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 model B+, 2 model B and 3 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.
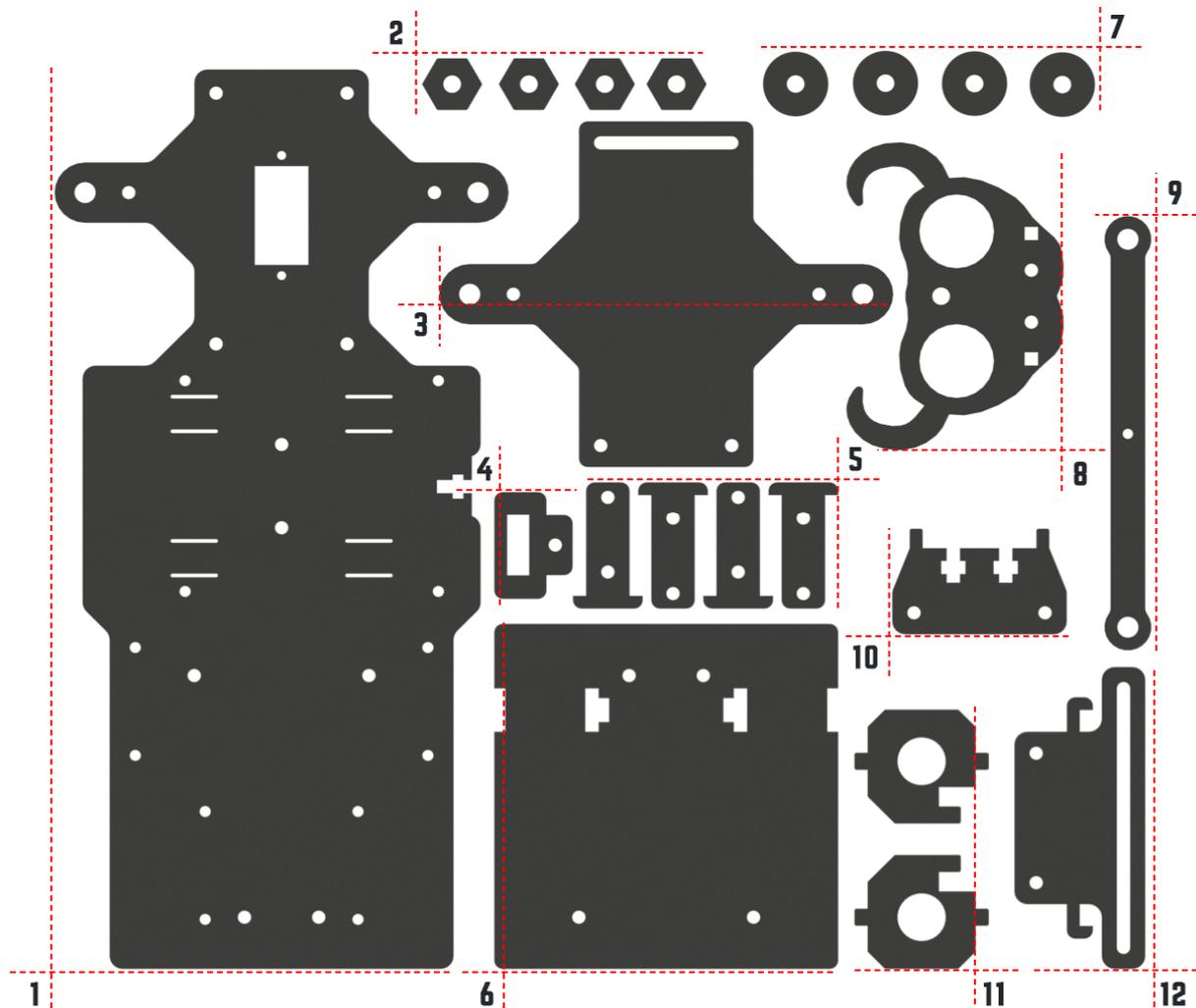
Contents

Components List

Acrylic Plates

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

1

Servo Accessories

The following three parts will be used in the servo package:
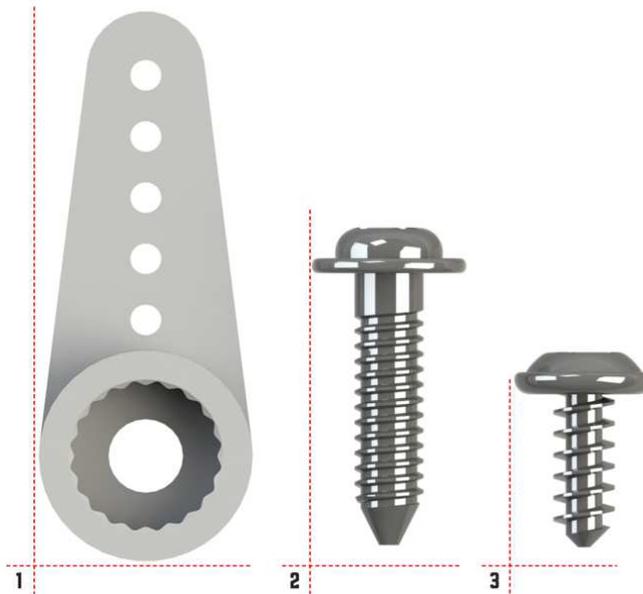
1. Rocker Arm
2. Rocker Arm Screw
3. Rocker Arm Fixing Screw

Mechanical Fasteners

Name	Component	Qty.
M2x8 Screw		2
M2.5x6 Screw		12
M3x8 Screw		8
M3x8 Countersunk Screw		2
M3x10 Screw		9
M3x30 Screw		4

2

M4x25 Screw		2
M2 Nut		2
M2.5 Nut		12
M3 Nut		23
M4 Self-locking Nut		2
M2.5x8 Copper Standoff		16
M3x25 Copper Standoff		8
4x11x4 F694ZZ Flange Bearing		2

Wires

100mm HX2.54 5-Pin Jumper Wire		1
50mm HX-2.54 4-Pin Jumper Wire		1
50mm HX-2.54 2-Pin Jumper Wire		1
100mm HX-2.54 2-Pin Jumper Wire		1
200mm HX2.54 5-Pin Jumper Wire		1
200mm HX-2.54 4-Pin Jumper Wire		1

3

200mm HX2.54 3-Pin Jumper Wire		1

PCB
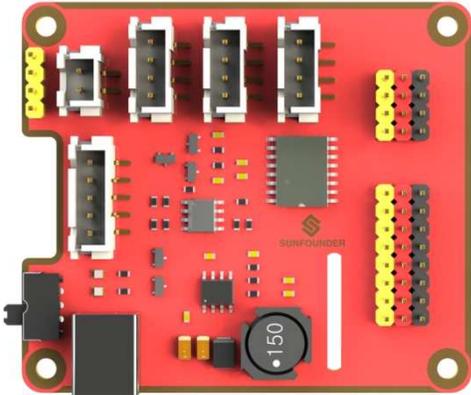
Robot HATS		1
PCA9685 PWM Driver		1
TB6612 Motor Driver		1
5-CH Line Follower Module		1
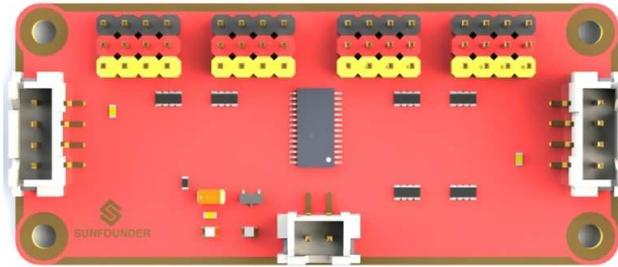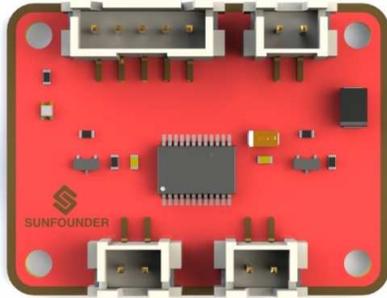
Ultrasonic Obstacle Avoidance Module		1
Light Follower Module		1
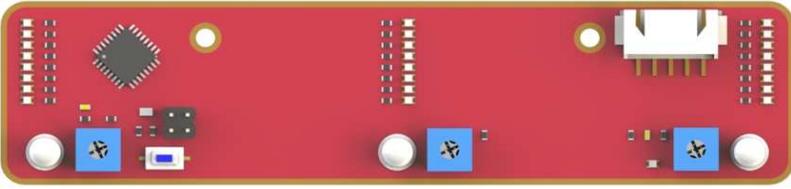
Other Components

2x18650 Battery Holder		1
DC Gear Motor		2
SunFounder SF0180 Servo		1
Rear Wheel		2
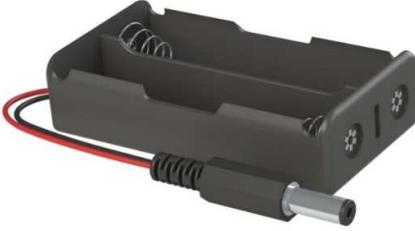
5

Front Wheel		2
USB Wi-Fi Adapter		1
Ribbon (30cm)		1

Tools

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

6

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!
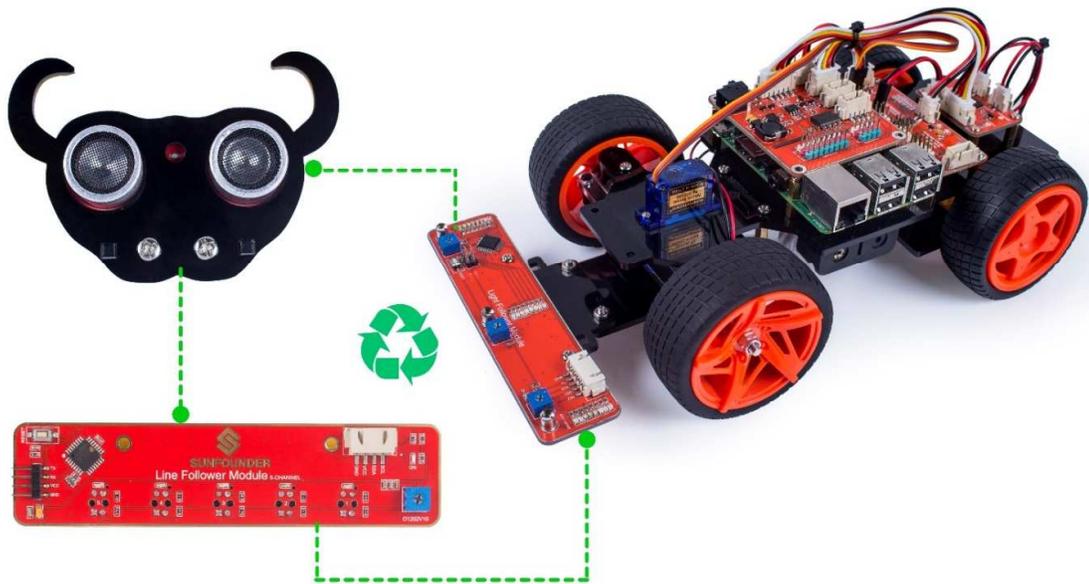
Building the Car

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!

Fixing Rear Wheels

Assemble the **Motor Support Plate** into the **Back Half Chassis** as shown below.

7
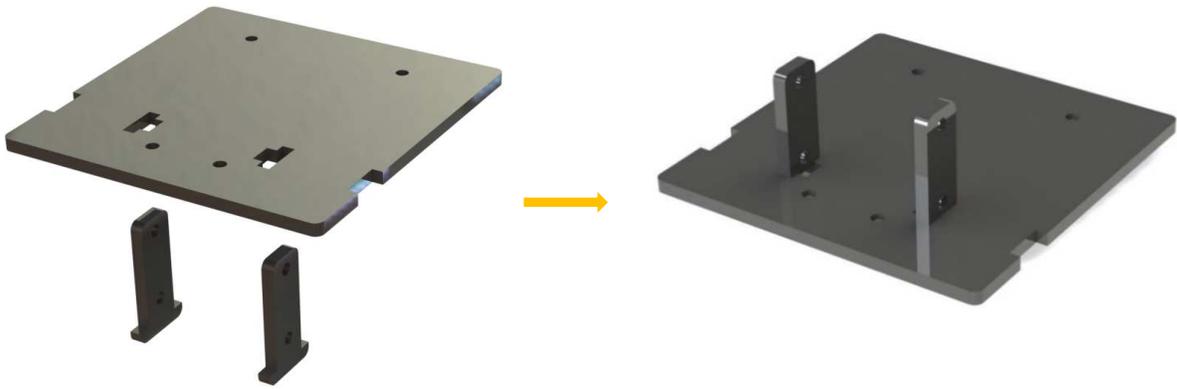
Assemble the two motors with four **M3x30 screws** and **M3 nuts**. Pay attention to place the motors with wires inward, providing convenience for connecting the circuit.

Place wires
inwards

8
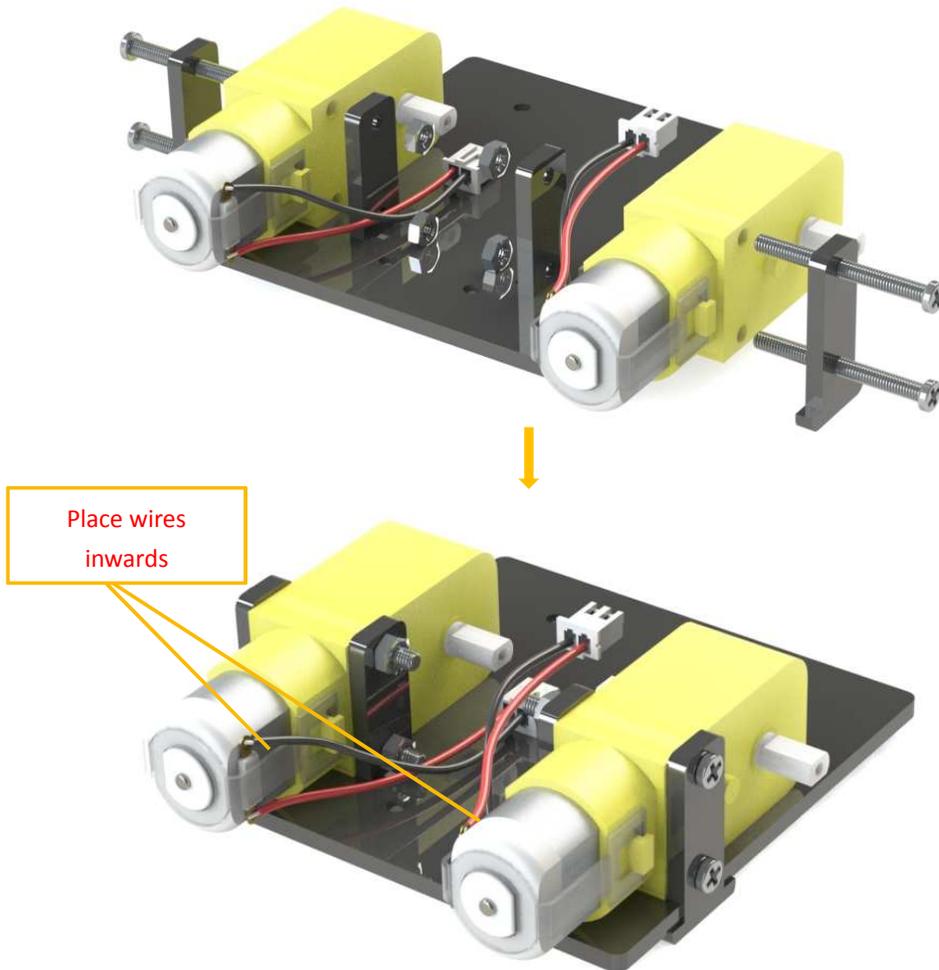
Insert four **M3x25 copper standoffs** through the acrylic plate into four **M3 nuts** as shown below:
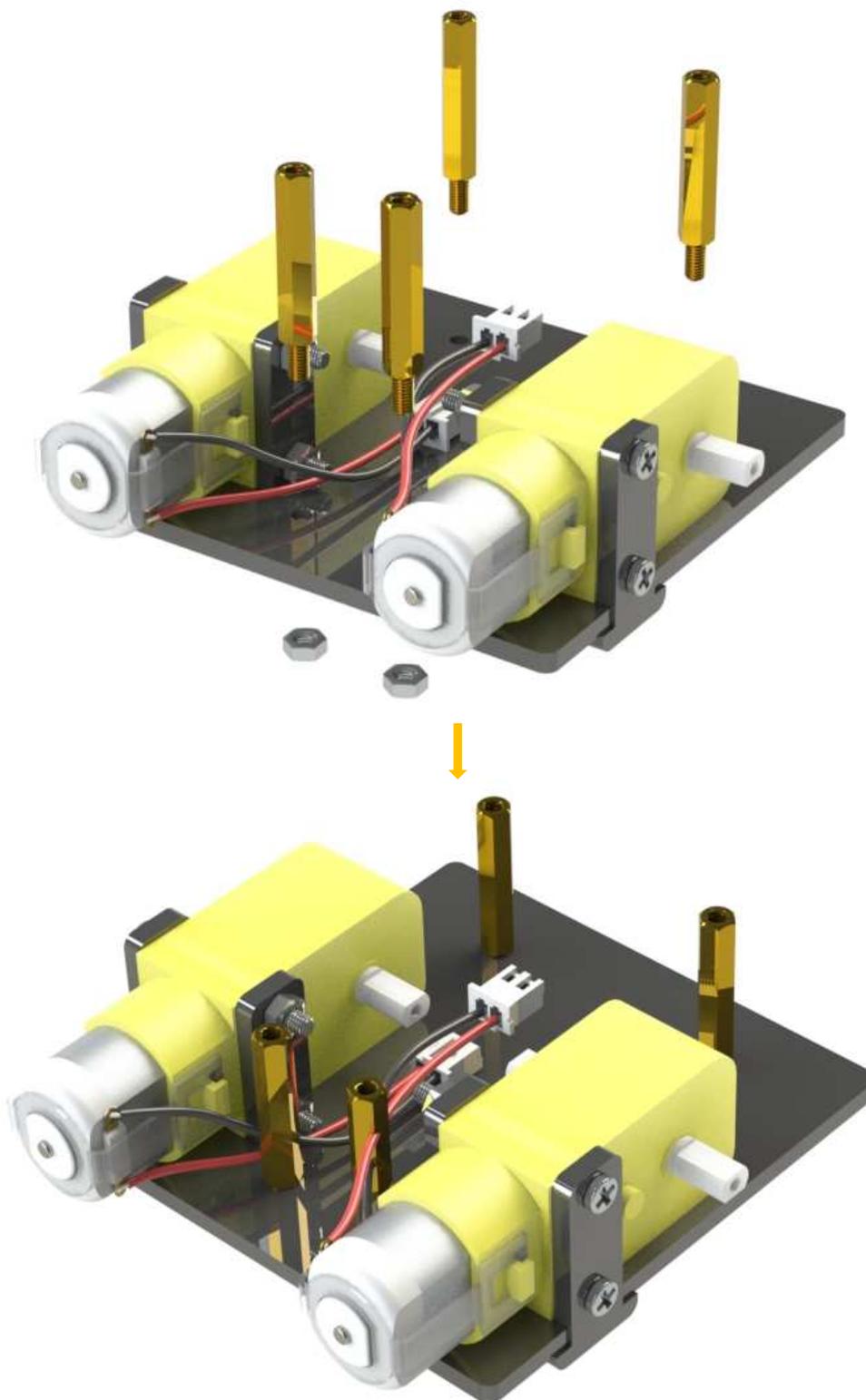
So now this part is completed. You can put it aside for now.

