sunfounder"
key_mgmt=WPA-PSK
priority=1
}
```

You need to replace "**WiFi-A**" with your custom name of WiFi and "**Sunfounder**" with your password. By doing these, the Raspbian system will move this file to the target directory automatically to overwrite the original WIFI configuration file when it runs next time. After doing this step, you also need to **keep the USB card reader plug in your computer.**

- Start SSH

To use the function of remote control of the Raspberry Pi, you need to start SSH firstly that is a more reliable protocol providing security for remote login sessions and other network services. Generally, SSH of Raspberry Pi is in a disabled state. Additionally, if you want to run it, you need to create a file named SSH under directory /boot/.



Now, the Raspbian system is configured. You can plug out the USB card reader and then plug the Micro SD card into the Raspberry Pi.

- Get the IP Address

You are also recommended to use the power adapter of Raspberry Pi to power your car for that the first test will take a long time.

After the Raspberry Pi is powered on, we need to get the IP address of it. There are many ways to know the IP address, and two of them are listed as follows.

1. Checking via the router

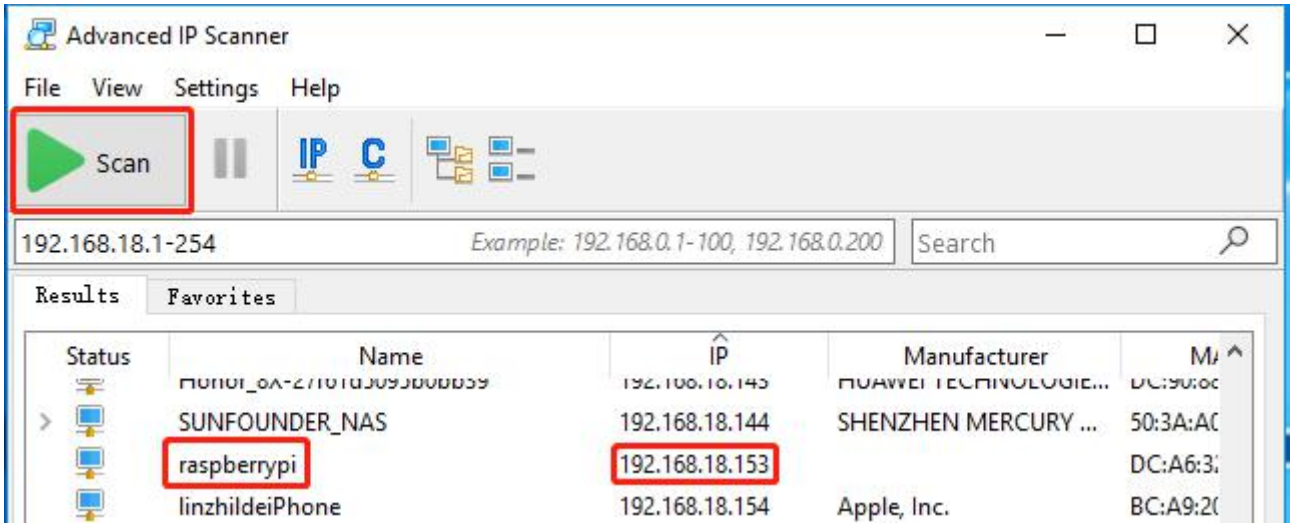
If you have permission to log in the router(such as a home network), you can check the addresses assigned to Raspberry Pi on the admin interface of router.

The default hostname of the system, Raspbian is **raspberrypi**, and you need to find it. (If you are using ArchLinuxARM system, please find alarmpi.)

2. Network Segment Scanning

You can also use network scanning to look up the IP address of Raspberry Pi. You can apply the software, **Advanced IP scanner(download from Google)**.

Click **Scan** and the name of all connected devices will be displayed. Similarly, the default hostname of the Raspbian system is **raspberrypi**, now you need to find the hostname and its IP.



- Use the SSH Remote Control

We can open the Bash Shell of Raspberry Pi by applying SSH. Bash is the standard default shell of Linux. The Shell itself is a program written in C that is the bridge linking the customers and Unix/Linux. Moreover, it can help to complete most of the work needed.

➢ For Linux or/Mac OS X Users

Step 1

Go to **Applications**->**Utilities**, find the **Terminal**, and open it.

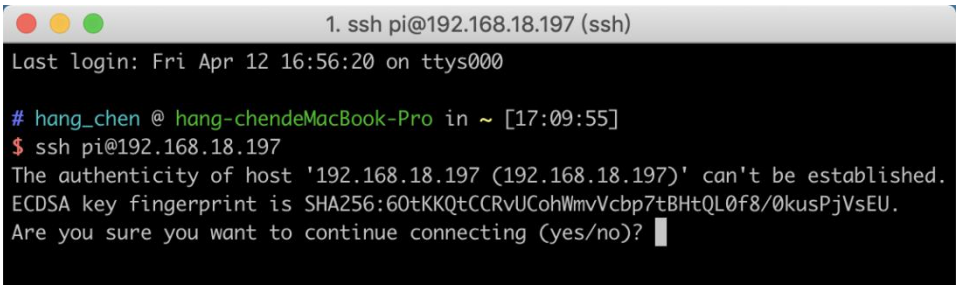
Step 2

Type in **ssh pi@ip_address** . "pi"is your username and "ip_address" is your IP address. For example:

```
ssh pi@192.168.18.197
```

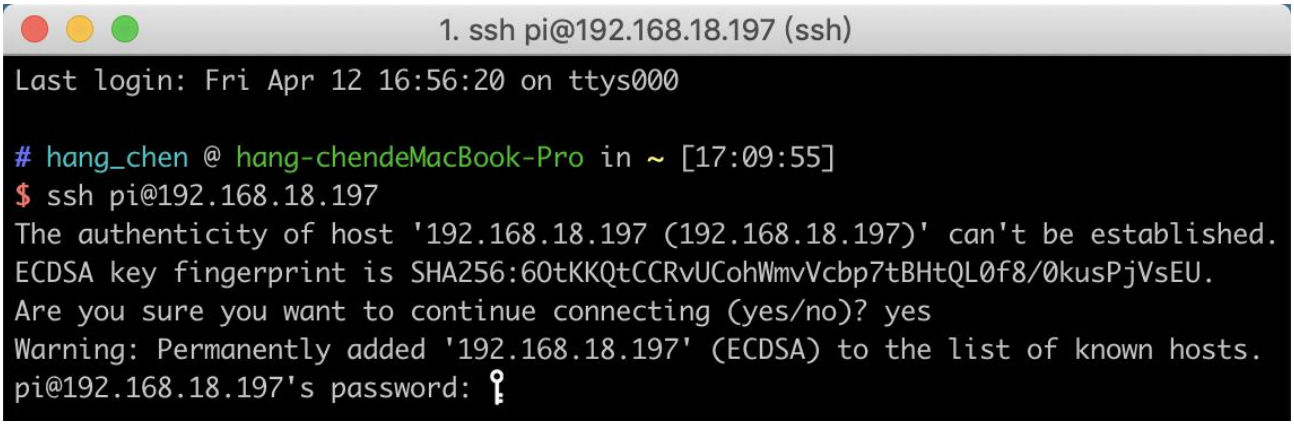
Step 3

Input"yes".



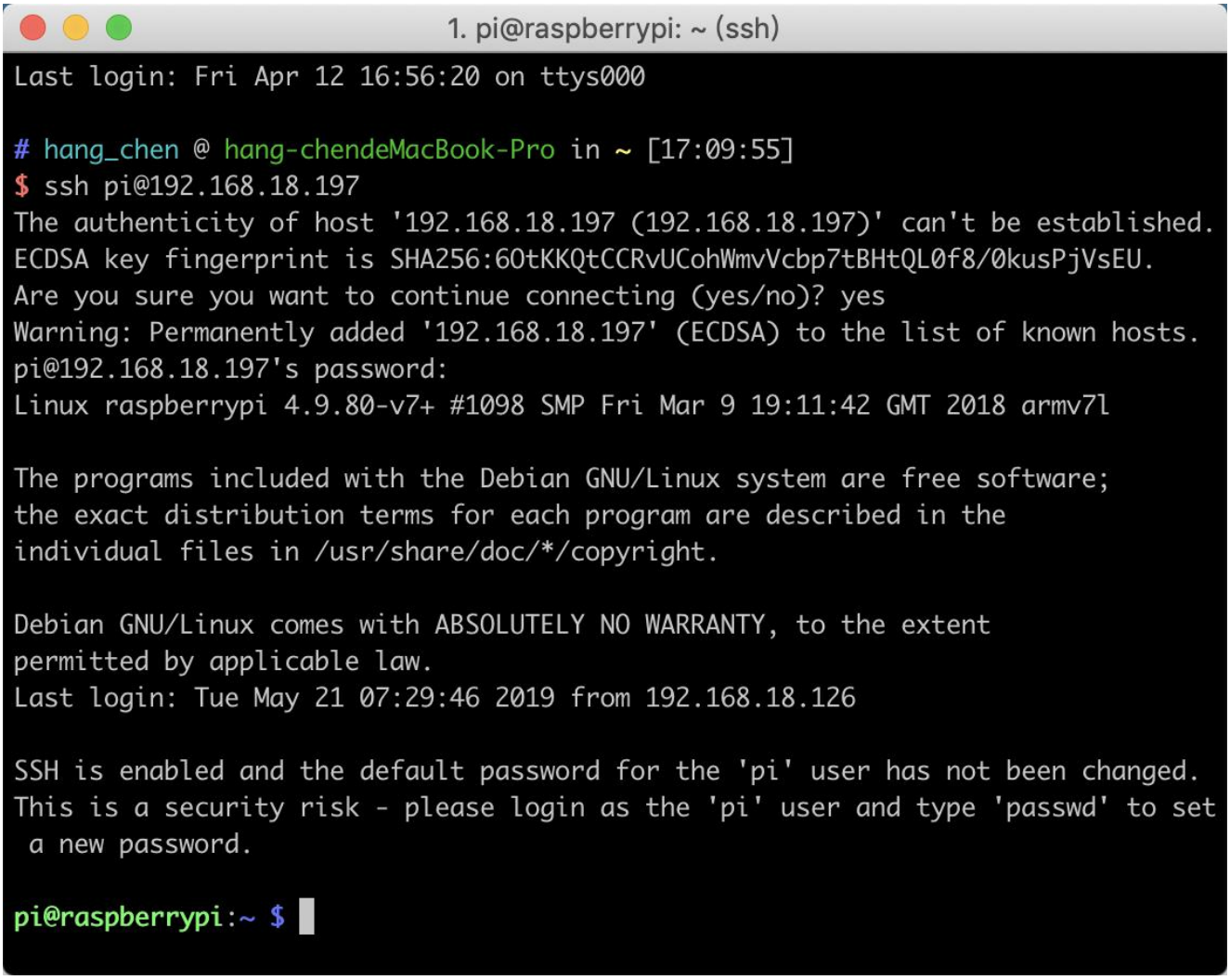
Step 4

Input the passcode and the default password is **raspberry**.



Step 5

We now get the Raspberry Pi connected and are ready to go to the next step.



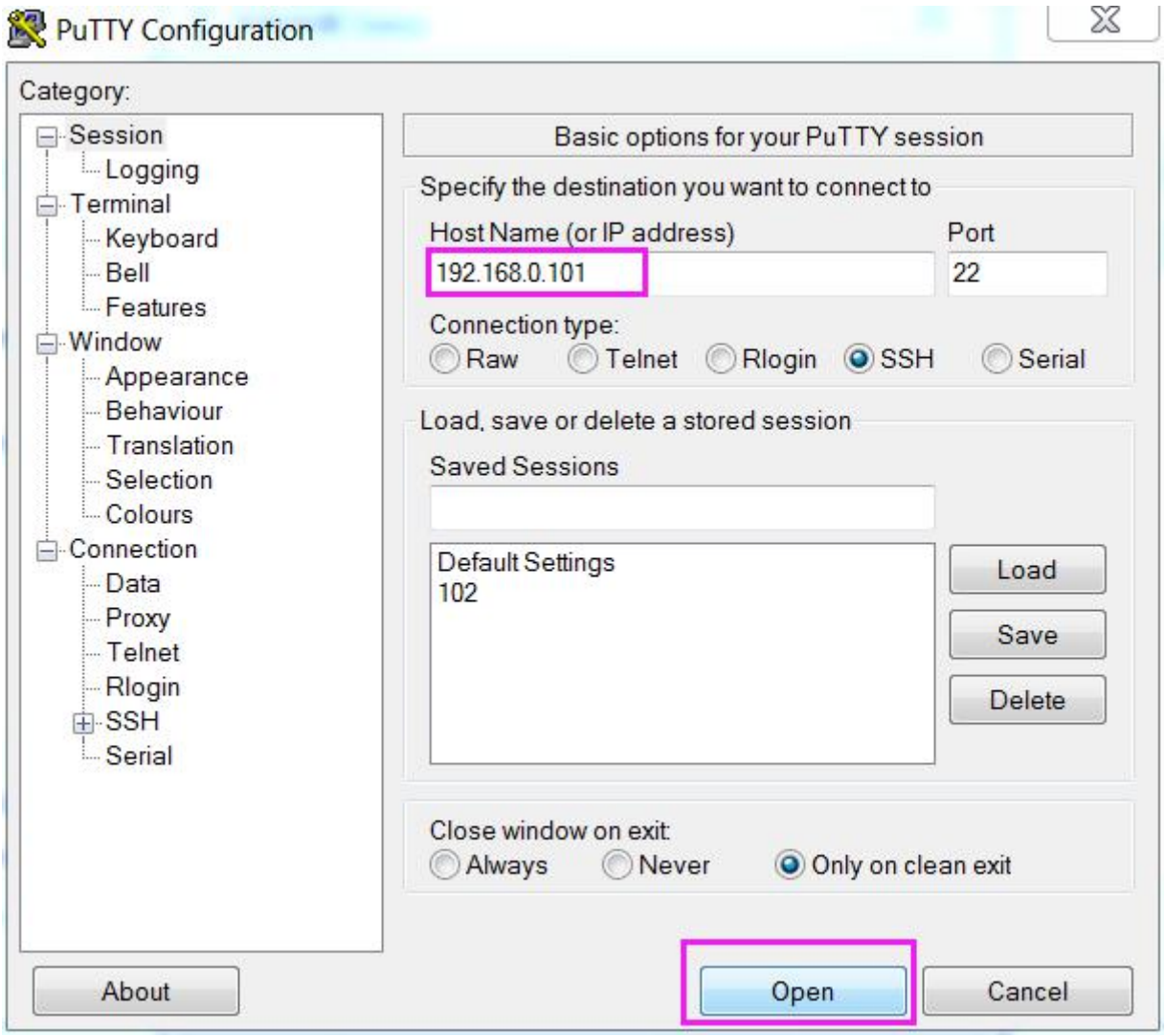
Note: When you input the password, the characters do not display on window accordingly, which is normal. What you need is to input the correct passcode.

➢ For Windows Users

If you're a Windows user, you can use SSH with the application of some software. Here, we recommend **PuTTY**(You can download from Google).

Step 1

Download PuTTY. Open PuTTY and click **Session** on the left tree-alike structure. Enter the IP address of the RPi in the text box under **Host Name (or IP address)** and **22** under **Port** (by default it is 22).

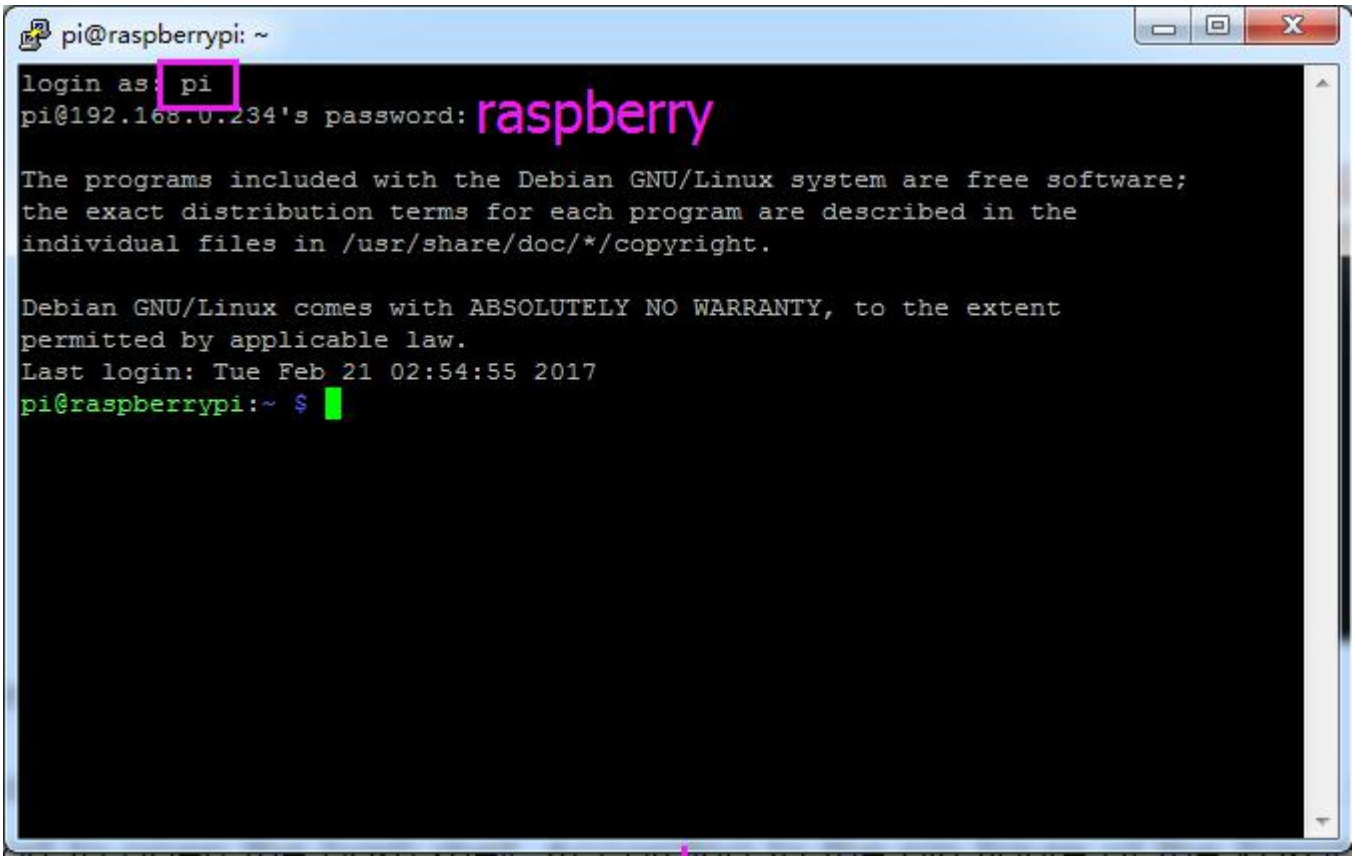


Step 2

Click **Open**. Note that when you first log in to the Raspberry Pi with the IP address, there prompts a security reminder. Just click **Yes**.

Step 3

When the PuTTY window prompts "**login as:**", type in "**pi**"(the user name of the RPi), and **password:** "raspberry" (the default one, if you haven't changed it).



Note: When you input the password, the characters do not display on window accordingly, which is normal. What you need is to input the correct password.

Here, we get the Raspberry Pi connected and it is time to conduct the next steps.

Servo Configuration

Since the servos used in this kit are adjusted by software and there's no such physical sticking point as other servos, here we need to configure the servo via software. First you need to finish some software installation before the configuration.

Note: Please do forget to put in the battery and slide the power switch to ON in following chapters.

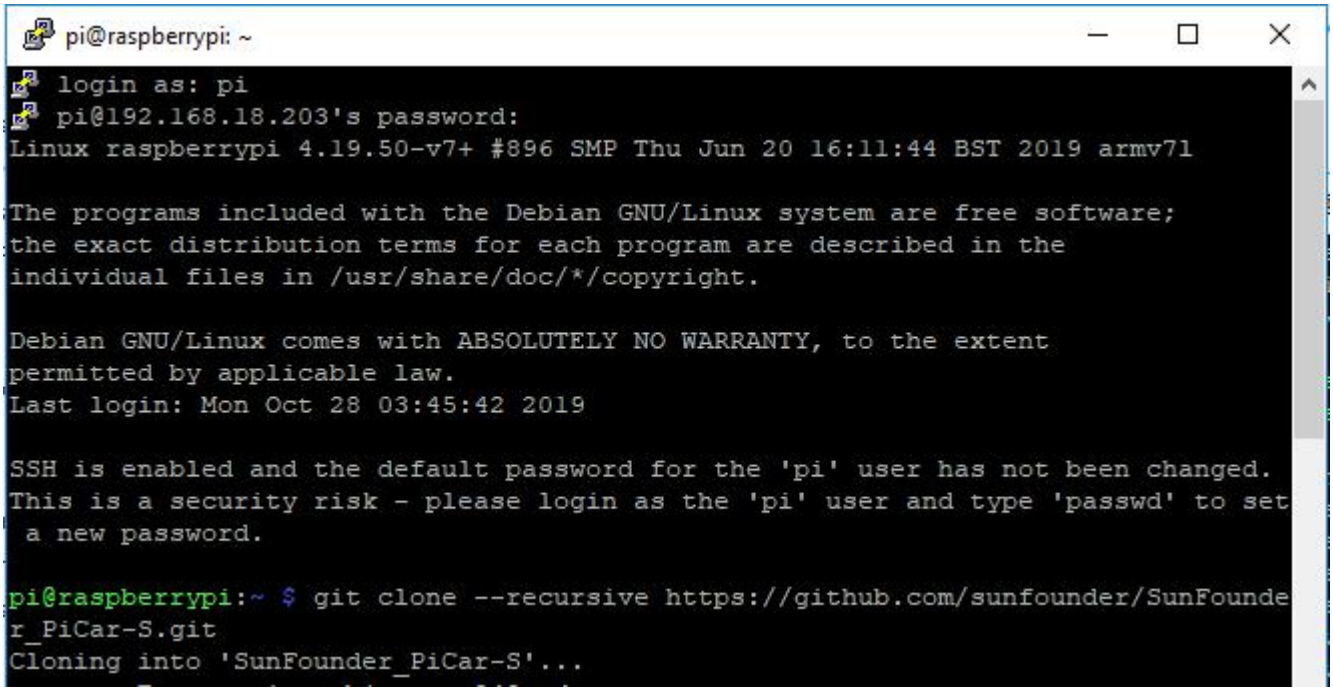
Get Source Code

You can find the source code in our Github repositories. Download the source code by *git clone*:

