SunFounder Robot HAT

www.sunfounder.com

Apr 02, 2024

CONTENTS

1 Features **3**

2 Hardware Introduction **5**
- 2.1 Pinout . . . 5
- 2.2 Pin Mapping . . . 8
- 2.3 Digital IO . . . 8
- 2.4 ADC . . . 10
- 2.5 PWM . . . 12
- 2.6 I2C . . . 13
- 2.7 SPI . . . 14
- 2.8 UART . . . 15
- 2.9 Buttons . . . 15
- 2.10 Speaker and Speaker Port . . . 15
- 2.11 Motor Port . . . 16
- 2.12 Battery Level Indicator . . . 16

3 About the Battery **17**

4 Install the `robot-hat` Module **19**

5 Install `i2samp.sh` for the Speaker **21**

6 On-Board MCU **23**
- 6.1 Introduce . . . 23
- 6.2 ADC . . . 23
- 6.3 PWM . . . 24
  - 6.3.1 Changing PWM Frequency . . . 24
  - 6.3.2 Pulse width . . . 24
  - 6.3.3 Prescaler . . . 24
  - 6.3.4 Period . . . 25
  - 6.3.5 PWM Timer(IMPORTANT) . . . 25
  - 6.3.6 Example . . . 25
- 6.4 Reset MCU . . . 26

7 Reference **27**
- 7.1 class `Pin` . . . 27
- 7.2 class `ADC` . . . 30
- 7.3 class `PWM` . . . 31
- 7.4 class `Servo` . . . 33
- 7.5 module `motor` . . . 34
  - 7.5.1 class `Motors` . . . 34

7.5.2 class Motor . . . 37
7.6 module modules . . . 37
7.6.1 class Ultrasonic . . . 37
7.6.2 class ADXL345 . . . 38
7.6.3 class RGB_LED . . . 39
7.6.4 class Buzzer . . . 40
7.6.5 class Grayscale_Module . . . 42
7.7 class Robot . . . 43
7.8 class Music . . . 46
7.9 class TTS . . . 51
7.10 module utils . . . 53
7.11 class FileDB . . . 54
7.12 class I2C . . . 56
7.13 class _Basic_class . . . 57

8 Some Projects **59**
8.1 Control Servos and Motors . . . 59
8.2 DIY Car . . . 61
8.3 Read from Photoresistor Module . . . 62
8.4 Read from Ultrasonic Module . . . 64
8.5 Plant Monitor . . . 65
8.6 Say Something . . . 68
8.7 Security System . . . 69
8.8 Community Tutorials . . . 71

9 FAQ **73**
9.1 Q1: Can the battery be connected while providing power to the Raspberry Pi at the same time? . . . 73
9.2 Q2: Can the Robot HAT be used while charging? . . . 73
9.3 Q3: Why is there no sound from the speaker? . . . 73

Python Module Index **75**

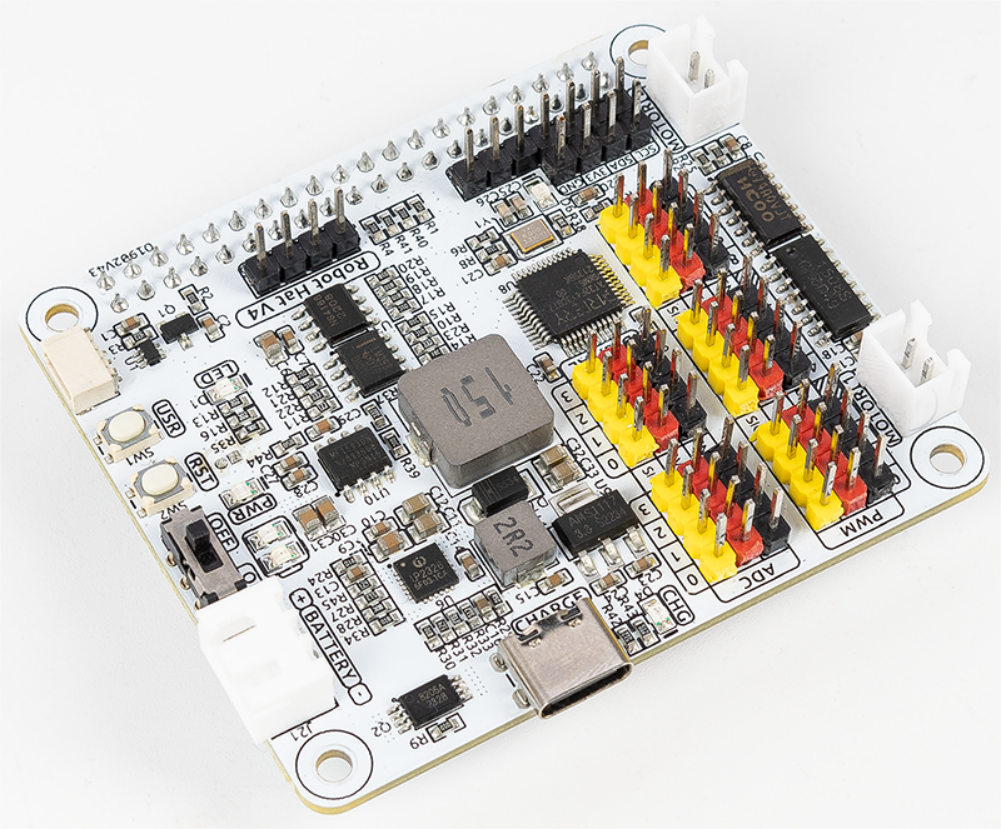
Index **77**

Thanks for choosing our Robot HAT.

Note: This document is available in the following languages.

-
-
-

Please click on the respective links to access the document in your preferred language.



Robot HAT is a multifunctional expansion board that allows Raspberry Pi to be quickly turned into a robot. An MCU is on board to extend the PWM output and ADC input for the Raspberry Pi, as well as a motor driver chip, Bluetooth module, I2S audio module and mono speaker. As well as the GPIOs that lead out of the Raspberry Pi itself.

It also comes with a Speaker, which can be used to play background music, sound effects and implement TTS functions to make your project more interesting.

Accepts 7-12V PH2.0 2pin power input with 2 power indicators. The board also has a user available LED and a button for you to quickly test some effects.

In this document, you will get a full understanding of the interface functions of the Robot HAT and the usage of these interfaces through the Python `robot-hat` library provided by SunFounder.

CHAPTER

ONE

FEATURES

- Shutdown Current: < 0.5mA
- Power Input: USB Type-C, 5V/2A
- Charging Power: 5V/2A 10W
- Output Power: 5V/3A
- Included Batteries: 2 x 3.7V 18650 Lithium-ion Batteries, XH2.0 3P Interface
- Battery Protection: Reverse polarity protection
- Charging Protection: Input undervoltage protection, input overvoltage protection, charging balance, overheat protection
- Onboard Charging Indicator Light: CHG
- Onboard Power Indicator Light: PWR
- Onboard 2 Battery Level Indicator LEDs
- Onboard User LED, 2 tactile switches
- Motor Driver: 5V/1.8A x 2
- 4-channel 12-bit ADC
- 12-channel PWM
- 4-channel digital signals
- Onboard SPI interface, UART interface, I2C interface
- Mono Speaker: 81W

Table 1: Electrical Characteristics

Parameters:	Minimum Value:	Typical Value:	Maximum Value:	Unit:
Input Voltage:	4.25	5	8.4	V
Battery Input Voltage:	6.0	7.4	8.4	V
Overcharge Protection (Battery):		8.3		V
Input Undervoltage Protection:	4.15	4.25	4.35	V
Input Overvoltage Protection:	8.3	8.4	8.5	V
Charging Current (5V):			2.0	A
Output Current (5V):			3.0	A
Output Voltage:	5.166	5.246	5.327	V
Charging Overheat Protection:	125	135	145	°C
DC-DC Overheat Protection:	70	75	80	°C
Motor Output Current:			1.8	A

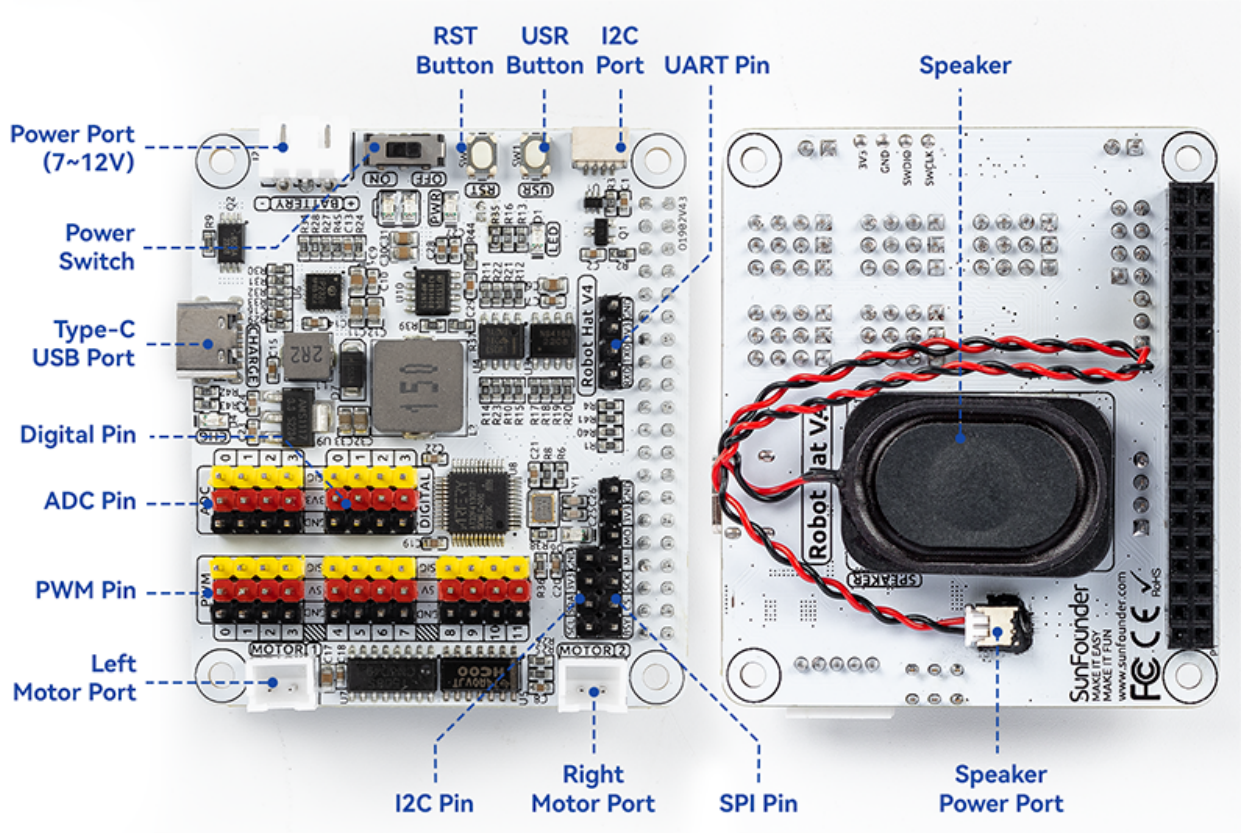
CHAPTER

TWO

HARDWARE INTRODUCTION

The Robot Hat V4 features 2 lithium battery charging, 5V/3A DC-DC discharge, I2S audio output and speaker, a simple battery level indicator, microcontroller-based PWM and ADC drivers, as well as motor drivers.

2.1 Pinout



Power Port

- 7-12V PH2.0 3pin power input.
- Powering the Raspberry Pi and Robot HAT at the same time.

Power Switch

- Turn on/off the power of the robot HAT.

- When you connect power to the power port, the Raspberry Pi will boot up. However, you will need to switch the power switch to ON to enable Robot HAT.

Type-C USB Port

- Insert the Type-C cable to charge the battery.
- At the same time, the charging indicator lights up in red color.
- When the battery is fully charged, the charging indicator turns off.
- If the USB cable is still plugged in about 4 hours after it is fully charged, the charging indicator will blink to prompt.

Digital Pin

- 4-channel digital pins, D0-D3.
- Pin: *Digital IO*.
- API: *class Pin*.

ADC Pin

- 4-channel ADC pins, A0-A3.
- Pin: *ADC*.
- API: *class ADC*.

PWM Pin

- 12-channel PWM pins, P0-P11.
- Pin: *PWM*.
- API: *class PWM*.

Left/Right Motor Port

- 2-channel XH2.54 motor ports.
- Pin: *Motor Port*.
- API: *module motor*, 1 for left motor port, 2 for right motor port.

I2C Pin and I2C Port

- **I2C Pin**: P2.54 4-pin interface.
- **I2C Port**: SH1.0 4-pin interface, which is compatible with QWIIC and STEMMA QT.
- These I2C interfaces are connected to the Raspberry Pi's I2C interface via GPIO2 (SDA) and GPIO3 (SCL).
- Pin: *I2C*.
- API: *class I2C*.

SPI Pin

- P2.54 7-pin SPI interface.
- Pin: *SPI*.

UART Pin

- P2.54 4-pin interface.
- Pin: *UART*.

RST Button

- The RST button, when using Ezblock, serves as a button to restart the Ezblock program.
- If not using Ezblock, the RST button does not have a predefined function and can be fully customized according to your needs.
- Pin: *Buttons*.
- API: *class Pin*

USR Button

- The functions of USR Button can be set by your programming. (Pressing down leads to a input "0"; releasing produces a input "1".)
- API: *class Pin*, you can use `Pin("SW")` to define it.
- Pin: *Buttons*.

Battery Indicator

- Two LEDs light up when the voltage is higher than 7.6V.
- One LED lights up in the 7.15V to 7.6V range.
- Below 7.15V, both LEDs turn off.
- *Battery Level Indicator*.

Speaker and Speaker Port

- **Speaker**: This is a 2030 audio chamber speaker.
- **Speaker Port**: The Robot HAT is equipped with onboard I2S audio output, along with a 2030 audio chamber speaker, providing a mono sound output.
- Pin: *Speaker and Speaker Port*.
- API: *class Music*

2.2 Pin Mapping

Table 1: Raspberry Pi IO

Robot Hat V4	Raspberry Pi	Robot Hat V4	Raspberry Pi
NC	3V3	5V	5V
SDA	SDA	5V	5V
SCL	SCL	GND	GND
D1	GPIO4	TXD	TXD
GND	GND	RXD	RXD
D0	GPIO17	I2S BCLK	GPIO18
D2	GPIO27	GND	GND
D3	GPIO22	MOTOR 1 DIR	GPIO23
NC	3V3	MOTOR 2 DIR	GPIO24
SPI MOSI	MOSI	GND	GND
SPI MISO	MISO	USR BUTTON	GPIO25
SPI SCLK	SCLK	SPI CE0	CE0
GND	GND	NC	CE1
NC	ID_SD	NC	ID_SC
MCU Reset	GPIO5	GND	GND
(SPI)BSY	GPIO6	Board Identifier 2	GPIO12
Board Identifier 1	GPIO13	GND	GND
I2S LRCLK	GPIO19	RST BUTTON	GPIO16
USER LED	GPIO26	NC	GPIO20
GND	GND	I2S SDATA	GPIO21

2.3 Digital IO

Robot HAT has 4 sets of 3Pin digital pins.