Upper Plate

Mount the **M2.5x8 copper standoffs** and **M2.5 nuts** into the **upper plate** first. There are three PCBs to be installed onto the plate and four copper standoffs are needed for each. So here 12 holes should be used, marked with cross as shown below:
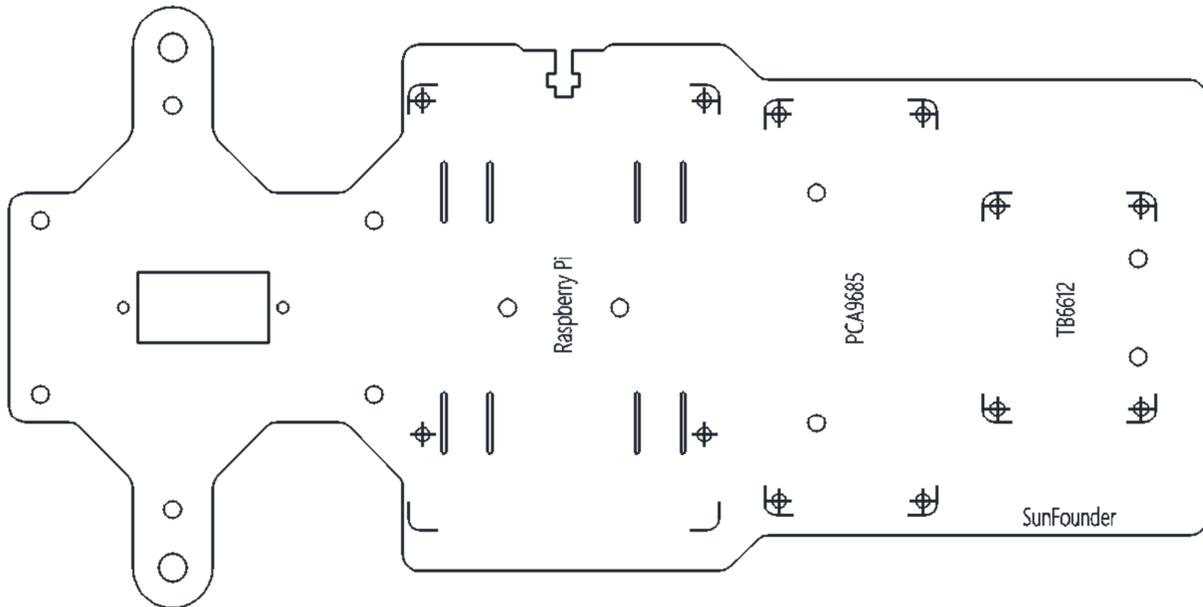
Assemble the **M2.5x8 copper standoffs** and **M2.5 nuts** as shown below. Pay attention that the side the logo carved should face up.
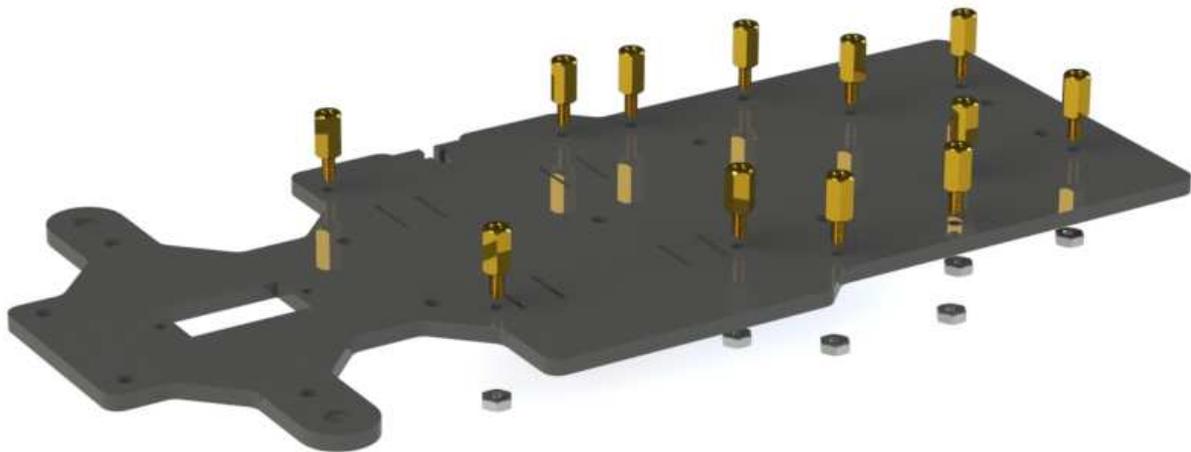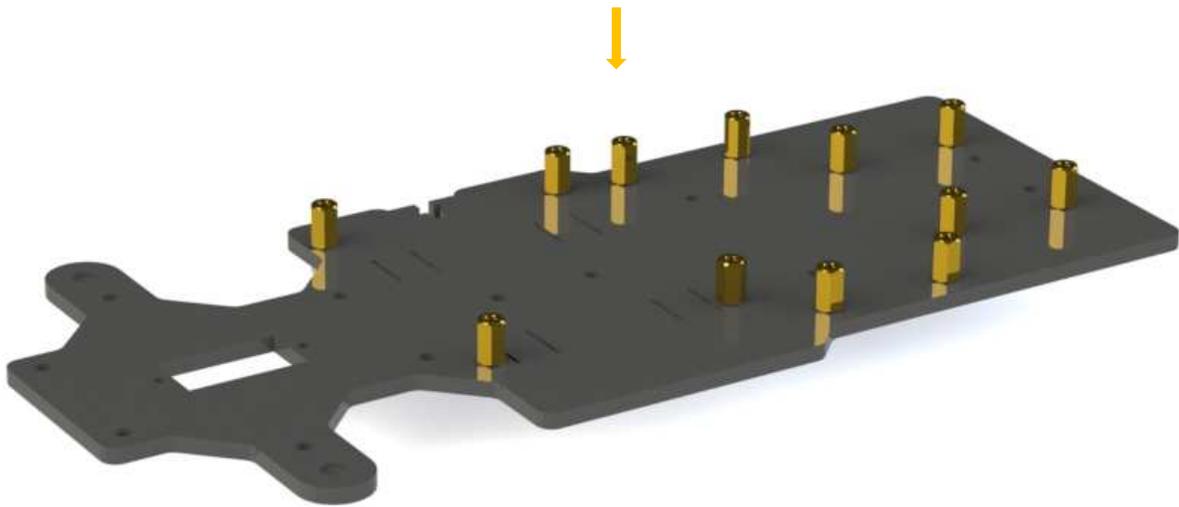
Battery Holder

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.

11
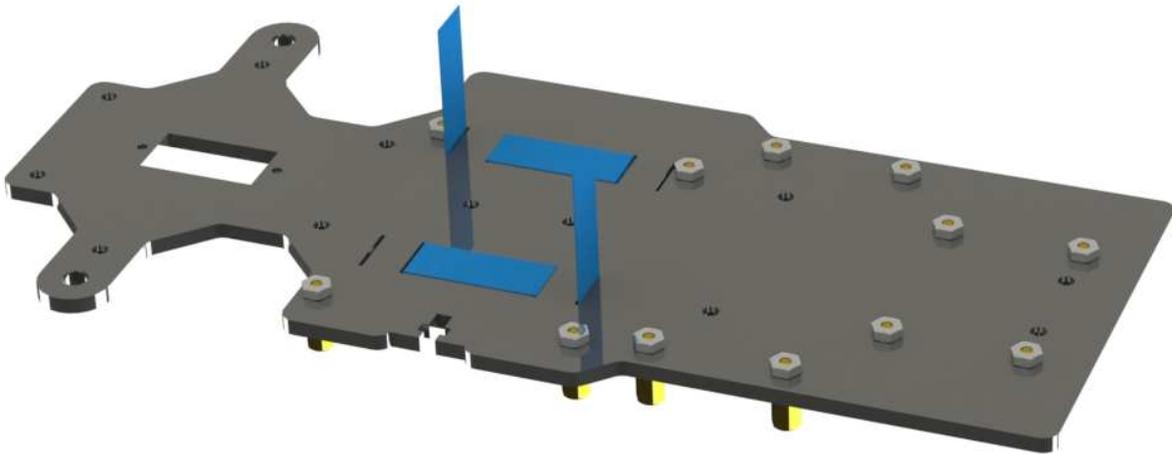
Fasten the battery holder with two **M3x8 countersunk screws** and two **M3 nuts**: pay attention to the direction of battery holder's wire.

12
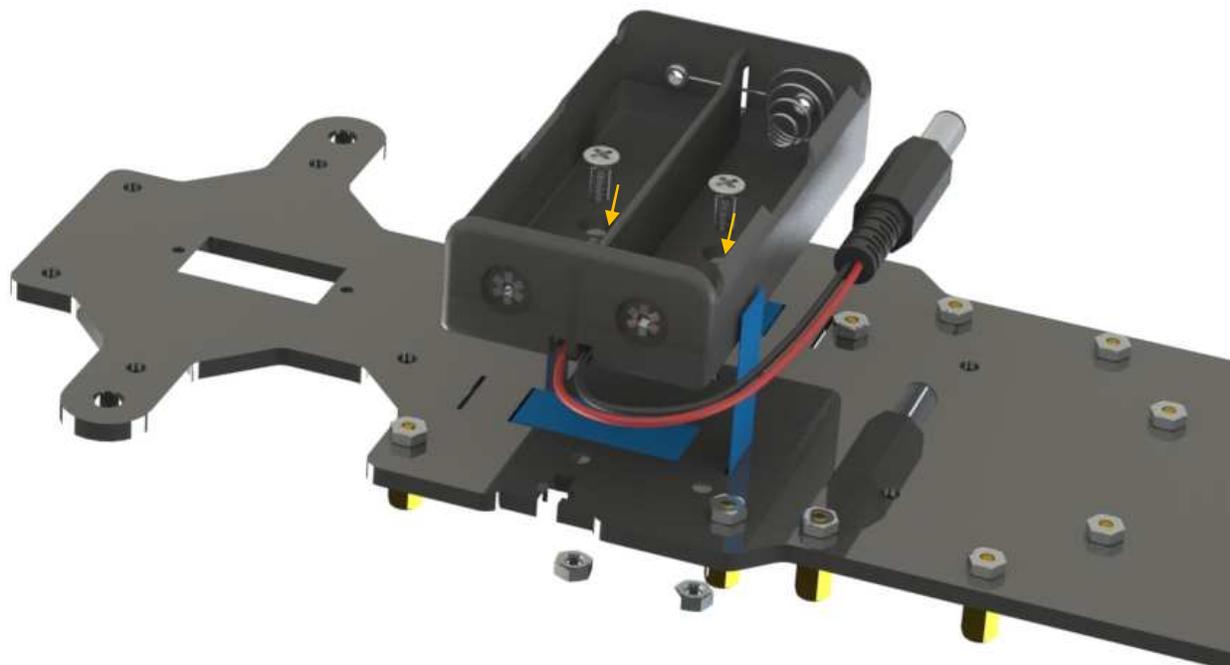
Rear Wheels (Driving)

Mount the assembled rear wheel driving part onto the Upper Plate with four **M3x8 screws**:

13
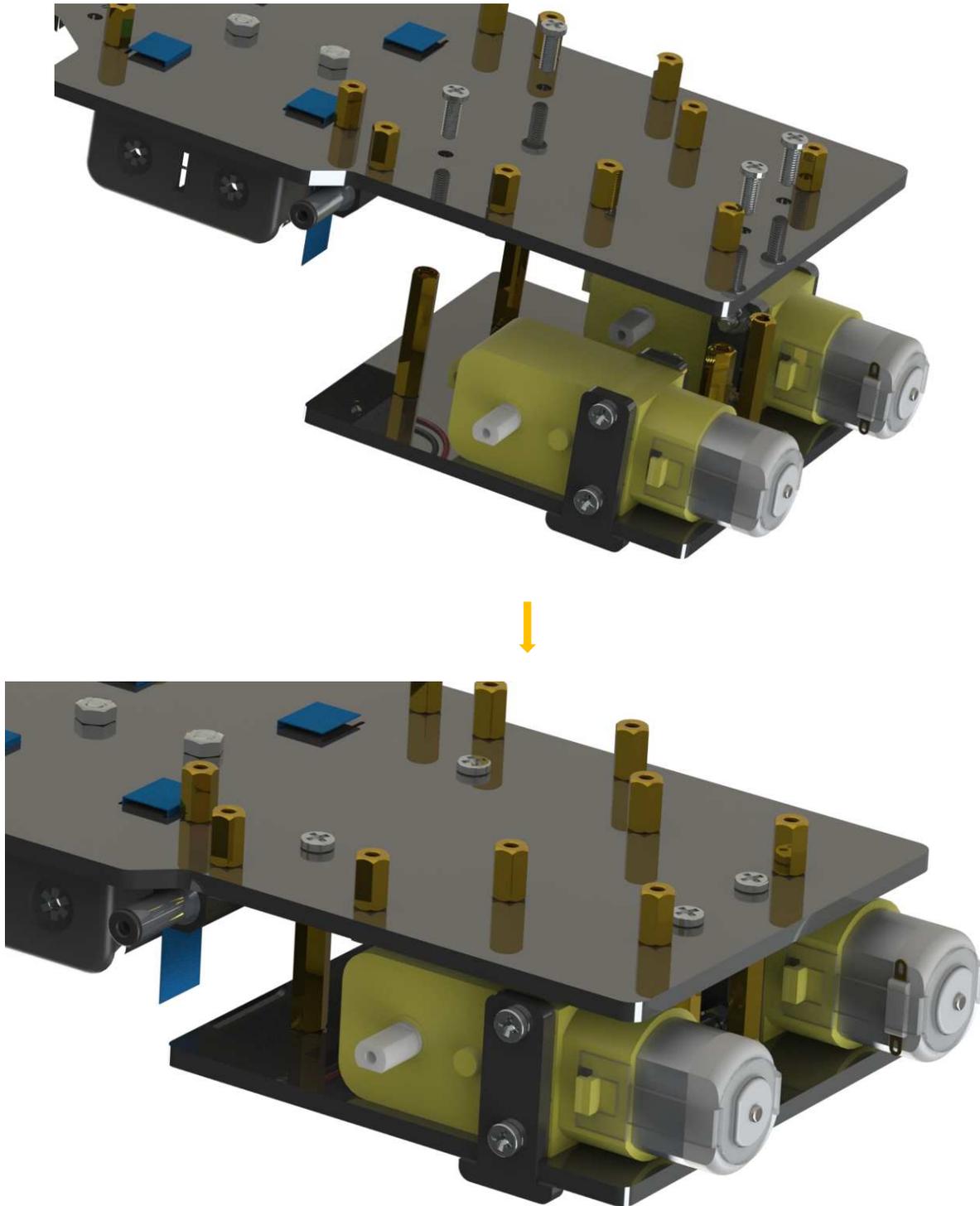
Assemble the **rear wheels**:

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

14
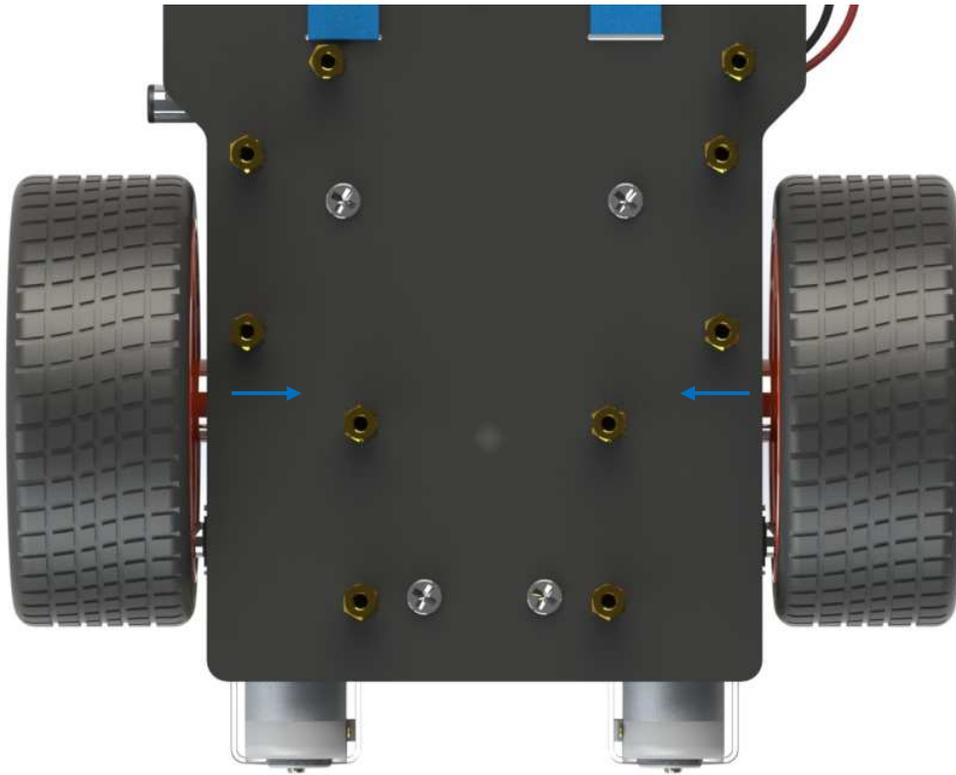
TF Card Guard

Mount the **TF Card Guard** plate onto the side near the front wheel with an **M3x10 screw** and an **M3 nut**. First place the nut into the slot from the underneath, and then insert the screw into the nut through the plate.
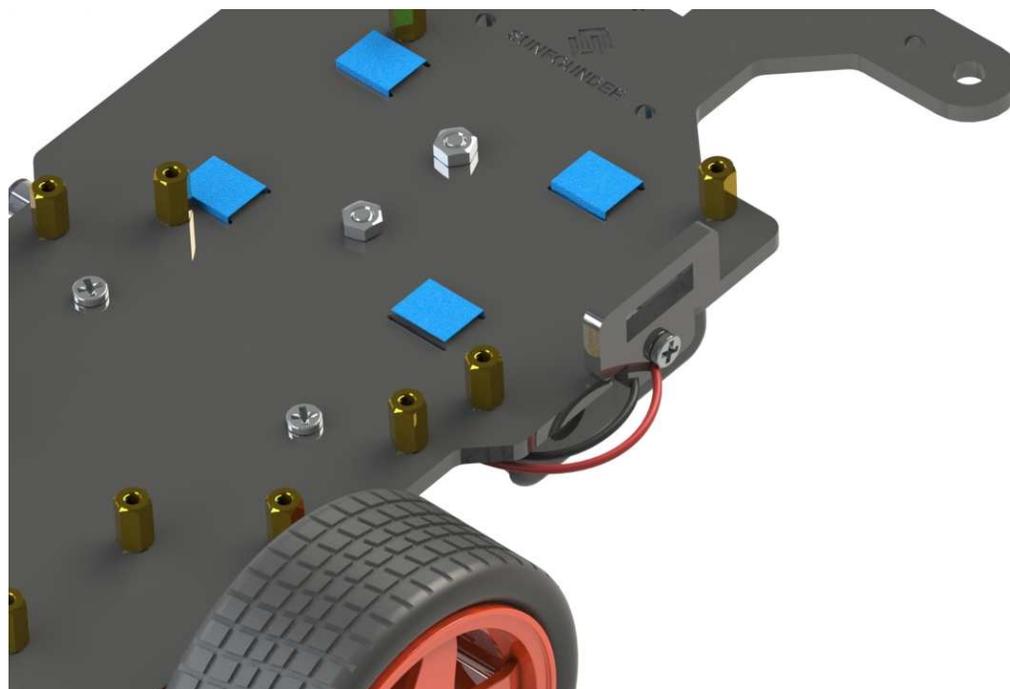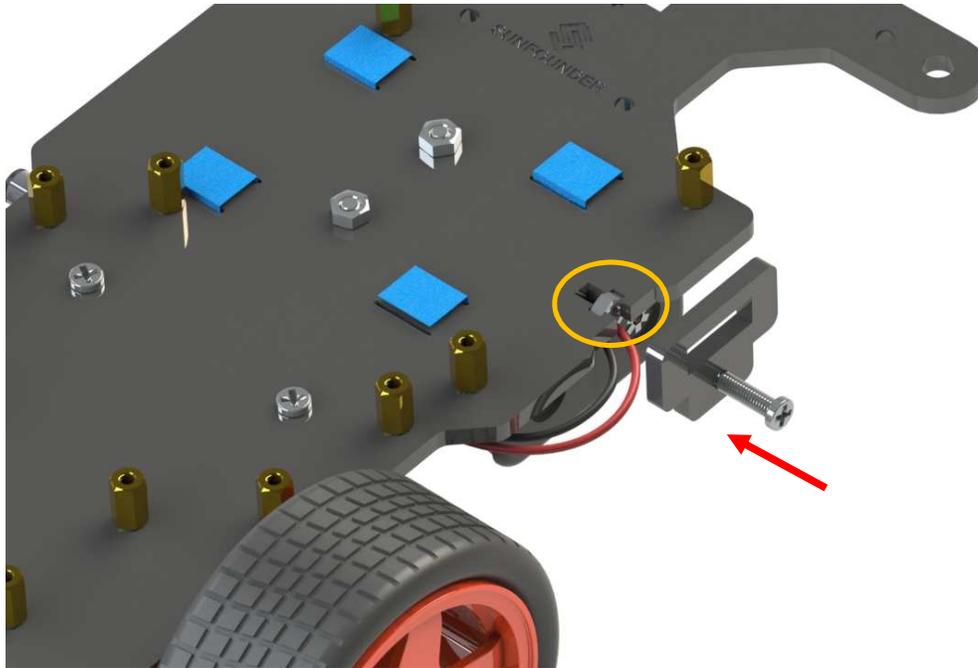
Front Half Chassis

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

16
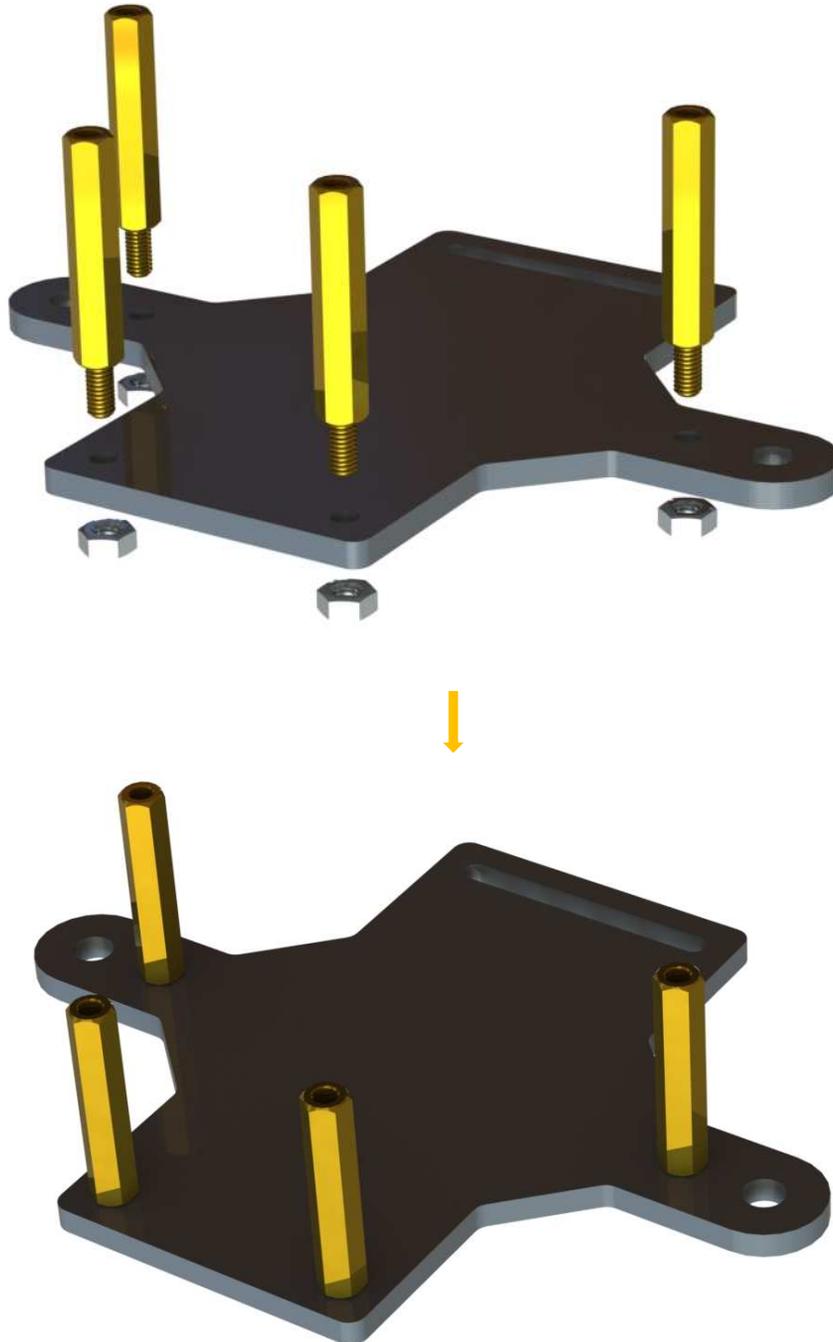
Front Wheels

Insert an **M4x25 screw** through a **Flange Bearing** (pay attention to the direction – the flange near the cap of the screw), a **Steering Connector**, 2 **Bearing Shields**, 2 **Hex Front Wheel Fixing Plates**, and a **front wheel**, into an **M4 Self-locking Nut** (note the direction) as shown below:

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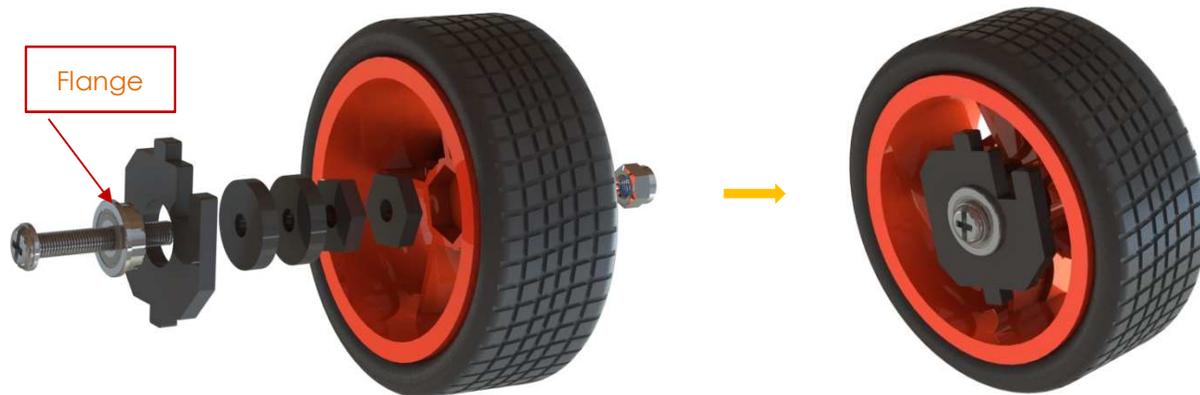
The Self-locking Nut should be screwed tight enough. It would be better to tighten the screw until the wheel and Steering Connector cannot move first, then loosen the screw a little, so that the Steering Plate can just move. Thus, the wheel can turn flexibly when the connection would not be too loose.

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.

17

Steering Part

Take out the **Rocker Arm** and the **Rocker Arm Screw** (the longer one):
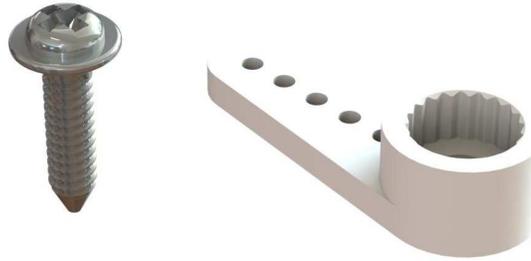
Connect the **Steering Linkage** and the **rocker arm** with the screw. **Note:** Insert it into the first hole of the arm (as indicated by the arrow below) which is the farthest from the gears. Since the screw is larger than the hole, you should try to screw it hardly so as tight to the arm. Don't worry of the arm which is soft.
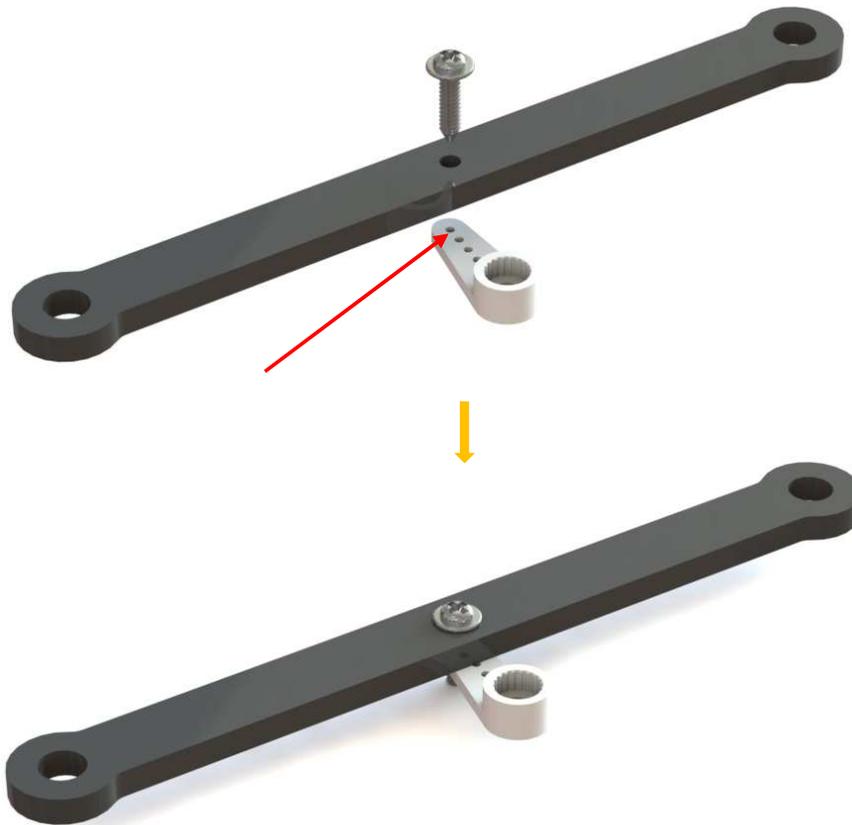
And also fasten them as tightly as possible, and then loosen the screw a little so the Steering Linkage can move flexibly.