```
git clone --recursive https://github.com/sunfounder/SunFounder_PiCar-S.git
```

Note: Please pay attention to your typing – if you get the prompt of entering your user name and password, you may have typed wrong. If unluckily you did so, press Ctrl + C to exit and try again.

Check by the **ls** command, then you can see the code directory *SunFounder_PiCar-S*:



Go to the Code Directory

```
cd ~/SunFounder_PiCar-S/
```

Enter the code directory and you can see the installation script:

```
pi@raspberrypi:~ $ cd ~/SunFounder_PiCar-S/
pi@raspberrypi:~/SunFounder_PiCar-S $ ls
example  install_dependencies  LICENSE  maps  README.md  show
pi@raspberrypi:~/SunFounder_PiCar-S $
```

Install the Environment via Script

You can get all the required software and configuration done with the installation script. If you want to do step by step instead, refer to the operations in **Appendix 1: Installing Manually**.

```
sudo ./install_dependencies
```

Notes:

1. The installation script will install the required components and configure for the running environment. Make sure your Raspberry Pi is connected to the Internet during the installation, or it would fail.
2. The Raspberry Pi will prompt you to reboot after the installation. You're recommended to type in **yes** to reboot.

Set the Servo to 90 Degrees

After reboot, type in the command:

```
picar
```

You can see three commands here.

```
pi@raspberrypi:~ $ picar
Usage:  picar [Command] [value]
Commands:
  servo-install              Set 16 channel servos to 90 degree for installation
  front-wheel-test [chn]     Test the steering servo connect to chn, chn default
 0
  rear-wheel-test            Test the rear wheel
```

The first one **servo-install** is for **servo adjustment**, which is used after the front wheels are assembled. The servo will rotate to 90 degrees after this command is run, so we will use this command here.

```
picar servo-install
```

```
pi@raspberrypi:~ $ picar servo-install
Servo now is set to 90 degree.
```

Notes:

If the "OSError: [Errno 121] Remote I/O error" error message appears, open raspi-config:

```
sudo raspi-config
```

Then choose 5 Interfacing Options → P5 I2C → <YES> to enable I2C service.

After the code is running, insert the rocker arm into the servo. You will see the rocker arm is rotate in clockwise and counterclockwise, then stop at a specific location. It means the servo is good. If the any of the conditions below happened to your servo, your servo is bad:

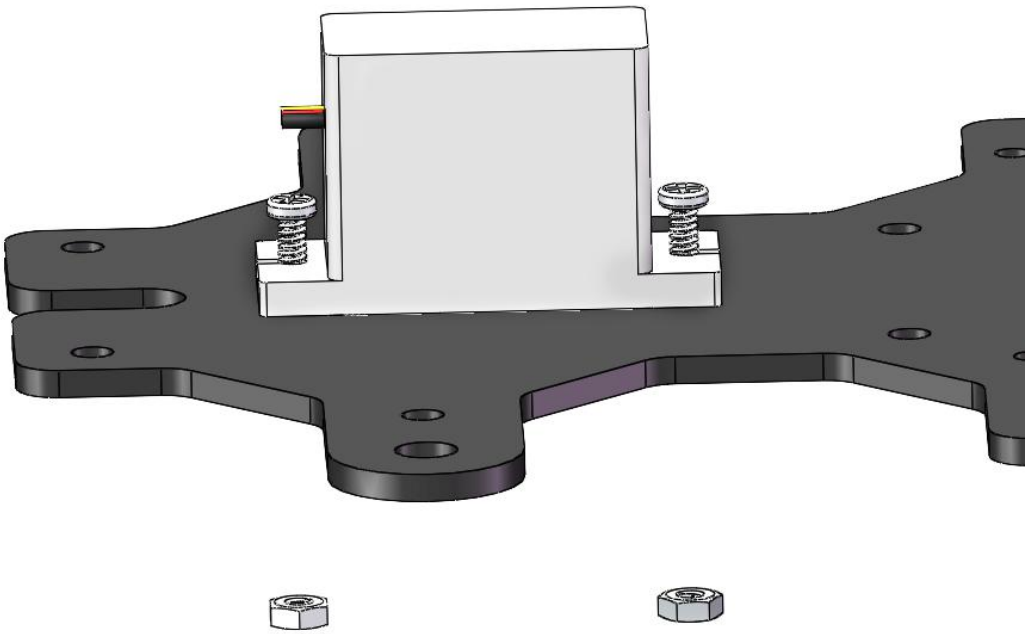
1) Noisy, hot.
2) If unplug the servo line and rotate the rocker arm, it sounds like “ka” “ka” “ka” or there has no sounds of gear driving.
3) Rotate slowly but continuously.

If you find one of the conditions above, please send e-mail to **service@sunfounder.com** . We will change a new one to you. If it is broken in the process of using or assembling, you should go to the official website www.sunfounder.com to buy.

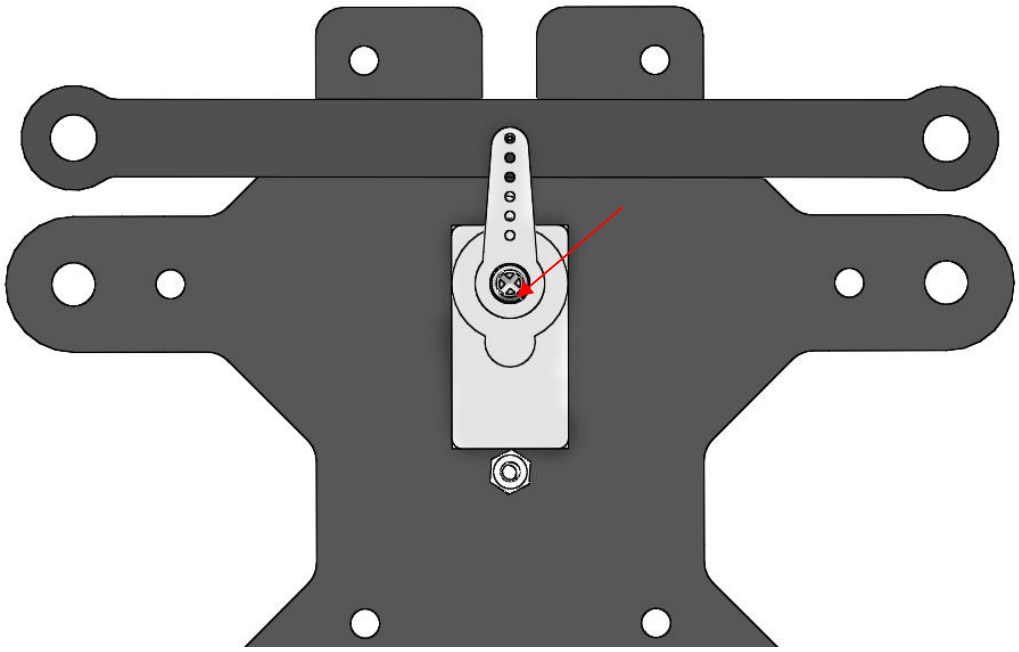
Build the Rest of the Car

Please keep the command **servo-install** running in the whole process of assembly.

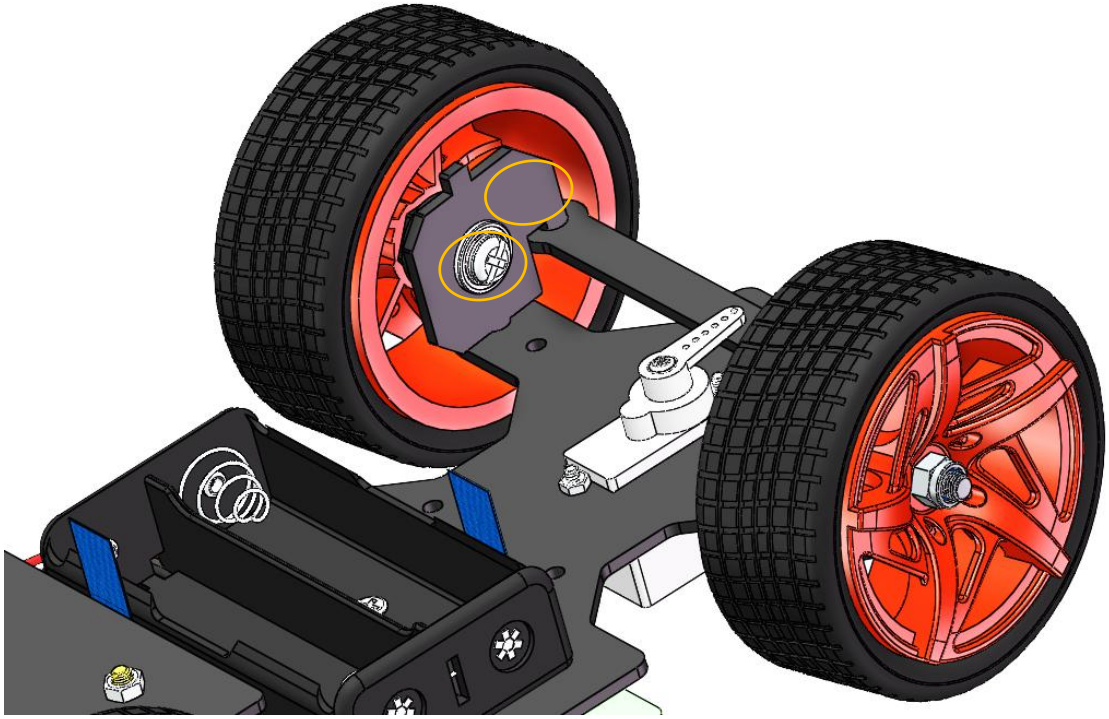
Mount the steering servo to the Upper Plate with two **M2x8 Screws** and two **M2 nuts** (pay attention to the direction of the servo wires):



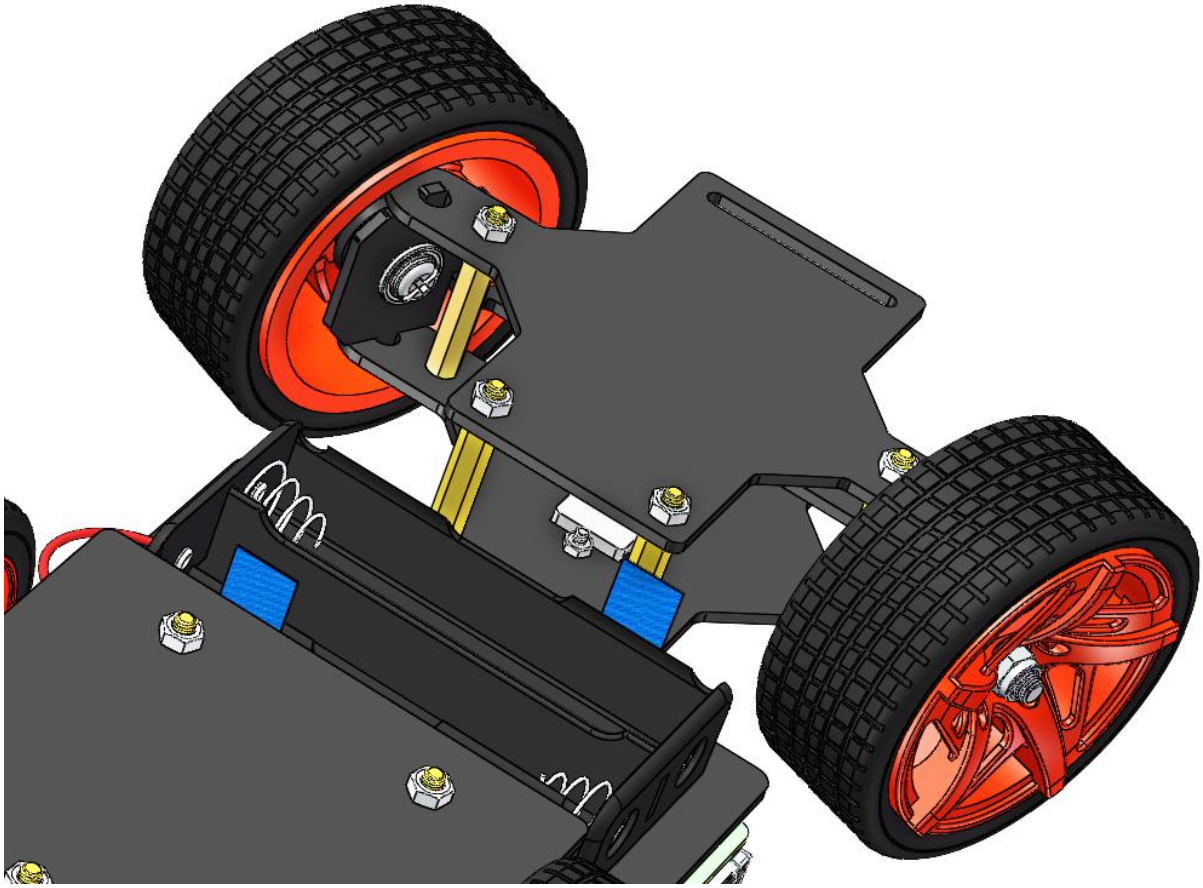
Connect the **Steering Linkage** and the **Rocker Arm** with **Rocker Arm Fixing Screw** (the shortest).



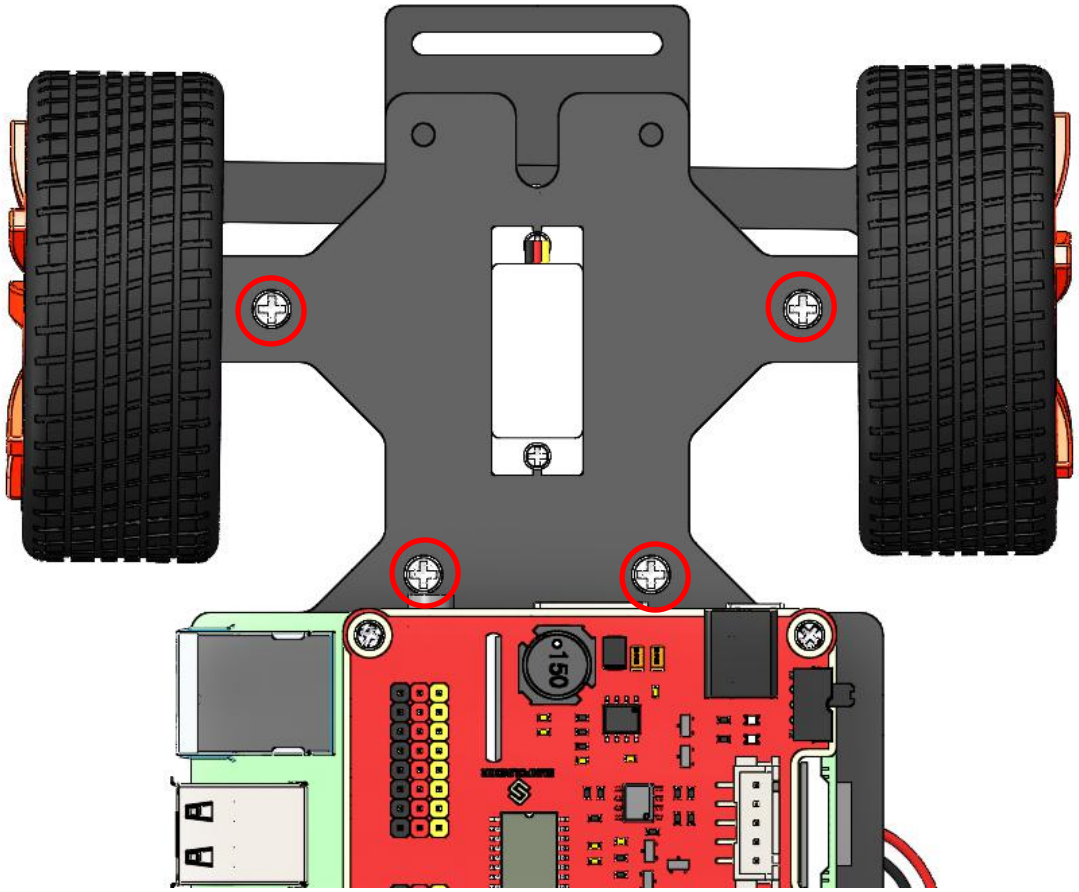
Mount the wheels onto the Upper Plate carefully.



Then put the assembled Front Half Chassis onto the Upper Plate with standoffs aligned with the holes.



Hold them carefully, turn upside down, and fasten the standoffs and the Upper Plate with four **M3x8 screws**:



So now, the whole assembly is DONE! Congratulations!

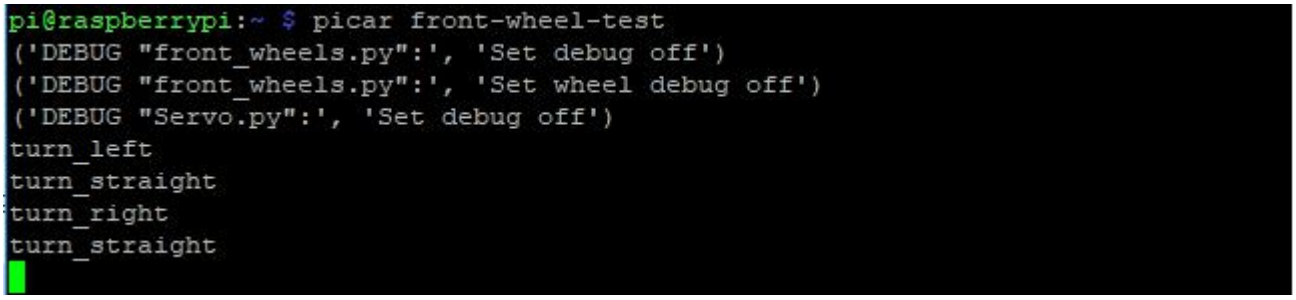
Calibration

Calibrate the Servo

Remember the commands to adjust the servo to 90 degrees previously? Now, let's talk about the other two commands.

The second command **front-wheel-test** is used to test whether the front wheels can turn flexibly after assembly. When you run this command, it will drive them to turn left and right.

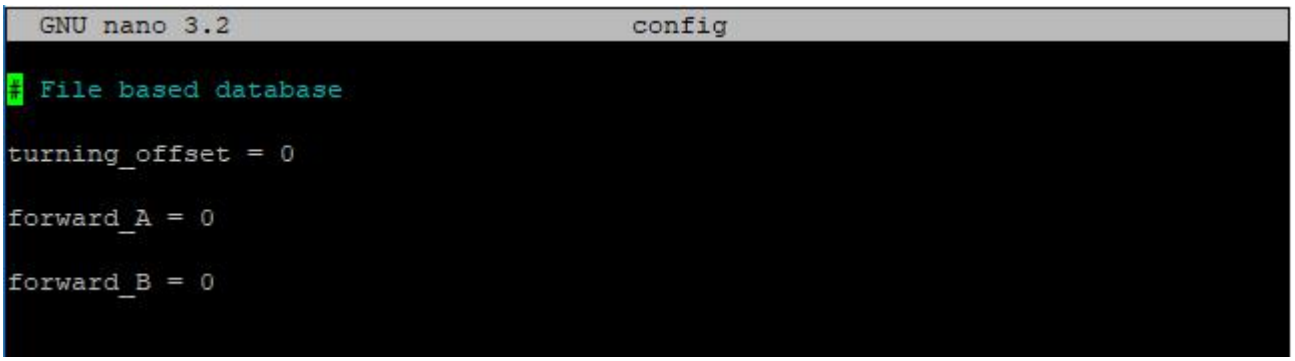
```
picar front-wheel-test
```



You may find the direction of the front wheels is not facing exactly front when they are in the straight status. If there is an obvious deviation from the middle line of the front chassis, reassemble the servo and run **servo-install** again; if it is just a little deviation (like about 0~15 degrees), it can be adjusted by software.

Get into the folder ***SunFounder_PiCar/picar:***