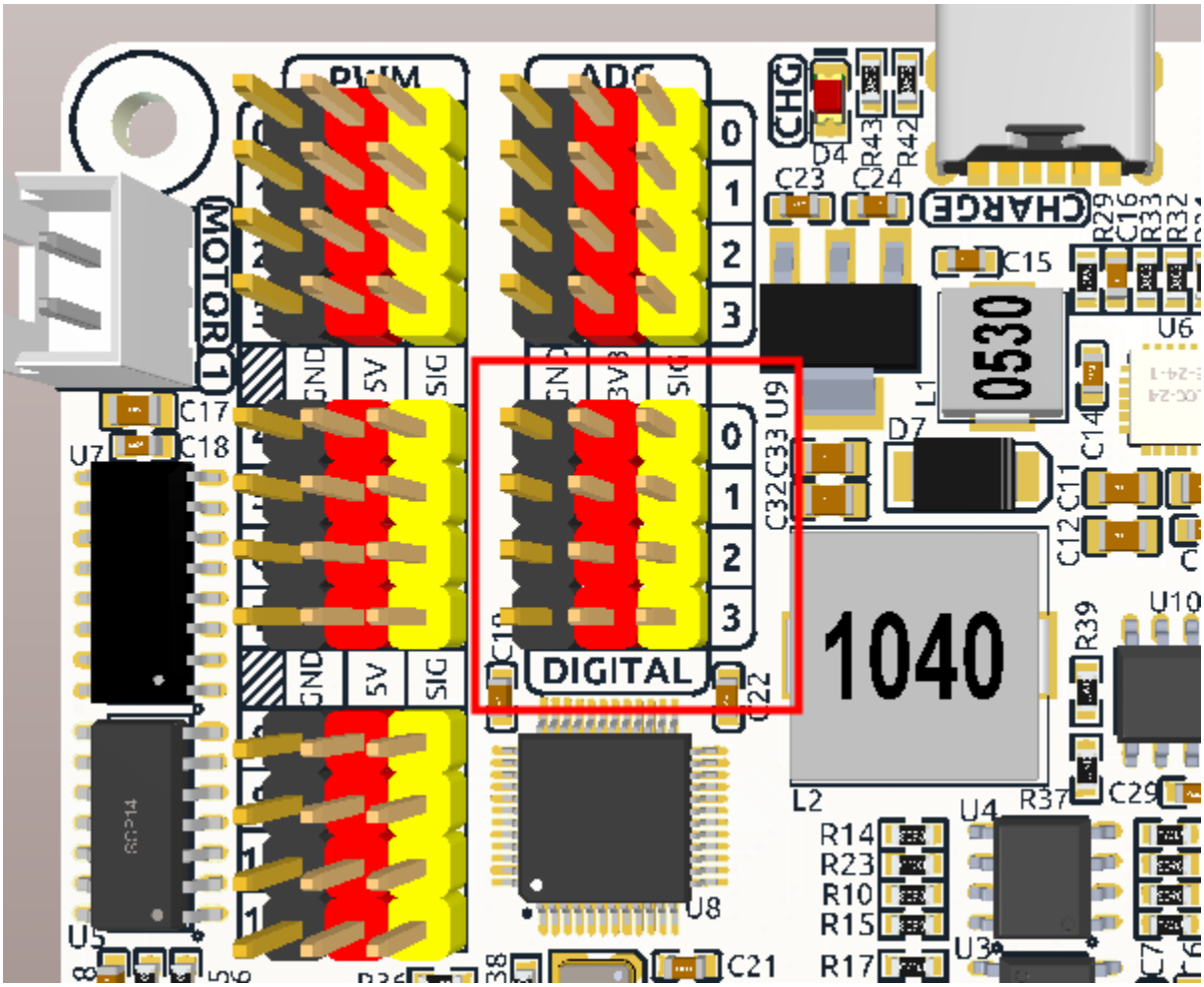
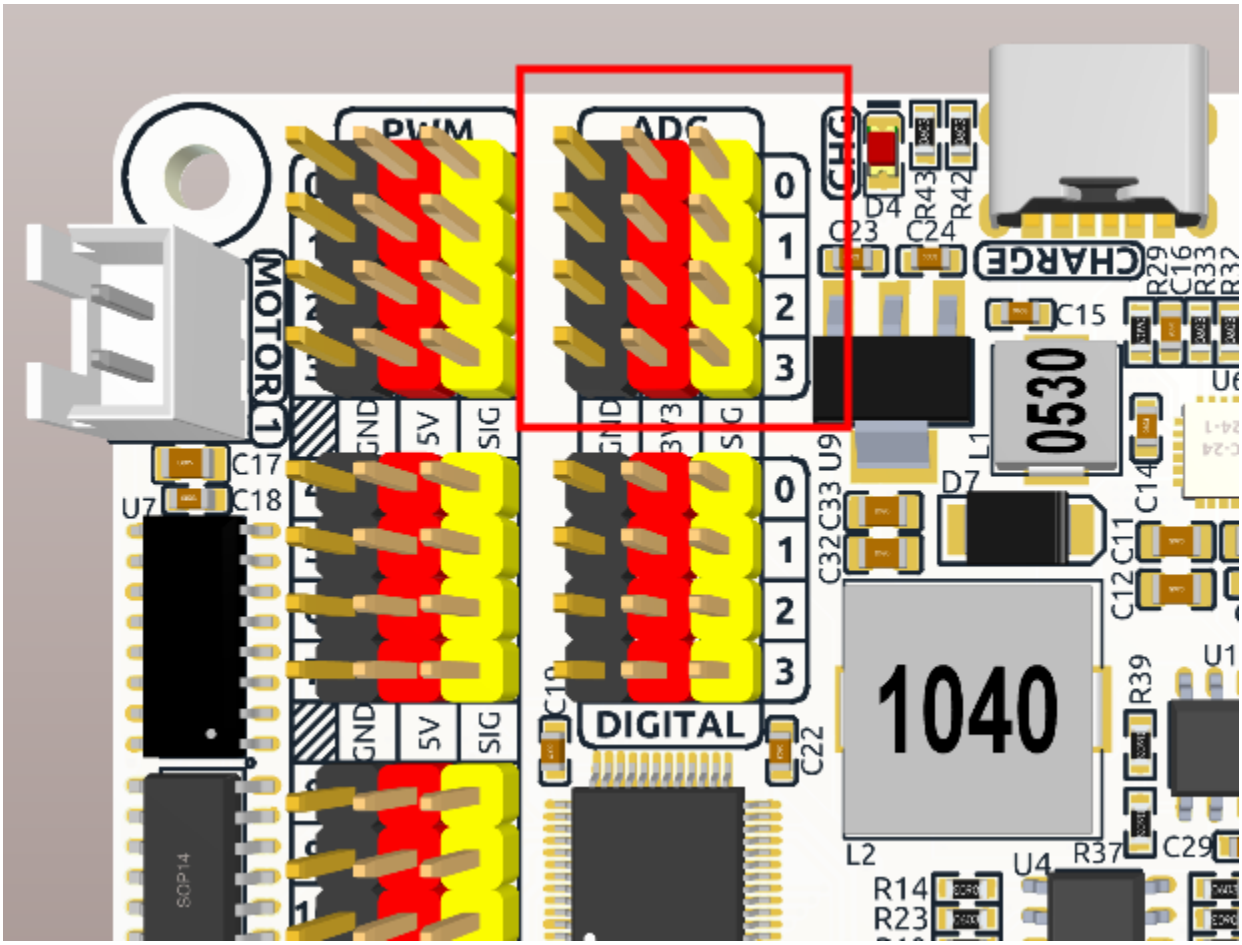


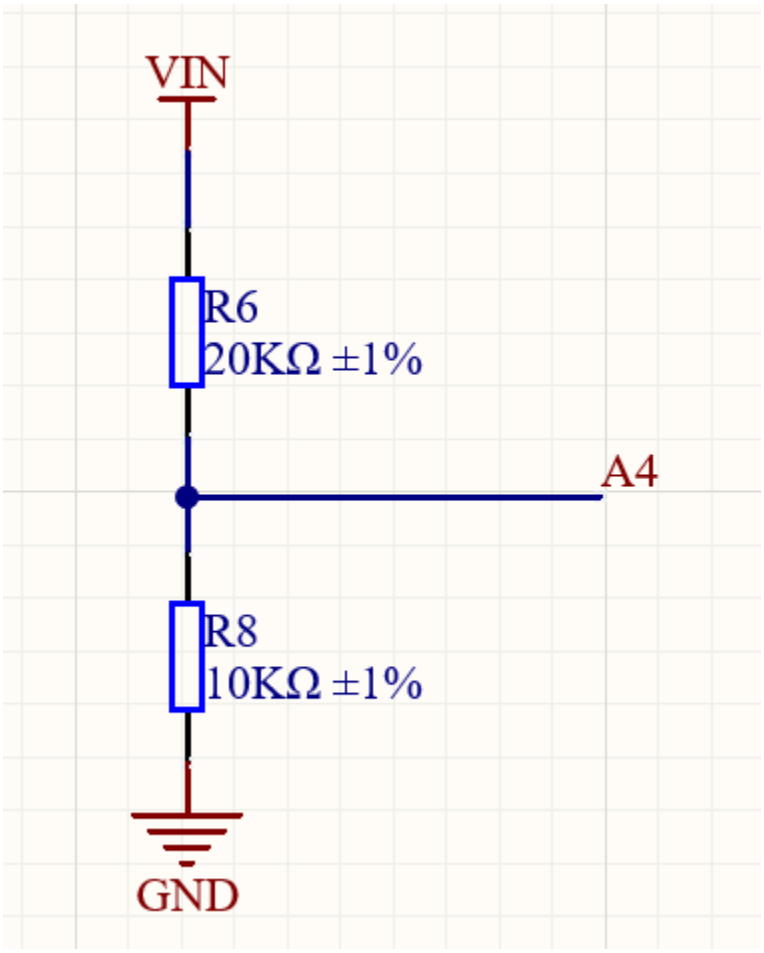
Table 2: Digital IO

Robot Hat V4	Raspberry Pi
D0	GPIO17
D1	GPIO4
D2	GPIO27
D3	GPIO22

2.4 ADC



The Robot HAT features four sets of 3Pin ADC (Analog to Digital Converter) pins, each spaced 2.54mm apart. These pins operate at a 3.3V power supply. The ADC function, offering 12-bit precision, is facilitated by an onboard microcontroller. Detailed instructions for reading ADC values are provided in the *On-Board MCU* section.

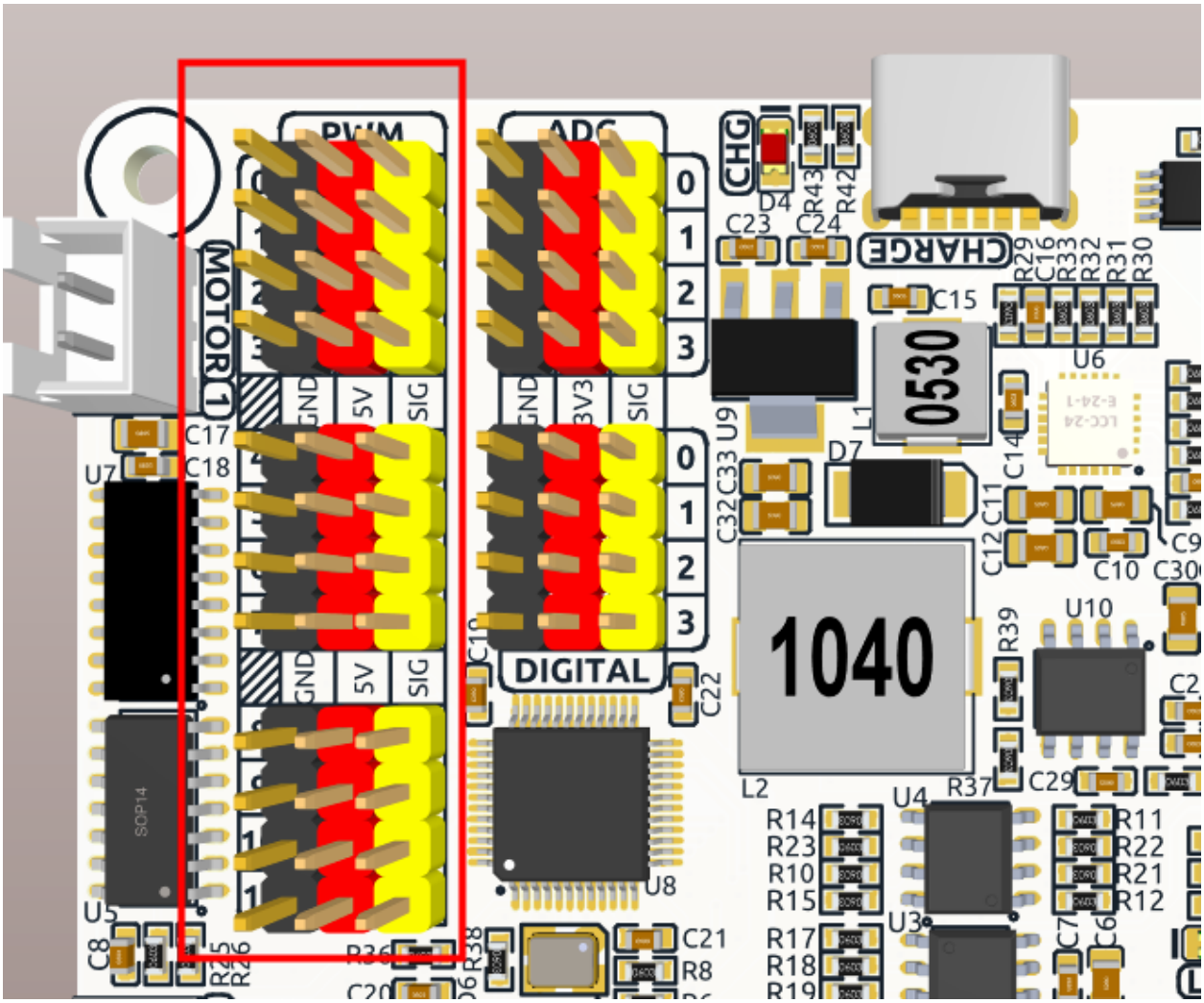


Also, ADC channel A4 is connected to the battery through a voltage divider using resistors, which will be used to measure the battery voltage to estimate the approximate battery charge.

The voltage divider ratio is 20K/10K, so:

- A4 voltage (Va4) = value_A4 / 4095.0 * 3.3
- Battery voltage (Vbat) = Va4*3
- Battery voltage (Vbat) = value_A4 / 4095.0 * 3.3 * 3

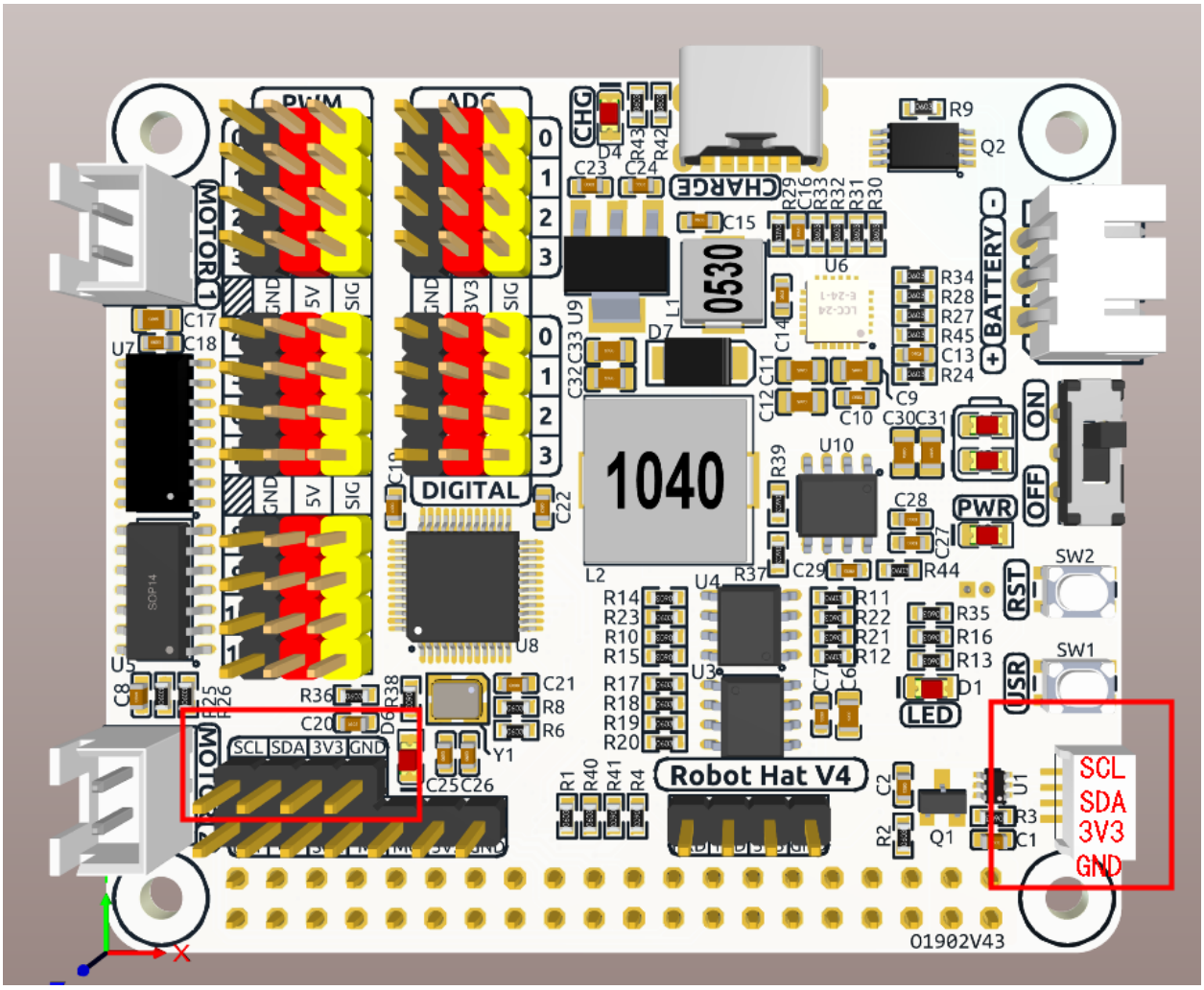
2.5 PWM



Robot HAT has 4 sets of 3Pin PWM pins, each spaced 2.54mm apart, and the power supply is 5V. The method of using the PWM is described in detail in *On-Board MCU*.

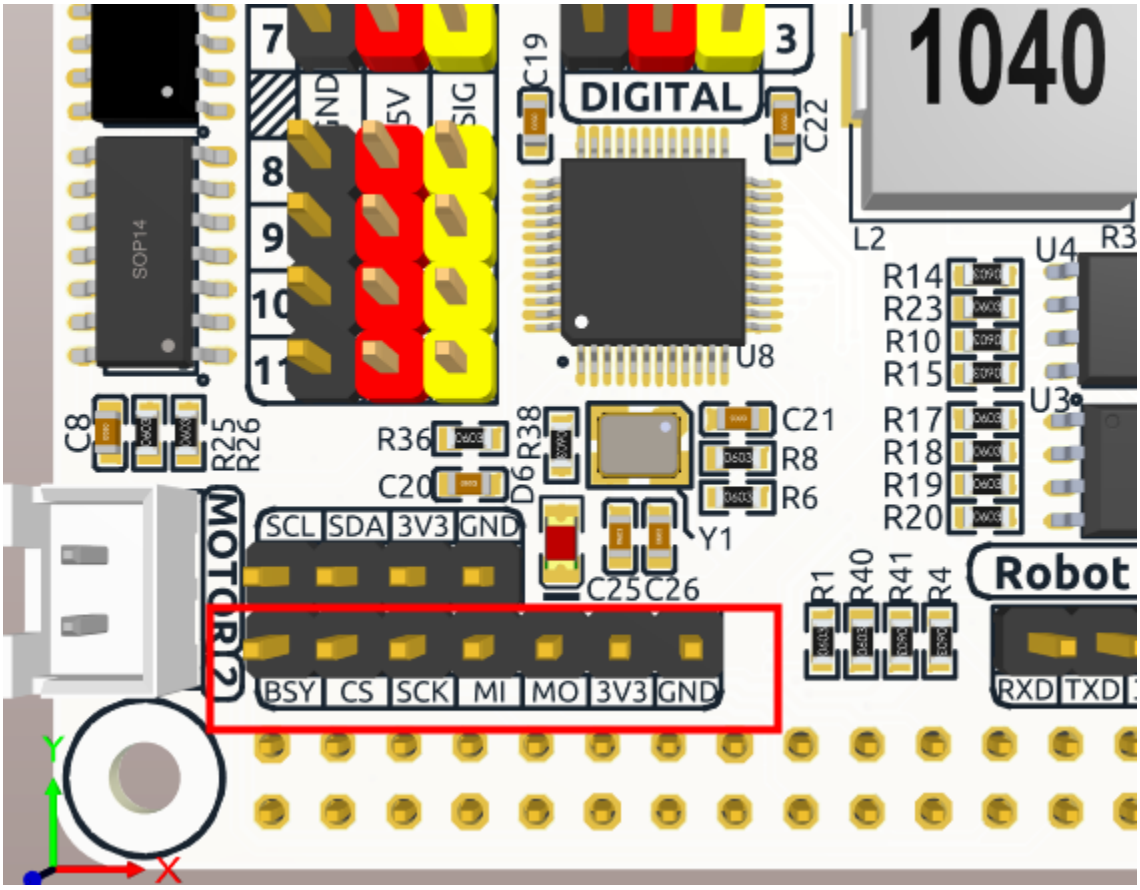
Note: PWM13 & 14 channels are used for motor drive.