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):
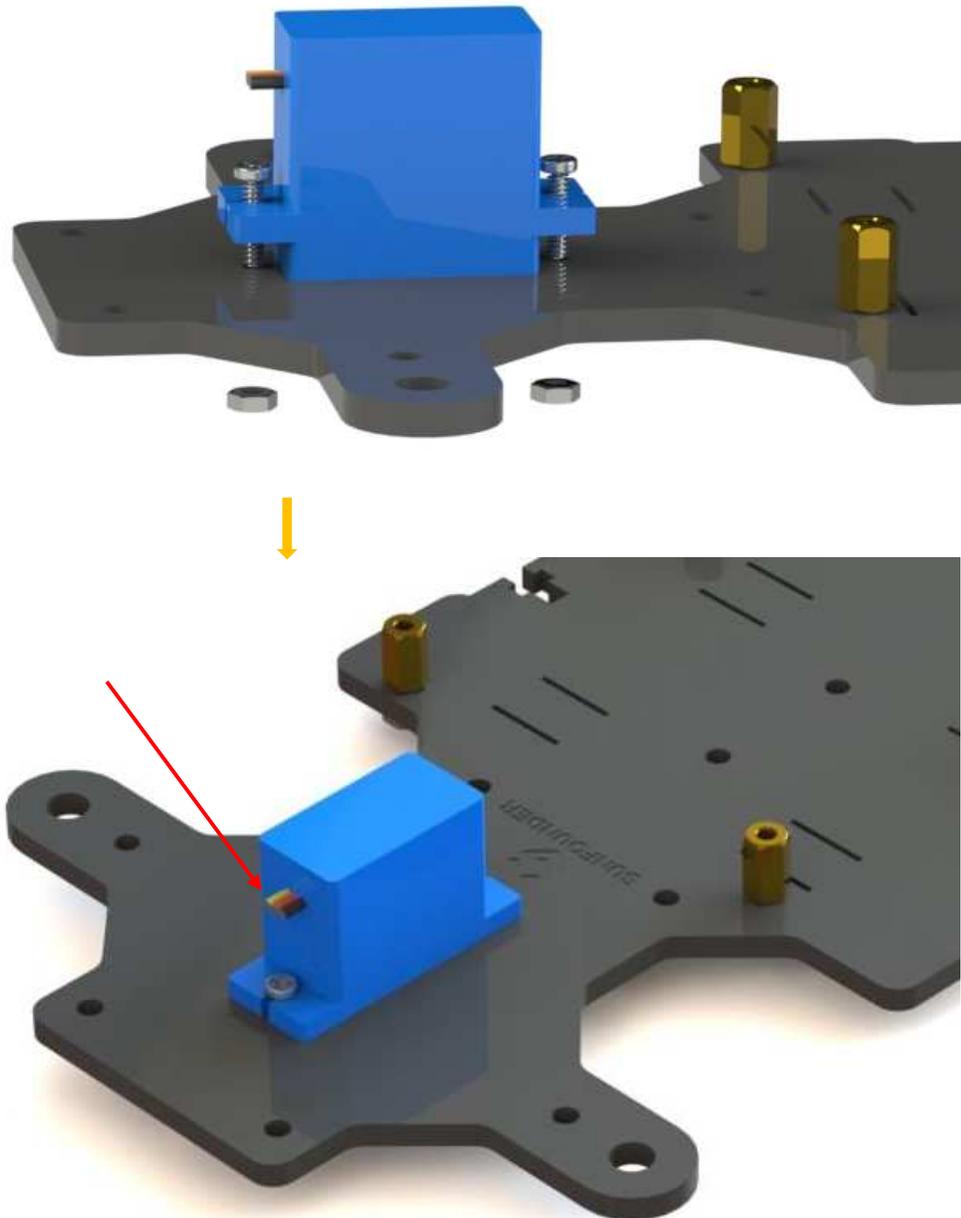
Till here the car assembly is mostly done. But **don't hurry** to assemble the front wheels, because the servo has not been adjusted yet. An unadjusted servo may get stuck and further get hot and burnt when it is connected to power to use as it is. To adjust, you can use your own way, like uploading an Arduino program to the servo which makes it rotate to the 90 degrees and stop, or using the servo adjustment code provided in this kit.

For the latter method, since we need to connect the related driver board and Raspberry Pi to adjust the servo, now let's first assemble the **PCBs** onto the car, which is much easier than previous assembly.

19

PCB Assembly

Assemble the **Raspberry Pi** (TF Card inserted) with four **M2.5x8 copper standoffs**, then plug the **Robot HATS** onto it, and fix the **PCA9685 PWM Driver**, the **TB6612 Motor Driver** and the **Robot HATS** with twelve **M2.5x6 screws**.
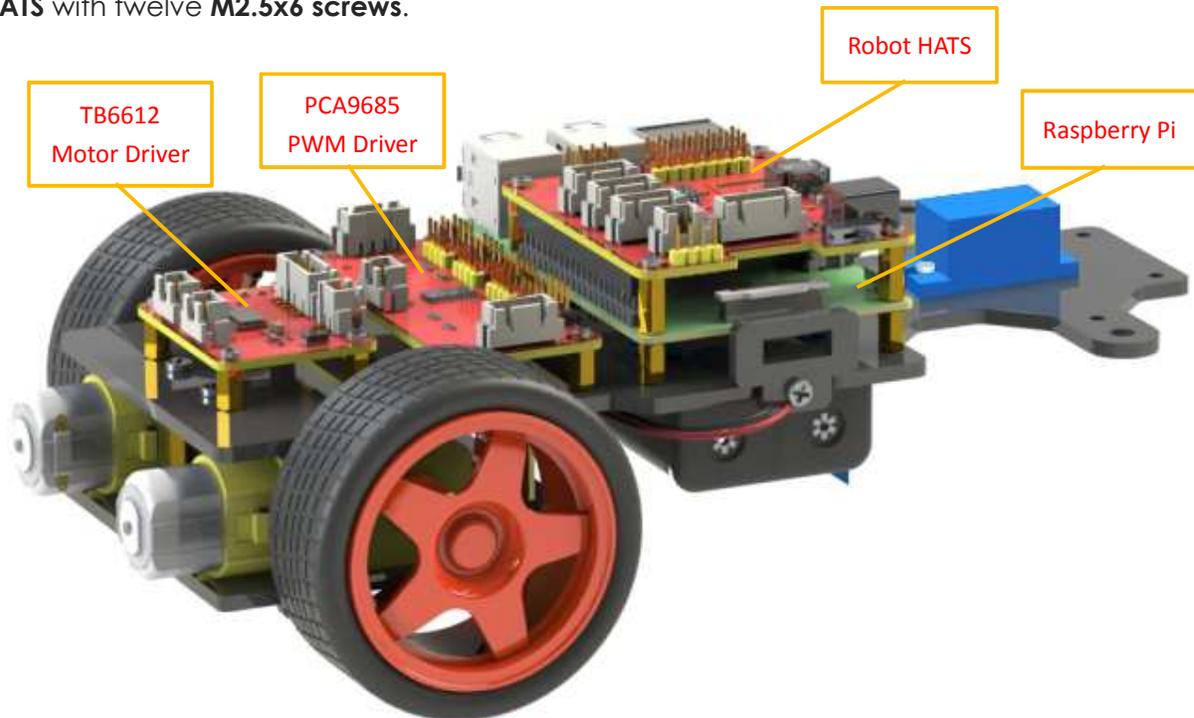
Circuits Building
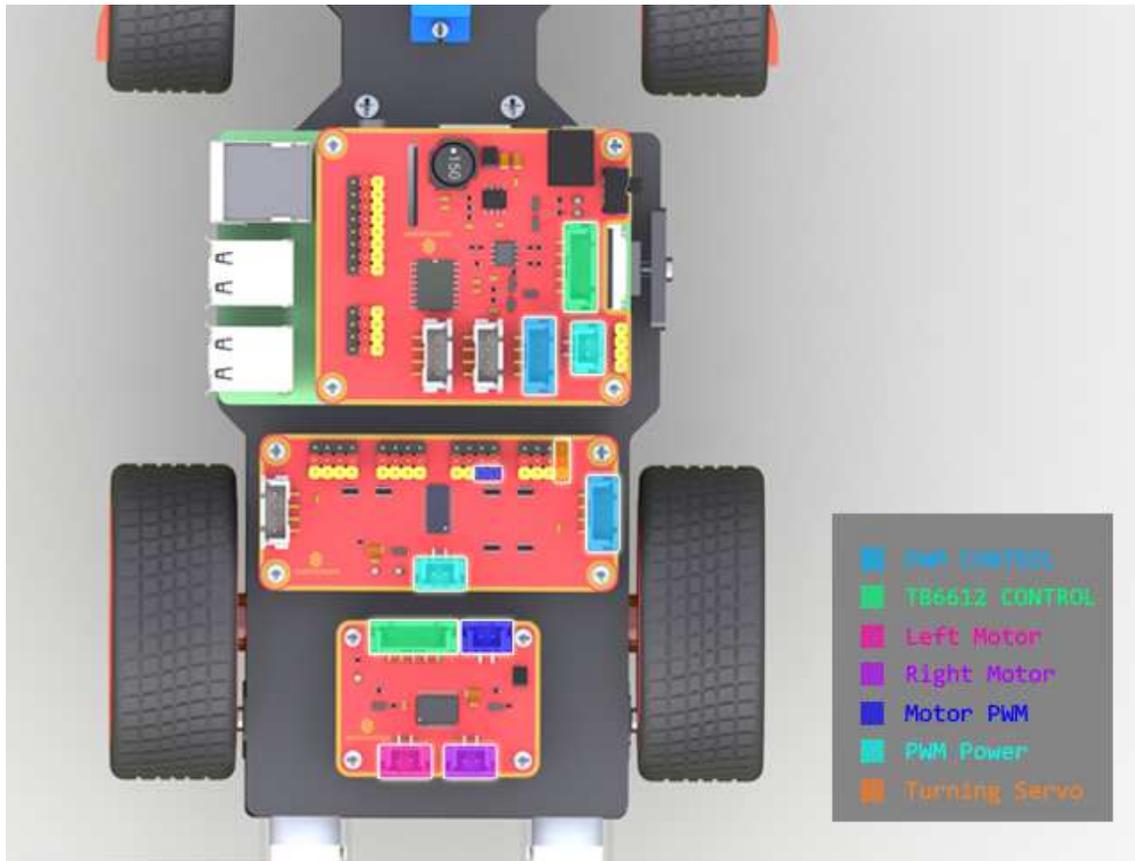
Connect the **PWM CONTROL** of the Robot HATS with the PCA9685 PWM Driver, and the **TB6612 CONTROL** with the TB6612 Motor Driver.

Then the **Left Motor** and the **Right Motor** of the TB6612 to two motors, and the **Motor PWM** with the No. 4 and 5 PWM channel of the PCA9685.

Connect the **PWM Power** of the PCA9685 with the Robot HATS, and its channel 0 of with the **Turning Servo**.
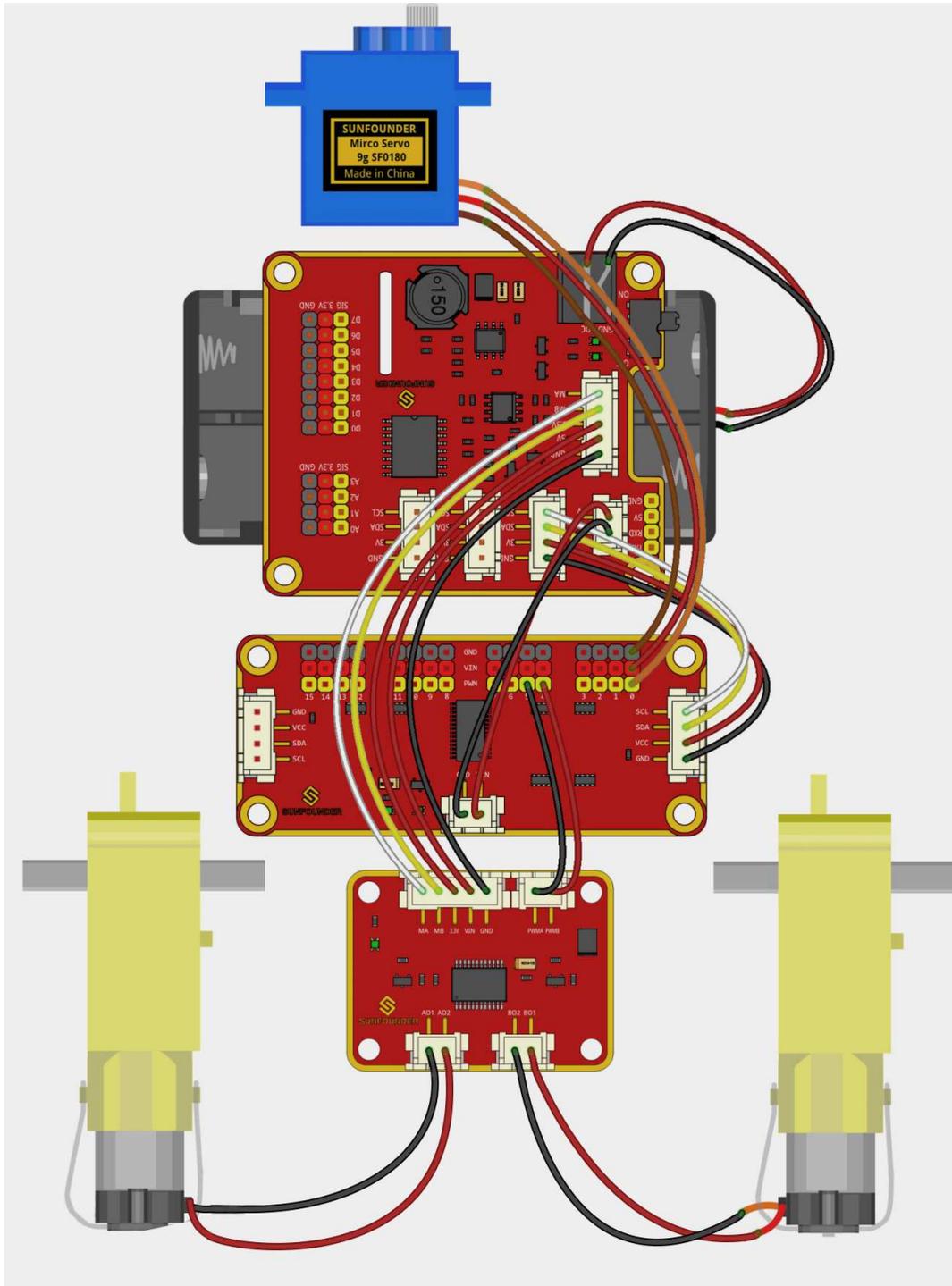
So now the circuit boards are all installed onto the car and the wiring is done. But still you're not ready to adjust the servo yet. First you need to complete some software installation.

Software Installation

1. Log into Raspberry Pi

The TF card onto which the Raspbian has been burnt is inserted into the Raspberry Pi before. Now power the Raspberry Pi.

The installation may take a long time, so you're recommended to so you're recommended to **supply the Raspberry Pi via a USB cable** *and* **then power the Robot HAT with batteries** *in case of power cut-off thus causing a sudden shutdown and file damage of the Raspberry Pi. You can directly power it via the Micro USB port because the Robot HATS won't get damaged by it due to the built-pin protective circuit.*

Plug in the USB Wi-Fi dongle (skip this if you use a Raspberry Pi 3 with the WiFi) and complete the setting in the way you're comfortable with.

Log into the Raspberry Pi via ssh or ssh tools like PuTTY.