```
cd /home/pi/SunFounder_PiCar/picar
sudo nano config
```



Open the **config** file under the folder with an editor. You can see a few parameters. The value of **turning_offset** is used to adjust the front wheels. Its value is **0** by default. If you want to make the front wheels **turn right a bit**, just modify it to a **larger numbe**r; to make it more **towards the left**, you can set it **smaller** (it can even be a negative

number).

But **DO NOT** over-configure the wheels (recommended a value between -30 and 30), or the servo may be stuck and broken.

After changing the value of turning_offset, press **Ctrl** + **O** to save the changes, and press **Ctrl** + **X** to exit. Run the command **picar servo-install** to check the front wheel's status.

```
picar servo-install
```

If the front wheels is still not facing the exact front, you may need to edit the file **config** for a couple of times. The front wheels may need to be adjusted about 3 to 5 times usually. We can move on to calibration of the rear wheels when the front wheels are done.

Calibrate the Motors

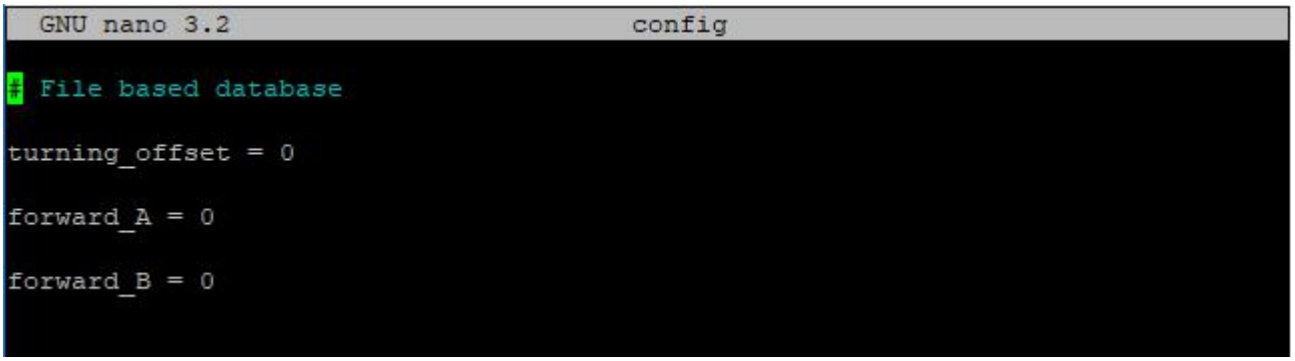
Since the wiring of the two DC motors is random, the VCC and GND of a motor may be connected to the wheel reversely, causing the wheel to spin forward when it should do backward as configured in the code. Thus we can use the third command which will drive the rear wheels to simultaneously speed up and slow down alternately.

```
picar rear-wheel-test
```

```
pi@raspberrypi:~/SunFounder_PiCar/picar $ picar rear-wheel-test
('DEBUG "back_wheels.py":', 'Set debug off')
('DEBUG "TB6612.py":', 'Set debug off')
('DEBUG "TB6612.py":', 'Set debug off')
('DEBUG "PCA9685.py":', 'Set debug off')
('Forward, speed =', 0)
('Forward, speed =', 1)
('Forward, speed =', 2)
('Forward, speed =', 3)
('Forward, speed =', 4)
```

Check whether both the two rear wheels rotate direction is the same as the screen. Note that the two wheels are driven by the two motors separately. It may happen that one rotates forward, while the other does backwards. If so, we need to adjust one or both two wheels which rotate reversely under that command.

```
cd /home/pi/SunFounder_PiCar/picar
sudo nano config
```



forward_A and **forward_B** are to change the default spinning direction of the two motors. The value can only be **0** or **1**, which represents clockwise and counterclockwise rotation. By default, it's **0** for both parameters. Thus if a wheel spins reversely, you only need to change the corresponding parameter for the wheel to **1**.

Press **Ctrl** + **O** to save the changes, and press **Ctrl** + **X** to exit.

Run the command **picar rear-wheel-test** again to check whether the rear wheels are rotating in accordance with the command.

```
picar rear-wheel-test
```

Copy *config* to the directory *example* under *PiCar-S*.

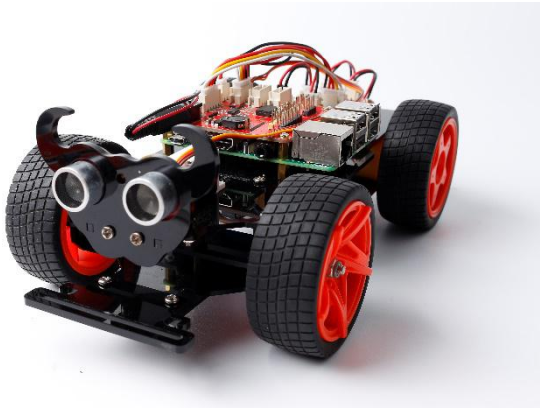
```
cp config ~/SunFounder_PiCar-S/example
```

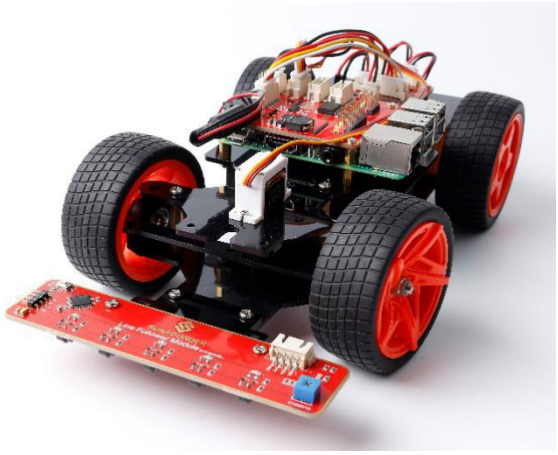
Arming the Car!

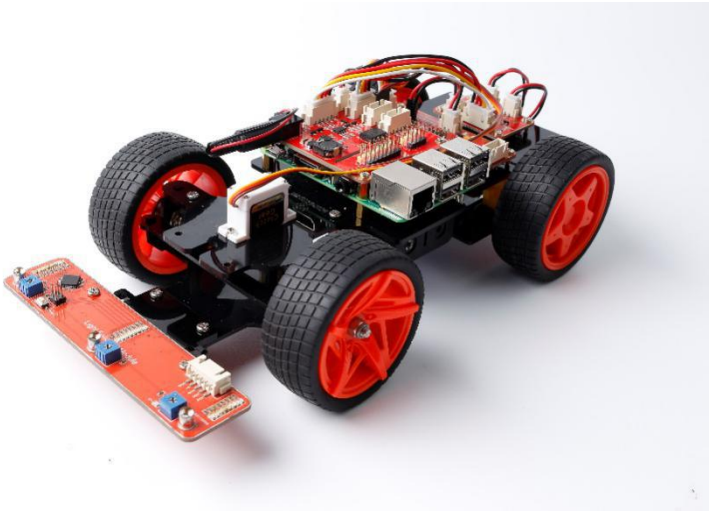
A car without sensor modules is unarmed just like a man without sight and hearing, thus he has no feeling for the surrounding environment. So what we are going to do is arm the car, allowing it to detect the surroundings. Now let's turn the **PiCar** into the **PiCar-S**.

What exactly is the PiCar-S? ------- We arm the PiCar with some sensors, which endow the car with the ability to collect and process the data. The **sensor modules** to the **PiCar** is what the **cartridges** to the **game console**; they are added to the basic design of the game and thus richening the play. It's also similar to the code. The processor will use *SunFounder_PiCar* to drive the car's movement, and call the corresponding code package for different modules (***SunFounder_Light_Follower***, ***SunFounder_Line_Follower***, ***SunFounder_Ultrasonic_Avoidance***).

Assemble the desired sensor module according to the wiring in corresponding module instructions below. Have fun with **The Transformer**!







Obstacle Avoidance

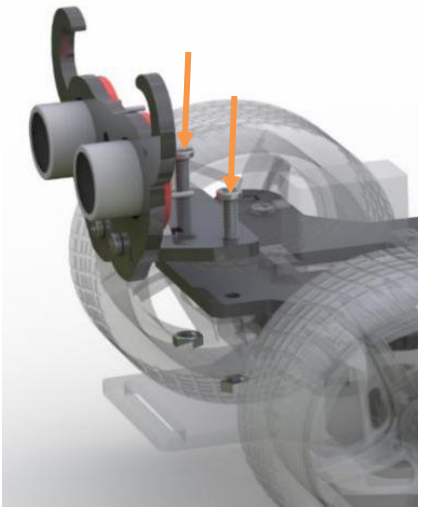
- How it Works

The ultrasonic obstacle avoidance module detects and transfers the collected data to Raspberry Pi that can calculate the distance from the obstacle. The Pi will send a command to adjust the front wheels and rear wheels direction and rotation to control the PiCar-S walk away from the obstacle if there is one.

- Procedures

Step 1 Assembly

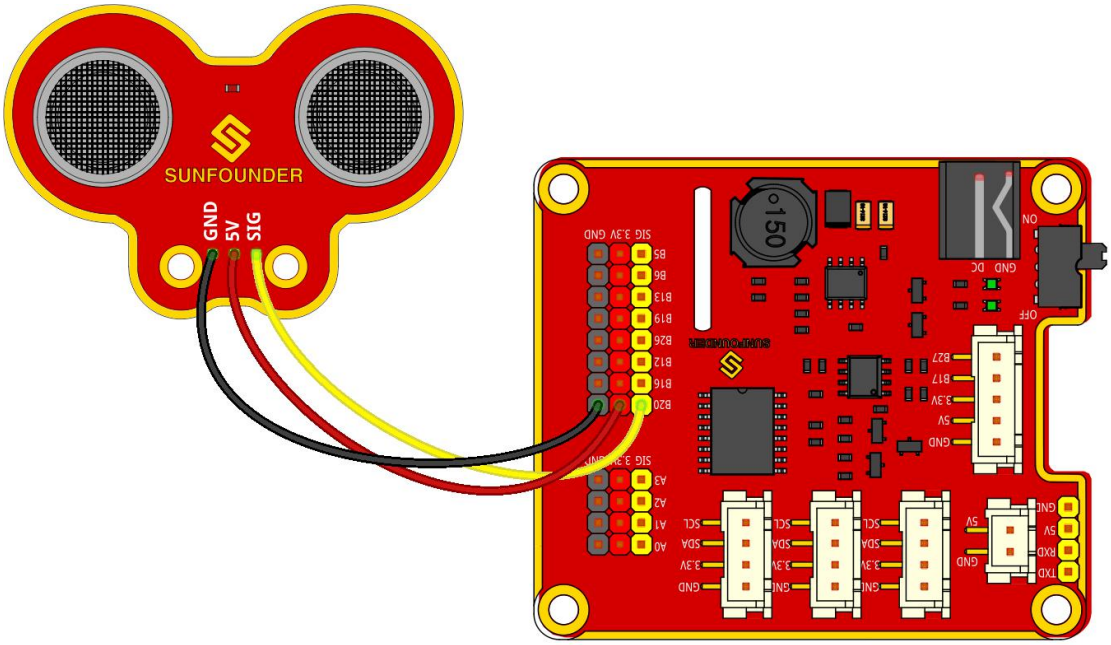
Connect the ultrasonic module and the ultrasonic connector to the ultrasonic support with **M3*10 screws** and **M3 nuts**. Reminder: It would be easier to place the nuts into the slots with your fingers to hold underneath. Then assemble them to the Upper Plate with **M3*10 screws** and **M3 nuts**.



Step 2 Wiring

Connect the ultrasonic obstacle avoidance to Robot HATS with a 3-pin anti-reverse cable as shown below.

Ultrasonic module can have a 5V or 3.3V power supply. Here, we give it a 3.3V power supply.



Step 3 Test

First, test the ultrasonic obstacle avoidance module before applying.