2.6 I2C



The Robot HAT has two I2C interfaces. One is the P2.54 4-pin interface, and the other is the SH1.0 4-pin interface, which is compatible with QWIIC and STEMMA QT. These I2C interfaces are connected to the Raspberry Pi's I2C interface via GPIO2 (SDA) and GPIO3 (SCL). The board also features an *On-Board MCU*, and the two signal lines have 10K pull-up resistors.

2.7 SPI

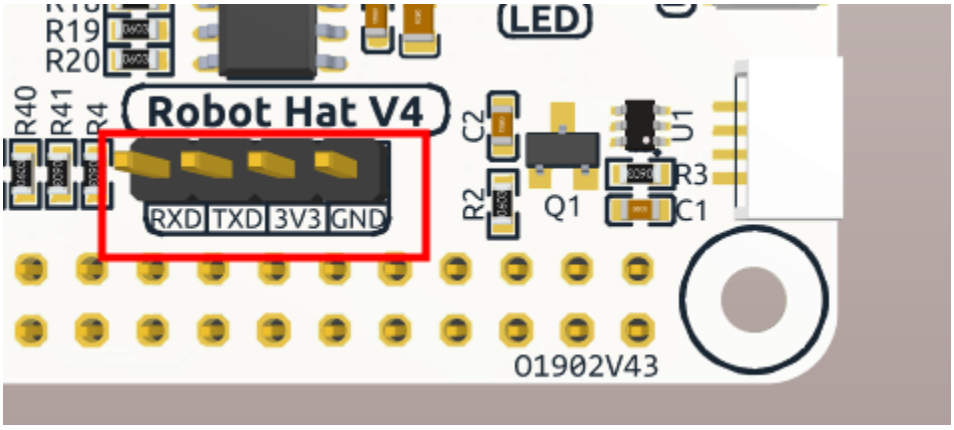


The SPI interface of the Robot HAT is a 7-pin P2.54 interface. It connects to the SPI interface of the Raspberry Pi and includes an additional I/O pin that can be used for purposes such as interrupts or resets.

Table 3: SPI

Robot Hat V4	Raspberry Pi
BSY	GPIO6
CS	CE0(GPIO8)
SCK	SCLK(GPIO11)
MI	MISO(GPIO9)
MO	MOSI(GPIO10)
3V3	3.3V Power
GND	Ground

2.8 UART



The UART interface of the Robot HAT is a 4-pin P2.54 interface. It connects to the Raspberry Pi's GPIO14 (TXD) and GPIO15 (RXD) pins.

2.9 Buttons

The Robot HAT comes with 1 LED and 2 buttons, all directly connected to the Raspberry Pi's GPIO pins. The RST button, when using Ezblock, serves as a button to restart the Ezblock program. If not using Ezblock, the RST button does not have a predefined function and can be fully customized according to your needs.

Table 4: LED & Button

Robot Hat V4	Raspberry Pi
LED	GPIO26
USR	GPIO25
RST	GPIO16

2.10 Speaker and Speaker Port

The Robot HAT is equipped with onboard I2S audio output, along with a 2030 audio chamber speaker, providing a mono sound output.

Table 5: I2S

I2S	Raspberry Pi
LRCLK	GPIO19
BCLK	GPIO18
SDATA	GPIO21

2.11 Motor Port

The motor driver of the Robot HAT supports 2 channels and can be controlled using 2 digital signals for direction and 2 PWM signals for speed control.

Table 6: Motor Driver

Motor	IO
Motor1 Dir	GPIO23
Motor1 Power	PWM13
Motor2 Dir	GPIO24
Motor2 Power	PWM12

2.12 Battery Level Indicator

The battery level indicator on the Robot HAT monitors the battery voltage using a voltage divider method and serves as a reference for estimating the battery level. The relationship between the LED and voltage is as follows:

Table 7: Battery Level

LED Battery	Total Voltage
2 LEDs on	Greater than 7.6V
1 LED on	Greater than 7.15V
Both LEDs off	Less than 7.15V

When any one of the batteries reaches or exceeds 4.1V while the others are below that threshold, the charging current of that specific battery will be reduced.

CHAPTER

THREE

ABOUT THE BATTERY

Battery

- **VCC**: Battery positive terminal, here there are two sets of VCC and GND is to increase the current and reduce the resistance.
- **Middle**: To balance the voltage between the two cells and thus protect the battery.
- **GND**: Negative battery terminal.

This is a custom battery pack made by SunFounder consisting of two 18650 batteries with a capacity of 2000mAh. The connector is PH2.0-3P, which can be charged directly after being inserted into the shield.

Features

- Battery charge: 5V/2A

- Battery output: 5V/5A
- Battery capacity: 3.7V 2000mAh x 2
- Battery life: 90min
- Battery charge time: 130min
- Connector: PH2.0, 3P

CHAPTER

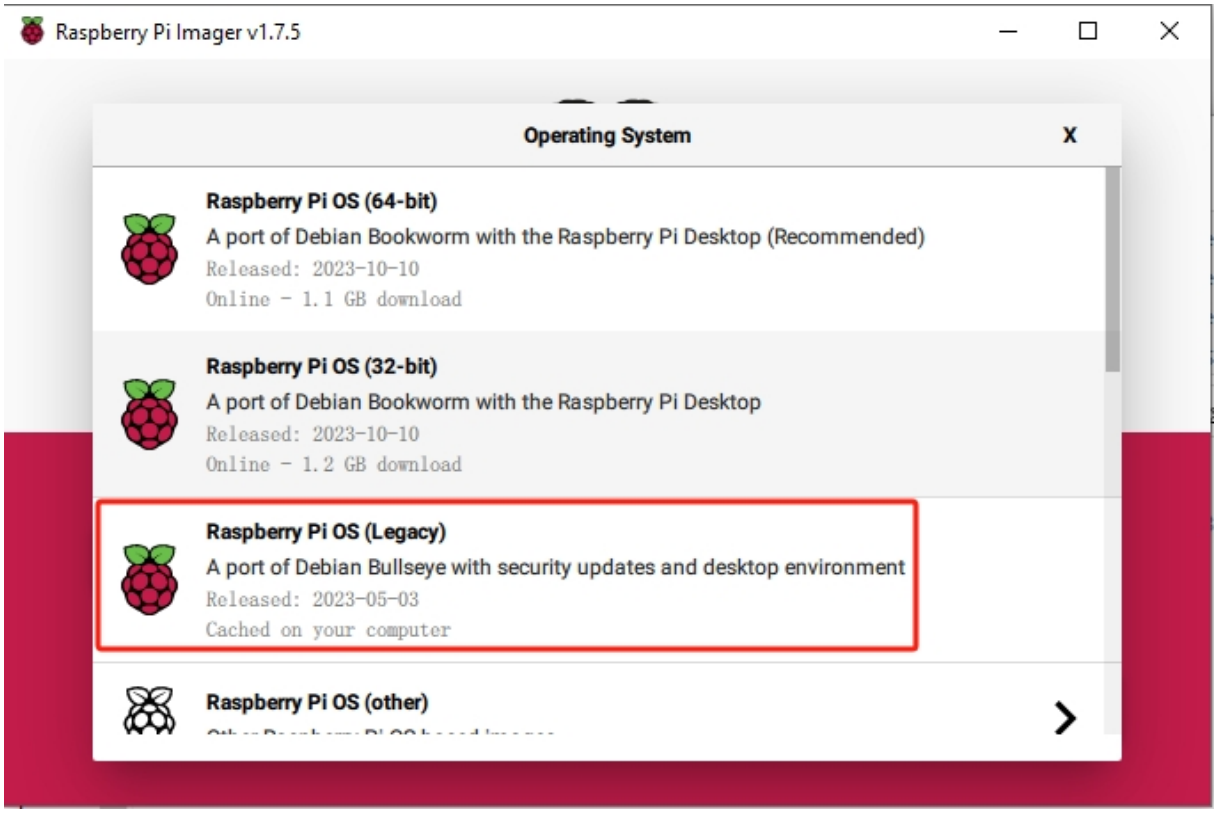
FOUR

INSTALL THE `ROBOT-HAT` MODULE

`robot-hat` is the supported library for the Robot HAT.

Warning:

- When installing the Raspberry Pi OS, please use the **Raspberry Pi OS (Legacy)** version - **Debian Bullseye**.
- If the version you install is **Bookworm**, the **Speaker** will not function properly.



1. Update your system.

   Make sure you are connected to the Internet and update your system:

   ```
   sudo apt update
   sudo apt upgrade
   ```

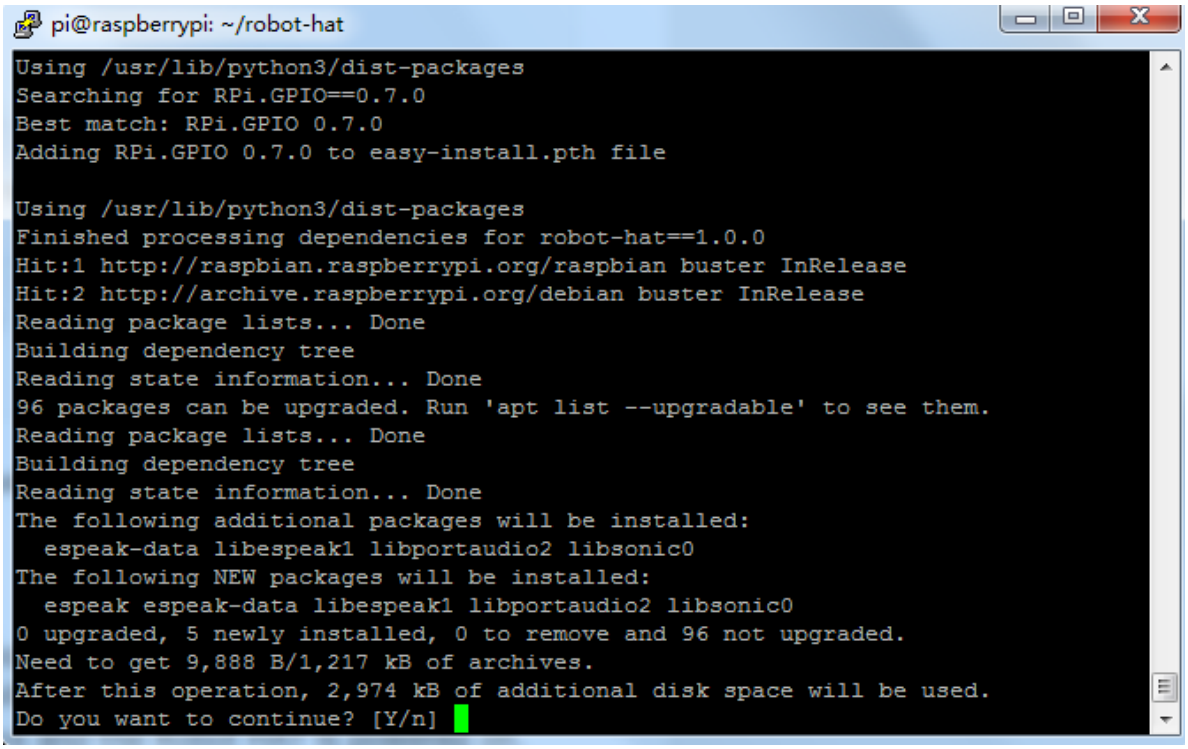
Note: Python3 related packages must be installed if you are installing the **Lite** version OS.

```
sudo apt install git python3-pip python3-setuptools python3-smbus
```

2. Type this command into the terminal to install the `robot-hat` package.

```
cd ~/
git clone -b v2.0 https://github.com/sunfounder/robot-hat.git
cd robot-hat
sudo python3 setup.py install
```

Note: Run `setup.py` to download some necessary components. You may have a network problem and the download may fail. At this point you may need to download again. In the following cases, type `Y` and press `Enter` to continue the process.



CHAPTER

FIVE

INSTALL I2SAMP.SH FOR THE SPEAKER

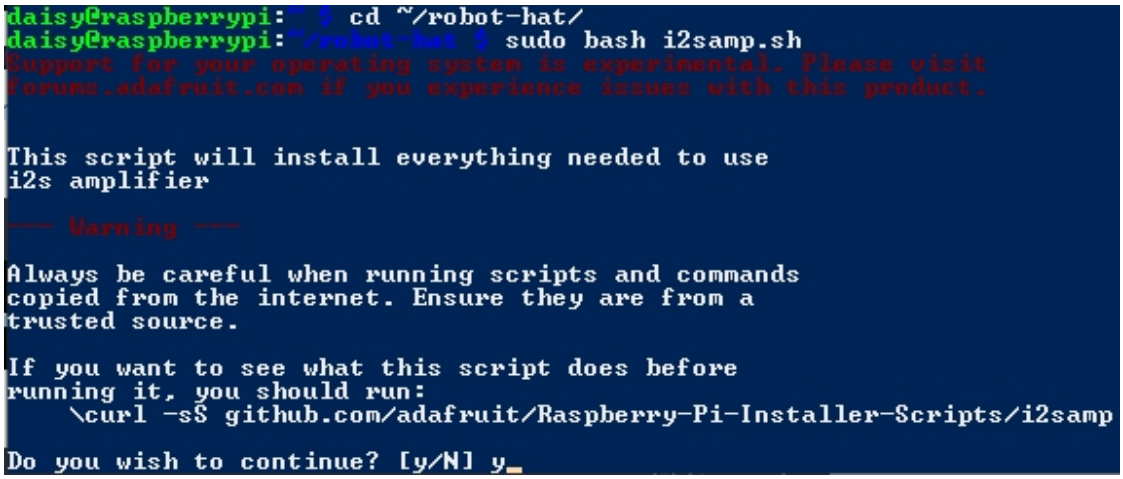
The i2samp.sh is a sophisticated Bash script specifically designed for setting up and configuring an I2S (Inter-IC Sound) amplifier on Raspberry Pi and similar devices. Licensed under the MIT license, it ensures compatibility with a range of hardware and operating systems, conducting thorough checks before proceeding with any installation or configuration.

If you want your speaker to work properly, you definitely need to install this script.

The steps are as follows:

```
cd ~/robot-hat
sudo bash i2samp.sh
```

Type y and press enter to continue running the script.



Type y and press enter to run /dev/zero in the background.

```
Do you wish to continue? [y/N] y

Checking hardware requirements...

Adding Device Tree Entry to /boot/config.txt
dtoverlay=hifiberry-dac
dtoverlay=i2s-mmap

Commenting out Blacklist entry in
/etc/modprobe.d/raspi-blacklist.conf

Disabling default sound driver
Configuring sound output

Installing aplay systemd unit

You can optionally activate '/dev/zero' playback in
the background at boot. This will remove all
popping/clicking but does use some processor time.

Activate '/dev/zero' playback in background? [RECOMMENDED] [y/N] y_
```

Type `y` and press `enter` to restart the Raspberry pi.


```
Do you wish to continue? [y/N] y

Checking hardware requirements...

Adding Device Tree Entry to /boot/config.txt
dtoverlay=hifiberry-dac
dtoverlay=i2s-mmap

Commenting out Blacklist entry in
/etc/modprobe.d/raspi-blacklist.conf

Disabling default sound driver
Configuring sound output

Installing aplay systemd unit

You can optionally activate '/dev/zero' playback in
the background at boot. This will remove all
popping/clicking but does use some processor time.

Activate '/dev/zero' playback in background? [RECOMMENDED] [y/N] y_
```


Warning: If there is no sound after restarting, you may need to run the `i2samp.sh` script several times.

CHAPTER

SIX

ON-BOARD MCU

The Robot HAT comes with an AT32F413CBT7 microcontroller from Artery. It is an ARM Cortex-M4 processor with a maximum clock frequency of 200MHz. The microcontroller has 128KB of Flash memory and 32KB of SRAM.

The onboard PWM and ADC are driven by the microcontroller. Communication between the Raspberry Pi and the microcontroller is established via the I2C interface. The I2C address used for communication is 0x14 (7-bit address format).

6.1 Introduce

The on board MCU RESET pin is connected to Raspberry Pi GPIO 5, or `MCURST` for *robot_hat.Pin*. The MCU using 7-bit address `0x14`.

6.2 ADC

Register addresses is 3 byte, 0x170000 to 0x140000 are ADC channels 0 to 3. The ADC precision is 12 bit, and the value is 0 to 4095. See more details in *robot_hat.ADC*.

Address	Description
`0x170000`	ADC channel 0
`0x160000`	ADC channel 1
`0x150000`	ADC channel 2
`0x140000`	ADC channel 3
`0x130000`	ADC channel 4 (Battery Level)

Example:

Read Channel 0 ADC value:

```
from smbus import SMBus
bus = SMBus(1)

# smbus only support 8 bit register address, so write 2 byte 0 first
bus.write_word_data(0x14, 0x17, 0)
msb = bus.read_byte(0x14)
lsb = bus.read_byte(0x14)
value = (msb << 8) | lsb
```

6.3 PWM

PWM have 1 byte register with 2 byte values.

6.3.1 Changing PWM Frequency

Frequency is defined with prescaler and period.

To set frequency first you need to define the period you want. Like on Arduino, normaly is 255, or like PCA9685 is 4095.

CPU clock is 72MHz, Then you can calculate the prescaler from your desire frequency

> prescaler = 72MHz / (Period + 1) / Frequency - 1

Or if you don't care about the period, there's a way to calculate both period and prescaler from frequency. See *robot_hat.PWM.freq()*.

6.3.2 Pulse width

To control the channel pulse width is rather simple, just write the value to the register.

But what is the value? If you want to set the PWM to 50% pulse width, you need to know exactly what the period is. Base on the above calculation, if you set the period to 4095, then set pulse value to 2048 is about 50% pulse width.