2. 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
```

```
pi@raspberrypi:~ $ git clone --recursive https://github.com/sunfounder/SunFounder_PiCar-S.git
Cloning into 'SunFounder_PiCar-S'...
remote: Counting objects: 162, done.
remote: Total 162 (delta 0), reused 0 (delta 0), pack-reused 162
Receiving objects: 100% (162/162), 61.98 KiB | 36.00 KiB/s, done.
Resolving deltas: 100% (80/80), done.
Checking connectivity... done.
Submodule 'example/SunFounder_Light_Follower' (https://github.com/sunfounder/SunFounder_Light_Follower.git) registered for path 'example/SunFounder_Light_Follower'
Submodule 'example/SunFounder_Line_Follower' (https://github.com/sunfounder/SunFounder_Line_Follower.git) registered for path 'example/SunFounder_Line_Follower'
Submodule 'example/SunFounder_Ultrasonic_Avoidance' (https://github.com/sunfounder/SunFounder_Ultrasonic_Avoidance.git) registered for path 'example/SunFounder_Ultrasonic_Avoidance'
Cloning into 'example/SunFounder_Light_Follower'...
remote: Counting objects: 15, done.
remote: Total 15 (delta 0), reused 0 (delta 0), pack-reused 15
Unpacking objects: 100% (15/15), done.
Checking connectivity... done.
Submodule path 'example/SunFounder_Light_Follower': checked out '10b3ccea709f34221c6dc137aca6cb00abfca417'
```

23

```
Cloning into 'example/SunFounder_Line_Follower'...
remote: Counting objects: 8, done.
remote: Total 8 (delta 0), reused 0 (delta 0), pack-reused 8
Unpacking objects: 100% (8/8), done.
Checking connectivity... done.
Submodule path 'example/SunFounder_Line_Follower': checked out '9560e7adbb52a883d438b
78de90d185fc168fed5'
Cloning into 'example/SunFounder_Ultrasonic_Avoidance'...
remote: Counting objects: 24, done.
remote: Total 24 (delta 0), reused 0 (delta 0), pack-reused 24
Unpacking objects: 100% (24/24), done.
Checking connectivity... done.
Submodule path 'example/SunFounder_Ultrasonic_Avoidance': checked out
'9e0349b2155fa7585021bfb979643e5f98aa51c5'
```

Check by the ls command, then you can see the code directory:

SunFounder_PiCar-S/

```
pi@raspberrypi:~ $ ls
Desktop      Downloads      Pictures      python_games        Templates
Documents    Music          Public        SunFounder_PiCar-S  Videos
```

3. 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  README.md  show
```

4. Install the Environment

Installing Automatically by 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 **Installing Manually** below.

```
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 reboot after the installation, so you need to log in again.*

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
python 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
```

Adjust the Servo to 90 Degrees

After reboot, type in the command:

```
picar
```

```
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
```

You can see three commands here.

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.

```
pi@raspberrypi:~ $ picar servo-install
pi@raspberrypi:~ $
```

Now the servo may or may not make noises to rotates to 90 degrees. Script will end almost

25

immediately, but all servos will keep at 90 degrees. You can continue the rest assembly.

Warning: After power is on, a short circuit may happen because the exposed contacts of the PCB may be touched by conductive objects like screw or screwdriver. If it occurs, please remove the conductor immediately or unplug the power as quickly as you can.

Build the Rest of the Car

Connect the Steering Linkage and the rocker arm. **Make sure the rocker arm is in an angle as shown below**. If there is just a littile deviation from the right position of the rocker arm, it can be fine tuned by software. Try to spin the arm on the shaft. If it's unmovable, it means the servo is adjusted to 90 degrees. But if it spins, check whether the wiring is correct. Besides, **the screw is quite sharp at the end, so be careful to assemble in case of getting hurt**.
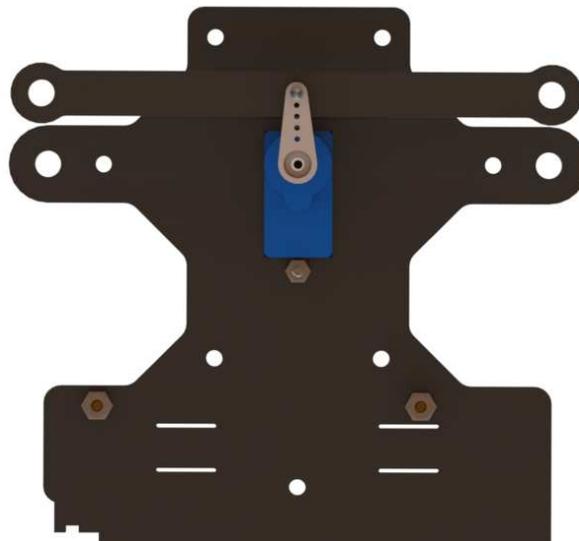
If everything is OK, take out the **Rocker Arm Fixing Screw** to connect them as shown below.
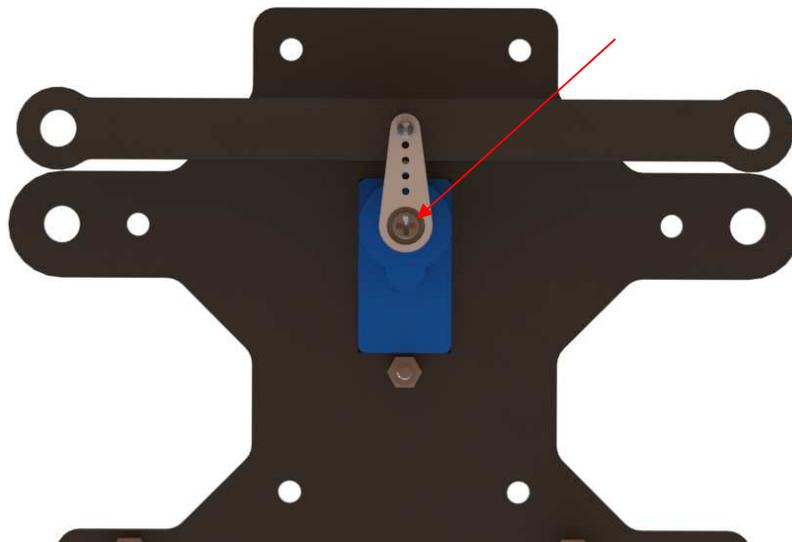
Take out the assembled front wheels and the Upper Plate, and mount the wheels onto the Upper Plate carefully: insert one bulge of the Steering Connector at either wheel into the hole on the Steering Linkage plate, and another bulge into the Upper Plate, and similar with the other wheel.
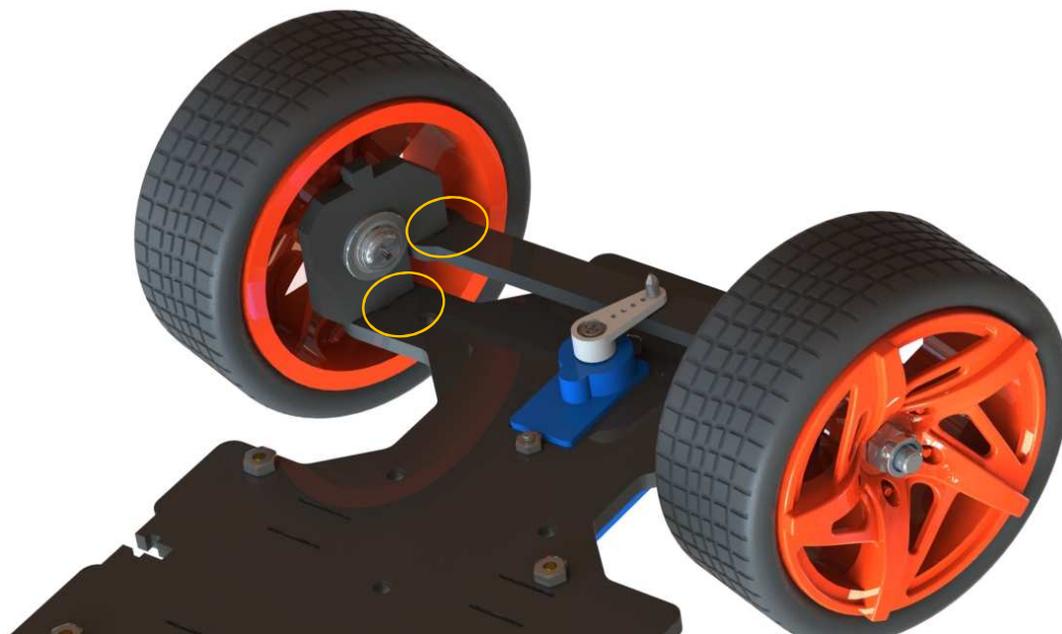
Then put the assembled Front Half Chassis onto the Upper Plate with standoffs aligned with the holes.
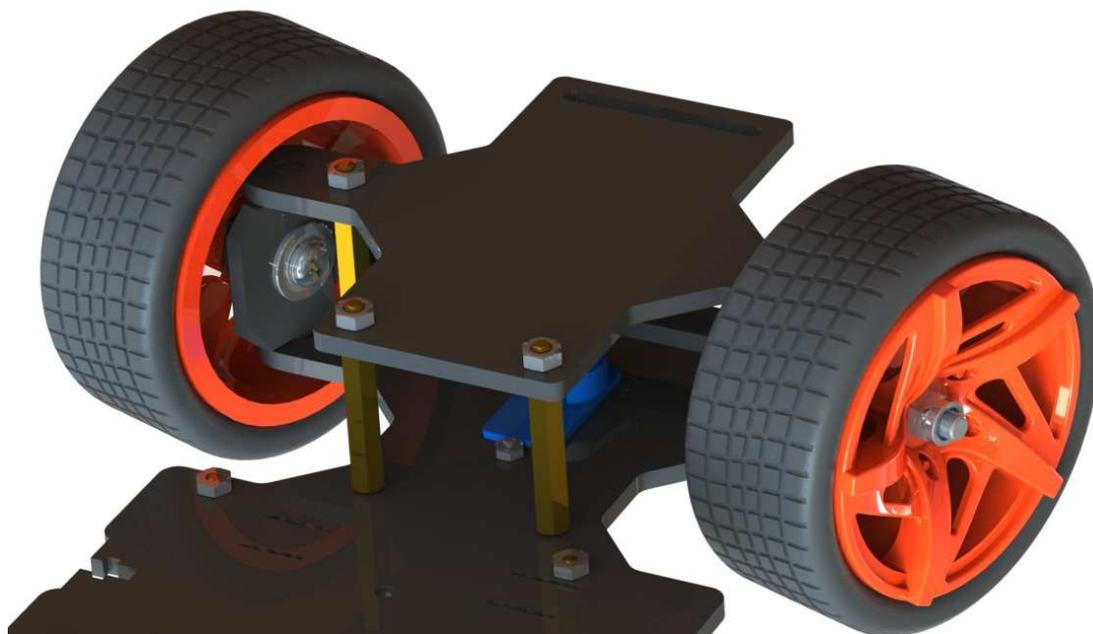
Hold them carefully, turn upside down, and fasten the standoffs and the Uppe Plate with four **M3x8 screws**:

27
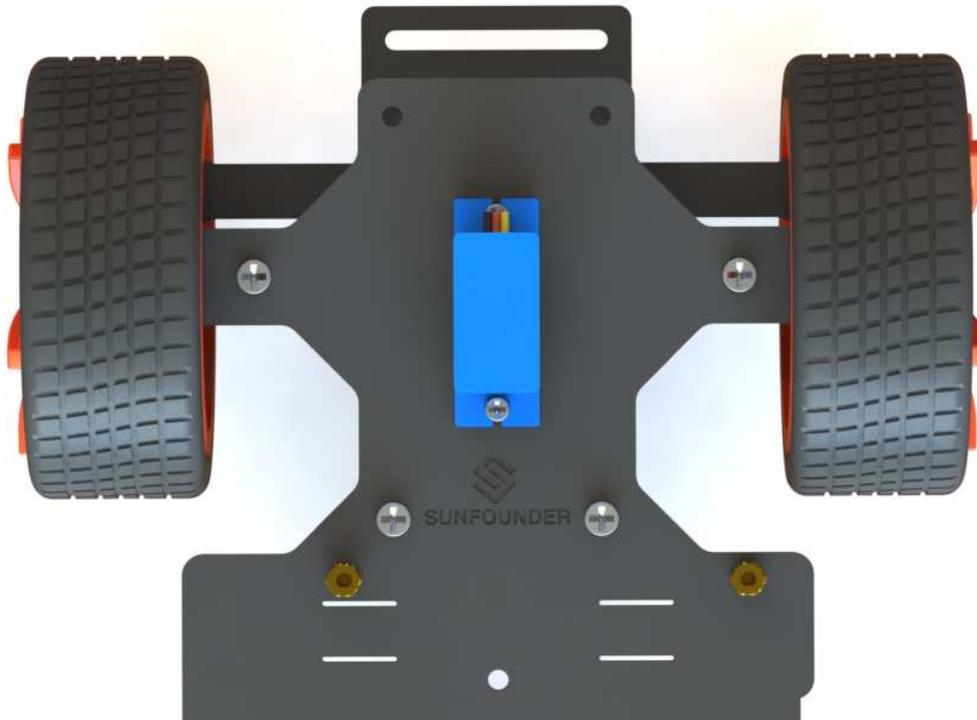
Now, the whole assembly, the wiring and code are all **DONE**! Congrats! Enjoy the satisfaction and thrill.

Configuration

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. Besides, there is an optional parameter **[chn]**. It will be used only when you change the PWM channel of the servo. If you use the default channel 0 according to the wiring diagram, there is no need to enter **[chn]**.

```
pi@raspberrypi:~ $ picar front-wheel-test
DEBUG "front_wheels.py": Set debug off
DEBUG "front_wheels.py": Set wheel debug off
DEBUG "Servo.py": Set debug off
turn_left
turn_straight
turn_right
turn_straight
turn_left
turn_straight
turn_right
```

28

```
^Cturn_straight
```

You may find the direction of the front wheels is not facing exatly front when they are in the straight status (they will return to the status anytime the front wheels program stops running). If there is an obvious deviation from the middle line of the front chassis, remove 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
ls
```

```
pi@raspberrypi:~ $ cd /home/pi/SunFounder_PiCar/picar
pi@raspberrypi:~/SunFounder_PiCar/picar $ ls
back_wheels.py  filedb.py         __init__.py      SunFounder_PCA9685
config          front_wheels.py   PCF8591.py       SunFounder_TB6612
pi@raspberrypi:~/SunFounder_PiCar/picar $
```

Open the **config** file under the folder with an editor. You can see a few parameters. Modify the value of **turning_offset** (the first parameter) to adjust the front wheels. Its value is **0** by default, and you can turn the wheels left by decreasing the value, or turn right by increasing it. For example, if you want to make the front wheels turn right a bit, just modify it to a larger number; 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.

```
pi@raspberrypi:~/SunFounder_PiCar/picar $ nano config
```

```
  GNU nano 2.2.6                        File: config

# File based database


turning_offset = 10


forward_A = 1


forward_B = 1




                        [ Read 9 lines ]
^G Get Help   ^O WriteOut   ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

29

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.

```
pi@raspberrypi:~/SunFounder_PiCar/picar $ picar servo-install
pi@raspberrypi:~/SunFounder_PiCar/picar $
```

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.

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.

```
pi@raspberrypi:~ $ 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
Forward, speed = 5
Forward, speed = 6
Forward, speed = 7
Forward, speed = 8
Forward, speed = 9
Forward, speed = 10
        .

        .

        .
Backward, speed = 10
Backward, speed = 9
Backward, speed = 8
Backward, speed = 7
Backward, speed = 6
Backward, speed = 5
Backward, speed = 4
Backward, speed = 3
Backward, speed = 2
Backward, speed = 1
Finished, motor stop
```

Check whether both the two rear wheels rotate direction is the same as the screen promt.

30

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.

The two other parameters in the file **config** are to set the rear wheels: `forward_A` and `forward_B`. They can 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 **1** for both parameters. Thus if a wheel spins reversely, you only need to change the corresponding parameter for the wheel to **0**.

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.