```
cd ~/SunFounder_PiCar-S/example/
python3 test_ultrasonic_module.py
```

```
pi@raspberrypi:~/SunFounder_PiCar-S/example $ python3 test_ultrasonic_module.py
distance 4 cm
Less than 10
distance 4 cm
Less than 10
distance 4 cm
Less than 10
distance 4 cm
Less than 10
distance 4 cm
Less than 10
distance 4 cm
Less than 10
distance 5 cm
Less than 10
distance 6 cm
```

You may find that the distance measurement may be not that accurate. It doesn't matter. This 25kHz ultrasonic module is not a commonly used one, but one has a **horizontal detecting range of about 30~40 degrees**. Thus the distance measured may be not so accurate, but that small range provides convenience for obstacle avoidance. Besides, since the Raspberry Pi is not a real-time operating system, the inaccurate time calculation will affect the accuracy of distance measurement too. However, this ultrasonic module is precise enough for obstacle avoidance.

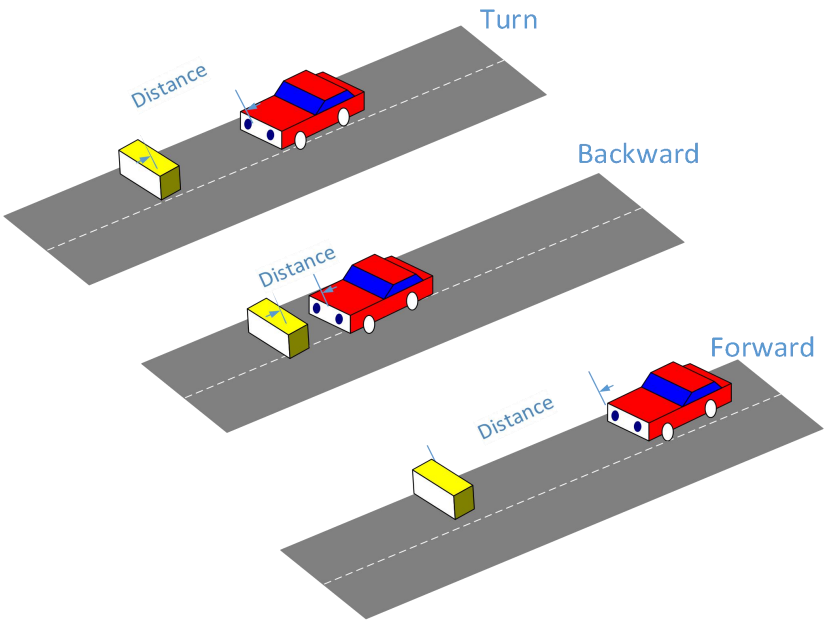
Step 4. Get on the road!

Now we have a general idea of the ultrasonic module's effect after the test above. Let'

s run the code of the ultrasonic obstacle avoidance.

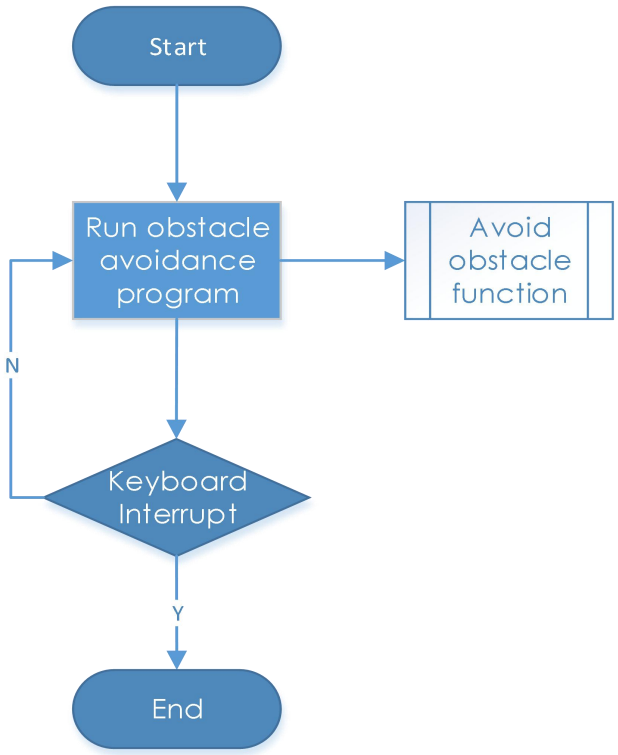
```
python3 ultra_sonic_avoid.py
```

The PiCar-S starts running now. Just place the car on the ground. It will follow the program to turn when it detects an obstacle; if the obstacle is too close, it will move backwards, and turn left/right. You can also modify the threshold of obstacle detecting range and that of moving backwards in the code.



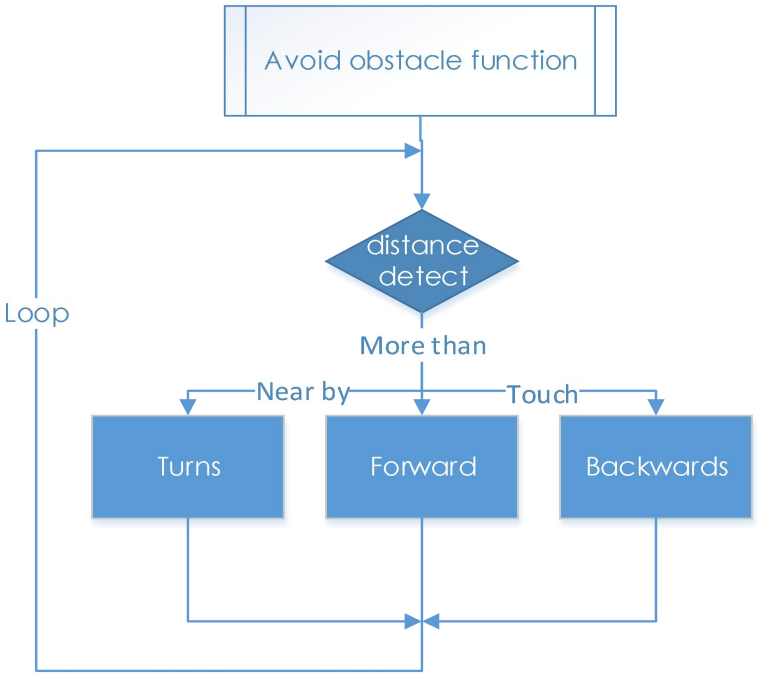
- Code Explanation for ultra_sonic_avoid.py

Whole Work Flow

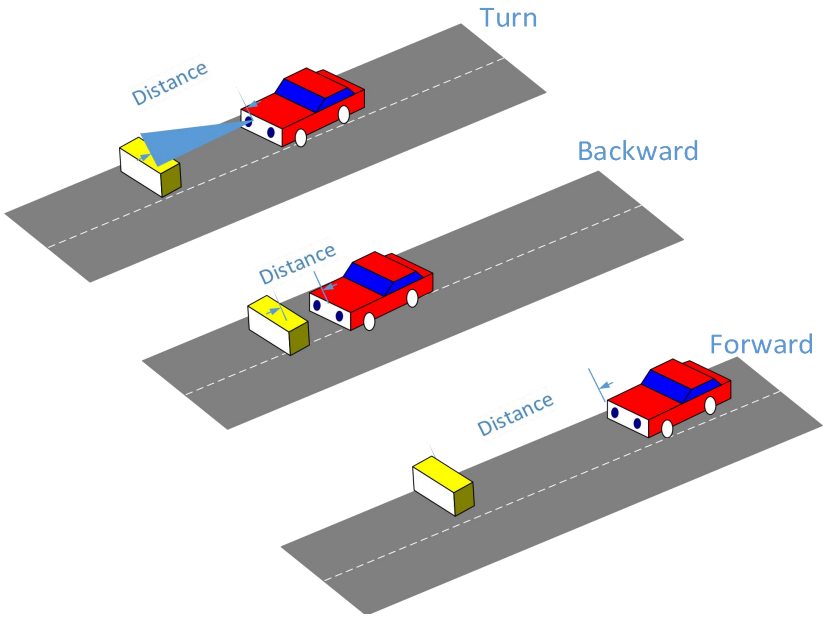


The ultrasonic module returns a digital value, i.e., High or Low level, and the interval time between two levels returned can be converted to the distance to the obstacle. Thus, we call the time module in Python for timing here. The formula to calculate the distance is written in the ultrasonic module's driver. The main program just calls the corresponding program to get the distance value.

Subflow of the Obstacle Avoidance Function



When the car starts, it will detect obstacles and measure the distance in cycle, make judgement, and take actions. Here are three cases: when the distance to the obstacle is equals to the threshold, the car will turn directions; when the distance is less than the threshold, the car will move backwards before turning direction; when the distance is more than the threshold, it will keep moving forwards.



Functions Explanation

ua = Ultra_Sonic.UltraSonic_Avoidance(17)

Create an object **ua** of a *UltraSonic_Avoidance* class in the Ultra_Sonic module. The number in the round bracket is the initial parameter, which represents the pin number the SIG of the module is connected to. Since the BCM naming method is applied, the corresponding pin on the Raspberry Pi is #17.

back_distance and **turn_distance**, two constants are to set the thresholds of the ranging distance.

while() loop

When the detected distance is less than the **back_distance**, the car will move backwards; when it is between **back_distance** and **turn_distance**, the car will turn a direction (you can set the turning angle in the aforementioned parameter **turning_angle** and the angle can be a positive or negative number, for turning left or turning right respectively; **NOTE** that the number of the turning angle should be **-90 to 90** considering the servo's max rotation degrees, or the servo may be burnt.); when the detected distance is greater than the **turn_distance**, the car will keep moving forward.

bw.backward(), making the rear wheels rotate backwards; **bw.forward()**, making the rear wheels spin forward. These two functions in the rear wheel driving module back_wheels are to set the wheel's rotating direction.

bw.set_speed(speed), function in the back_wheels, to set the wheel's rotating speed. The larger the number (within the range 0-100) is, the faster the wheel rotates.

fw.turn(angle), function in the back_wheels, to set the turning angle. The angle is 90 when the car moves straight forwards; reduce the number to turn left, and increase it to turn right.

fw.turn_straight(), making the front wheels return to the angle of moving straight forwards.

More:

back_distance and **turn_distance**

Try to modify the constants to make the car back off and turn away in a desired distance and angle as you like during the obstacle avoidance.

Light Following

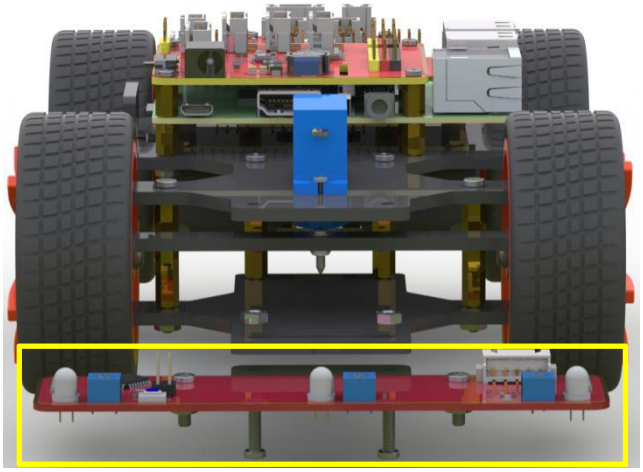
- How It Works

The light follower module detects light sources in the surroundings, and transfers the data to the processor. The processor analyzes the data and finds the direction of the light resource, so it will send a command to control the movement of the front and rear wheels to approach the resource.

- Procedures

Step 1 Assembly

Connect the light follower to the Sensor Connector with **M3*10 screws** and **M3 nuts**, and then assemble them to the car with two **M3*10 screws** and two **M3 nuts**. You're suggested to hold the nuts underneath with your fingers.



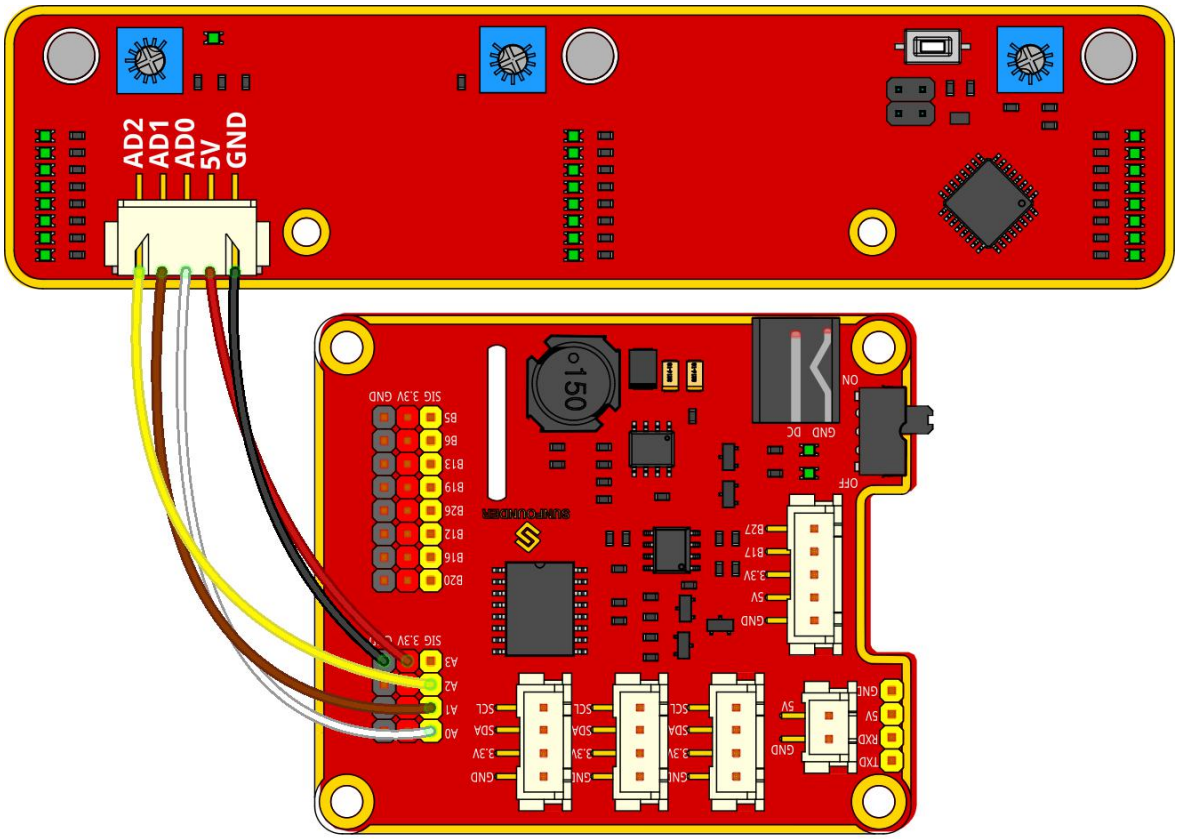
Step 2 Wiring

Connect the light follower to the Robot HATS with a 5-pin anti-reverse cable as shown below.

Light Follower	Robot HATS
AD2	A2
AD1	A1
AD0	A0
5V	3.3V
GND	GND

Note: You may wonder why we connect 5V to 3.3. Well, since the working voltage of the STM8 chip on the light follower is 2.7-5.5V, we can connect it to 3.3V here. **DO NOT** connect 5V to 5V! All the analog ports on the Robot HATS are led from the PCA8591, which is powerd by 3.3V. Therefore, if the voltage is between 3.3V-5V, the output value will always be 255, thus the PCA8591 may be damaged if connected to 5V. Remember to connect to **3.3V**.

The wiring is shown as below:



Step 3 Test

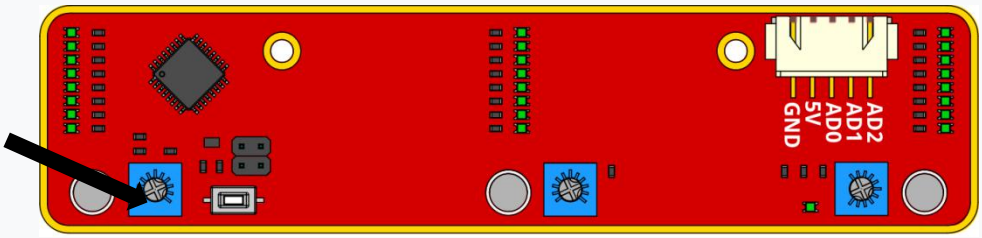
Let's test the light follower first.