Address	Description
`0x20`	Set PWM channel 0 **On Value**
`0x21`	Set PWM channel 1 **On Value**
`0x22`	Set PWM channel 2 **On Value**
`0x23`	Set PWM channel 3 **On Value**
`0x24`	Set PWM channel 4 **On Value**
`0x25`	Set PWM channel 5 **On Value**
`0x26`	Set PWM channel 6 **On Value**
`0x27`	Set PWM channel 7 **On Value**
`0x28`	Set PWM channel 8 **On Value**
`0x29`	Set PWM channel 9 **On Value**
`0x2A`	Set PWM channel 10 **On Value**
`0x2B`	Set PWM channel 11 **On Value**
`0x2C`	Set Motor 2 speed **On Value**
`0x2D`	Set Motor 1 speed **On Value**

6.3.3 Prescaler

Register from 0x40 is to set the PWM prescaler. ranges from 0~65535. There are only 4 timers for all 14 channels. See *PWM Timer(IMPORTANT)*

Address	Description
`0x40`	Set timer 0 **Prescaler**
`0x41`	Set timer 1 **Prescaler**
`0x42`	Set timer 2 **Prescaler**
`0x43`	Set timer 3 **Prescaler**

6.3.4 Period

Register from 0x44 is to set the PWM period. ranges from 0~65535. There are only 4 timers for all 14 channels. See *PWM Timer(IMPORTANT)*

Address	Description
0x44	Set timer 0 **Period**
0x45	Set timer 1 **Period**
0x46	Set timer 2 **Period**
0x47	Set timer 3 **Period**

6.3.5 PWM Timer(IMPORTANT)

What is PWM Timer? PWM Timer is a tool to turn on and off the PWM channel for you.

The MCU only have 4 timers for PWM: which means you cannot set frequency on different channels at with the same timer.

Example: if you set frequency on channel 0, channel 1, 2, 3 will be affected. If you change channel 2 frequency, channel 0, 1, 3 will be override.

This happens like if you want to control both a passive buzzer (who changes frequency all the time) and servo (who needs a fix frequency of 50Hz). Then you should seperate them into two different timer.

Timer	PWM Channel
Timer 0	0, 1, 2, 3
Timer 1	4, 5, 6, 7
Timer 2	8, 9, 10, 11
Timer 3	12, 13(for motors)

6.3.6 Example

```
from smbus import SMBus
bus = SMBus(1)

# Set timer 0 period to 4095
bus.write_word_data(0x14, 0x44, 4095)
# Set frequency to 50Hz,
freq = 50
# Calculate prescaler
prescaler = int(72000000 / (4095 + 1) / freq) - 1
# Set prescaler
bus.write_word_data(0x14, 0x40, prescaler)

# Set channel 0 to 50% pulse width
bus.write_word_data(0x14, 0x20, 2048)
```

6.4 Reset MCU

Currently the firmware reads a fix 3 byte value, then it can return ADC values or control PWM. Thats why ADC register need 3byte with the latter 2 byte is 0.

And if your program is interrupted in the middle of the communication, the firmware may stuck and offset the data. Even we have timeout on waiting on 3 byte datas.

If so, you need to reset the MCU. To reset it. You can use the robot_hat command:

```
robot_hat reset_mcu
```

Or you can do it in your python code:

```
from robot_hat import reset_mcu
reset_mcu()
```

Or you can just pull down the reset pin (GPIO 5) for 10 ms, then pull it back up for another 10ms, as that's what `reset_mcu` dose.

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(5, GPIO.OUT)
GPIO.output(5, GPIO.LOW)
time.sleep(0.01)
GPIO.output(5, GPIO.HIGH)
time.sleep(0.01)
```

CHAPTER

SEVEN

REFERENCE

Robot Hat Library

7.1 class Pin

Example

```
# Import Pin class
from robot_hat import Pin

# Create Pin object with numeric pin numbering and default input pullup enabled
d0 = Pin(0, Pin.IN, Pin.PULL_UP)
# Create Pin object with named pin numbering
d1 = Pin('D1')

# read value
value0 = d0.value()
value1 = d1.value()
print(value0, value1)

# write value
d0.value(1) # force input to output
d1.value(0)

# set pin high/low
d0.high()
d1.off()

# set interrupt
led = Pin('LED', Pin.OUT)
switch = Pin('SW', Pin.IN, Pin.PULL_DOWN)
def onPressed(chn):
    led.value(not switch.value())
switch.irq(handler=onPressed, trigger=Pin.IRQ_RISING_FALLING)
```

API

class robot_hat.**Pin**(*pin*, *mode=None*, *pull=None*, **args*, ***kwargs*)

Bases: *_Basic_class*

Pin manipulation class

OUT = 1

Pin mode output

IN = 2

Pin mode input

PULL_UP = 17

Pin internal pull up

PULL_DOWN = 18

Pin internal pull down

PULL_NONE = None

Pin internal pull none

IRQ_FALLING = 33

Pin interrupt falling

IRQ_RISING = 34

Pin interrupt falling

IRQ_RISING_FALLING = 35

Pin interrupt both rising and falling

__init__(*pin*, *mode=None*, *pull=None*, **args*, ***kwargs*)

Initialize a pin

Parameters

- **pin** (*int/str*) – pin number of Raspberry Pi
- **mode** (*int*) – pin mode(IN/OUT)
- **pull** (*int*) – pin pull up/down(PUD_UP/PUD_DOWN/PUD_NONE)

setup(*mode*, *pull=None*)

Setup the pin

Parameters

- **mode** (*int*) – pin mode(IN/OUT)
- **pull** (*int*) – pin pull up/down(PUD_UP/PUD_DOWN/PUD_NONE)

dict(*_dict=None*)

Set/get the pin dictionary

Parameters

_dict (*dict*) – pin dictionary, leave it empty to get the dictionary

Returns

pin dictionary

Return type

dict

__call__(*value*)

Set/get the pin value

Parameters

value (*int*) – pin value, leave it empty to get the value(0/1)

Returns
pin value(0/1)

Return type
int

`value`(*value: bool = None*)

Set/get the pin value

Parameters
value (*int*) – pin value, leave it empty to get the value(0/1)

Returns
pin value(0/1)

Return type
int

`on`()

Set pin on(high)

Returns
pin value(1)

Return type
int

`off`()

Set pin off(low)

Returns
pin value(0)

Return type
int

`high`()

Set pin high(1)

Returns
pin value(1)

Return type
int

`low`()

Set pin low(0)

Returns
pin value(0)

Return type
int

`irq`(*handler*, *trigger*, *bouncetime=200*, *pull=None*)

Set the pin interrupt

Parameters

- **handler** (*function*) – interrupt handler callback function
- **trigger** (*int*) – interrupt trigger(RISING, FALLING, RISING_FALLING)
- **bouncetime** (*int*) – interrupt bouncetime in miliseconds

name()

Get the pin name

Returns
pin name

Return type
str

7.2 class ADC

Example

```
# Import ADC class
from robot_hat import ADC

# Create ADC object with numeric pin numbering
a0 = ADC(0)
# Create ADC object with named pin numbering
a1 = ADC('A1')

# Read ADC value
value0 = a0.read()
value1 = a1.read()
voltage0 = a0.read_voltage()
voltage1 = a1.read_voltage()
print(f"ADC 0 value: {value0}")
print(f"ADC 1 value: {value1}")
print(f"ADC 0 voltage: {voltage0}")
print(f"ADC 1 voltage: {voltage1}")
```

API

class robot_hat.**ADC**(*chn*, *address=None*, **args*, ***kwargs*)

Bases: *I2C*

Analog to digital converter

__init__(*chn*, *address=None*, **args*, ***kwargs*)

Analog to digital converter

Parameters
chn (*int/str*) – channel number (0-7/A0-A7)

read()

Read the ADC value

Returns
ADC value(0-4095)

Return type
int

read_voltage()

Read the ADC value and convert to voltage

Returns
Voltage value(0-3.3(V))

Return type
float

7.3 class PWM

Example

```
# Import PWM class
from robot_hat import PWM

# Create PWM object with numeric pin numbering and default input pullup enabled
p0 = PWM(0)
# Create PWM object with named pin numbering
p1 = PWM('P1')


# Set frequency will automatically set prescaller and period
# This is easy for device like Buzzer or LED, which you care
# about the frequency and pulse width percentage.
# this usually use with pulse_width_percent function.
# Set frequency to 1000Hz
p0.freq(1000)
print(f"Frequence: {p0.freq()} Hz")
print(f"Prescaler: {p0.prescaler()}")
print(f"Period: {p0.period()}")
# Set pulse width to 50%
p0.pulse_width_percent(50)

# Or set prescaller and period, will get a frequency from:
# frequency = PWM.CLOCK / prescaler / period
# With this setup you can tune the period as you wish.
# set prescaler to 64
p1.prescaler(64)
# set period to 4096 ticks
p1.period(4096)
print(f"Frequence: {p1.freq()} Hz")
print(f"Prescaler: {p1.prescaler()}")
print(f"Period: {p1.period()}")
# Set pulse width to 2048 which is also 50%
p1.pulse_width(2048)
```

API

class robot_hat.**PWM**(*channel*, *address=None*, **args*, ***kwargs*)

Bases: *I2C*

Pulse width modulation (PWM)

REG_CHN = 32

Channel register prefix

REG_PSC = 64

Prescaler register prefix

REG_ARR = 68

Period registor prefix

CLOCK = 72000000.0

Clock frequency

__init__(*channel*, *address=None*, **args*, ***kwargs*)

Initialize PWM

Parameters
channel (*int/str*) – PWM channel number(0-13/P0-P13)

freq(*freq=None*)

Set/get frequency, leave blank to get frequency

Parameters
freq (*float*) – frequency(0-65535)(Hz)

Returns
frequency

Return type
float

prescaler(*prescaler=None*)

Set/get prescaler, leave blank to get prescaler

Parameters
prescaler (*int*) – prescaler(0-65535)

Returns
prescaler

Return type
int

period(*arr=None*)

Set/get period, leave blank to get period

Parameters
arr (*int*) – period(0-65535)

Returns
period

Return type
int

pulse_width(*pulse_width=None*)

Set/get pulse width, leave blank to get pulse width

Parameters
pulse_width (*float*) – pulse width(0-65535)

Returns
pulse width

Return type
float

pulse_width_percent(*pulse_width_percent=None*)

Set/get pulse width percentage, leave blank to get pulse width percentage

Parameters
pulse_width_percent (*float*) – pulse width percentage(0-100)

Returns
pulse width percentage

Return type
float

7.4 class Servo

Example

```
# Import Servo class
from robot_hat import Servo

# Create Servo object with PWM object
servo0 = Servo("P0")

# Set servo to position 0, here 0 is the center position,
# angle ranges from -90 to 90
servo0.angle(0)

# Sweep servo from 0 to 90 degrees, then 90 to -90 degrees, finally back to 0
import time
for i in range(0, 91):
    servo0.angle(i)
    time.sleep(0.05)
for i in range(90, -91, -1):
    servo0.angle(i)
    time.sleep(0.05)
for i in range(-90, 1):
    servo0.angle(i)
    time.sleep(0.05)


# Servos are all controls with pulse width, some
# from 500 ~ 2500 like most from SunFounder.
# You can directly set the pulse width

# Set servo to 1500 pulse width (-90 degree)
servo0.pulse_width_time(500)
# Set servo to 1500 pulse width (0 degree)
servo0.pulse_width_time(1500)
# Set servo to 1500 pulse width (90 degree)
servo0.pulse_width_time(2500)
```

API

class robot_hat.**Servo**(*channel*, *address=None*, **args*, ***kwargs*)

Bases: *PWM*

Servo motor class

__init__(*channel*, *address=None*, **args*, ***kwargs*)

Initialize the servo motor class

Parameters
channel (*int/str*) – PWM channel number(0-14/P0-P14)

angle(*angle*)

Set the angle of the servo motor

Parameters
angle (*float*) – angle(-90~90)

pulse_width_time(*pulse_width_time*)

Set the pulse width of the servo motor

Parameters
pulse_width_time (*float*) – pulse width time(500~2500)

7.5 module motor

7.5.1 class Motors

Example

Initilize

```
# Import Motor class
from robot_hat import Motors

# Create Motor object
motors = Motors()
```

Directly control a motor. Motor 1/2 is according to PCB mark

```
# Motor 1 clockwise at 100% speed
motors[1].speed(100)
# Motor 2 counter-clockwise at 100% speed
motors[2].speed(-100)
# Stop all motors
motors.stop()
```

Setup for high level control, high level control provides functions from simple forword, backward, left, right, stop to more complex like joystick control, motor directions calibration, etc.

Note: All these setup only need to run once, and will save in a config file. Next time you load Motors class, it will load from config file.

```
# Setup left and right motors
motors.set_left_id(1)
motors.set_right_id(2)
# Go forward and see if both motor directions are correct
```

(continues on next page)

(continued from previous page)

```
motors.forward(100)
# if you found a motor is running in the wrong direction
# Use these function to correct it
motors.set_left_reverse()
motors.set_right_reverse()
# Run forward again and see if both motor directions are correct
motors.forward(100)
```

Now control the robot

```
import time

motors.forward(100)
time.sleep(1)
motors.backward(100)
time.sleep(1)
motors.turn_left(100)
time.sleep(1)
motors.turn_right(100)
time.sleep(1)
motors.stop()
```

API

class robot_hat.**Motors**(*db='/root/.config/robot-hat/robot-hat.conf'*, **args*, ***kwargs*)

Bases: *_Basic_class*

__init__(*db='/root/.config/robot-hat/robot-hat.conf'*, **args*, ***kwargs*)

Initialize motors with robot_hat.motor.Motor

Parameters

db (*str*) – config file path

__getitem__(*key*)

Get specific motor

stop()

Stop all motors

property left

left motor

property right

right motor

set_left_id(*id*)

Set left motor id, this function only need to run once It will save the motor id to config file, and load the motor id when the class is initialized

Parameters

id (*int*) – motor id (1 or 2)

set_right_id(*id*)

Set right motor id, this function only need to run once It will save the motor id to config file, and load the motor id when the class is initialized

Parameters
id (*int*) – motor id (1 or 2)

`set_left_reverse()`

Set left motor reverse, this function only need to run once It will save the reversed status to config file, and load the reversed status when the class is initialized

Returns
if currently is reversed

Return type
bool

`set_right_reverse()`

Set right motor reverse, this function only need to run once It will save the reversed status to config file, and load the reversed status when the class is initialized

Returns
if currently is reversed

Return type
bool

`speed(left_speed, right_speed)`

Set motor speed

Parameters

- **left_speed** (*float*) – left motor speed(-100.0~100.0)
- **right_speed** (*float*) – right motor speed(-100.0~100.0)

`forward(speed)`

Forward

Parameters
speed (*float*) – Motor speed(-100.0~100.0)

`backward(speed)`

Backward

Parameters
speed (*float*) – Motor speed(-100.0~100.0)

`turn_left(speed)`

Left turn

Parameters
speed (*float*) – Motor speed(-100.0~100.0)

`turn_right(speed)`

Right turn

Parameters
speed (*float*) – Motor speed(-100.0~100.0)

7.5.2 class Motor

Example

```
# Import Motor class
from robot_hat import Motor, PWM, Pin

# Create Motor object
motor = Motor(PWM("P13"), Pin("D4"))

# Motor clockwise at 100% speed
motor.speed(100)
# Motor counter-clockwise at 100% speed
motor.speed(-100)

# If you like to reverse the motor direction
motor.set_is_reverse(True)
```

API

class robot_hat.**Motor**(*pwm*, *dir*, *is_reversed=False*)

Bases: object

__init__(*pwm*, *dir*, *is_reversed=False*)

Initialize a motor

Parameters

- **pwm** (robot_hat.pwm.PWM) – Motor speed control pwm pin
- **dir** (robot_hat.pin.Pin) – Motor direction control pin

speed(*speed=None*)

Get or set motor speed

Parameters

speed (*float*) – Motor speed(-100.0~100.0)

set_is_reverse(*is_reverse*)

Set motor is reversed or not

Parameters

is_reverse (*bool*) – True or False

7.6 module modules

7.6.1 class Ultrasonic

Example

```
# Import Ultrasonic and Pin class
from robot_hat import Ultrasonic, Pin

# Create Motor object
us = Ultrasonic(Pin("D2"), Pin("D3"))
```

(continues on next page)

(continued from previous page)

```
# Read distance
distance = us.read()
print(f"Distance: {distance}cm")
```

API

class robot_hat.modules.**Ultrasonic**(*trig*, *echo*, *timeout=0.02*)

__init__(*trig*, *echo*, *timeout=0.02*)

7.6.2 class ADXL345

Example

```
# Import ADXL345 class
from robot_hat import ADXL345

# Create ADXL345 object
adxl = ADXL345()
# or with a custom I2C address
adxl = ADXL345(address=0x53)

# Read acceleration of each axis
x = adxl.read(adxl.X)
y = adxl.read(adxl.Y)
z = adxl.read(adxl.Z)
print(f"Acceleration: {x}, {y}, {z}")

# Or read all axis at once
x, y, z = adxl.read()
print(f"Acceleration: {x}, {y}, {z}")
# Or print all axis at once
print(f"Acceleration: {adxl.read()}")
```

API

class robot_hat.**ADXL345**(**args*, *address: int = 83*, *bus: int = 1*, ***kwargs*)

Bases: *I2C*

ADXL345 modules

X = 0

X

Y = 1

Y

Z = 2

Z

__init__(**args*, *address: int = 83*, *bus: int = 1*, ***kwargs*)

Initialize ADXL345

Parameters

address (*int*) – address of the ADXL345

read(*axis: int = None*) → Union[float, List[float]]

Read an axis from ADXL345

Parameters

axis (*int*) – read value(g) of an axis, ADXL345.X, ADXL345.Y or ADXL345.Z, None for all axis

Returns

value of the axis, or list of all axis

Return type

float/list

7.6.3 class RGB_LED

Example

```
# Import RGB_LED and PWM class
from robot_hat import RGB_LED, PWM

# Create RGB_LED object for common anode RGB LED
rgb = RGB_LED(PWM(0), PWM(1), PWM(2), common=RGB_LED.ANODE)
# or for common cathode RGB LED
rgb = RGB_LED(PWM(0), PWM(1), PWM(2), common=RGB_LED.CATHODE)