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

```
pi@raspberrypi:~/SunFounder_PiCar/picar $ 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**!

Ultrasonic Obstacle Avoidance

How the Obstacle Avoidance 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.

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.

Procedures

Step 1. Assembly

Connect the ultrasonic module and the ultrasonic module connector to the ultrasonic module 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**.

33

Step 2. Wiring

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

Ultrasonic Obstacle Avoidance	Robot HATS
SIG	D0
5V	3.3V
GND	GND
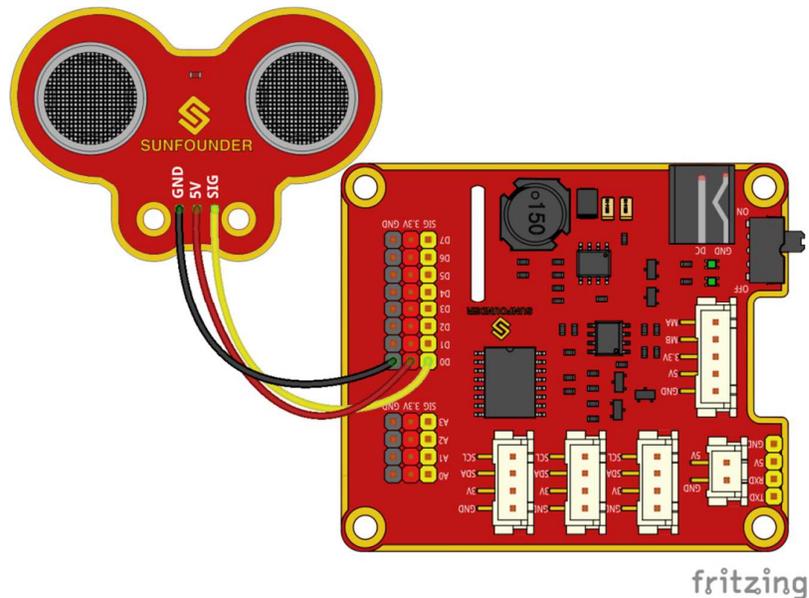
Test for Ultrasonic Obstacle Avoidance

First, test the ultrasonic obstacle avoidance module before applying.

Get into the **directory example:**

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

Run the `test` code:

```
python test_ultrasonic_module.py
```

```
pi@raspberrypi:~ $ cd ~/SunFounder_PiCar-S/example/
pi@raspberrypi:~/SunFounder_PiCar-S/example $ python test_ultrasonic_module.py
distance 32 cm Over 17
distance 32 cm Over 17
distance 29 cm Over 17
distance 29 cm Over 17
Read distance error
distance 21 cm Over 17
distance 21 cm Over 17
distance 18 cm Over 17
distance 18 cm Over 17
distance 16 cm less than 17
distance 15 cm less than 17
```

```
distance 13 cm less than 17
distance 10 cm less than 17
Read distance error
```

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 3. Run the Code

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.

Get into the directory:

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

Run the code:

```
python ultra_sonic_avoid.py
```

Get on the road!

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.

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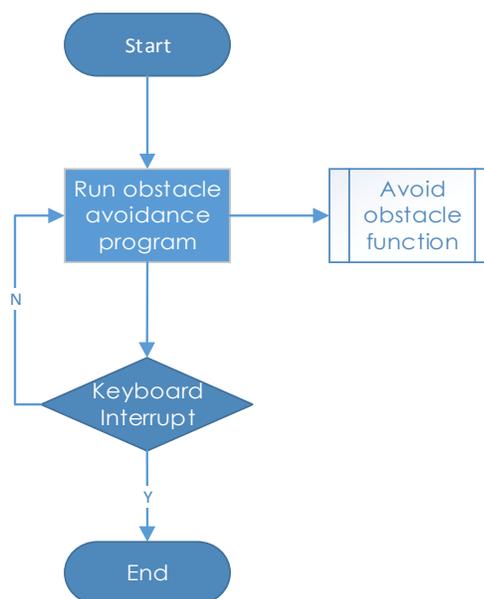
The ultrasonic module returns a digital value, i.e., High or Low level, and the interval time between two levels returned can be converted to the distance to the obstacle. Thus, we call the time module in Python for timing here. The formula to calculate the distance is written in the ultrasonic module's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

Subflow of the Obstacle Avoidance Function
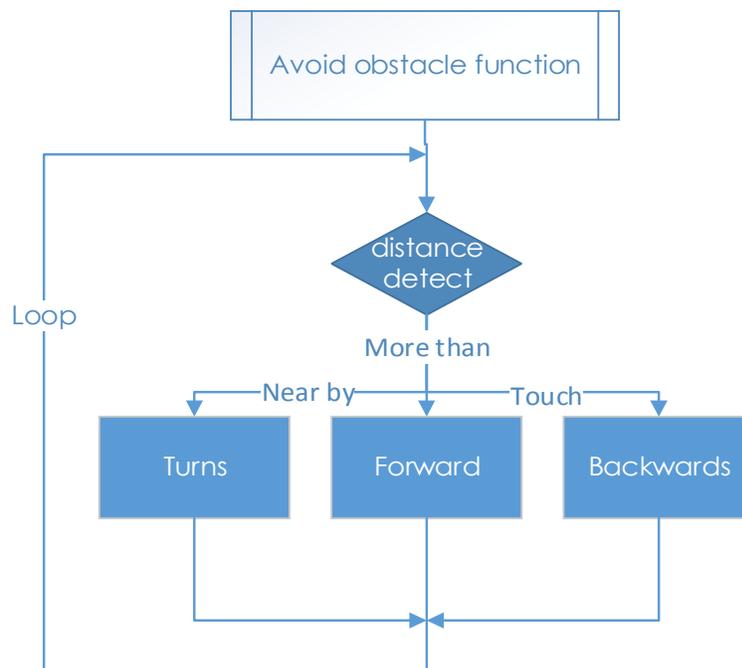
When the car starts, it will detect obstacles and measure the distance in cycle, make judgement, and take actions. Here are three cases: when the distance to the obstacle is **x** (the ranging distance threshold) away, the car will turn directions; when the distance is less than **x**, the car will move backwards before turning direction; when the distance is more than **x**, it will keep moving forwards.

36
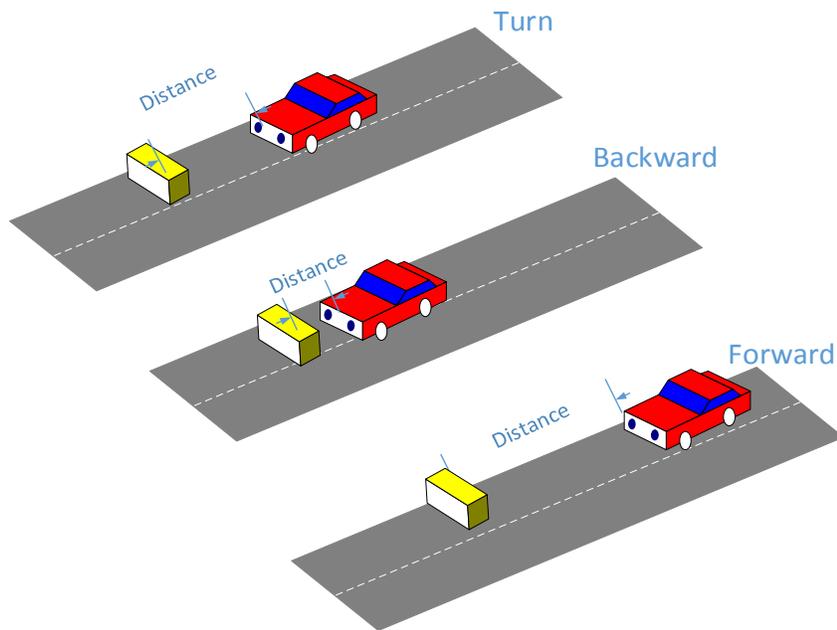
Code Explanation

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

37

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.

Principle

This light follower module applies phototransistors as photosensitive components. Similar to common transistors, the phototransistor can amplify current, but the current at the collector terminal is controlled not only by the circuit and current of the base area, but also by light emission. Usually, the base terminal is not exposed, but some may still do for temperature compensation and extra control.

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.

38

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:
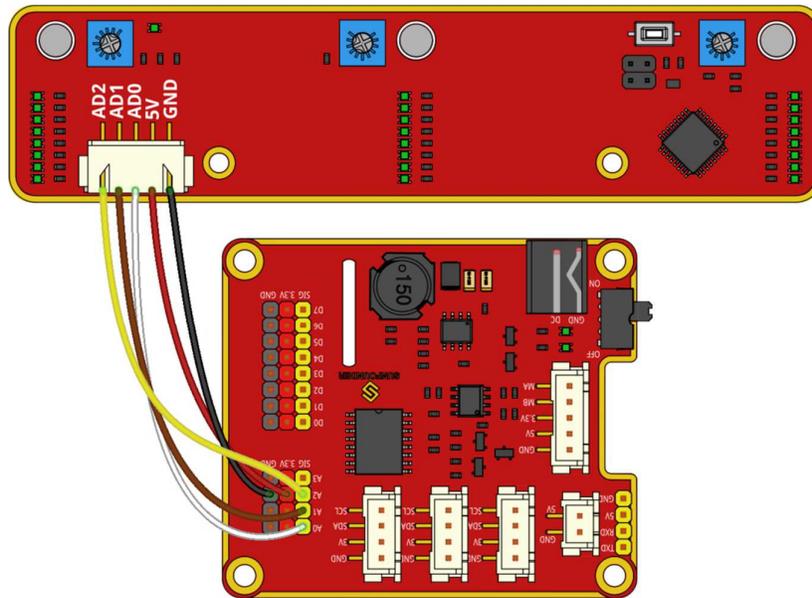
fritzing

Test for Light Follower

Let's test the light follower first.

Get into the directory example

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

Run the test code

```
python test_light_module.py
```

```
pi@raspberrypi:~ $ cd ~/SunFounder_PiCar-S/example/
pi@raspberrypi:~/SunFounder_PiCar-S/example $ python test_light_module.py
[237, 233, 237]
[236, 230, 236]
[232, 238, 246]
[237, 240, 251]
[145, 175, 200]
[228, 234, 74]
[219, 226, 23]
[220, 192, 21]
[233, 238, 23]
[236, 241, 22]
```

Expose the phototransistors to the light spot of the flashlight. When you increase the light intensity, more LEDs light up, and the ouput values decrease. Here we can rotate the blue adjustable resistor to change the values under the same light luminance. If there is no light, only one LED will light up with the output value as 255; if the light is the brightest, all the LED will light up, and it would be better to adjust the output value to 10-25 under this circumstance.

Place the module in an open field with even light exposure so the experiment can be free of the interference of unsteady ambient light thus the sensor is more sensitive. Check this way: expose

40

the light follower module under the natural light (or other bright conditions) to get the output value; then shine the flashlight on the phototransistors to get another output value. A place with a gap of 20+ between these two values would be a better choice, so that we can get an obvious effect when the car follows light.

Step 3 Run the Code

Get into the directory

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

Run the code

```
python light_follower.py
```

Calibrate the light following

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.

Picar-S follows light

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.

Software 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.

41

2. Here write two main functions/modules including light following calibration and light following in the main program.
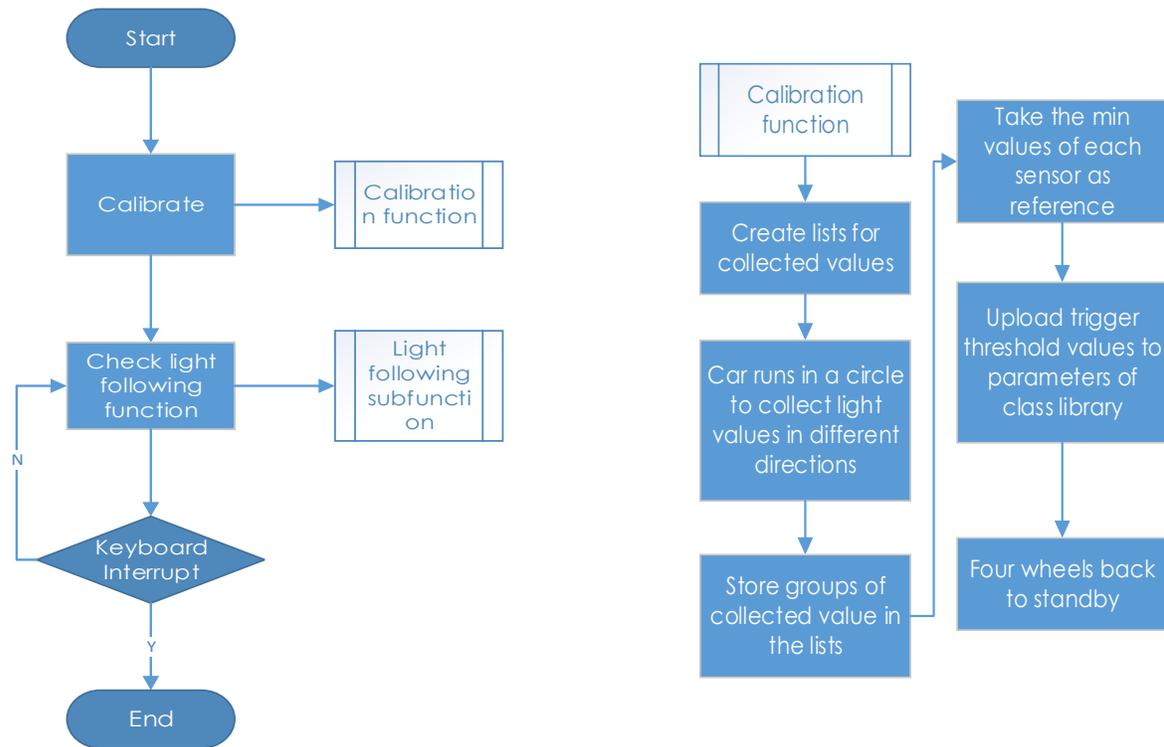
Subflow of Light Follower Calibration Function

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 ouput 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

42
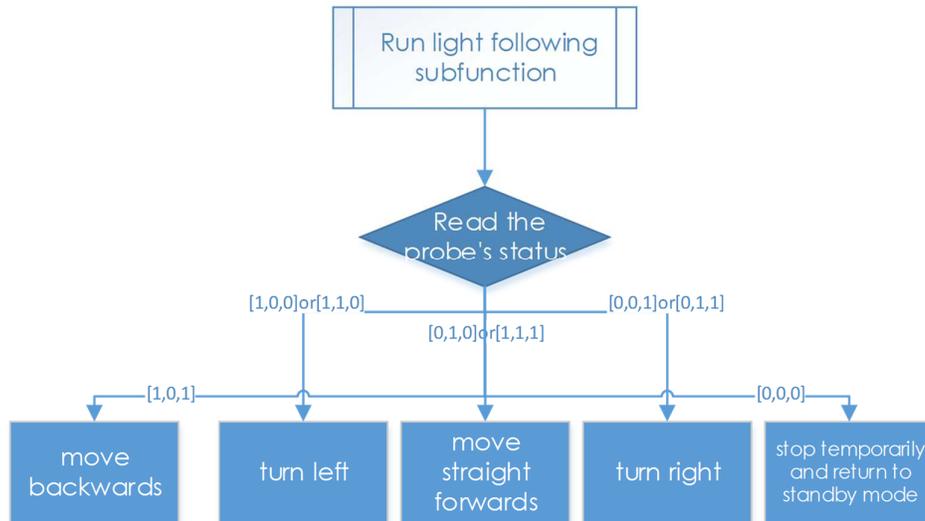
The light follower includes three phototransistors, thus its status list is composed of three elements which represent 8 statuses (based on permutation and combination). And here we need to set related responses to these statuses.

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-

Small turning
[1,1,0]

Straight forward
[0,1,0]

Large turning
[0,0,1]

43
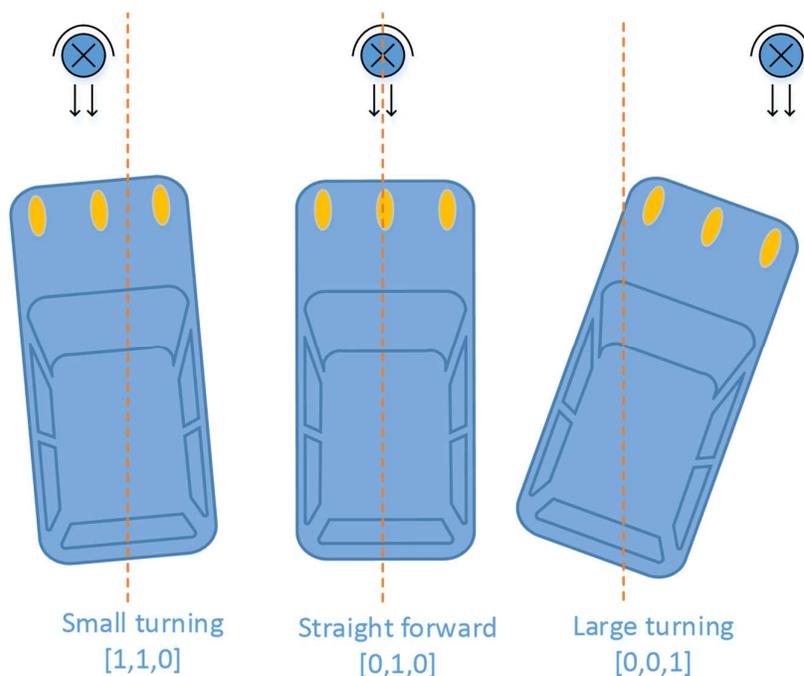
angle and small-angle turning. If the light is at the central left side (status `[1,1,0]`), we should apply a small-angle turning; if the light is at the edge of the left side (status `[1,0,0]`), we should apply a large-angle turning.

Code 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()`

44

```
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 to Follow Lines

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.

Principle

The TCRT5000 IR sensor on the line follower has a TX and RX inside. If a bright color surface is detected in front of the sensor, such as a white paper (about 1cm away), most of the IR rays will be reflected; if a dark color like black surface is detected, most of the rays will be absorbed, while only a small amount will be reflected. The RX will output different analog signals according to the intensity of the reflected IR light.

The ADC chip on the HATS can receive analog signal and convert it into digital signal, and transfer to the Raspberry Pi.