```
cd ~/SunFounder_PiCar-S/example/
python3 test_light_module.py
```

```
pi@raspberrypi:~/SunFounder_PiCar-S/example $ python3 test_light_module.py
a0 = 200        a1 = 220        a2 = 204
a0 = 201        a1 = 221        a2 = 204
a0 = 200        a1 = 220        a2 = 204
a0 = 200        a1 = 220        a2 = 204
a0 = 201        a1 = 220        a2 = 204
a0 = 201        a1 = 221        a2 = 205
a0 = 201        a1 = 221        a2 = 205
a0 = 201        a1 = 220        a2 = 204
a0 = 201        a1 = 220        a2 = 204
a0 = 201        a1 = 221        a2 = 204
```

Expose the phototransistors to the light spot of the flashlight. When you increase the light intensity, more LEDs light up, and the output values decrease.

Here we can rotate the blue adjustable resistor to change the values under the same light luminance. The best status is as follows:



1) When there is only one LED lights up, the output value is 255

2) When the light is the brightest and all the LED light up, the output value is about 10-25.

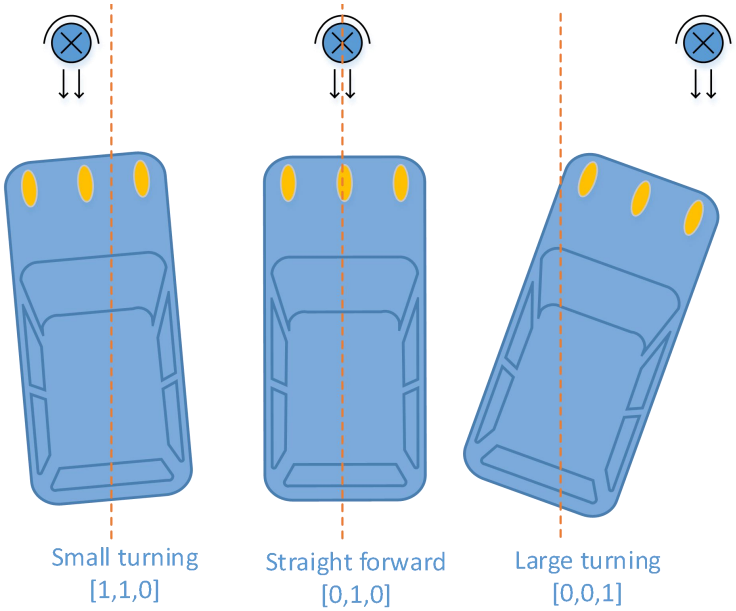
Step 4. Get on the road!

```
python3 light_follower.py
```

```
^Cpi@raspberrypi:~/SunFounder_PiCar-S/example $ python3 light_follower.py
DEBUG "front_wheels.py": Set debug off
DEBUG "front_wheels.py": Set wheel debug off
DEBUG "Servo.py": Set debug off
DEBUG "back_wheels.py": Set debug off
DEBUG "TB6612.py": Set debug off
DEBUG "TB6612.py": Set debug off
DEBUG "PCA9685.py": Set debug off
calibrating.....
calibrate 1
calibrate 2
calibrate 3
calibrate 4
```

The car will enter the light following configuration mode when we run the code above. It will keep turning to the right in a circle to gather the information of light condition in different directions. So just place the car in an open field and wait.

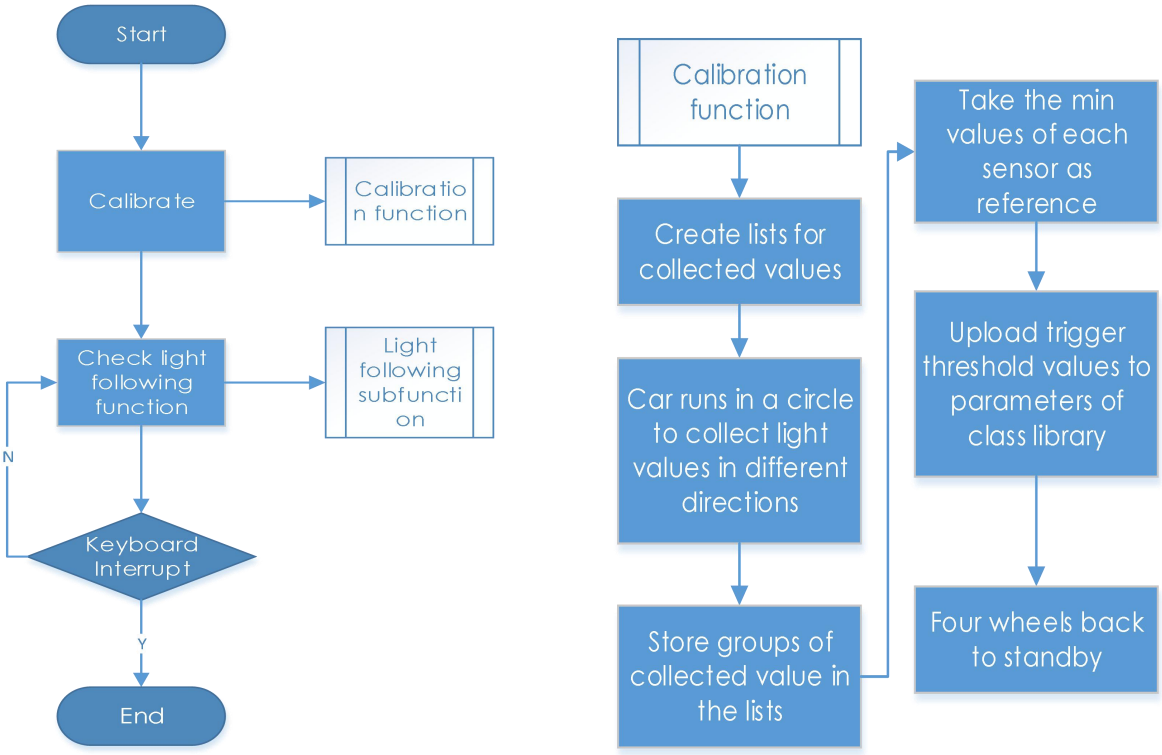
When the calibration is done, the car will stop temporarily. Shine a flashlight on the light follower module, and the car will just follow the light spot as you moves it.



- Code Explanation for light_follower.py

Whole Work Flow

1. Light-sensitive sensors need to be calibrated before actual use because of complex light conditions in the environment. It gathers the information of the ambient light luminance. The car can follow light only when the light source is brighter than the surroundings.



Here write two main functions/modules including light following calibration and light following in the main program.

Subflow of Light Follower Calibration Function

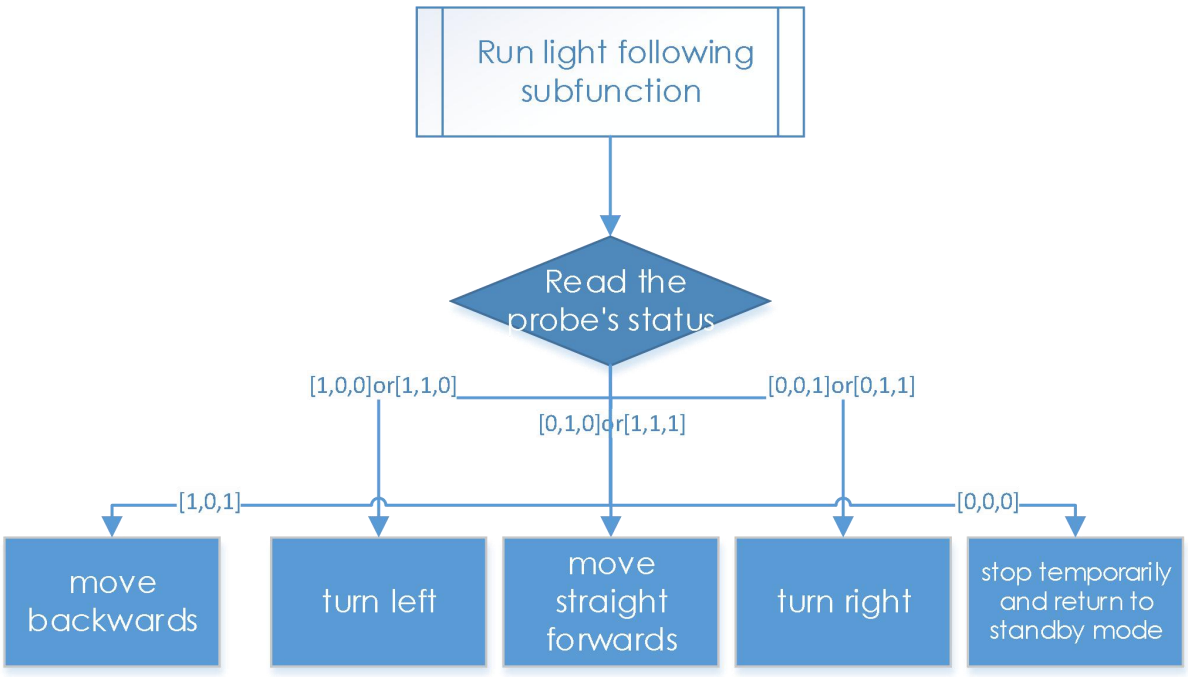
We need to configure three light-sensitive components separately, so we set three lists to store the values in A0, A1, and A2 collected for multiple times. Then pick out the minimum values, which are the output analog values in the brightest conditions.

Since the light source we use is much brighter than the ambient light, we should take the output values in the brightest conditions as reference.

Besides, we should set a threshold value - when the gap between the collected value of the light source and that of the environment is beyond the threshold, trigger the value switching to 0 or 1. Here we use [0,0,0] to represent the three photoresistors' status when they are not triggered. "0" will become "1" when the value detected of the corresponding photoresistor is higher than the threshold. Thus we can set the related action of the car according to the three-element list.

If there is light detected, the car will move and follow it; if there is no light detected, the car will stop temporarily and keep turning to detect in a circle.

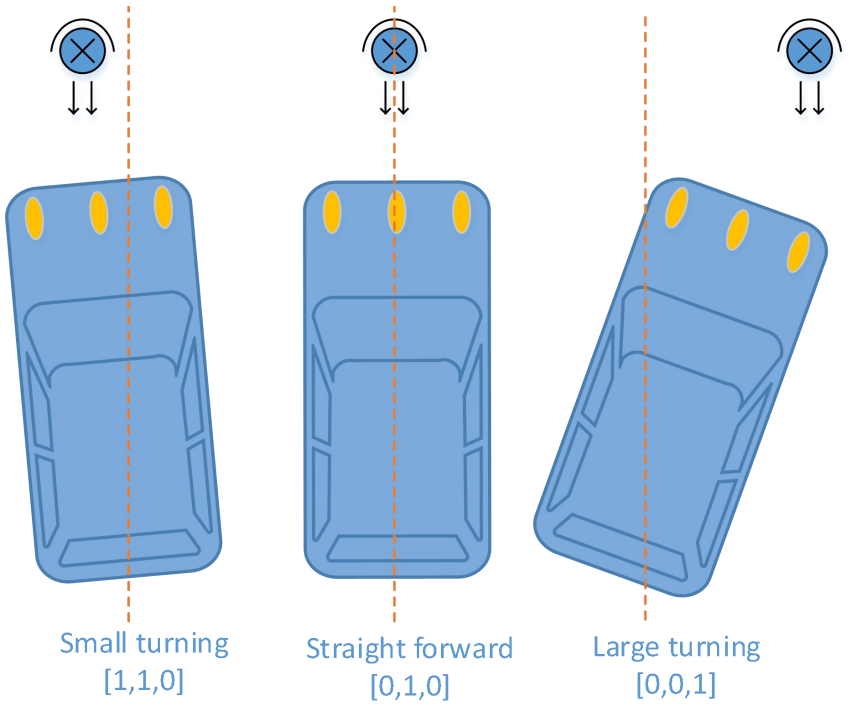
Subflow of Light Following Function



The light follower includes three phototransistors, thus its status list is composed of three elements which represent 8 statuses (based on permutation and combination). And here we need to set related responses to these statuses.

The three elements show the status of the three probes: 1 represents light detected,

and 0, for none. For example, [1,0,0] shows that light is detected only by the left probe, meaning th light source is at the left of the car, thus setting the car's response action as turning left; [1,1,0] means that light is detected on the left and central probes, thus its response action should be set turning left too; and set it as turning right the same way according to the corresponding status. When there is no light detected, the status is [0,0,0], so we set the response action to stop and return to the standby mode.



Here, we need to set another variable – the steering angle – to distinguish between the large-angle and small-angle turning. If the light is at the central left side (status [1,1,0]), we should apply a small-angle turning; if the light is at the edge of the left side (status [1,0,0]), we should apply a large-angle turning.

Functions Explanation

To understand the code, take the software subflows above for reference.

Three Python modules are used in the code including the imported **light_follower_module**, **front_wheels**, and **back_wheel** previously. They are drivers for this kit, respectively for light following, front wheels and rear wheels.

The related classes have been defined here. When the modules are applied to use, objects will be created for related classes, and different parts of hardware will be driven by calling a function by the class object.

For example, for the light following module, we create an object named **lf**:

lf = Light_Follower.Light_Follower()

Then we can call the function by a class object.

A0 = lf.read_analog()[0]

This function **read_analog()** will return a list with three elements, which stores the detected analog values of three probes. Here we use **A0 = lf.read_analog()[0]**, **A1 = lf.read_analog()[1]**, and **A2 = lf.read_analog()[2]** to store three elements of returned value separately into the variables A0-A2.

A **for()** loop is used here cycling 10 times, that is the car will acquire the analog values ten times when the car drives in a circle under the calibration mode. The minimum values will be taken as reference here. If you need more samples, just increase the times of the loop.

Store the detected values to a list in each loop by the **env0_list.append(A0)** function. When the loop ends, the built-in list function **reference[0] = min(env0_list)** in Python will pick out the minimum in the list.

lt_status_now = lf.read_flashlight()

This is to read the status of the module, which will return a 3-element list. This function is used to solve the possible problem caused by brightness-adjustable flashlights. They blink repeatedly due to brightness change by PWM method, so we add this function to the driver library to prevent the car from moving and stopping repeatedly when the light source lights up and goes out quickly or changes luminance by ON/OFF ratio.

fw.turn(turning_angle)

Function for front wheels steering. The main program will call this function if the front wheels are applied for steering. The parameter is the turning angle.

bw.forward()

bw.set_speed(forward_speed)

Here we need two functions for rear wheels. The first function is to control the rotating direction as forward (the function for backwards is **bw.backward()**). The second one is to set the rotating speed of the wheels; the parameter is the speed value (range: 0-100). The bigger the parameter is, the faster the wheel rotates.

Line Following

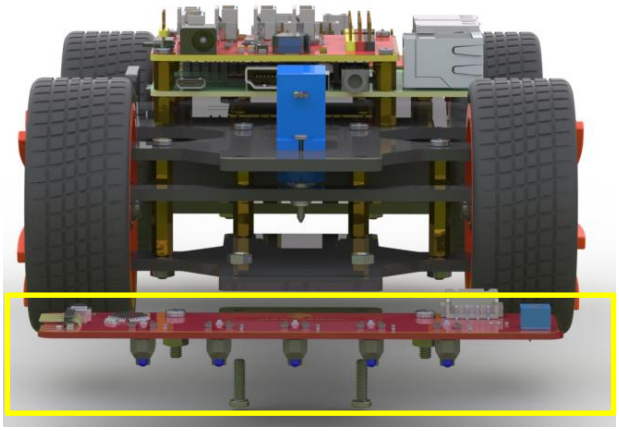
- How it works

The line follower detects lines in the surrounding environment, and transfers the data to the processor. The processor analyzes the data, and sends a command to control the movement of front wheels and rear wheels.

- Procedures

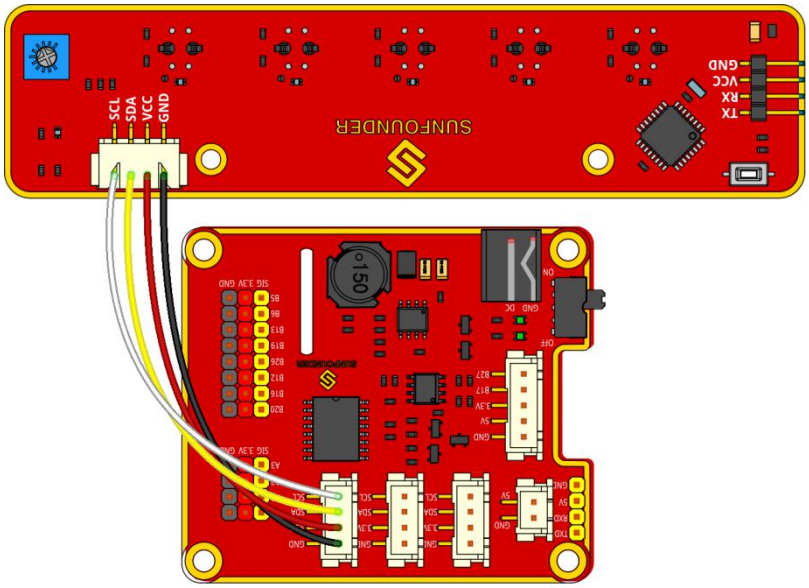
Step 1 Assembly

Connect the light follower to the Sensor Connector with **M3*10 screws** and **M3 nuts**, and then assemble them to the car with two **M3*10 screws** and two **M3 nuts**. You're suggested to hold the nuts underneath with your fingers.



Step 2 Wiring

Connect the line follower module to the Robot HATS with a 4-pin anti-reverse cable as shown below.



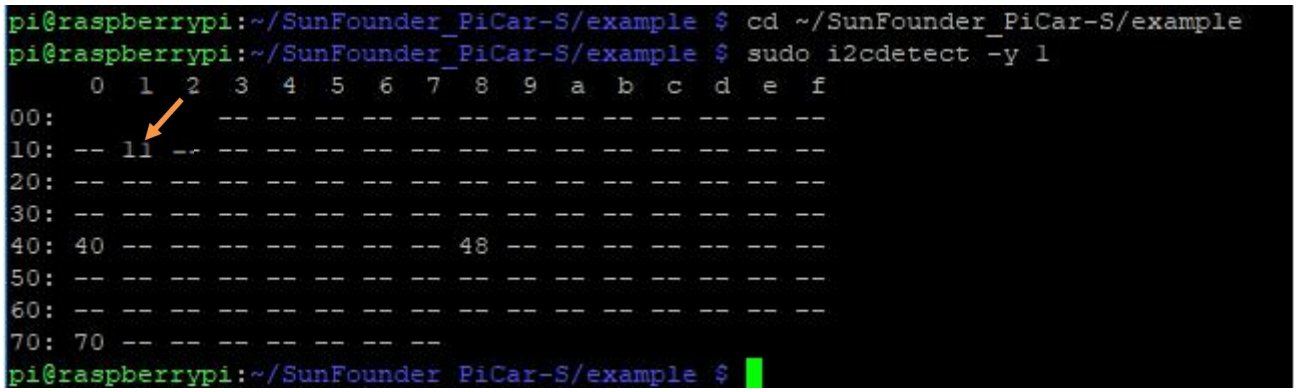
Step 3 Test

Get into the directory example:

```
cd ~/SunFounder_PiCar-S/example
```

Check whether any i2c device is recognized or not via i2c-tools

```
sudo i2cdetect -y 1
```



We can see 11 is the line follower's i2c address. If it is not shown, it proves your wiring is not correct and the i2c communication with Raspberry Pi fails too. You need to check the wiring before the next step.

Run the test code.

```
python3 test_line_module.py
```

Note: For the better working of line following module, we should adjust its sensitivity. The steps are as follows:

Place the module on the white surface, read the value ; place it on balck surface, and read value.

Calculate the difference, rotate potentiometer on the line following module toward the

clockwise and anticlockwise till the diffence reaches up to the maximum. Now the debugging is finished.

Step 4 Starts Running!

Run the line follower code

```
python3 line_follower.py
```

A prompt of calibration will be printed on the screen when the program starts to run. We will calibrate the module on a white surface first: place all the five probes of the line follower above a white board. The prompt of completed calibration will be printed on the screen a few seconds later. Then let's move on to calibration on black line. Also the prompt of starting is printed on the screen, and then place all the probes above the black lines. And the prompt of calibration completed will be printed on the screen a few seconds later.

When the module calibration is all completed, we can run the car then. Place the PiCar-S with probes above the black line on the white board, and then it will go forward following the line itself.

How to make a track for line following

To make a track for the car to follow a black line, you need to prepare the following materials:

A large sheet of paper, a roll of black tape (as black lines), a hard card board (the size depending on the size of the track) or a flat surface like the floor or desk.

1. Spread the paper out smoothly on the hard board, and paste on the board or flat surface.

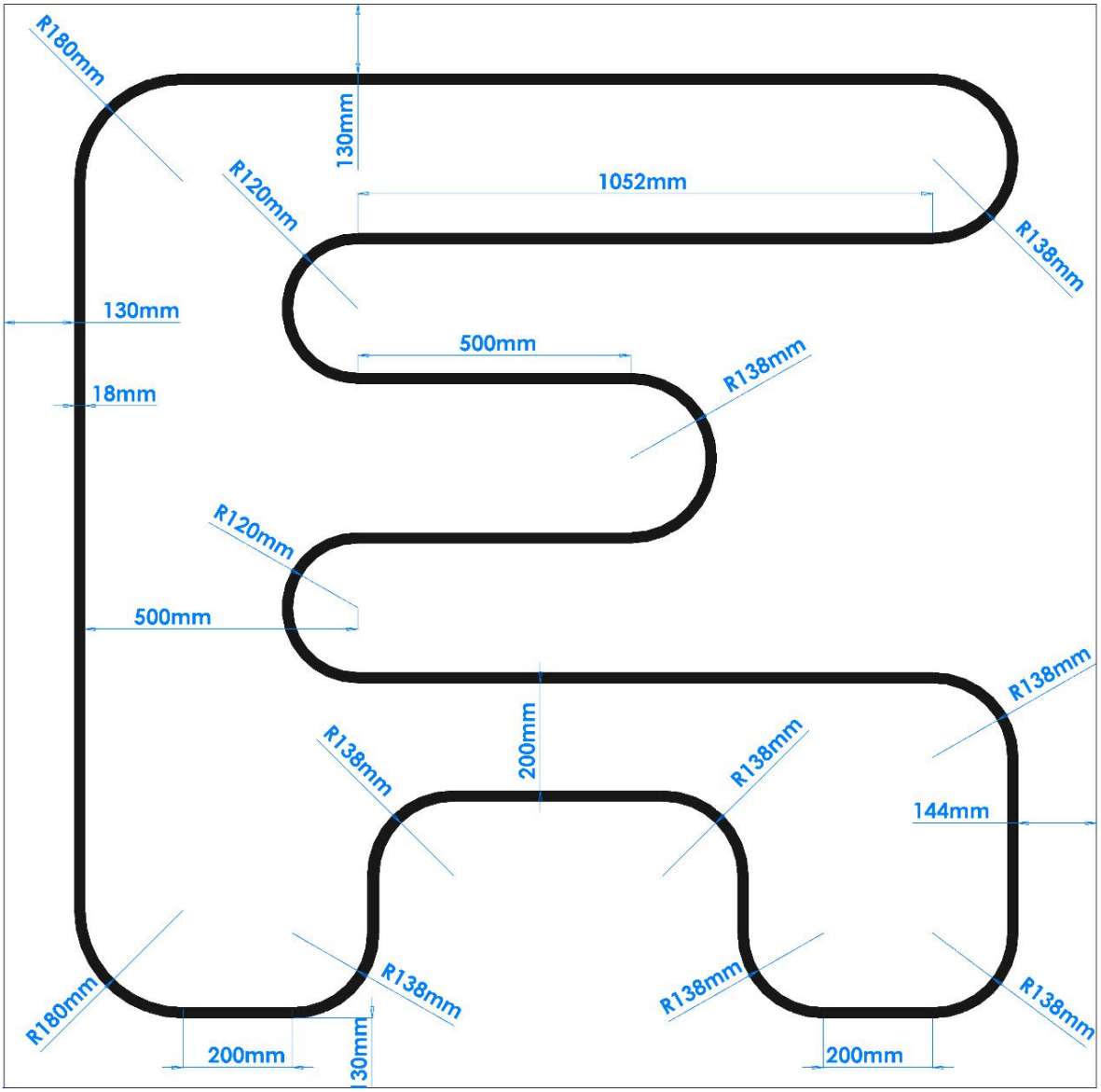
2. Paste the tape on the paper.

Rules for making:

1. Width of the black line: about 18-30mm, nearly the distance between two probes, no more than the minimum distance of two nonadjacent probes
2. The gap between two lines: more than 125mm, which is the width of the whole module, to prevent the car from getting confused when detecting two lines at the same time.
3. The semidiameter of curves: more than 138mm. When the front wheels turn left or

right 45 degrees, the semidiameter of the path by which the car turns is equal to the wheelbase (the distance between the center of the front wheels and rear wheels). The car won't be able to turn and pass the curve smoothly if the semidiameter of the curve is too small.

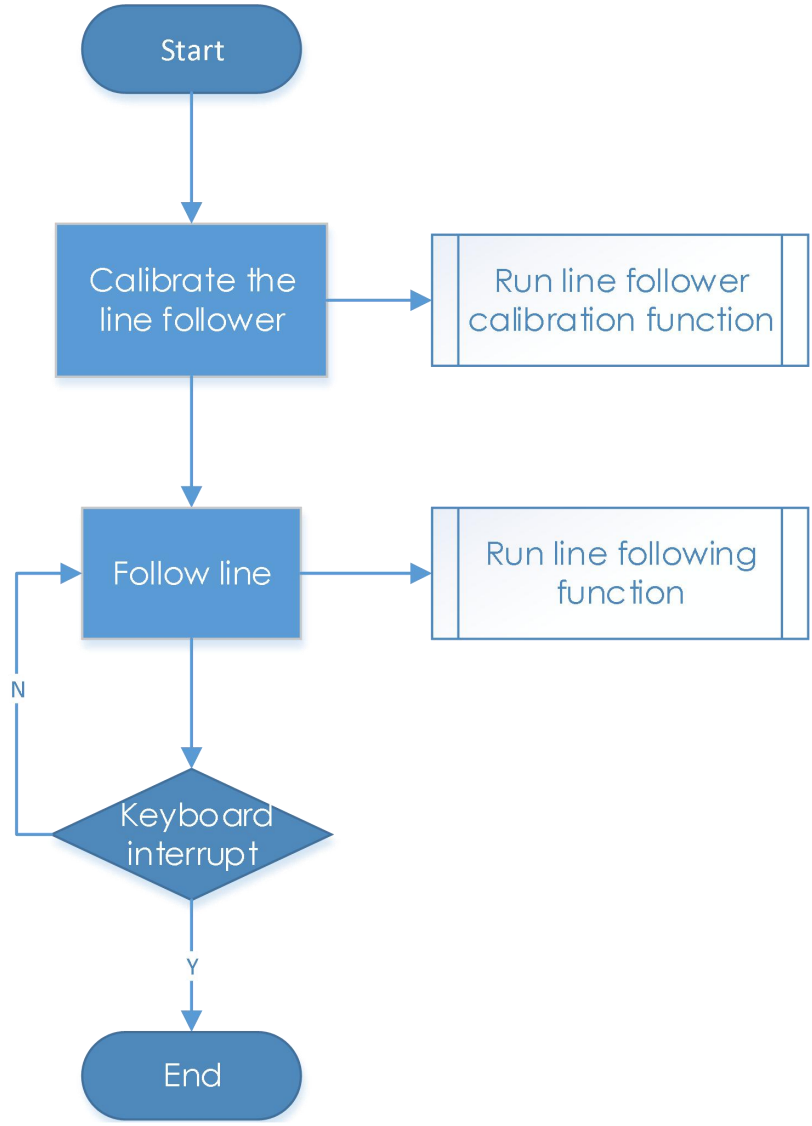
A track sample is shown as below (the original map file can be found under folder **map** in **github**):



- Code Explanation of line_follower.py

Whole Work Flow

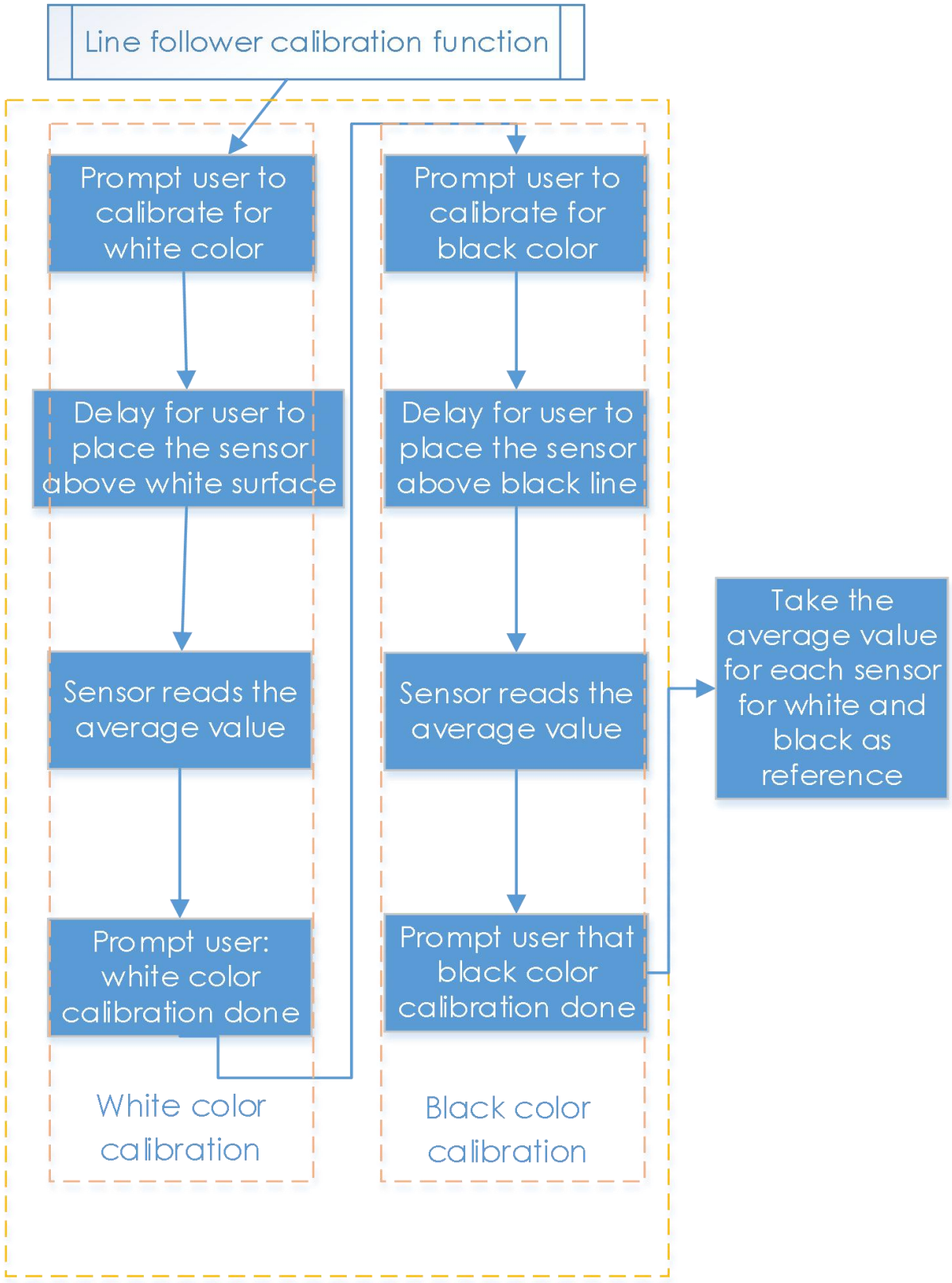
Considering the interference of negative environment factors, we need to calibrate the line follower sensor before actual use.



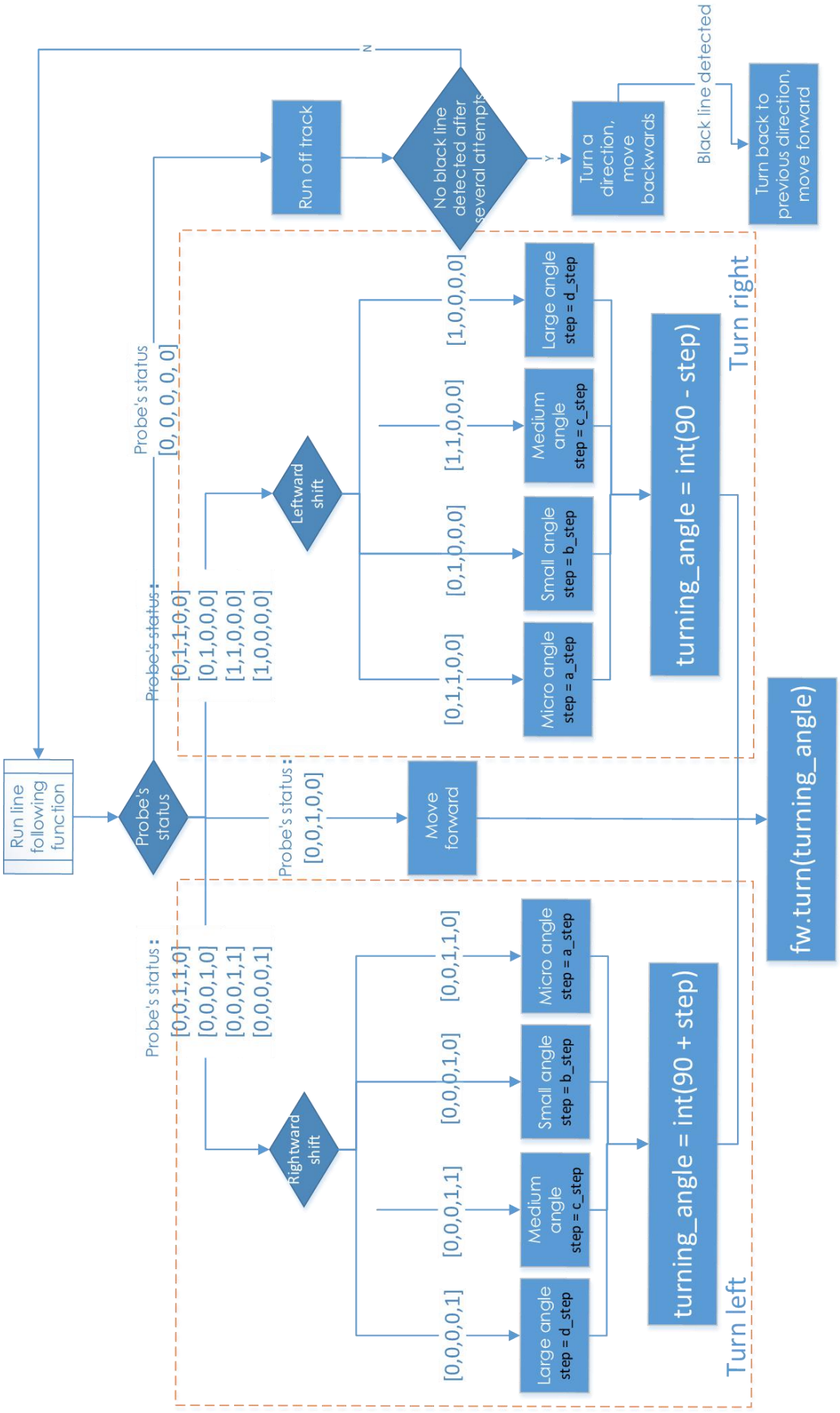
Here two main functions including the line follower calibration and line following are included in the main program.

Subflow of Line Follower Calibration Function

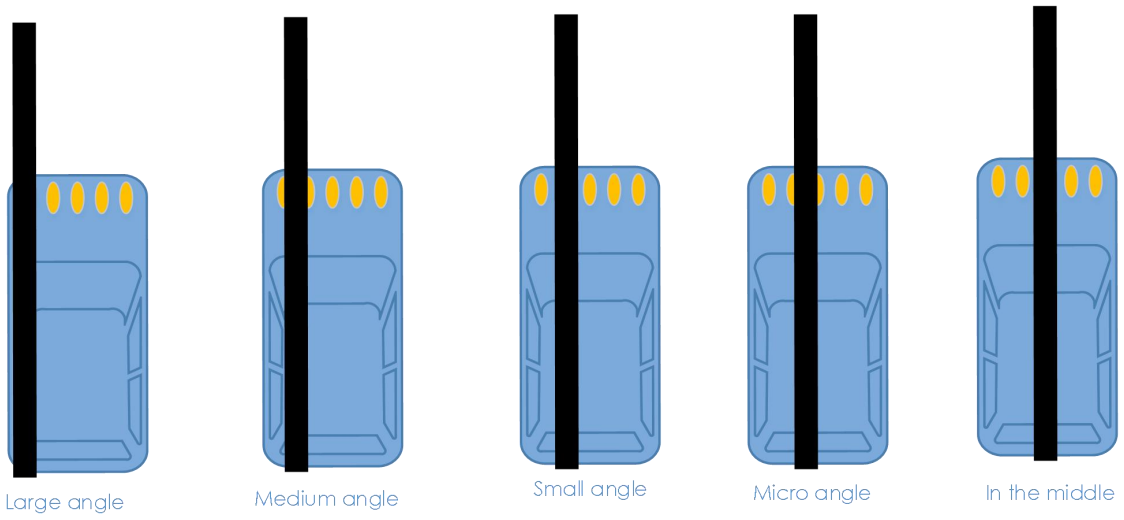
When we run the line follower configuration, we will start from white color, then black color, which is more like the upper limit and lower limit of the sensor. Then we take the average value of black and white as reference value: if the detected value is higher than the reference, it should be white; if the detected is lower than the reference, it should be black. We will show the five detectors' status by 5 elements [0,0,0,0,0].


Line follower calibration function
Prompt user to calibrate for white color
Delay for user to place the sensor above white surface
Sensor reads the average value
Prompt user: white color calibration done
White color calibration
Prompt user to calibrate for black color
Delay for user to place the sensor above black line
Sensor reads the average value
Prompt user that black color calibration done
Black color calibration
Take the average value for each sensor for white and black as reference

Subflow of Line Following Function

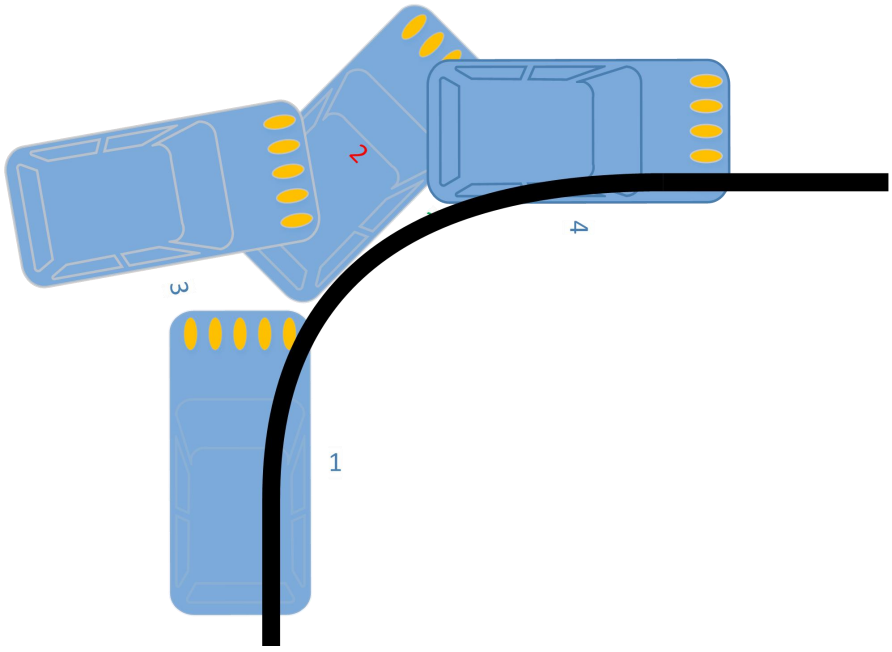


In the line following function, we set the turning angle of the servo in different levels according to the detection results of the probes. If the line in front of the car is detected as a small curve, then the car will turn a small angle; if it is a big one, the car will turn a large angle. Thus, here we set four angle-turning constants: a_step, b_step, c_step, and d_step.



When the car moves forward originally, the servo is in 90 degrees. To drive the car to turn left, the servo should be in 90+step degrees; to turn right, the servo should be in 90-step degrees.

There is a special case: if the car runs off the track, and all the probes cannot detect the black lines any more, then it will continue the program below.



In some case, especially when the car turns in a direction when the semi diameter of the curve is very small (1), the car may run out of the track and cannot detect any black

line (2). If there is no response program in such case, the car will be unable to follow the line again. Thus we set the response program to let the car move backwards in the opposite direction (3), and then turn back to the original direction until a black line is detected again and move forward (4).

Functions Explanation

The logic of the code is just as shown in the flow chart above.

Three Python modules are used in the code, including the imported **SunFounder_Line_Follower**, **front_wheels**, and **back_wheels**. They are the drivers for this kit, respectively for line following , front wheels, and rear wheels

The related classes have been defined here. When the modules are applied to use, objects will be created for related classes, and different parts of hardware will be driven by calling a function by the class object.

Similar to the line following module, we create an object named **lf**:

lf = Line_Follower_module.Line_Follower(references=REFERENCES)

The parameter is initial, and then we can apply the function by calling a class object.

lf.read_digital()

This function is used to read the analog signal of all probes, and convert it into digital signal. If the signal is larger than the reference, the corresponding parameter will be 0; if it is lower than the reference, the parameter will be 1. There are five probes, thus we will get a 5-parameter list.

fw.turn(turning_angle)

The function for front wheels' turning. The main program will call this function if applying the front wheels for turning. The parameter is the turning angle.

bw.forward()

bw.set_speed(forward_speed)

Here we need two functions for rear wheels. One is to control the rotating direction as forward (for rotating backwards, **bw.backward()**). The second one is to set the rotating speed; the parameter is the speed value (range 0~100). The bigger the parameter is, the faster the wheel rotates.

Combination

So, this smart car now is smart in three separate features. But, you think only one sensor module is not enough? Try to combine those sensor modules in one! Here we can show you an experiment - light following with obstacle avoidance for reference.

When the car runs with the light follower, sometimes it may crash into obstacles when following the light, and it's not quite convenient to let the car move back (though we've set the car to move backward if the array is [1,0,1], it's hard to acquire these values since the the car is moving and the light cannot be exactly as required sometimes). So we consider Also, you can let the car move backwards by a paper board or your foot, which is quite easy.

Check below the program of this example.

Assemble the light follower module and ultrasonic obstacle avoidance module on the car first.

Log into the Raspberry Pi on your computer via ssh, and get into the directory