# Set color with 24 bit int
rgb.color(0xFF0000) # Red
# Set color with RGB tuple
rgb.color((0, 255, 0)) # Green
# Set color with RGB List
rgb.color([0, 0, 255]) # Blue
# Set color with RGB hex string starts with "#"
rgb.color("#FFFF00") # Yellow
```

API

class robot_hat.**RGB_LED**(*r_pin: PWM*, *g_pin: PWM*, *b_pin: PWM*, *common: int = 1*)

Simple 3 pin RGB LED

ANODE = 1

Common anode

CATHODE = 0

Common cathode

__init__(*r_pin: PWM*, *g_pin: PWM*, *b_pin: PWM*, *common: int = 1*)

Initialize RGB LED

Parameters

- **r_pin** (robot_hat.PWM) – PWM object for red
- **g_pin** (robot_hat.PWM) – PWM object for green
- **b_pin** (robot_hat.PWM) – PWM object for blue
- **common** (*int*) – RGB_LED.ANODE or RGB_LED.CATHODE, default is ANODE

Raises

- **ValueError** – if common is not ANODE or CATHODE
- **TypeError** – if r_pin, g_pin or b_pin is not PWM object

color(*color: Union[str, Tuple[int, int, int], List[int], int]*)

Write color to RGB LED

Parameters

color (*str/int/tuple/list*) – color to write, hex string starts with "#", 24-bit int or tuple of (red, green, blue)

7.6.4 class Buzzer

Example

Imports and initialize

```
# Import Buzzer class
from robot_hat import Buzzer
# Import Pin for active buzzer
from robot_hat import Pin
# Import PWM for passive buzzer
from robot_hat import PWM
# import Music class for tones
from robot_hat import Music
# Import time for sleep
import time

music = Music()
# Create Buzzer object for passive buzzer
p_buzzer = Buzzer(PWM(0))
# Create Buzzer object for active buzzer
a_buzzer = Buzzer(Pin("D0"))
```

Active buzzer beeping

```
while True:
    a_buzzer.on()
    time.sleep(0.5)
    a_buzzer.off()
    time.sleep(0.5)
```

Passive buzzer Simple usage

```
# Play a Tone for 1 second
p_buzzer.play(music.note("C3"), duration=1)
# take adventage of the music beat as duration
# set song tempo of the beat value
music.tempo(120, 1/4)
# Play note with a quarter beat
p_buzzer.play(music.note("C3"), music.beat(1/4))
```

Passive buzzer Manual control

```
# Play a tone
p_buzzer.play(music.note("C4"))
# Pause for 1 second
time.sleep(1)
```

(continues on next page)

(continued from previous page)

```
# Play another tone
p_buzzer.play(music.note("C5"))
# Pause for 1 second
time.sleep(1)
# Stop playing
p_buzzer.off()
```

Play a song! Baby shark!

```
music.tempo(120, 1/4)

# Make a Shark-doo-doo function as is all about it
def shark_doo_doo():
    p_buzzer.play(music.note("C5"), music.beat(1/8))
    p_buzzer.play(music.note("C5"), music.beat(1/8))
    p_buzzer.play(music.note("C5"), music.beat(1/8))
    p_buzzer.play(music.note("C5"), music.beat(1/16))
    p_buzzer.play(music.note("C5"), music.beat(1/16 + 1/16))
    p_buzzer.play(music.note("C5"), music.beat(1/16))
    p_buzzer.play(music.note("C5"), music.beat(1/8))

# loop any times you want from baby to maybe great great great grandpa!
for _ in range(3):
    print("Measure 1")
    p_buzzer.play(music.note("G4"), music.beat(1/4))
    p_buzzer.play(music.note("A4"), music.beat(1/4))
    print("Measure 2")
    shark_doo_doo()
    p_buzzer.play(music.note("G4"), music.beat(1/8))
    p_buzzer.play(music.note("A4"), music.beat(1/8))
    print("Measure 3")
    shark_doo_doo()
    p_buzzer.play(music.note("G4"), music.beat(1/8))
    p_buzzer.play(music.note("A4"), music.beat(1/8))
    print("Measure 4")
    shark_doo_doo()
    p_buzzer.play(music.note("C5"), music.beat(1/8))
    p_buzzer.play(music.note("C5"), music.beat(1/8))
    print("Measure 5")
    p_buzzer.play(music.note("B4"), music.beat(1/4))
    time.sleep(music.beat(1/4))
```

API

class robot_hat.**Buzzer**(*buzzer: Union[*PWM, Pin*]*)

__init__(*buzzer: Union[*PWM, Pin*]*)

Initialize buzzer

Parameters

pwm (*robot_hat.PWM/robot_hat.Pin*) – PWM object for passive buzzer or Pin object for active buzzer

on()

Turn on buzzer

off()

Turn off buzzer

freq(*freq: float*)

Set frequency of passive buzzer

Parameters

freq (*int/float*) – frequency of buzzer, use Music.NOTES to get frequency of note

Raises

TypeError – if set to active buzzer

play(*freq: float*, *duration: float = None*)

Play freq

Parameters

- **freq** (*float*) – freq to play, you can use Music.note() to get frequency of note
- **duration** (*float*) – duration of each note, in seconds, None means play continuously

Raises

TypeError – if set to active buzzer

7.6.5 class Grayscale_Module

Example

```
# Import Grayscale_Module and ADC class
from robot_hat import Grayscale_Module, ADC

# Create Grayscale_Module object, reference should be calculate from the value␣
↪reads on white
# and black ground, then take the middle as reference
gs = Grayscale_Module(ADC(0), ADC(1), ADC(2), reference=2000)

# Read Grayscale_Module datas
datas = gs.read()
print(f"Grayscale Module datas: {datas}")
# or read a specific channel
l = gs.read(gs.LEFT)
m = gs.read(gs.MIDDLE)
r = gs.read(gs.RIGHT)
print(f"Grayscale Module left channel: {l}")
print(f"Grayscale Module middle channel: {m}")
print(f"Grayscale Module right channel: {r}")

# Read Grayscale_Module simple states
state = gs.read_status()
print(f"Grayscale_Module state: {state}")
```

API

class robot_hat.**Grayscale_Module**(*pin0:* ADC, *pin1:* ADC, *pin2:* ADC, *reference: int = None*)

3 channel Grayscale Module

LEFT = 0

Left Channel

MIDDLE = 1

Middle Channel

RIGHT = 2

Right Channel

__init__(*pin0: ADC*, *pin1: ADC*, *pin2: ADC*, *reference: int = None*)

Initialize Grayscale Module

Parameters

- **pin0** (*robot_hat.ADC/int*) – ADC object or int for channel 0
- **pin1** (*robot_hat.ADC/int*) – ADC object or int for channel 1
- **pin2** (*robot_hat.ADC/int*) – ADC object or int for channel 2
- **reference** (*1*3 list, [int, int, int]*) – reference voltage

reference(*ref: list = None*) → list

Get Set reference value

Parameters

ref (*list*) – reference value, None to get reference value

Returns

reference value

Return type

list

read_status(*datas: list = None*) → list

Read line status

Parameters

datas (*list*) – list of grayscale datas, if None, read from sensor

Returns

list of line status, 0 for white, 1 for black

Return type

list

read(*channel: int = None*) → list

read a channel or all datas

Parameters

channel (*int/None*) – channel to read, leave empty to read all. 0, 1, 2 or Grayscale_Module.LEFT, Grayscale_Module.CENTER, Grayscale_Module.RIGHT

Returns

list of grayscale data

Return type

list

7.7 class Robot

Example

```
# Import Robot class
from robot import Robot

# Create a robot(PiSloth)
robot = Robot(pin_list=[0, 1, 2, 3], name="pisloth")

robot.move_list["forward"] = [
```

(continues on next page)

(continued from previous page)

```
    [0, 40, 0, 15],
    [-30, 40, -30, 15],
    [-30, 0, -30, 0],

    [0, -15, 0, -40],
    [30, -15, 30, -40],
    [30, 0, 30, 0],
    ]

robot.do_action("forward", step=3, speed=90)
```

API

class robot_hat.**Robot**(*pin_list*, *db='/root/.config/robot-hat/robot-hat.conf'*, *name=None*, *init_angles=None*, *init_order=None*, ***kwargs*)

Bases: *_Basic_class*

Robot class

This class is for makeing a servo robot with Robot HAT

There are servo initialization, all servo move in specific speed. servo offset and stuff. make it easy to make a robot. All Pi-series robot from SunFounder use this class. Check them out for more details.

PiSloth: https://github.com/sunfounder/pisloth

PiArm: https://github.com/sunfounder/piarm

PiCrawler: https://github.com/sunfounder/picrawler

move_list = {}

Preset actions

max_dps = 428

Servo max Degree Per Second

__init__(*pin_list*, *db='/root/.config/robot-hat/robot-hat.conf'*, *name=None*, *init_angles=None*, *init_order=None*, ***kwargs*)

Initialize the robot class

Parameters

- **pin_list** (*list*) – list of pin number[0-11]
- **db** (*str*) – config file path
- **name** (*str*) – robot name
- **init_angles** (*list*) – list of initial angles
- **init_order** (*list*) – list of initialization order(Servos will init one by one in case of sudden huge current, pulling down the power supply voltage. default order is the pin list. in some cases, you need different order, use this parameter to set it.)

new_list(*default_value*)

Create a list of servo angles with default value

Parameters

default_value (*int or float*) – default value of servo angles

Returns
list of servo angles

Return type
list

`servo_write_raw`(*angle_list*)

Set servo angles to specific raw angles

Parameters
angle_list (*list*) – list of servo angles

`servo_write_all`(*angles*)

Set servo angles to specific angles with original angle and offset

Parameters
angles (*list*) – list of servo angles

`servo_move`(*targets*, *speed=50*, *bpm=None*)

Move servo to specific angles with speed or bpm

Parameters

- **targets** (*list*) – list of servo angles
- **speed** (*int or float*) – speed of servo move
- **bpm** (*int or float*) – beats per minute

`do_action`(*motion_name*, *step=1*, *speed=50*)

Do prefix action with motion_name and step and speed

Parameters

- **motion_name** (*str*) – motion
- **step** (*int*) – step of motion
- **speed** (*int or float*) – speed of motion

`set_offset`(*offset_list*)

Set offset of servo angles

Parameters
offset_list (*list*) – list of servo angles

`calibration`()

Move all servos to home position

`reset`()

Reset servo to original position

7.8 class Music

Warning:

- You need to add sudo when running this script, in case the speaker doesn't work.
- *Q3: Why is there no sound from the speaker?*.

Example

Initialize

```
# Import Music class
from robot_hat import Music

# Create a new Music object
music = Music()
```

Play tones

```
# You can directly play a frequency for specific duration in seconds
music.play_tone_for(400, 1)

# Or use note to get the frequency
music.play_tone_for(music.note("Middle C"), 0.5)
# and set tempo and use beat to get the duration in seconds
# Which make's it easy to code a song according to a sheet!
music.tempo(120)
music.play_tone_for(music.note("Middle C"), music.beat(1))

# Here's an example playing Greensleeves
set_volume(80)
music.tempo(60, 1/4)

print("Measure 1")
music.play_tone_for(music.note("G4"), music.beat(1/8))
print("Measure 2")
music.play_tone_for(music.note("A#4"), music.beat(1/4))
music.play_tone_for(music.note("C5"), music.beat(1/8))
music.play_tone_for(music.note("D5"), music.beat(1/8 + 1/16))
music.play_tone_for(music.note("D#5"), music.beat(1/16))
music.play_tone_for(music.note("D5"), music.beat(1/8))
print("Measure 3")
music.play_tone_for(music.note("C5"), music.beat(1/4))
music.play_tone_for(music.note("A4"), music.beat(1/8))
music.play_tone_for(music.note("F4"), music.beat(1/8 + 1/16))
music.play_tone_for(music.note("G4"), music.beat(1/16))
music.play_tone_for(music.note("A4"), music.beat(1/8))
print("Measure 4")
music.play_tone_for(music.note("A#4"), music.beat(1/4))
music.play_tone_for(music.note("G4"), music.beat(1/8))
music.play_tone_for(music.note("G4"), music.beat(1/8 + 1/16))
music.play_tone_for(music.note("F#4"), music.beat(1/16))
```

(continues on next page)

(continued from previous page)

```
music.play_tone_for(music.note("G4"), music.beat(1/8))
print("Measure 5")
music.play_tone_for(music.note("A4"), music.beat(1/4))
music.play_tone_for(music.note("F#4"), music.beat(1/8))
music.play_tone_for(music.note("D4"), music.beat(1/4))
music.play_tone_for(music.note("G4"), music.beat(1/8))
print("Measure 6")
music.play_tone_for(music.note("A#4"), music.beat(1/4))
music.play_tone_for(music.note("C5"), music.beat(1/8))
music.play_tone_for(music.note("D5"), music.beat(1/8 + 1/16))
music.play_tone_for(music.note("D#5"), music.beat(1/16))
music.play_tone_for(music.note("D5"), music.beat(1/8))
print("Measure 7")
music.play_tone_for(music.note("C5"), music.beat(1/4))
music.play_tone_for(music.note("A4"), music.beat(1/8))
music.play_tone_for(music.note("F4"), music.beat(1/8 + 1/16))
music.play_tone_for(music.note("G4"), music.beat(1/16))
music.play_tone_for(music.note("A4"), music.beat(1/8))
print("Measure 8")
music.play_tone_for(music.note("A#4"), music.beat(1/8 + 1/16))
music.play_tone_for(music.note("A4"), music.beat(1/16))
music.play_tone_for(music.note("G4"), music.beat(1/8))
music.play_tone_for(music.note("F#4"), music.beat(1/8 + 1/16))
music.play_tone_for(music.note("E4"), music.beat(1/16))
music.play_tone_for(music.note("F#4"), music.beat(1/8))
print("Measure 9")
music.play_tone_for(music.note("G4"), music.beat(1/4 + 1/8))
music.play_tone_for(music.note("G4"), music.beat(1/4 + 1/8))
print("Measure 10")
music.play_tone_for(music.note("F5"), music.beat(1/4 + 1/8))
music.play_tone_for(music.note("F5"), music.beat(1/8))
music.play_tone_for(music.note("E5"), music.beat(1/16))
music.play_tone_for(music.note("D5"), music.beat(1/8))
print("Measure 11")
music.play_tone_for(music.note("C5"), music.beat(1/4))
music.play_tone_for(music.note("A4"), music.beat(1/8))
music.play_tone_for(music.note("F4"), music.beat(1/8 + 1/16))
music.play_tone_for(music.note("G4"), music.beat(1/16))
music.play_tone_for(music.note("A4"), music.beat(1/8))
print("Measure 12")
music.play_tone_for(music.note("A#4"), music.beat(1/4))
music.play_tone_for(music.note("G4"), music.beat(1/8))
music.play_tone_for(music.note("G4"), music.beat(1/8 + 1/16))
music.play_tone_for(music.note("F#4"), music.beat(1/16))
music.play_tone_for(music.note("G4"), music.beat(1/8))
print("Measure 13")
music.play_tone_for(music.note("A4"), music.beat(1/4))
music.play_tone_for(music.note("F#4"), music.beat(1/8))
music.play_tone_for(music.note("D4"), music.beat(1/4 + 1/8))
print("Measure 14")
music.play_tone_for(music.note("F5"), music.beat(1/4 + 1/8))
music.play_tone_for(music.note("F5"), music.beat(1/8))
```

(continues on next page)

(continued from previous page)

```
music.play_tone_for(music.note("E5"), music.beat(1/16))
music.play_tone_for(music.note("D5"), music.beat(1/8))
print("Measure 15")
music.play_tone_for(music.note("C5"), music.beat(1/4))
music.play_tone_for(music.note("A4"), music.beat(1/8))
music.play_tone_for(music.note("F4"), music.beat(1/8 + 1/16))
music.play_tone_for(music.note("G4"), music.beat(1/16))
music.play_tone_for(music.note("A4"), music.beat(1/8))
print("Measure 16")
music.play_tone_for(music.note("A#4"), music.beat(1/8 + 1/16))
music.play_tone_for(music.note("A4"), music.beat(1/16))
music.play_tone_for(music.note("G4"), music.beat(1/8))
music.play_tone_for(music.note("F#4"), music.beat(1/8 + 1/16))
music.play_tone_for(music.note("E4"), music.beat(1/16))
music.play_tone_for(music.note("F#4"), music.beat(1/8))
print("Measure 17")
music.play_tone_for(music.note("G4"), music.beat(1/4 + 1/8))
music.play_tone_for(music.note("G4"), music.beat(1/4 + 1/8))
```

Play sound

```
# Play a sound
music.sound_play("file.wav", volume=50)
# Play a sound in the background
music.sound_play_threading("file.wav", volume=80)
# Get sound length
music.sound_length("file.wav")
```

Play Music