## 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.

45
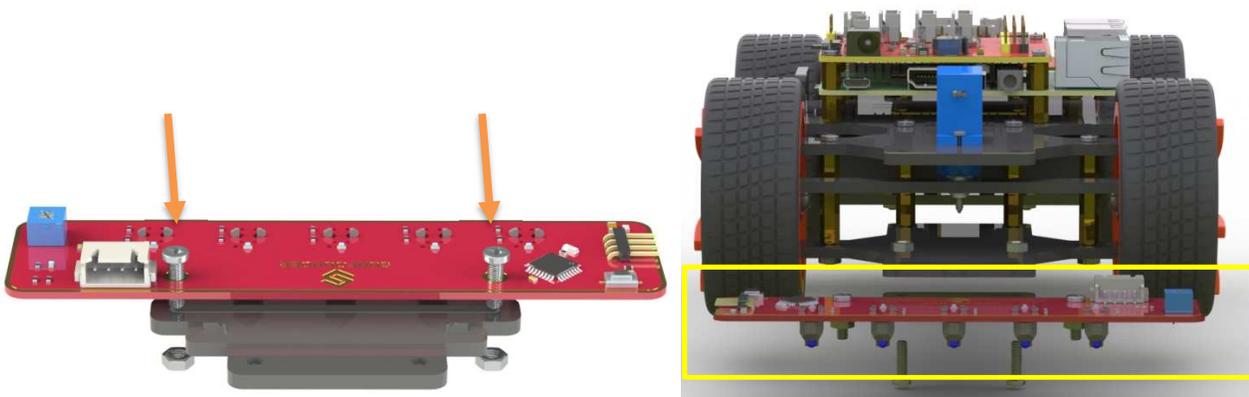
Step 2 Wiring

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

Line Follower	Robot HATS
SCL	SCL
SDA	SDA
VCC	3.3V
GND	GND
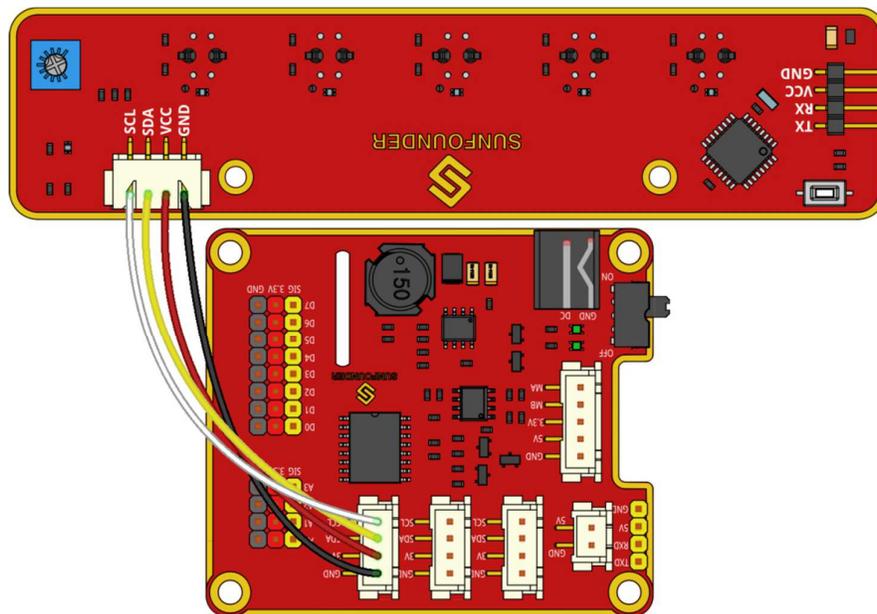
fritzing

Test for Line Follower

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
```

```
pi@raspberrypi:~ $ cd ~/SunFounder_PiCar-S/example/
pi@raspberrypi:~/SunFounder_PiCar-S/example $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- 11 -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
```

46

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

```
python test_line_module.py
```

```
pi@raspberrypi:~/SunFounder_PiCar-S/example/ $ python test_line_module.py
[58, 38, 84, 121, 112]
[58, 38, 84, 121, 112]
[58, 38, 84, 121, 112]
[58, 38, 84, 121, 112]
[59, 38, 84, 121, 112]
[58, 38, 84, 121, 112]
[58, 38, 84, 121, 112]
[58, 38, 84, 121, 112]
[58, 38, 84, 121, 112]
[58, 38, 84, 121, 112]
```

Calibrate the line follower

The IR sensor on the line follower should be calibrated before application. Run the line following code *python Line_Follower.py*. The main program will print the data of 5 channels. Place each of the five probes above the black line, and then above a white surface, so we can get the gap between the two values. Adjust the potentiometer, and repeat the previous operations to get the gap again. Thus we can get the optimal resistance of the pot when the gap is the max value.

Step 3 Run the Code

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

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

Run the line follower code

```
python line_follower.py
```

Calibrate the line follower on the car

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.

Starts Running!

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

47

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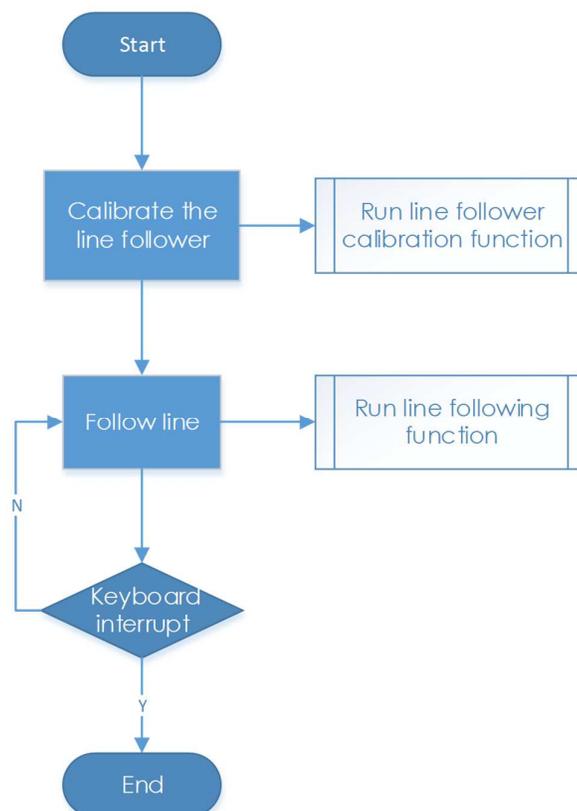
A track sample is shown as below (the original map file can be found under folder **map** in **github**):


48
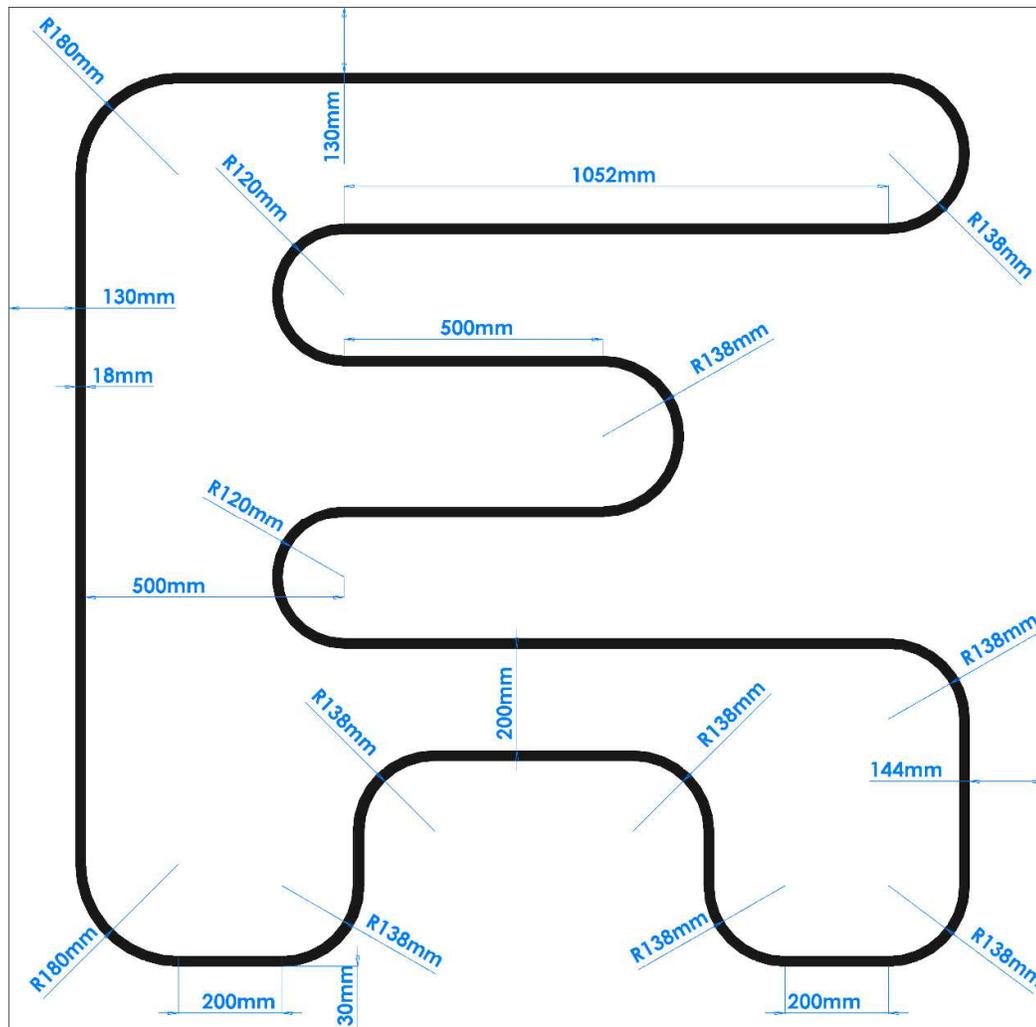
Software Flow

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].
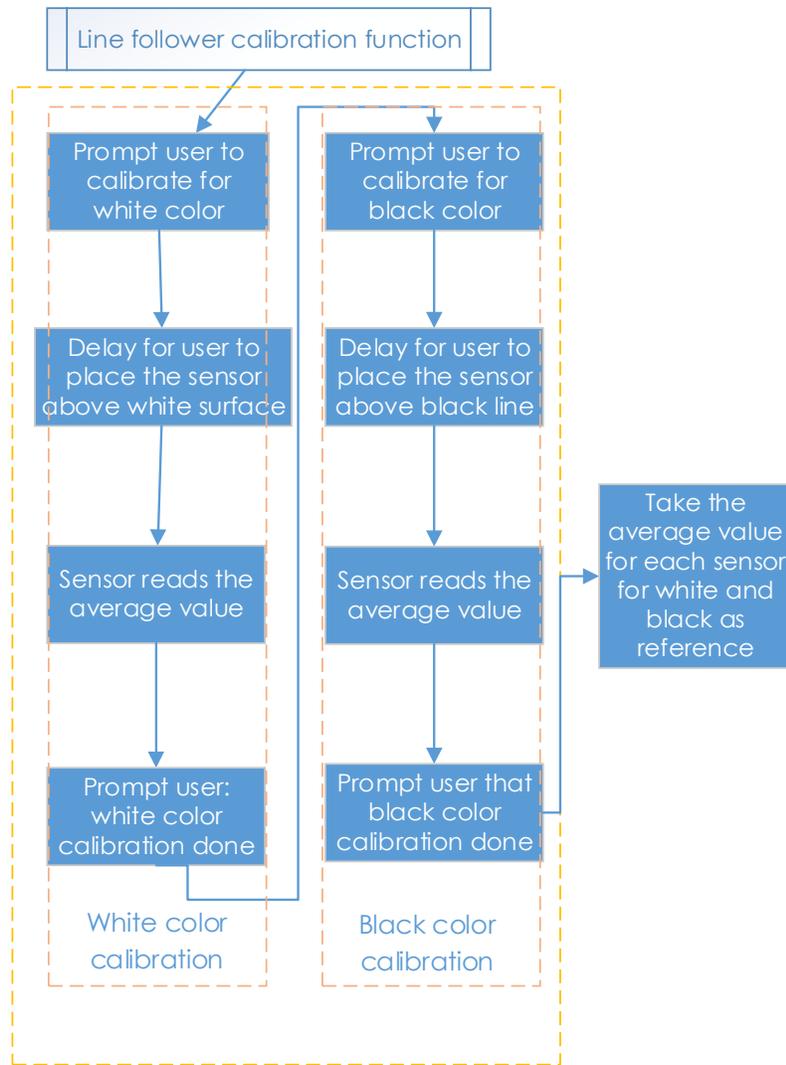
Subflow of Line Following Function

50
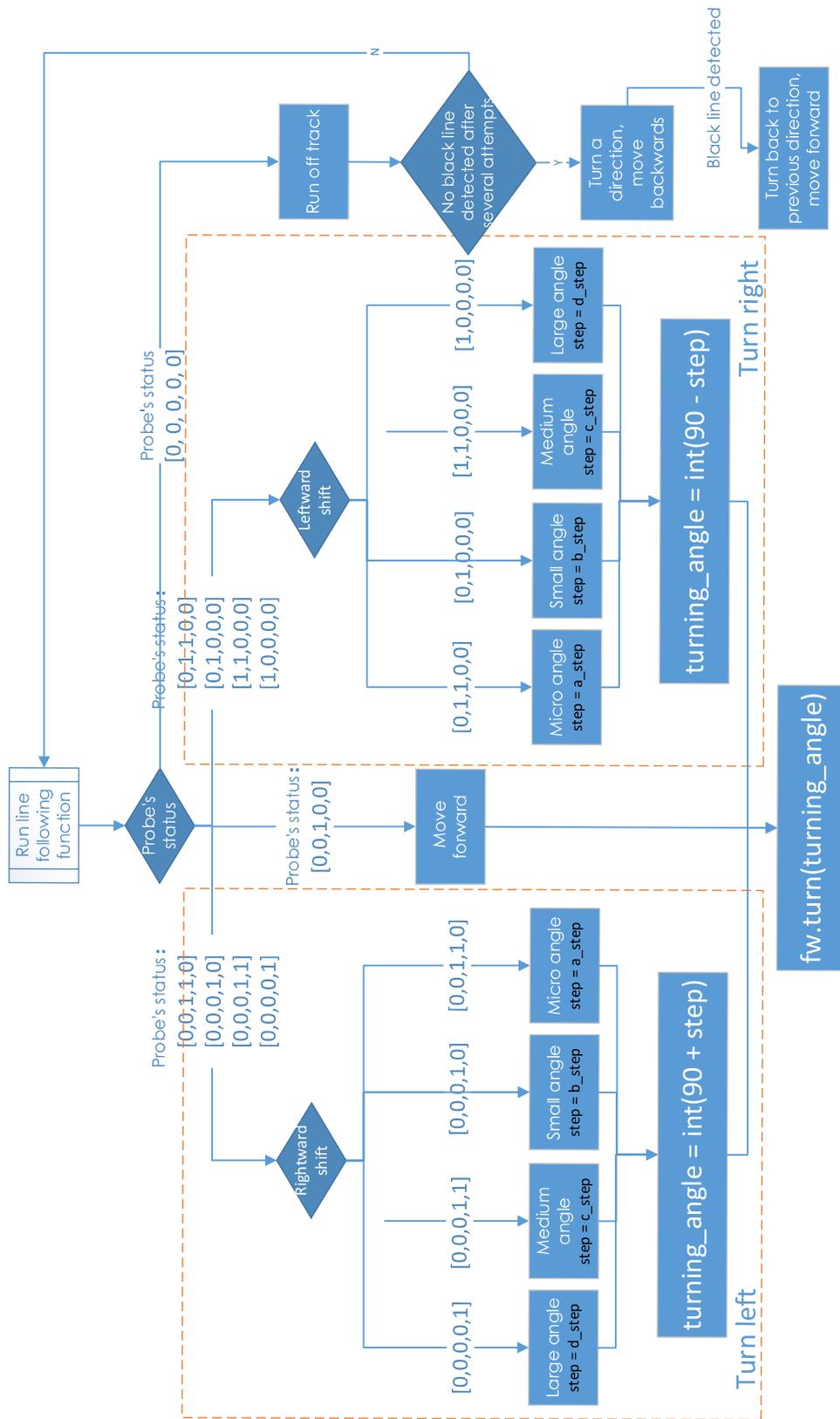
In the line following function, we set the turning angle of the servo in different levels according

51

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.

| Large angle | Medium angle | Small angle | Micro angle | In the middle |
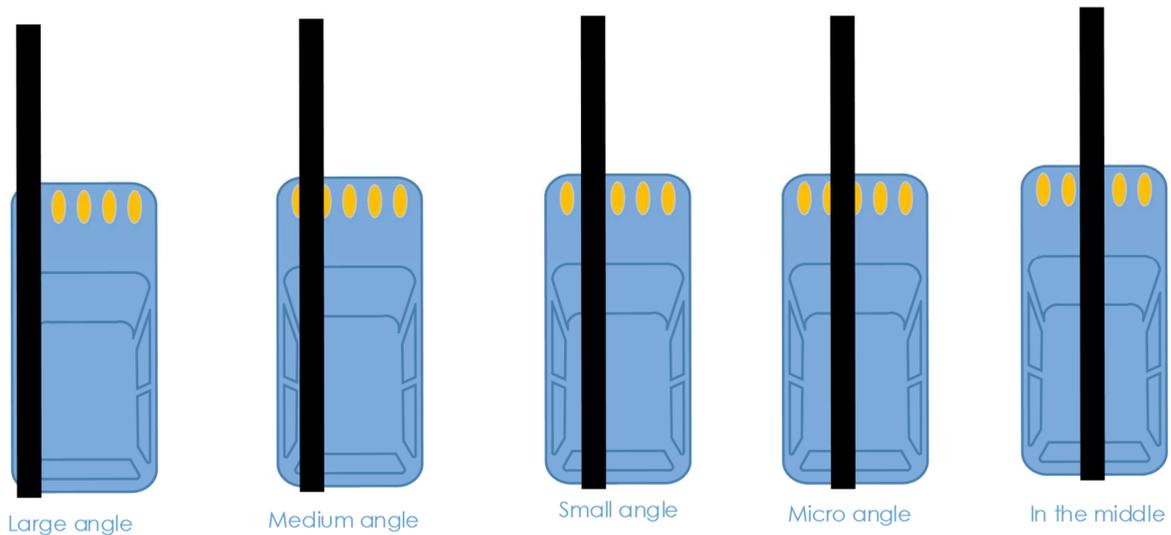
There is a special case: if the car runs off the track, and all the probes cannot detect the black lines any more, then it will continue the program below.

In some case, especially when the car turns in a direction when the semidiameter 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

52
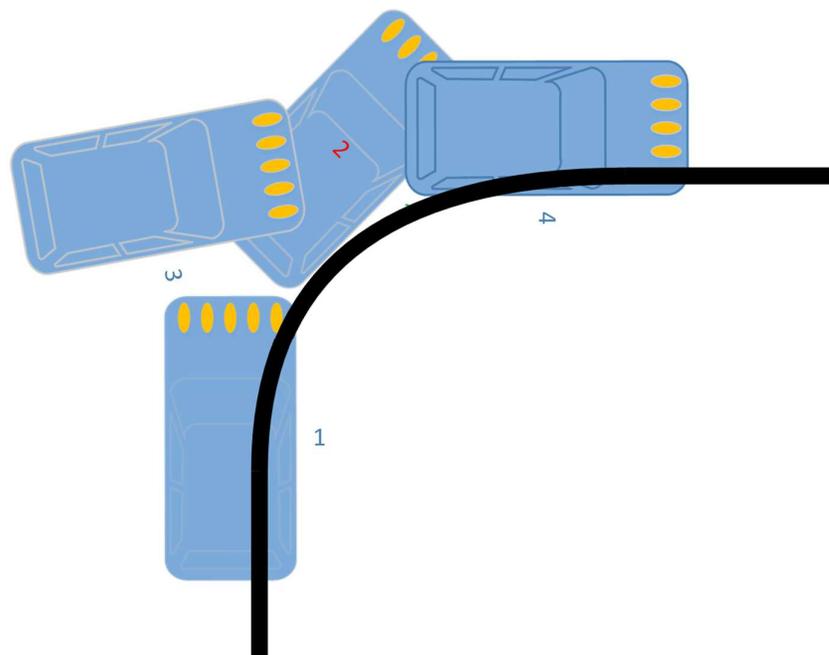
response program to let the car move backwards in the opposite direction (3), and then turn back to the original direction until a black line is detected again and move forward (4).

Code 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.

53

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 referece.

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