```
cd ~/SunFounder_PiCar-S/example
```

Run the code.

```
python3 light_with_obsavoidance.py
```

- How it works

Set the obstacle avoidance as a superior priority than light following: if there is an obstacle in front of the car, it walk away from the obstacle and back to the track; if not, then the car will keep follow light.

Since the light following and obstacle avoidance of the car depend on the sensor modules, we set two functions to read the status of two sensors separately, and assign values to flags to be returned from the functions: **state_light()**, and **state_sonic()**.

In the function **state_sonic()**, the return value is **avoid_flag**.

If the car is **close to** an obstacle, it will return **avoid_flag = 2**;

if it is **too close to** the obstacle, it will return **avoid_flag = 1**;

if ahead **no obstacle** is detected near, it will return **avoid_flag = 0**.

In the function **state_light()**, the return value is **light_flag**.

If the light spot is **in front of** the car, it will return **light_flag = 0**;

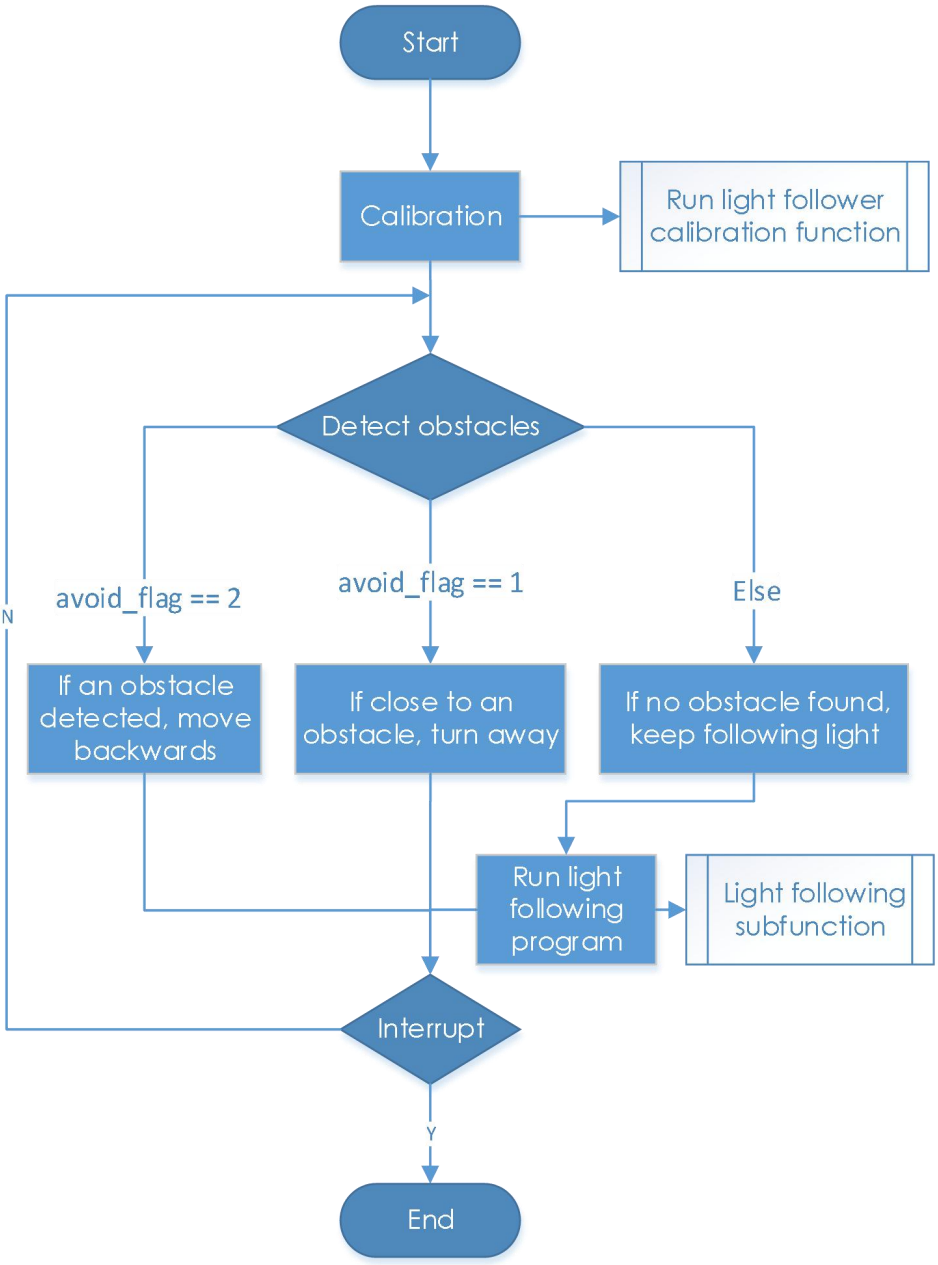
if the spot is **at the right side**, it will return **light_flag = 1**;

if the spot is **at the left side**, it will return **light_flag = 2**;

if the spot is **at the back**, it will return **light_flag = 3**;

if **no light spot** is detected, it will return **light_flag = 4**.

The main program **main()** will run the corresponding program according to **avoid_flag** and **light_flag**, and the **avoid_flag** is superior in priority.



Appendix 1: Installing Manually

1. Update the apt list.

```
sudo apt-get update
```

2. Install python-smbus.

```
sudo apt-get install python-smbus -y
```

3. Install the PiCar module.