```
# Play music
music.music_play("file.mp3")
# Play music in loop
music.music_play("file.mp3", loop=0)
# Play music in 3 times
music.music_play("file.mp3", loop=3)
# Play music in starts from 2 second
music.music_play("file.mp3", start=2)
# Set music volume
music.music_set_volume(50)
# Stop music
music.music_stop()
# Pause music
music.music_pause()
# Resume music
music.music_resume()
```

API

class robot_hat.**Music**

Bases: *_Basic_class*

Play music, sound affect and note control

NOTE_BASE_FREQ = 440

Base note frequency for calculation (A4)

NOTE_BASE_INDEX = 69

Base note index for calculation (A4) MIDI compatible

NOTES = [None, 'A0', 'A#0', 'B0', 'C1', 'C#1', 'D1', 'D#1', 'E1', 'F1', 'F#1', 'G1', 'G#1', 'A1', 'A#1', 'B1', 'C2', 'C#2', 'D2', 'D#2', 'E2', 'F2', 'F#2', 'G2', 'G#2', 'A2', 'A#2', 'B2', 'C3', 'C#3', 'D3', 'D#3', 'E3', 'F3', 'F#3', 'G3', 'G#3', 'A3', 'A#3', 'B3', 'C4', 'C#4', 'D4', 'D#4', 'E4', 'F4', 'F#4', 'G4', 'G#4', 'A4', 'A#4', 'B4', 'C5', 'C#5', 'D5', 'D#5', 'E5', 'F5', 'F#5', 'G5', 'G#5', 'A5', 'A#5', 'B5', 'C6', 'C#6', 'D6', 'D#6', 'E6', 'F6', 'F#6', 'G6', 'G#6', 'A6', 'A#6', 'B6', 'C7', 'C#7', 'D7', 'D#7', 'E7', 'F7', 'F#7', 'G7', 'G#7', 'A7', 'A#7', 'B7', 'C8']

Notes name, MIDI compatible

__init__()

Initialize the basic class

Parameters
debug_level (*str/int*) – debug level, 0(critical), 1(error), 2(warning), 3(info) or 4(debug)

time_signature(*top: int = None*, *bottom: int = None*)

Set/get time signature

Parameters

- **top** (*int*) – top number of time signature
- **bottom** (*int*) – bottom number of time signature

Returns
time signature

Return type
tuple

key_signature(*key: int = None*)

Set/get key signature

Parameters
key (*int/str*) – key signature use KEY_XX_MAJOR or String “#”, “##”, or “bbb”, “bbbb”

Returns
key signature

Return type
int

tempo(*tempo=None*, *note_value=0.25*)

Set/get tempo beat per minute(bpm)

Parameters

- **tempo** (*float*) – tempo
- **note_value** – note value(1, 1/2, Music.HALF_NOTE, etc)

Returns
tempo

Return type
int

`beat`(*beat*)

Calculate beat delay in seconds from tempo

Parameters
beat (*float*) – beat index

Returns
beat delay

Return type
float

`note`(*note*, *natural=False*)

Get frequency of a note

Parameters

- **note_name** (*string*) – note name(See NOTES)
- **natural** (*bool*) – if natural note

Returns
frequency of note

Return type
float

`sound_play`(*filename*, *volume=None*)

Play sound effect file

Parameters
filename (*str*) – sound effect file name

`sound_play_threading`(*filename*, *volume=None*)

Play sound effect in thread(in the background)

Parameters

- **filename** (*str*) – sound effect file name
- **volume** (*int*) – volume 0-100, leave empty will not change volume

`music_play`(*filename*, *loops=1*, *start=0.0*, *volume=None*)

Play music file

Parameters

- **filename** (*str*) – sound file name
- **loops** (*int*) – number of loops, 0:loop forever, 1:play once, 2:play twice, ...
- **start** (*float*) – start time in seconds
- **volume** (*int*) – volume 0-100, leave empty will not change volume

`music_set_volume`(*value*)

Set music volume

Parameters
value (*int*) – volume 0-100

music_stop()

Stop music

music_pause()

Pause music

music_resume()

Resume music

music_unpause()

Unpause music(resume music)

sound_length(*filename*)

Get sound effect length in seconds

Parameters
filename (*str*) – sound effect file name

Returns
length in seconds

Return type
float

get_tone_data(*freq: float*, *duration: float*)

Get tone data for playing

Parameters

- **freq** (*float*) – frequency
- **duration** (*float*) – duration in seconds

Returns
tone data

Return type
list

play_tone_for(*freq*, *duration*)

Play tone for duration seconds

Parameters

- **freq** (*float*) – frequency, you can use NOTES to get frequency
- **duration** (*float*) – duration in seconds

7.9 class TTS

Warning:

- You need to add sudo when running this script, in case the speaker doesn't work.
- *Q3: Why is there no sound from the speaker?*.

Example

```
# Import TTS class
from robot_hat import TTS

# Initialize TTS class
tts = TTS(lang='en-US')
# Speak text
tts.say("Hello World")
# show all supported languages
print(tts.supported_lang())
```

API

class robot_hat.**TTS**(*engine='pico2wave'*, *lang=None*, **args*, ***kwargs*)

Bases: *_Basic_class*

Text to speech class

SUPPORTED_LANGUAUE = ['en-US', 'en-GB', 'de-DE', 'es-ES', 'fr-FR', 'it-IT']

Supported TTS language for pico2wave

ESPEAK = 'espeak'

espeak TTS engine

PICO2WAVE = 'pico2wave'

pico2wave TTS engine

__init__(*engine='pico2wave'*, *lang=None*, **args*, ***kwargs*)

Initialize TTS class.

Parameters
engine (*str*) – TTS engine, TTS.PICO2WAVE or TTS.ESPEAK

say(*words*)

Say words.

Parameters
words (*str*) – words to say.

espeak(*words*)

Say words with espeak.

Parameters
words (*str*) – words to say.

pico2wave(*words*)

Say words with pico2wave.

Parameters
words (*str*) – words to say.

lang(**value*)

Set/get language. leave empty to get current language.

Parameters
value (*str*) – language.

supported_lang()

Get supported language.

Returns
supported language.

Return type
list

espeak_params(*amp=None*, *speed=None*, *gap=None*, *pitch=None*)

Set espeak parameters.

Parameters

- **amp** (*int*) – amplitude.
- **speed** (*int*) – speed.
- **gap** (*int*) – gap.
- **pitch** (*int*) – pitch.

7.10 module utils

robot_hat.utils.**set_volume**(*value*)

Set volume

Parameters
value (*int*) – volume(0~100)

robot_hat.utils.**run_command**(*cmd*)

Run command and return status and output

Parameters
cmd (*str*) – command to run

Returns
status, output

Return type
tuple

robot_hat.utils.**is_installed**(*cmd*)

Check if command is installed

Parameters
cmd (*str*) – command to check

Returns
True if installed

Return type
bool

robot_hat.utils.**mapping**(*x*, *in_min*, *in_max*, *out_min*, *out_max*)

Map value from one range to another range

Parameters

- **x** (*float/int*) – value to map
- **in_min** (*float/int*) – input minimum
- **in_max** (*float/int*) – input maximum

- **out_min** (*float/int*) – output minimum
- **out_max** (*float/int*) – output maximum

Returns
mapped value

Return type
float/int

robot_hat.utils.**get_ip**(*ifaces=['wlan0', 'eth0']*)

Get IP address

Parameters
ifaces (*list*) – interfaces to check

Returns
IP address or False if not found

Return type
str/False

robot_hat.utils.**reset_mcu**()

Reset mcu on Robot Hat.

This is helpful if the mcu somehow stuck in a I2C data transfer loop, and Raspberry Pi getting IOError while Reading ADC, manipulating PWM, etc.

robot_hat.utils.**get_battery_voltage**()

Get battery voltage

Returns
battery voltage(V)

Return type
float

7.11 class FileDB

Example

```
# Import fileDB class
from robot_hat import fileDB

# Create fileDB object with a config file
db = fileDB("./config")

# Set some values
db.set("apple", "10")
db.set("orange", "5")
db.set("banana", "13")

# Read the values
print(db.get("apple"))
print(db.get("orange"))
print(db.get("banana"))
```

(continues on next page)

(continued from previous page)

```
# Read an none existing value with a default value
print(db.get("pineapple", default_value="-1"))
```

Now you can checkout the config file `config` in bash.

```
cat config
```

API

class robot_hat.**fileDB**(*db: str*, *mode: str = None*, *owner: str = None*)

Bases: `object`

A file based database.

A file based database, read and write arguements in the specific file.

__init__(*db: str*, *mode: str = None*, *owner: str = None*)

Init the db_file is a file to save the datas.

Parameters

- **db** (*str*) – the file to save the datas.
- **mode** (*str*) – the mode of the file.
- **owner** (*str*) – the owner of the file.

file_check_create(*file_path: str*, *mode: str = None*, *owner: str = None*)

Check if file is existed, otherwise create one.

Parameters

- **file_path** (*str*) – the file to check
- **mode** (*str*) – the mode of the file.
- **owner** (*str*) – the owner of the file.

get(*name*, *default_value=None*)

Get value with data's name

Parameters

- **name** (*str*) – the name of the arguement
- **default_value** (*str*) – the default value of the arguement

Returns

the value of the arguement

Return type

str

set(*name*, *value*)

Set value by with name. Or create one if the arguement does not exist

Parameters

- **name** (*str*) – the name of the arguement
- **value** (*str*) – the value of the arguement

7.12 class I2C

Example

```
# Import the I2C class
from robot_hat import I2C

# You can scan for available I2C devices
print([f"0x{addr:02X}" for addr in I2C().scan()])
# You should see at least one device address 0x14, which is the
# on board MCU for PWM and ADC

# Initialize a I2C object with device address, for example
# to communicate with on board MCU 0x14
mcu = I2C(0x14)
# Send ADC channel register to read ADC, 0x10 is Channel 0, 0x11 is Channel 1, etc.
mcu.write([0x10, 0x00, 0x00])
# Read 2 byte for MSB and LSB
msb, lsb = mcu.read(2)
# Convert to integer
value = (msb << 8) + lsb
# Print the value
print(value)
```

For more information on the I2C protocol, see checkout adc.py and pwm.py

API

class robot_hat.**I2C**(*address=None*, *bus=1*, **args*, ***kwargs*)

Bases: *_Basic_class*

I2C bus read/write functions

__init__(*address=None*, *bus=1*, **args*, ***kwargs*)

Initialize the I2C bus

Parameters

- **address** (*int*) – I2C device address
- **bus** (*int*) – I2C bus number

scan()

Scan the I2C bus for devices

Returns
List of I2C addresses of devices found

Return type
list

write(*data*)

Write data to the I2C device

Parameters
data (*int/list/bytearray*) – Data to write

Raises
ValueError if write is not an int, list or bytearray

read(*length=1*)

Read data from I2C device

Parameters
length (*int*) – Number of bytes to receive

Returns
Received data

Return type
list

mem_write(*data*, *memaddr*)

Send data to specific register address

Parameters

- **data** (*int/list/bytearray*) – Data to send, int, list or bytearray
- **memaddr** (*int*) – Register address

Raises
ValueError – If data is not int, list, or bytearray

mem_read(*length*, *memaddr*)

Read data from specific register address

Parameters

- **length** (*int*) – Number of bytes to receive
- **memaddr** (*int*) – Register address

Returns
Received bytearray data or False if error

Return type
list/False

is_avaliable()

Check if the I2C device is avaliable

Returns
True if the I2C device is avaliable, False otherwise

Return type
bool

7.13 class _Basic_class

_Basic_class is a logger class for all class to log, so if you want to see logs of a class, just add a debug argument to it.

Example

```
# See PWM log
from robot_hat import PWM

# init the class with a debug argument
pwm = PWM(0, debug_level="debug")
```

(continues on next page)

(continued from previous page)

```
# run some functions and see logs
pwm.freq(1000)
pwm.pulse_width_percent(100)
```

API

class robot_hat.basic.**_Basic_class**(*debug_level='warning'*)

Basic Class for all classes

with debug function

DEBUG_LEVELS = {'critical': 50, 'debug': 10, 'error': 40, 'info': 20, 'warning': 30}

Debug level

DEBUG_NAMES = ['critical', 'error', 'warning', 'info', 'debug']

Debug level names

__init__(*debug_level='warning'*)

Initialize the basic class

Parameters

debug_level (*str/int*) – debug level, 0(critical), 1(error), 2(warning), 3(info) or 4(debug)

property debug_level

Debug level

CHAPTER

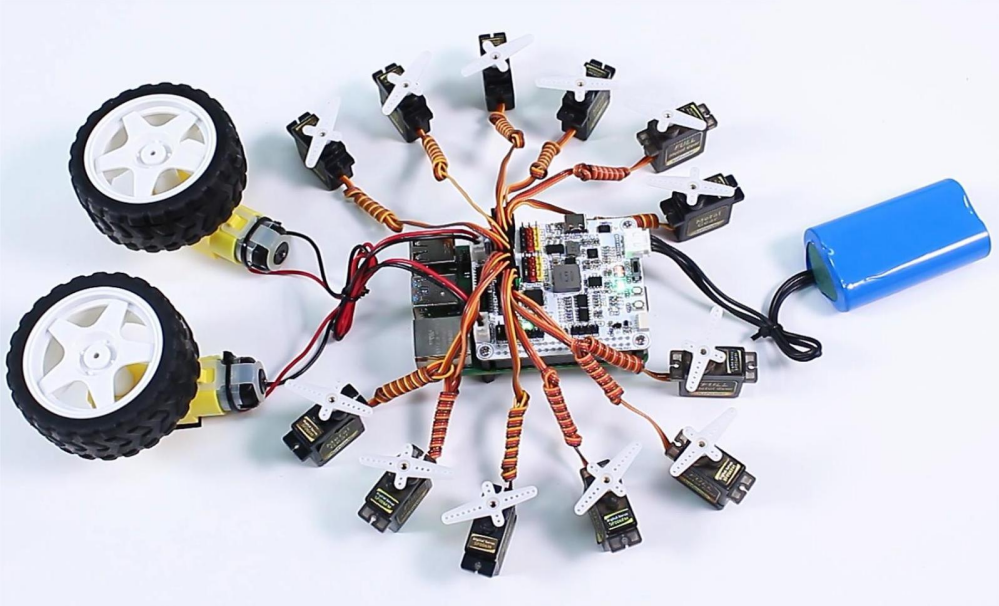
EIGHT

SOME PROJECTS

Here, you'll find a collection of fascinating projects, all implemented using the Robot HAT. We provide you with detailed code, giving you the opportunity to try these projects out for yourself.

8.1 Control Servos and Motors

In this project, we have 12 servos and two motors working simultaneously.



However, it's important to note that if your servos and motors have a high starting current, it's recommended to start them separately to avoid insufficient power supply current, which could lead to the Raspberry Pi restarting.

Code

```
from robot_hat import Servo, Motors
import time

# Create objects for 12 servos
servos = [Servo(f"P{i}") for i in range(12)]

# Create motor object
motors = Motors()
```

(continues on next page)

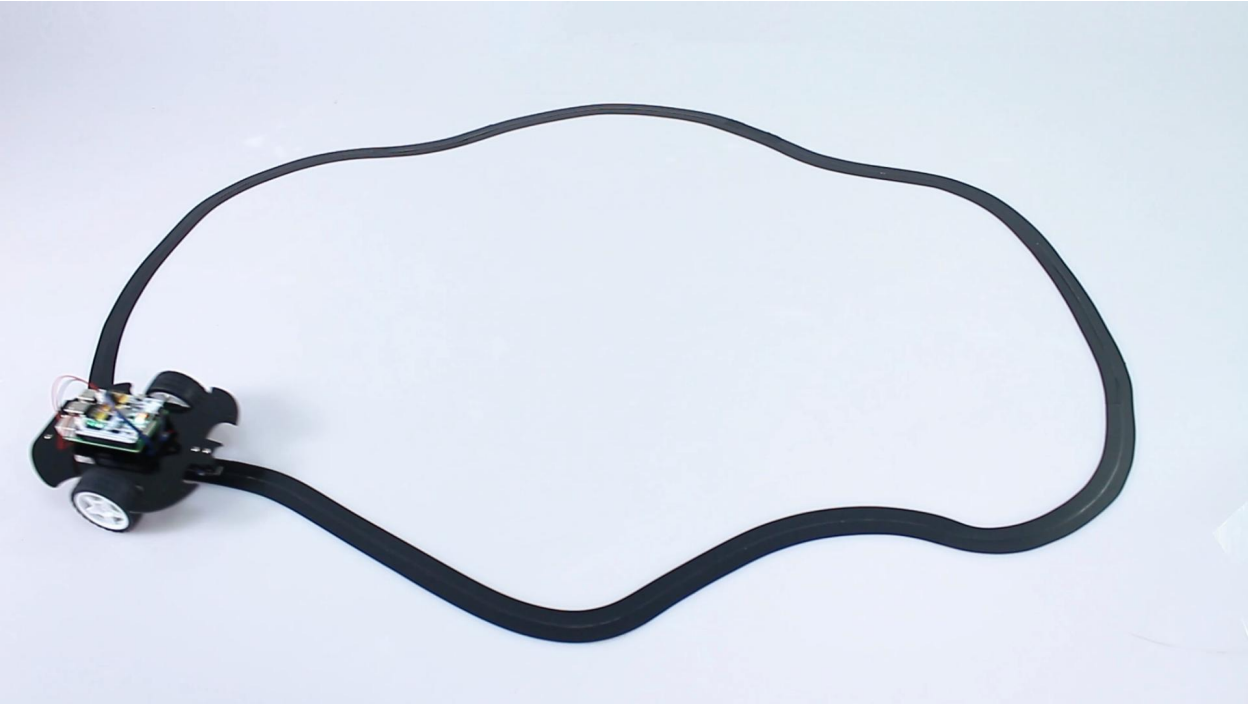
(continued from previous page)

```
def initialize_servos():
    """Set initial angle of all servos to 0."""
    for servo in servos:
        servo.angle(-90)
        time.sleep(0.1)  # Wait for servos to reach the initial position
    time.sleep(1)


def sweep_servos(angle_from, angle_to, step):
    """Control all servos to sweep from a start angle to an end angle."""
    if angle_from < angle_to:
        range_func = range(angle_from, angle_to + 1, step)
    else:
        range_func = range(angle_from, angle_to - 1, -step)

    for angle in range_func:
        for servo in servos:
            servo.angle(angle)
        time.sleep(0.05)

def control_motors_and_servos():
    """Control motors and servos in synchronization."""
    try:
        while True:
            # Motors rotate forward and servos sweep from -90 to 90 degrees
            motors[1].speed(80)
            time.sleep(0.01)
            motors[2].speed(80)
            time.sleep(0.01)
            sweep_servos(-90, 90, 5)
            time.sleep(1)

            # Motors rotate backward and servos sweep from 90 to -90 degrees
            motors[1].speed(-80)
            time.sleep(0.01)
            motors[2].speed(-80)
            time.sleep(0.01)
            sweep_servos(90, -90, 5)
            time.sleep(1)
    except KeyboardInterrupt:
        # Stop motors when Ctrl+C is pressed
        motors.stop()
        print("Motors stopped.")