```
python 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.

54
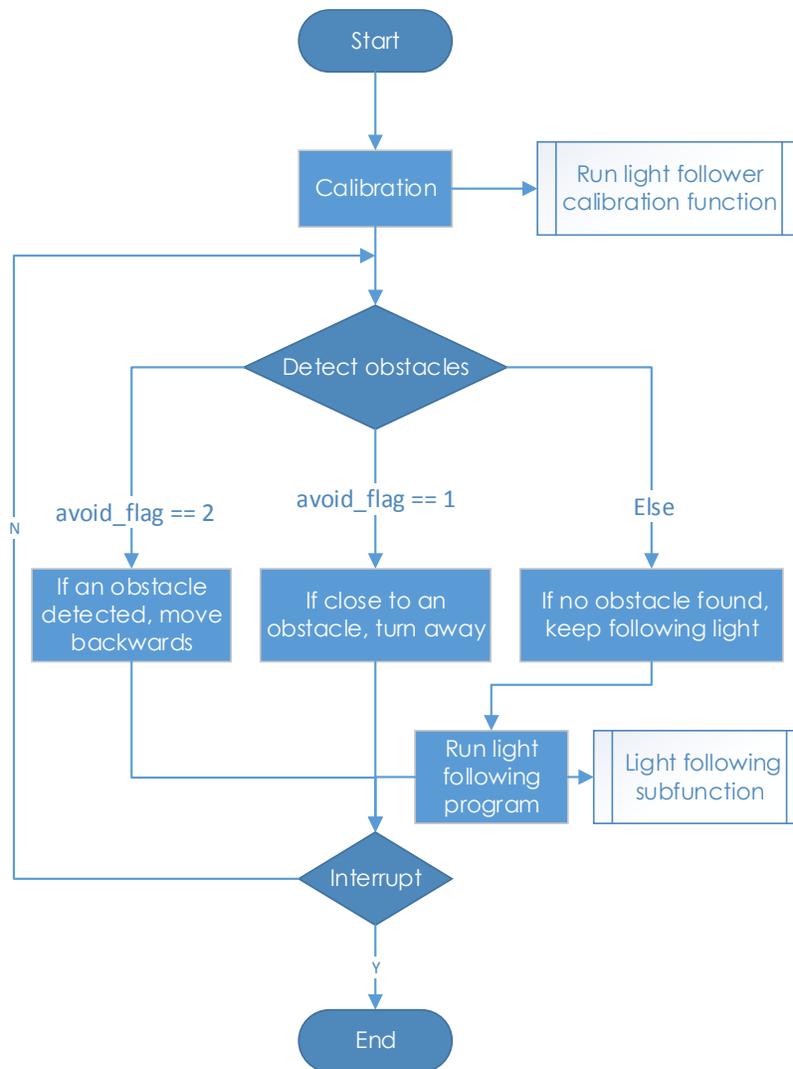
Since the light following and obstacle avoidance of the car depend on the sensor modules, we set two functions to read the status of two sensors separately, and assign values to flags to be returned from the functions: **state_light()**, and **state_sonic()**.

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**;

55

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

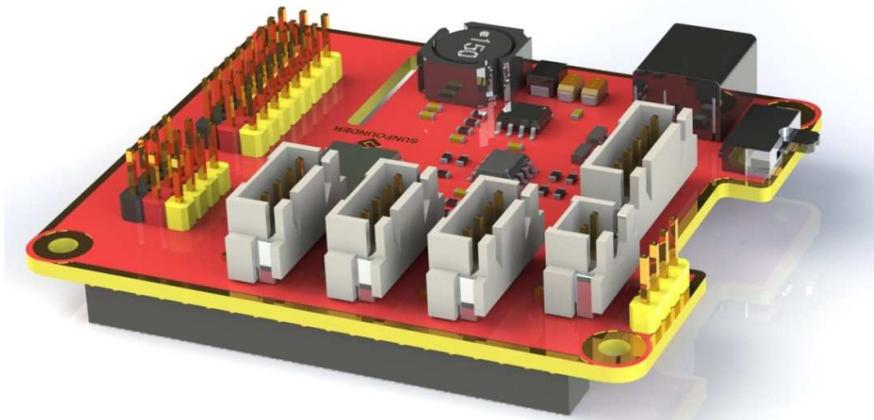if the spot is **at the back**, it will return `light_flag = 3`;

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

Appendix 1: Modules

Robot HATS

Robot HATS is a specially-designed HAT for a 40-pin Raspberry Pi and can work with Raspberry Pi model B+, 2 model B, and 3 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.
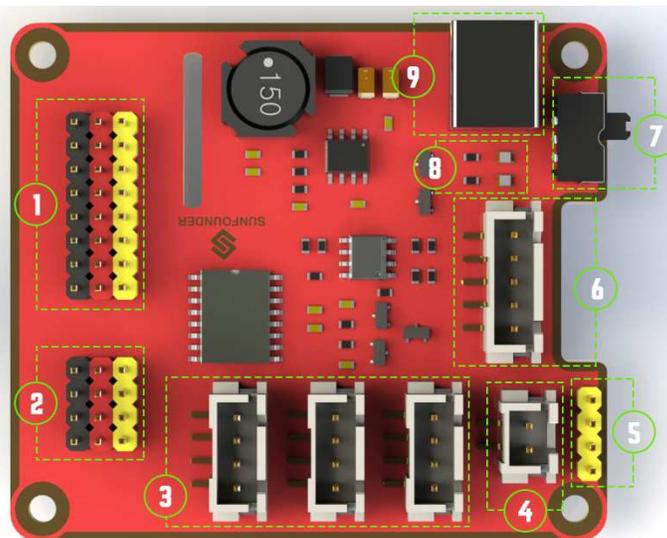

①. **Digital ports**: 3-wire digital sensor ports, signal voltage: 3.3V, VCC voltage: 3.3V.

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.

57

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

⑥. **TB6612 motor control ports**: includes 3.3V for the TB6612 chip, 5V for motors, and direction control of motors MA and MB; working with SunFounder TB6612 Motor driver.

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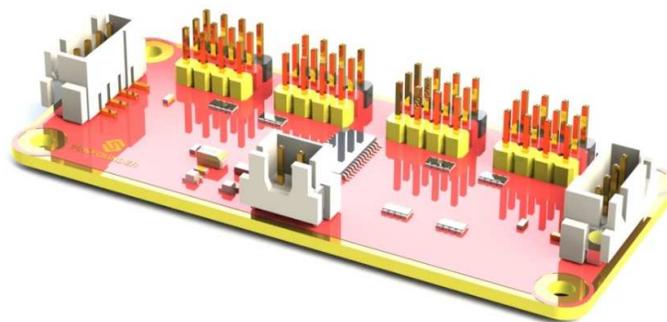
PCA9685 16-channel 12-bit I2C Bus PWM driver. It supports independent PWM output power and is easy to use 4-wire I2C port for connection in parallel, distinguished 3-color ports for PWM output.
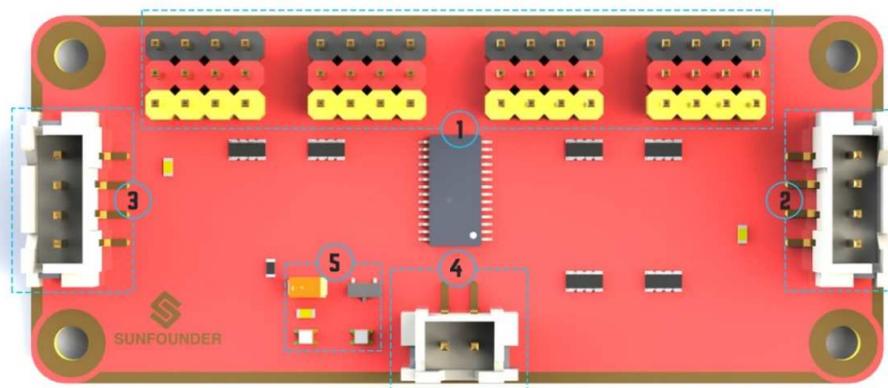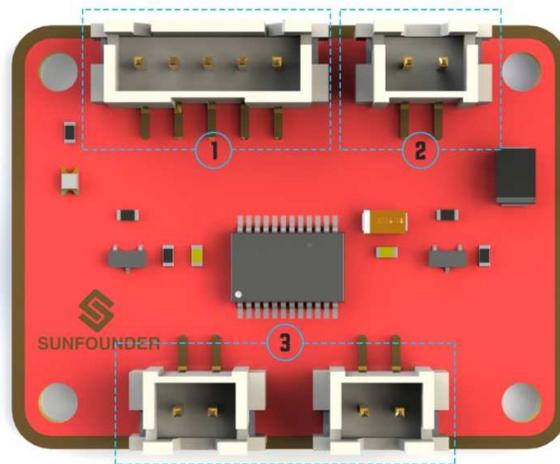

①. **PWM output ports**: 3-color ports, independent power PWM output port, connect to the servo directly.

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.

58
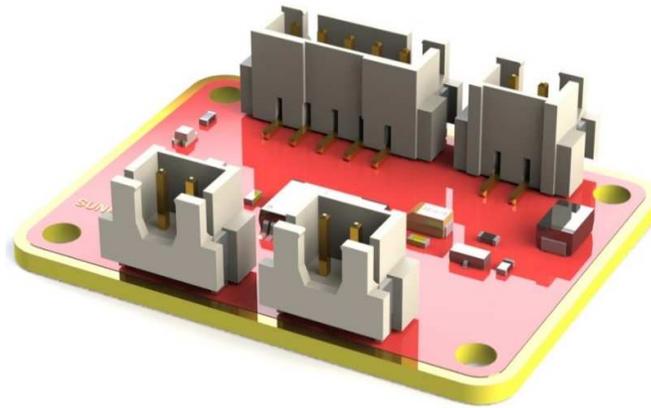
TB6612

The TB6612 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

59
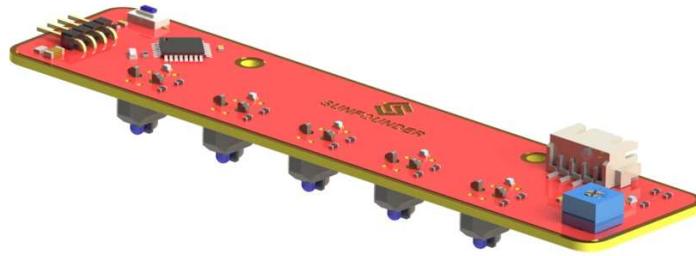
The TCRT5000 infrared photoelectric switch adopts a high transmit power infrared photodiode and a highly sensitive phototransistor. It works by applying the principle of objects' reflecting IR light – the light is emitted, then reflected, and sensed by the synchronous circuit. Then it determines whether 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
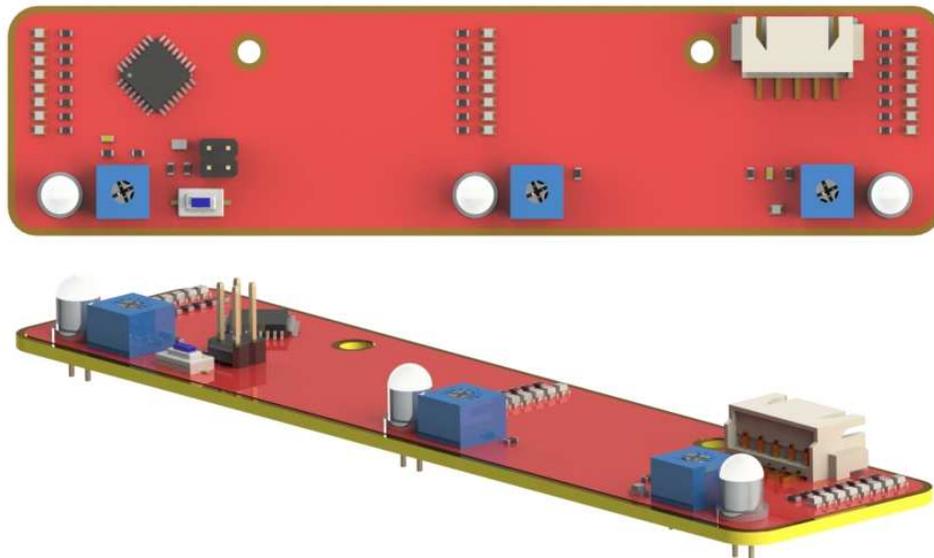
Phototransistor, or photo triode, is a photoelectric conversion component that senses the light intensity. The stronger light, the larger currents, and better conductivity of the triode.

Based on this principle, we use it for light following. It works by converting the conductivity to voltage output, which means the stronger light shone on it, the smaller voltage output. Then

60

convert the voltage into digital values by the A/D converter on the MCU. And we can tell the light intensity then by this value.

The module can be used to sense the ambient luminance. The blue potentiometer is to adjust the sensitivity – you can change it to adapt to specific circumstances. There are two mounting holes of 3mm diameter on the module for easy assembly on your robot.

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.

SunFounder SF0180 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.

61

Electrical Specifications:

Item	V = 4.8V	V = 6.0V
Operating speed (no load)	0.12 Sec/60°	0.09 Sec/60°
Running current (no load)	140 mA	200 mA
Stall torque (locked)	2.0 kg-cm	2.5 kg-cm
Stall current (locked)	520 mA	700 mA
Idle current (stop)	5 mA	5 mA

For more information, refer to SunFounder_PiCar-S/datasheet/SunFounder_SF0180.pdf in the code package on our Github page. .

WiFi Adapter

The Ralink RT5370 Wireless Adapter is used in the car. Below are the features:

USB: XC1291

Network Connectivity: Wireless-Wi-Fi 802.11b, Wireless-Wi-Fi 802.11g, Wireless-Wi-Fi 802.11n

Applicable Network Type: Ethernet

Transmission Rate: 150Mbps

DC Gear Motor

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

	Model	F130SA-11200-38V
Motor	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