```
cd ~
git clone --recursive https://github.com/sunfounder/SunFounder_PiCar.git
cd SunFounder_PiCar
python3 setup.py install
```

4. Enable I2C.

Edit the file /boot/config.txt

```
sudo nano /boot/config.txt
```

The "#" in front of each line is to comment the following contents which does not take effect in a sketch. The I2C configuration part is commented by default too. Add the following code at the end of the file, or delete the pound mark "#" at the beginning of related line; either way will do.

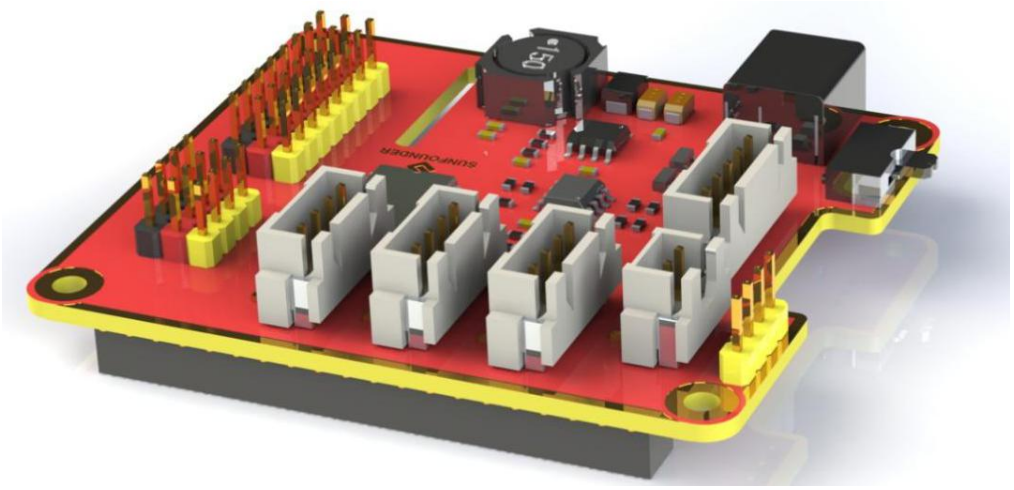
```
dtparam=i2c_arm=on
```

5. Reboot.

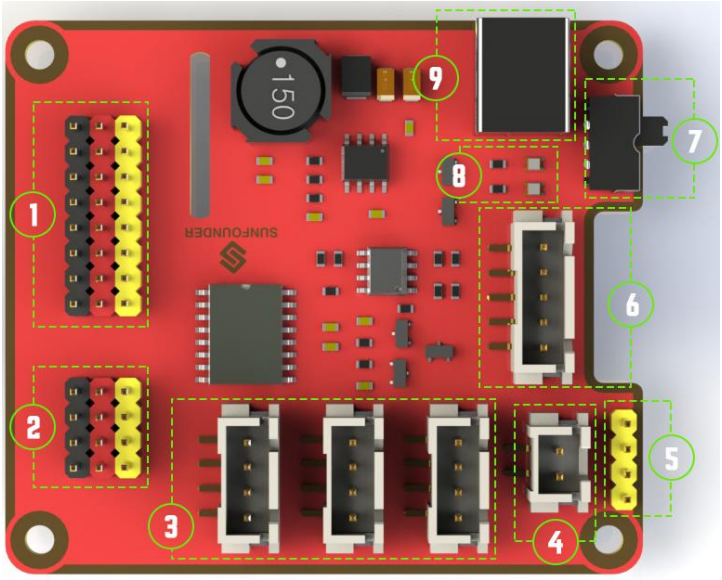
```
sudo reboot
```

Appendix 2: Modules

Robot HATS



Robot HATS is a specially-designed HAT for a 40-pin Raspberry Pi and can work with Raspberry Pi 3 model B, 3 model B +, and 4 model B. It supplies power to the Raspberry Pi from the GPIO ports. Thanks to the design of the ideal diode based on the rules of HATS, it can supply the Raspberry Pi via both the USB cable and the DC port thus protecting it from damaging the TF card caused by batteries running out of power. The PCF8591 is used as the ADC chip, with I2C communication, and the address 0x48.



1. **Digital ports**: 3-wire digital sensor ports, signal voltage: 3.3V, VCC voltage: 3.3V.

2. **Analog ports**: 3-wire 4-channel 8-bit ADC sensor port, reference voltage: 3.3V, VCC voltage: 3.3V.

3. **I2C ports**: 3.3V I2C bus ports

4. **5V power output**: 5V power output to PWM driver.

5. **UART port**: 4-wire UART port, 5V VCC, perfectly working with SunFounder FTDI Serial to USB.

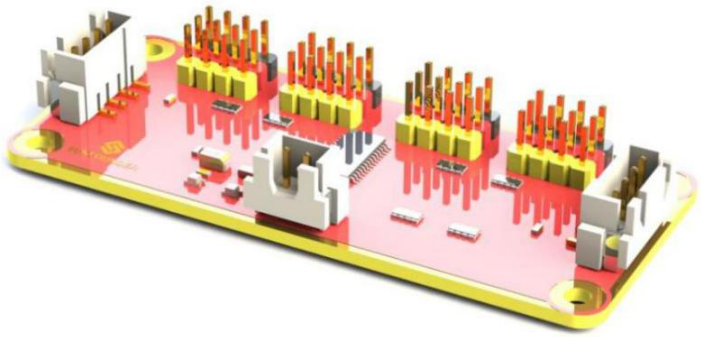
6. **Motor control ports**: 5V for motors, direction control of motors MA and MB and a floating pin NC; working with motor driver module.

7. **Switch**: power switch

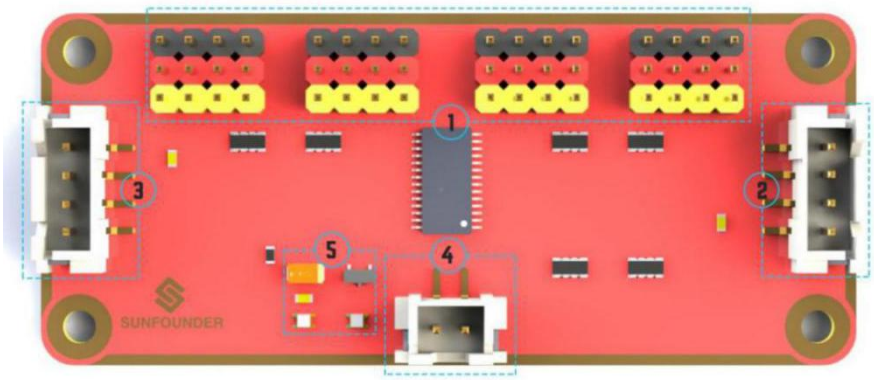
8. **Power indicators**: indicating the voltage – 2 indicators on: >7.9V; 1 indicator on: 7.9V~7.4V; no indicator on: <7.4V. To protect the batteries, you're recommended to take them out for charge when there is no indicator on. The power indicators depend on the voltage measured by the simple comparator circuit; the detected voltage may be lower than normal depending on loads, so it is just for reference.

9. **Power port**: 5.5/2.1mm standard DC port, input voltage: 8.4~7.4V (limited operating voltage: 12V~6V).

PCA9865



PCA9685 16-channel 12-bit I2C Bus PWM driver. It supports independent PWM output power and is easy to use 4-wire I2C port for connection in parallel, distinguished 3-color ports for PWM output.



1. **PWM output ports**: 3-color ports, independent power PWM output port, connect to the servo directly.

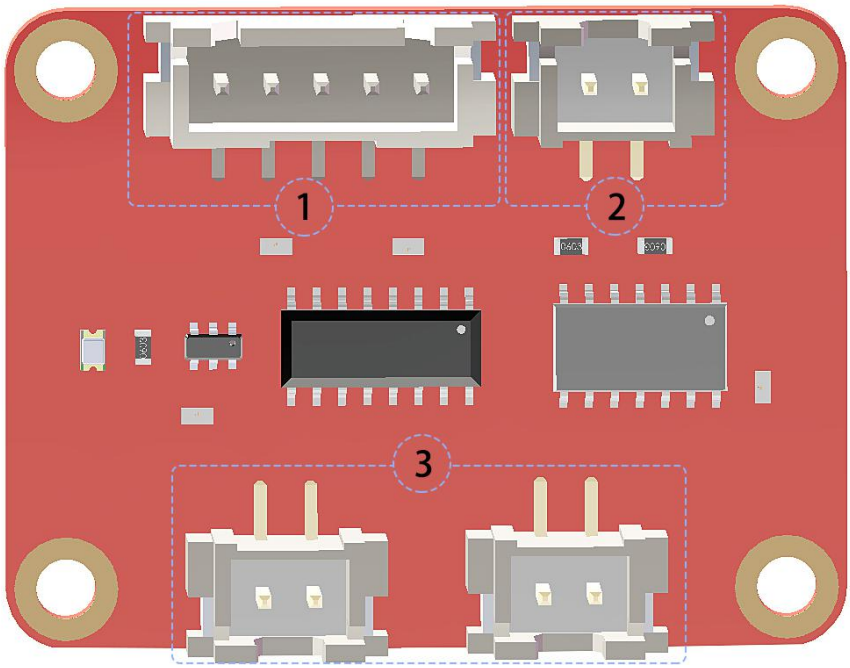
2 & 3. **I2C port**: 4-wire I2C port, can be used in parallel. Compatible with 3.3V/5.5V

4. **PWM power input**: 12V max.

5. **LED**: power indicator for the chip and for the PWM power input.

Motor Driver Module

The Motor Driver module is a low heat generation one and small packaged motor drive.

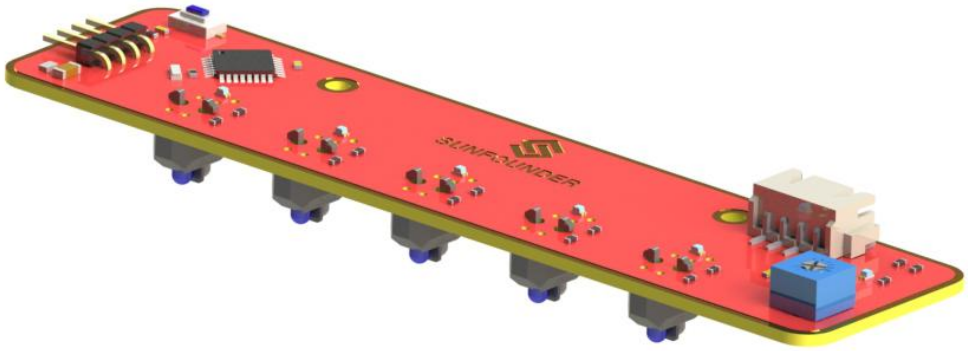


1. Power and motor control port: includes pins for supplying the chip and the motors and controlling the motors' direction

2. PWM input for the motors: PWM signal input for adjusting the speed of the two motors

3. Motor output port: output port for two motors

Line Follower Module

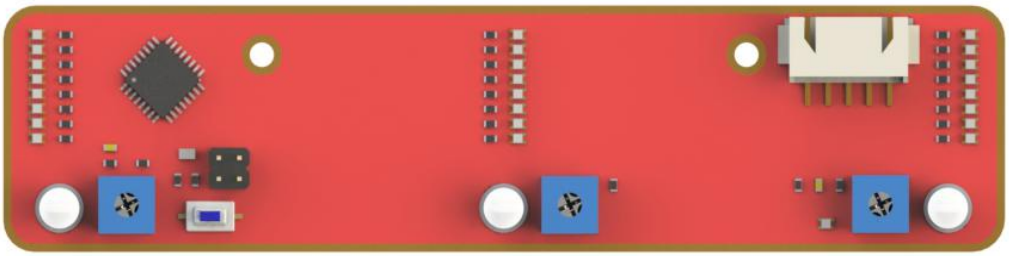


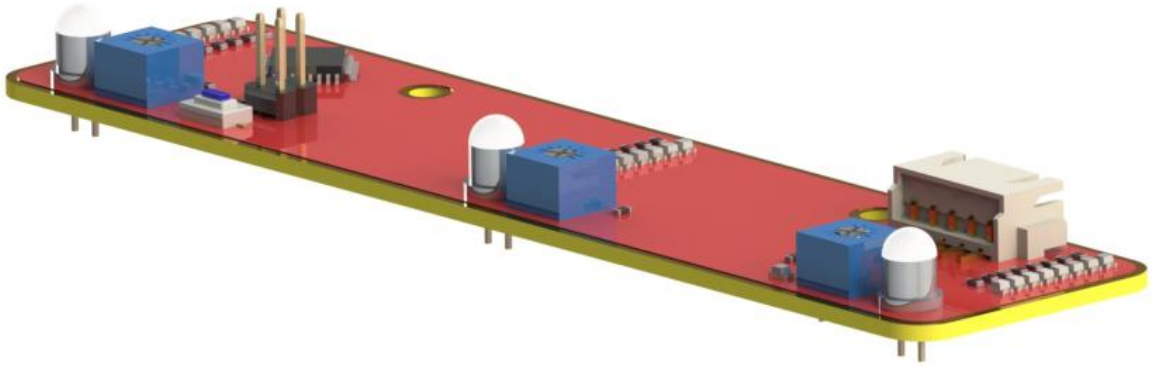
The TCRT5000 infrared photoelectric switch adopts a high transmit power infrared photodiode and a highly sensitive phototransistor. It works by applying the principle of objects' reflecting IR light – the light is emitted, then reflected, and sensed by the synchronous circuit. Then it determines whether there exists an object or not by the light intensity. It can easily identify black and white lines.

In other words, the different conduction levels of the phototransistor when it passes over black and white lines can generate different output voltages. Therefore, all we need to do is to collect data by the AD converter on the Atmega328 and then send the data to the master control board via I2C communication.

This module is an infrared tracking sensor one that uses 5 TRT5000 sensors. The blue LED of TRT5000 is the emission tube and after electrified it emits infrared light invisible to human eye. The black part of the sensor is for receiving; the resistance of the resistor inside changes with the infrared light received.

Light Follower Module





Phototransistor, also known as photodiode, is a device that converts light to current. Currents are generated when photons are absorbed in the P-N junction. When a reverse voltage is applied, the reverse current in the device will change with the light luminance. The stronger the light is, the larger the reverse current will be. Most phototransistors work this way.

The ADC chip on the HATS can receive 8-bit analog signals and convert them into integers, and transfer the signals to the Raspberry Pi. The Raspberry Pi will analyze the data to determine the direction of the brightest area (the light source), and further control the steering and movement of the four wheels to approach the light source.

You may need a light focused flashlight in this experiment. At least, the spot size of the torch should not be too big to reach all the 3 phototransistors on the module at the same time. Well, you can also shine the flashlight closer to the car to get a small spot size.

Ultrasonic Obstacle Avoidance Module

This module contains an ultrasonic sensor to detect the distance to an obstacle in front. It is usually used on robots to avoid obstacles. With the two holes, it can be assembled easily on the robot. There is power indicator light added onside to indicate power on/off. The 3-pin design makes it unique among most of the ultrasonic module in the market: the same pin to trigger and receive signals. The 3-pin anti-reverse cable included makes the wiring tighter and more convenient. Also the beautiful cartoon design and the red PCB add to its fascination.

The 25kHz ultrasonic module differs from common 40kHz ones. It is not so precise in distance measurement, but it has a wider detecting range of about 30 degrees. Thus it can detect the obstacle near ahead, and measure the distance if it's assembled on the PiCar. Though a 40kHz ultrasonic measures distance precisely, it can only detect obstacles which is right in front. Therefore, the 25kHz ultrasonic module is more preferably applicable to an obstacle avoidance car, or projects require a large detection range instead of accurate distance measurement.

Principle

Supply a short 10μS pulse to the SIG to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 25 kHz and raise its echo back to SIG. The echo is a distance object that is pulse width and the range in proportion. You can calculate the $Range$ through the $Time\ Interval$ between sending trigger signal and receiving echo signal.

Formula:

$$Range(m) = \frac{Time\ Interval \times 340_{m/s}}{2}$$

Or:

$$Range(cm) = \frac{Time\ Interval}{58}$$

Or:

$$Range(inchs) = \frac{Time\ Interval}{148}$$

We suggest to use over 60ms measurement cycle, so as to prevent trigger signal to the echo.

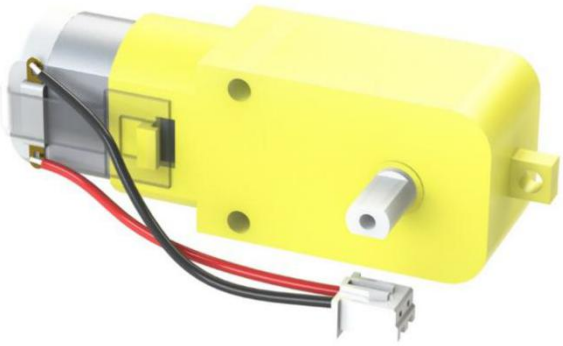
SunFounder SF006C Servo

The SunFounder SF0180 Servo is a 180-degree three-wire digital servo. It utilizes PWM signal of 60Hz and has no physical limit – only control by internal software to 180 degrees at most.

Electrical Specifications:

Item	V = 4.8V	V = 6.0V
Consumption Current* (No Load)	≦50mA	≦60mA
Stall Current	≦550mA	≦650mA
Rated Torque	≥0.6 kgf·cm	≥0.7 kgf·cm
Max. Torque	≥1.4 kgf.cm	≥1.6 kgf.cm
No Load Speed	≦0.14sec/60°	≦0.12sec/60°

DC Gear Motor



It's a DC motor with a speed reducing gear train. See the parameters below:

Motor	Model	F130SA-11200-38V
	Rated Voltage	4.5V-6V
	No-load Current	≤80mA
	No-load Speed	10000±10%
Gear Reducer	Gear Ratio	1:48
	Speed (no-load)	≈200rpm (≈180rmp in test)
	Current	≤120mA

Copyright Notice