# Initialize servos to their initial position
initialize_servos()

# Control motors and servos
control_motors_and_servos()
```

8.2 DIY Car

In addition to being suitable for simple experiments, the Robot HAT is ideal for use as a central controller in robotics, such as for smart cars.

In this project, we built a simple line-following car.



Code

```
from robot_hat import Motors, Pin
import time

# Create motor object
motors = Motors()

# Initialize line tracking sensor
line_track = Pin('D0')

def main():
    while True:
        # print("value", line_track.value())
        # time.sleep(0.01)
        if line_track.value() == 1:
            # If line is detected
            motors[1].speed(-60)  # Motor 1 forward
            motors[2].speed(20) # Motor 2 backward
            time.sleep(0.01)
        else:
            # If line is not detected
            motors[1].speed(-20) # Motor 1 backward
            motors[2].speed(60)  # Motor 2 forward
```

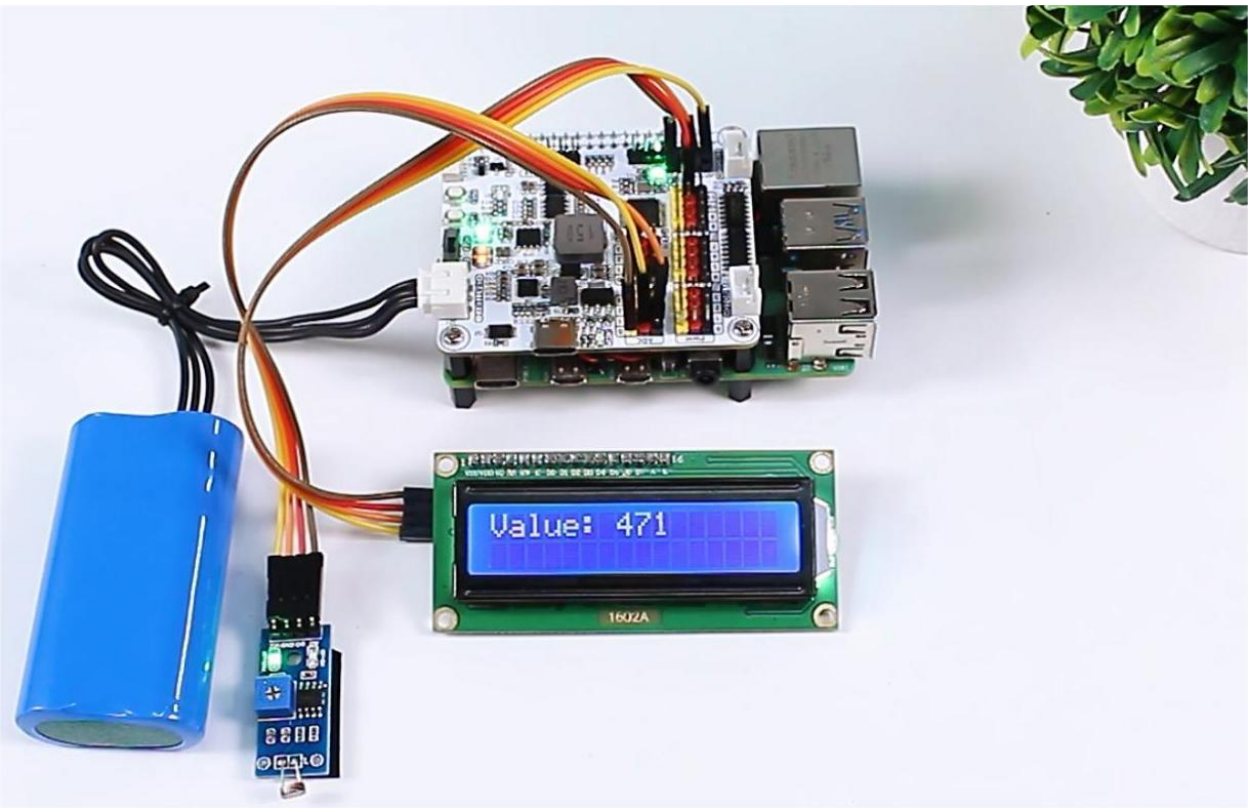
(continues on next page)

(continued from previous page)

```
            time.sleep(0.01)

def destroy():
    # Stop motors when Ctrl+C is pressed
    motors.stop()
    print("Motors stopped.")

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        destroy()
```

8.3 Read from Photoresistor Module

In this project, we detect the light intensity and display on the I2C LCD1602.



Steps

1. In this project, an I2C LCD1602 is used, so it's necessary to download the relevant libraries to make it work.

```
cd ~/
wget https://github.com/sunfounder/raphael-kit/blob/master/python/LCD1602.py
```

2. Install `smbus2` for I2C.

```
sudo pip3 install smbus2
```

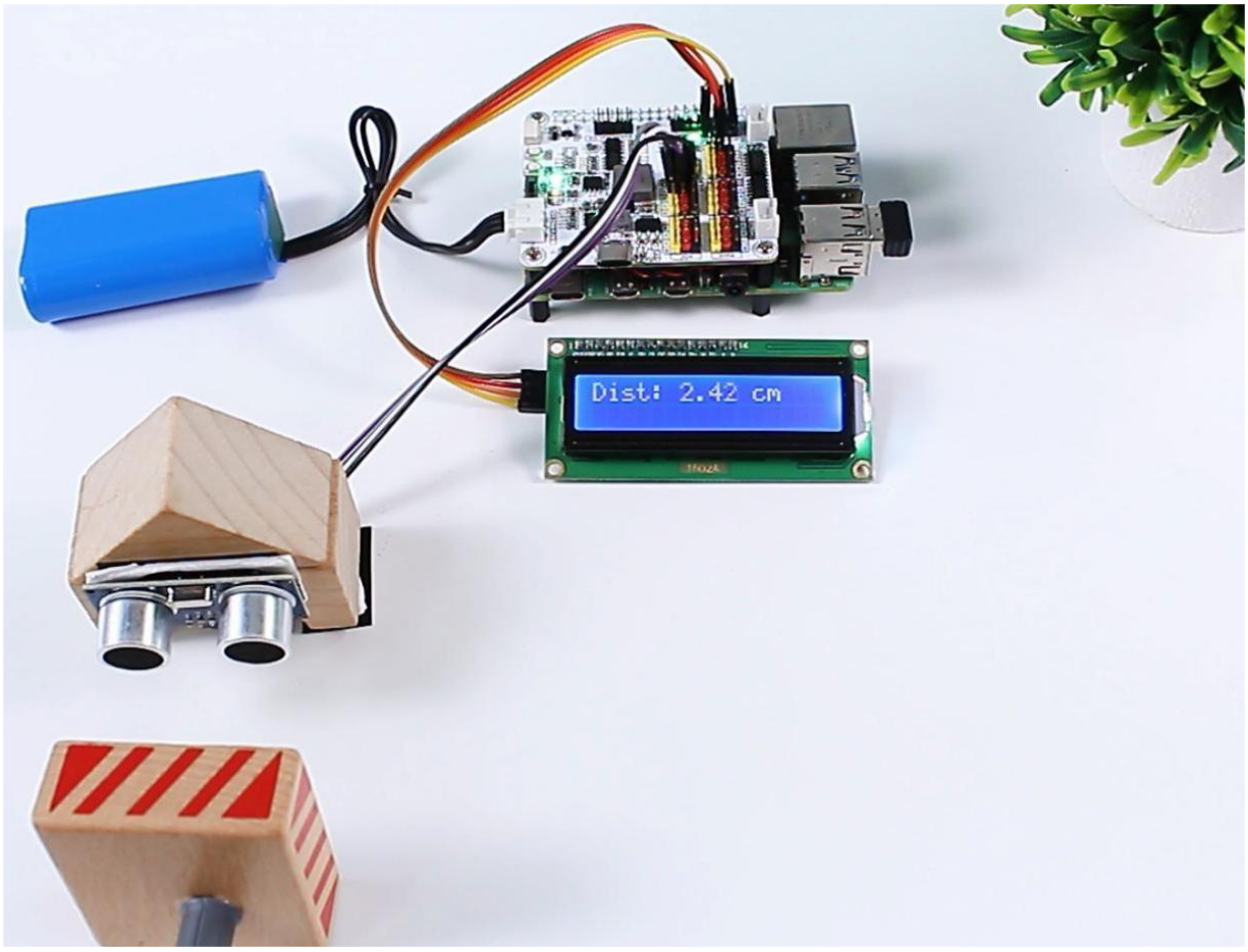
3. Save the following code to your Raspberry Pi and give it a name, for example, `photoresistor.ty`.

```
from robot_hat import ADC
import LCD1602
import time

# Create an ADC object to read the value from the photoresistor
a0 = ADC(0)

def setup():
    # Initialize the LCD1602
    LCD1602.init(0x27, 1)
    time.sleep(2)

def destroy():
    # Clear the LCD display
    LCD1602.clear()

def loop():
    while True:
        # Read the value from the photoresistor
        value0 = a0.read()
        # Display the read value on the LCD
        LCD1602.write(0, 0, 'Value: %d  ' % value0)
        # Reduce the refresh rate to update once per second
        time.sleep(0.2)

if __name__ == '__main__':
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
    except Exception as e:
        # Clear the LCD and print error message in case of an exception
        destroy()
        print("Error:", e)
```

4. Use the command `sudo python3 photoresistor.ty` to run this code.

8.4 Read from Ultrasonic Module

In this project, we use ultrasonic sensors to measure distance and display the readings on the I2C LCD1602.



Steps

1. In this project, an I2C LCD1602 is used, so it's necessary to download the relevant libraries to make it work.

```
cd ~/
wget https://github.com/sunfounder/raphael-kit/blob/master/python/LCD1602.py
```

2. Install `smbus2` for I2C.

```
sudo pip3 install smbus2
```

3. Save the following code to your Raspberry Pi and give it a name, for example, `ultrasonic.ty`.

```
from robot_hat import ADC, Ultrasonic, Pin
import LCD1602
import time

# Create ADC object for photoresistor
a0 = ADC(0)

```

(continues on next page)

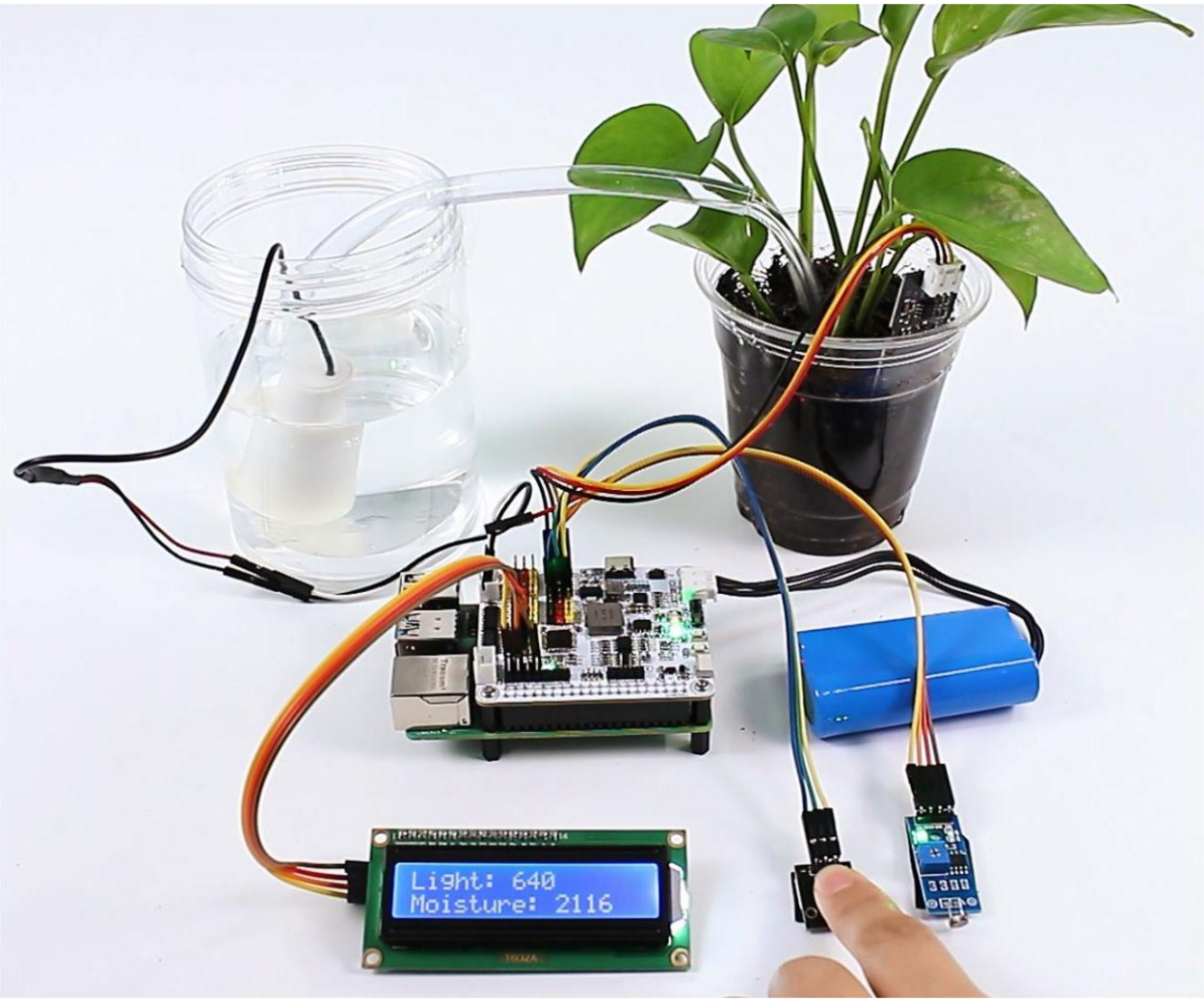
(continued from previous page)

```
# Create Ultrasonic object
us = Ultrasonic(Pin("D2"), Pin("D3")) //Trig to digital pin 2, echo to pin 3

def setup():
    # Initialize LCD1602
    LCD1602.init(0x27, 1)
    # Initial message on LCD
    LCD1602.write(0, 0, 'Measuring...')
    time.sleep(2)

def destroy():
    # Clear the LCD display
    LCD1602.clear()

def loop():
    while True:
        # Read distance from ultrasonic sensor
        distance = us.read()
        # Display the distance on the LCD
        if distance != -1:
            # Display the valid distance on the LCD
            LCD1602.write(0, 0, 'Dist: %.2f cm   ' % distance)

        # Update every 0.5 seconds
        time.sleep(0.2)

if __name__ == '__main__':
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
    except Exception as e:
        # Clear the LCD and print error message in case of an exception
        destroy()
        print("Error:", e)
```

4. Use the command `sudo python3 ultrasonic.ty` to run this code.

8.5 Plant Monitor

In this project, we detect both light intensity and soil moisture levels, and display them on the I2C LCD1602 . When you feel that the soil moisture is insufficient, you can press the button module to water the potted plant.



Steps

1. In this project, an I2C LCD1602 is used, so it's necessary to download the relevant libraries to make it work.

```
cd ~/
wget https://github.com/sunfounder/raphael-kit/blob/master/python/LCD1602.py
```

2. Install `smbus2` for I2C.

```
sudo pip3 install smbus2
```

3. Save the following code to your Raspberry Pi and give it a name, for example, `plant_monitor.ty`.

```
from robot_hat import ADC, Motors, Pin
import LCD1602
import time
import threading

from robot_hat.utils import reset_mcu

reset_mcu()
```

(continues on next page)

(continued from previous page)

```
time.sleep(.1)


# Initialize objects
light_sensor = ADC(1)
moisture_sensor = ADC(0)
motors = Motors()
button = Pin('D0')

# Thread running flag
running = True

def init_lcd():
    LCD1602.init(0x27, 1)
    time.sleep(2)

def update_lcd(light_value, moisture_value):
    LCD1602.write(0, 0, 'Light: %d  ' % light_value)
    LCD1602.write(0, 1, 'Moisture: %d  ' % moisture_value)

def read_sensors():
    light_value = light_sensor.read()
    time.sleep(0.2)
    moisture_value = moisture_sensor.read()
    time.sleep(0.2)
    return light_value, moisture_value

def control_motor():
    global running
    while running:
        button_pressed = button.value() == 0
        if button_pressed:
            motors[1].speed(80)
            time.sleep(0.1)
        else:
            motors[1].speed(0)
            time.sleep(0.1)
        time.sleep(0.1)

def setup():
    init_lcd()

def destroy():
    global running
    running = False
    LCD1602.clear()

def loop():
    global running
    while running:
        light_value, moisture_value = read_sensors()
        update_lcd(light_value, moisture_value)
```

(continues on next page)

(continued from previous page)

```
        time.sleep(.2)

if __name__ == '__main__':
    try:
        setup()
        motor_thread = threading.Thread(target=control_motor)
        motor_thread.start()
        loop()
    except KeyboardInterrupt:
        motor_thread.join()  # Wait for motor_thread to finish
        print("Program stopped")
    except Exception as e:
        print("Error:", e)
    finally:
        motors[1].speed(0)
        time.sleep(.1)
        destroy()
        print('end')
```

4. Use the command `sudo python3 plant_monitor.ty` to run this code.

8.6 Say Something

In this section, you'll learn how to convert text into speech and have Robot HAT speak it aloud.

Steps

1. We retrieve text from the command line to enable Robot HAT to articulate it. To achieve this, save the following code as a `.py` file, such as `tts.py`.

```
import sys
from robot_hat import TTS

# Check if there are enough command line arguments
if len(sys.argv) > 1:
    text_to_say = sys.argv[1]  # Get the first argument passed from the␣
↪command line
else:
    text_to_say = "Hello SunFounder"  # Default text if no arguments are␣
↪provided

# Initialize the TTS class
tts = TTS(lang='en-US')

# Read the text
tts.say(text_to_say)

# Display all supported languages
print(tts.supported_lang())
```

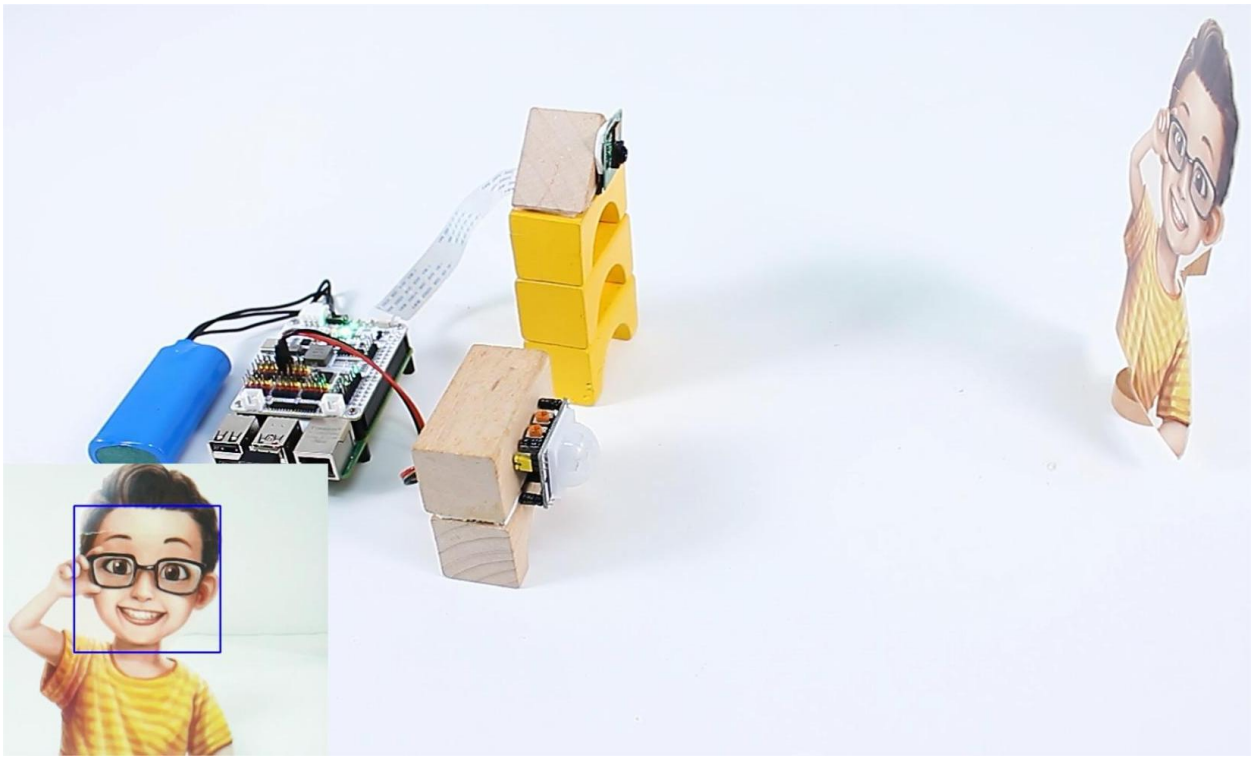
2. To make Robot HAT vocalize a specific sentence, you can use the following command: `sudo python3 tts.py "any text"` - simply replace `"any text"` with the desired phrase.

Note:

- *Q3: Why is there no sound from the speaker?*

8.7 Security System

In this project, we've created a simple security system. The PIR sensor detects if someone passes by, and then the camera activates. If a face is detected, it takes a picture and simultaneously delivers a warning message.



Steps

1. Install the `vilib` library for face detection.

```
cd ~/
git clone -b picamera2 https://github.com/sunfounder/vilib.git
cd vilib
sudo python3 install.py
```

2. Save the following code to your Raspberry Pi and give it a name, for example, `security.ty`.

```
import os
from time import sleep, time, strftime, localtime
from vilib import Vilib
from robot_hat import Pin, TTS


# Initialize the TTS class
```

(continues on next page)

(continued from previous page)

```
tts = TTS(lang='en-US')

# Display all supported languages
print(tts.supported_lang())

# Initialize the PIR sensor
pir = Pin('D0')

def camera_start():
    Vilib.camera_start()
    Vilib.display()
    Vilib.face_detect_switch(True)

def take_photo():
    _time = strftime('%Y-%m-%d-%H-%M-%S', localtime(time()))
    name = f'photo_{_time}'
    username = os.getlogin()
    path = f"/home/{username}/Pictures/"
    Vilib.take_photo(name, path)
    print(f'Photo saved as {path}{name}.jpg')

def main():
    motion_detected = False
    while True:
        # Check for motion
        if pir.value() == 1:
            if not motion_detected:
                print("Motion detected! Initializing camera...")
                camera_start()
                motion_detected = True
                sleep(2)  # Stabilization delay to confirm motion

            # Check for human face and take a photo
            if Vilib.detect_obj_parameter['human_n'] != 0:
                take_photo()
                # Read the text
                tts.say("Security alert: Unrecognized Individual detected.␣
↪Please verify identity")
                sleep(2)  # Delay after taking a photo

        # If no motion is detected, turn off the camera
        elif motion_detected:
            print("No motion detected. Finalizing camera...")
            Vilib.camera_close()
            motion_detected = False
            sleep(2)  # Delay before re-enabling motion detection

        sleep(0.1)  # Short delay to prevent CPU overuse

def destroy():
    Vilib.camera_close()
    print("Camera and face detection stopped.")
```

(continues on next page)

(continued from previous page)

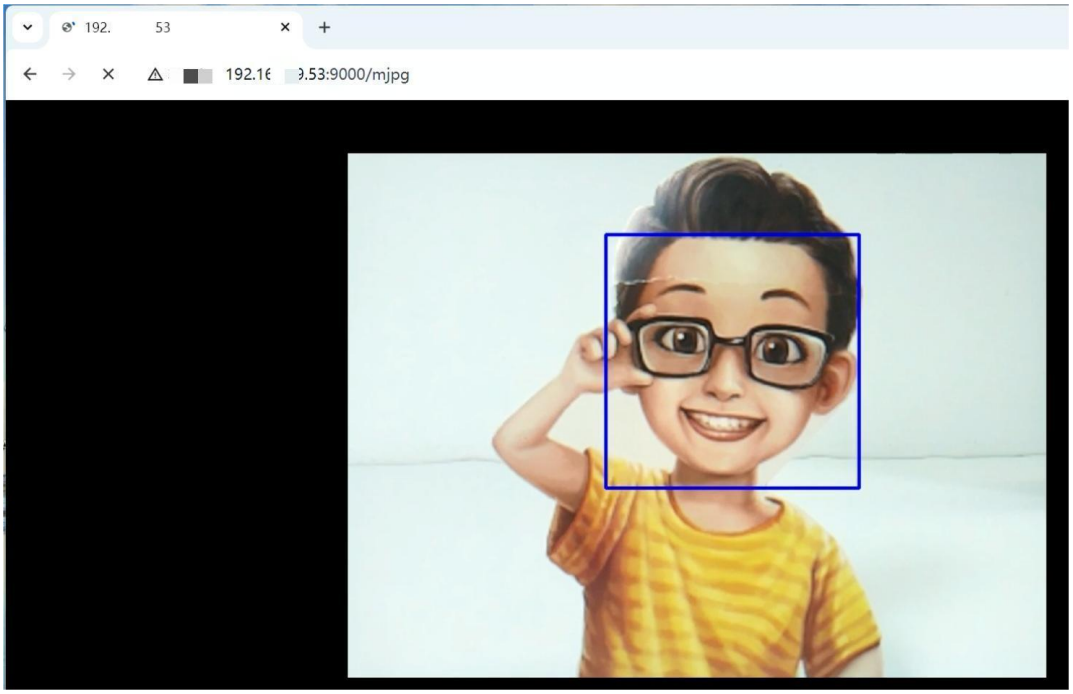
```
if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        destroy()
```

3. **Use the command `sudo python3 security.py` to run this code.**

   Note:

   - *Q3: Why is there no sound from the speaker?*

4. Open a web browser and enter `http://rpi_ip:9000/mjpg` to view the captured footage. Additionally, you can find the captured face images in `/home/{username}/Pictures/`.



8.8 Community Tutorials

-

This document summarizes the SunFounder Raspberry Pi Robot HAT, covering its purpose, compatibility, specifications, and testing:

- **Introduction**: Explains the Robot HAT's role in simplifying control for Raspberry Pi-based DIY robot projects.
- **Specifications**: Details the technical specs, including power input, battery details, ports, and motor driver features.
- **Ports Overview**: Describes various ports like Power, Digital, Analog, PWM, I2C, SPI, UART, and Motor Ports.

- **Additional Components**: Highlights extra components like buttons, LED, and speaker, with Raspberry Pi PIN mappings.
- **Setup and Testing**: Guides on mounting the Robot HAT, necessary components, and testing procedures for features like LED and servo motors.

CHAPTER

NINE

FAQ

9.1 Q1: Can the battery be connected while providing power to the Raspberry Pi at the same time?

A: Yes, the Robot HAT has a built-in anti-backflow diode that prevents the Raspberry Pi's power from flowing back into the Robot HAT.

9.2 Q2: Can the Robot HAT be used while charging?

A: Yes, the Robot HAT can be used while charging. When charging, the input power is boosted by the charging chip to charge the batteries, while also providing power to the DC-DC step-down for external use. The charging power is approximately 10W. If the external power consumption is too high for an extended period, the batteries may supplement the power, similar to how a mobile phone charges while in use. However, it is important to be mindful of the battery's capacity to avoid draining it completely during simultaneous charging and usage.

9.3 Q3: Why is there no sound from the speaker?

When your script is running but the speaker is not producing sound, there could be several reasons:

1. Check if the `i2samp.sh` script has been installed. For detailed instructions, please refer to: *Install i2samp.sh for the Speaker*.
2. When running scripts related to speakers, it's necessary to add `sudo` to obtain administrative privileges. For example, `sudo python3 tts.py`.
3. Don't using Raspberry Pi's built-in programming tools, like Geany to run Speaker-related scripts. These tools run with standard user privileges, while hardware control, such as managing speakers, often requires higher permissions.

PYTHON MODULE INDEX

r

robot_hat, 27
robot_hat.utils, 53

INDEX

Symbols

_Basic_class (*class in robot_hat.basic*), 58
__call__() (*robot_hat.Pin method*), 28
__getitem__() (*robot_hat.Motors method*), 35
__init__() (*robot_hat.ADC method*), 30
__init__() (*robot_hat.ADXL345 method*), 38
__init__() (*robot_hat.Buzzer method*), 41
__init__() (*robot_hat.Grayscale_Module method*), 43
__init__() (*robot_hat.I2C method*), 56
__init__() (*robot_hat.Motor method*), 37
__init__() (*robot_hat.Motors method*), 35
__init__() (*robot_hat.Music method*), 49
__init__() (*robot_hat.PWM method*), 32
__init__() (*robot_hat.Pin method*), 28
__init__() (*robot_hat.RGB_LED method*), 39
__init__() (*robot_hat.Robot method*), 44
__init__() (*robot_hat.Servo method*), 34
__init__() (*robot_hat.TTS method*), 52
__init__() (*robot_hat.basic._Basic_class method*), 58
__init__() (*robot_hat.fileDB method*), 55
__init__() (*robot_hat.modules.Ultrasonic method*), 38

A

ADC (*class in robot_hat*), 30
ADXL345 (*class in robot_hat*), 38
angle() (*robot_hat.Servo method*), 34
ANODE (*robot_hat.RGB_LED attribute*), 39

B

backward() (*robot_hat.Motors method*), 36
beat() (*robot_hat.Music method*), 50
Buzzer (*class in robot_hat*), 41

C

calibration() (*robot_hat.Robot method*), 45
CATHODE (*robot_hat.RGB_LED attribute*), 39
CLOCK (*robot_hat.PWM attribute*), 32
color() (*robot_hat.RGB_LED method*), 39

D

debug_level (*robot_hat.basic._Basic_class property*), 58
DEBUG_LEVELS (*robot_hat.basic._Basic_class attribute*), 58
DEBUG_NAMES (*robot_hat.basic._Basic_class attribute*), 58
dict() (*robot_hat.Pin method*), 28
do_action() (*robot_hat.Robot method*), 45

E

ESPEAK (*robot_hat.TTS attribute*), 52
espeak() (*robot_hat.TTS method*), 52
espeak_params() (*robot_hat.TTS method*), 53

F

file_check_create() (*robot_hat.fileDB method*), 55
fileDB (*class in robot_hat*), 55
forward() (*robot_hat.Motors method*), 36
freq() (*robot_hat.Buzzer method*), 42
freq() (*robot_hat.PWM method*), 32

G

get() (*robot_hat.fileDB method*), 55
get_battery_voltage() (*in module robot_hat.utils*), 54
get_ip() (*in module robot_hat.utils*), 54
get_tone_data() (*robot_hat.Music method*), 51
Grayscale_Module (*class in robot_hat*), 42

H

high() (*robot_hat.Pin method*), 29

I

I2C (*class in robot_hat*), 56
IN (*robot_hat.Pin attribute*), 28
irq() (*robot_hat.Pin method*), 29
IRQ_FALLING (*robot_hat.Pin attribute*), 28
IRQ_RISING (*robot_hat.Pin attribute*), 28
IRQ_RISING_FALLING (*robot_hat.Pin attribute*), 28
is_avaliable() (*robot_hat.I2C method*), 57
is_installed() (*in module robot_hat.utils*), 53

K

key_signature() (*robot_hat.Music method*), 49

L

lang() (*robot_hat.TTS method*), 52
LEFT (*robot_hat.Grayscale_Module attribute*), 42
left (*robot_hat.Motors property*), 35
low() (*robot_hat.Pin method*), 29

M

mapping() (*in module robot_hat.utils*), 53
max_dps (*robot_hat.Robot attribute*), 44
mem_read() (*robot_hat.I2C method*), 57
mem_write() (*robot_hat.I2C method*), 57
MIDDLE (*robot_hat.Grayscale_Module attribute*), 42
module
    robot_hat, 27
    robot_hat.utils, 53
Motor (*class in robot_hat*), 37
Motors (*class in robot_hat*), 35
move_list (*robot_hat.Robot attribute*), 44
Music (*class in robot_hat*), 48
music_pause() (*robot_hat.Music method*), 51
music_play() (*robot_hat.Music method*), 50
music_resume() (*robot_hat.Music method*), 51
music_set_volume() (*robot_hat.Music method*), 50
music_stop() (*robot_hat.Music method*), 50
music_unpause() (*robot_hat.Music method*), 51

N

name() (*robot_hat.Pin method*), 30
new_list() (*robot_hat.Robot method*), 44
note() (*robot_hat.Music method*), 50
NOTE_BASE_FREQ (*robot_hat.Music attribute*), 48
NOTE_BASE_INDEX (*robot_hat.Music attribute*), 49
NOTES (*robot_hat.Music attribute*), 49

O

off() (*robot_hat.Buzzer method*), 41
off() (*robot_hat.Pin method*), 29
on() (*robot_hat.Buzzer method*), 41
on() (*robot_hat.Pin method*), 29
OUT (*robot_hat.Pin attribute*), 27

P

period() (*robot_hat.PWM method*), 32
PICO2WAVE (*robot_hat.TTS attribute*), 52
pico2wave() (*robot_hat.TTS method*), 52
Pin (*class in robot_hat*), 27
play() (*robot_hat.Buzzer method*), 42
play_tone_for() (*robot_hat.Music method*), 51
prescaler() (*robot_hat.PWM method*), 32
PULL_DOWN (*robot_hat.Pin attribute*), 28
PULL_NONE (*robot_hat.Pin attribute*), 28
PULL_UP (*robot_hat.Pin attribute*), 28
pulse_width() (*robot_hat.PWM method*), 32
pulse_width_percent() (*robot_hat.PWM method*), 32
pulse_width_time() (*robot_hat.Servo method*), 34
PWM (*class in robot_hat*), 31

R

read() (*robot_hat.ADC method*), 30
read() (*robot_hat.ADXL345 method*), 39
read() (*robot_hat.Grayscale_Module method*), 43
read() (*robot_hat.I2C method*), 56
read_status() (*robot_hat.Grayscale_Module method*), 43
read_voltage() (*robot_hat.ADC method*), 30
reference() (*robot_hat.Grayscale_Module method*), 43
REG_ARR (*robot_hat.PWM attribute*), 32
REG_CHN (*robot_hat.PWM attribute*), 31
REG_PSC (*robot_hat.PWM attribute*), 31
reset() (*robot_hat.Robot method*), 45
reset_mcu() (*in module robot_hat.utils*), 54
RGB_LED (*class in robot_hat*), 39
RIGHT (*robot_hat.Grayscale_Module attribute*), 43
right (*robot_hat.Motors property*), 35
Robot (*class in robot_hat*), 44
robot_hat
    module, 27
robot_hat.utils
    module, 53
run_command() (*in module robot_hat.utils*), 53

S

say() (*robot_hat.TTS method*), 52
scan() (*robot_hat.I2C method*), 56
Servo (*class in robot_hat*), 33
servo_move() (*robot_hat.Robot method*), 45
servo_write_all() (*robot_hat.Robot method*), 45
servo_write_raw() (*robot_hat.Robot method*), 45
set() (*robot_hat.fileDB method*), 55
set_is_reverse() (*robot_hat.Motor method*), 37
set_left_id() (*robot_hat.Motors method*), 35
set_left_reverse() (*robot_hat.Motors method*), 36
set_offset() (*robot_hat.Robot method*), 45
set_right_id() (*robot_hat.Motors method*), 35
set_right_reverse() (*robot_hat.Motors method*), 36
set_volume() (*in module robot_hat.utils*), 53
setup() (*robot_hat.Pin method*), 28
sound_length() (*robot_hat.Music method*), 51
sound_play() (*robot_hat.Music method*), 50
sound_play_threading() (*robot_hat.Music method*), 50
speed() (*robot_hat.Motor method*), 37
speed() (*robot_hat.Motors method*), 36
stop() (*robot_hat.Motors method*), 35
supported_lang() (*robot_hat.TTS method*), 52
SUPPORTED_LANGUAUE (*robot_hat.TTS attribute*), 52

T

tempo() (*robot_hat.Music method*), 49
time_signature() (*robot_hat.Music method*), 49
TTS (*class in robot_hat*), 52
turn_left() (*robot_hat.Motors method*), 36
turn_right() (*robot_hat.Motors method*), 36

U

Ultrasonic (*class in robot_hat.modules*), 38

V

value() (*robot_hat.Pin method*), 29

W

write() (*robot_hat.I2C method*), 56

X

X (*robot_hat.ADXL345 attribute*), 38

Y

Y (*robot_hat.ADXL345 attribute*), 38

Z

Z (*robot_hat.ADXL345 attribute*), 38