# SunFounder Robot HAT

**www.sunfounder.com**

**02.04.2024**

# Inhaltsverzeichnis

Danke, dass Sie sich für unser Robot HAT entschieden haben.

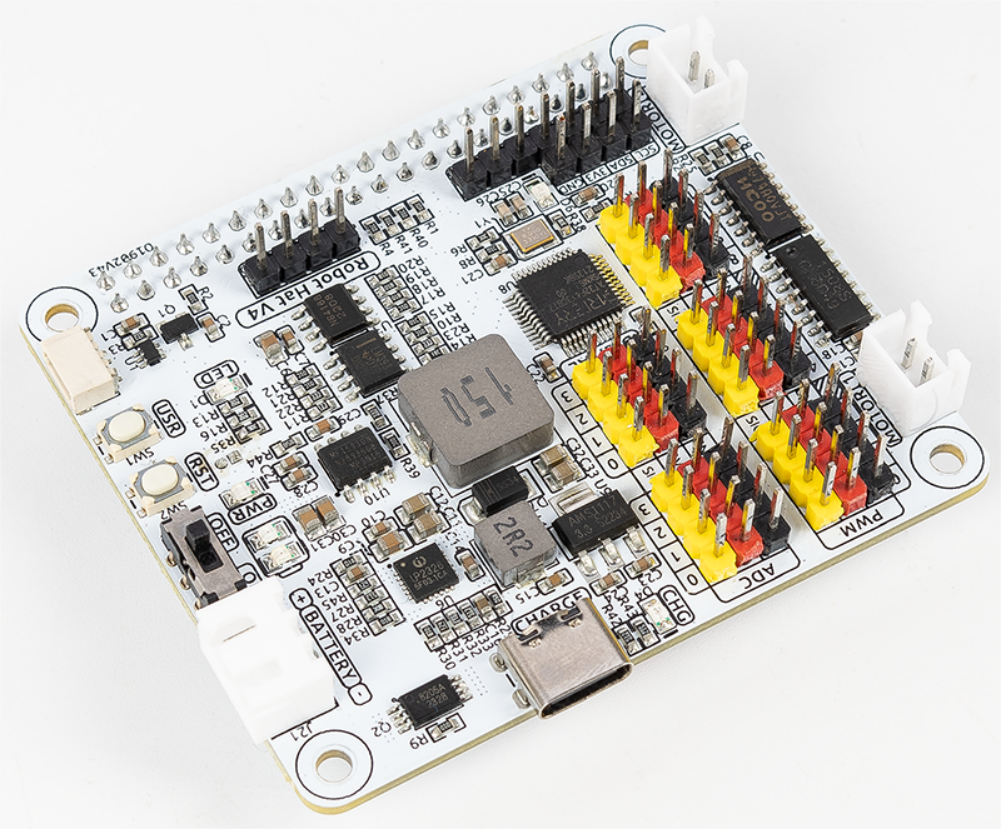
---

**Bemerkung:** Dieses Dokument ist in den folgenden Sprachen verfügbar.

- 
- 
- 

Bitte klicken Sie auf die jeweiligen Links, um das Dokument in Ihrer bevorzugten Sprache aufzurufen.

---




Das Robot HAT ist eine multifunktionale Erweiterungsplatine, die es ermöglicht, einen Raspberry Pi schnell in einen Roboter zu verwandeln. Ein MCU an Bord erweitert den PWM-Ausgang und den ADC-Eingang für den Raspberry Pi sowie einen Motorantriebschip, ein Bluetooth-Modul, ein I2S-Audiomodul und einen Monolautsprecher, sowie die GPIOs, die aus dem Raspberry Pi selbst herausführen.

Es verfügt auch über einen Lautsprecher, der zur Wiedergabe von Hintergrundmusik, Soundeffekten und zur Implementierung von TTS-Funktionen genutzt werden kann, um Ihr Projekt interessanter zu gestalten.

Akzeptiert einen 7-12V PH2.0 2-Pin-Stromanschluss mit 2 Leistungsindikatoren. Die Platine verfügt außerdem über eine benutzerzugängliche LED und einen Knopf, um schnell einige Effekte zu testen.

In diesem Dokument erhalten Sie ein vollständiges Verständnis der Schnittstellenfunktionen des Robot HAT und der Nutzung dieser Schnittstellen über die von SunFounder bereitgestellte Python-Bibliothek `robot-hat`.

KAPITEL 1

# Eigenschaften

- Abschaltstrom: < 0.5mA
- Stromversorgung: USB Typ-C, 5V/2A
- Ladeleistung: 5V/2A 10W
- Ausgangsleistung: 5V/3A
- Enthaltene Batterien: 2 x 3.7V 18650 Lithium-Ionen-Batterien, XH2.0 3P Schnittstelle
- Batterieschutz: Verpolungsschutz
- Ladeschutz: Eingangsunterspannungsschutz, Eingangsüberspannungsschutz, Ladebalance, Überhitzungsschutz
- An Bord Ladeanzeigeleuchte: CHG
- An Bord Leistungsanzeigeleuchte: PWR
- An Bord 2 Batteriestatusanzeige LEDs
- An Bord Benutzer-LED, 2 Tastschalter
- Motorsteuerung: 5V/1.8A x 2
- 4-Kanal 12-Bit-ADC
- 12-Kanal-PWM
- 4-Kanal Digitalsignale
- An Bord SPI-Schnittstelle, UART-Schnittstelle, I2C-Schnittstelle
- Mono Lautsprecher: 81W

Tab. 1: Elektrische Eigenschaften

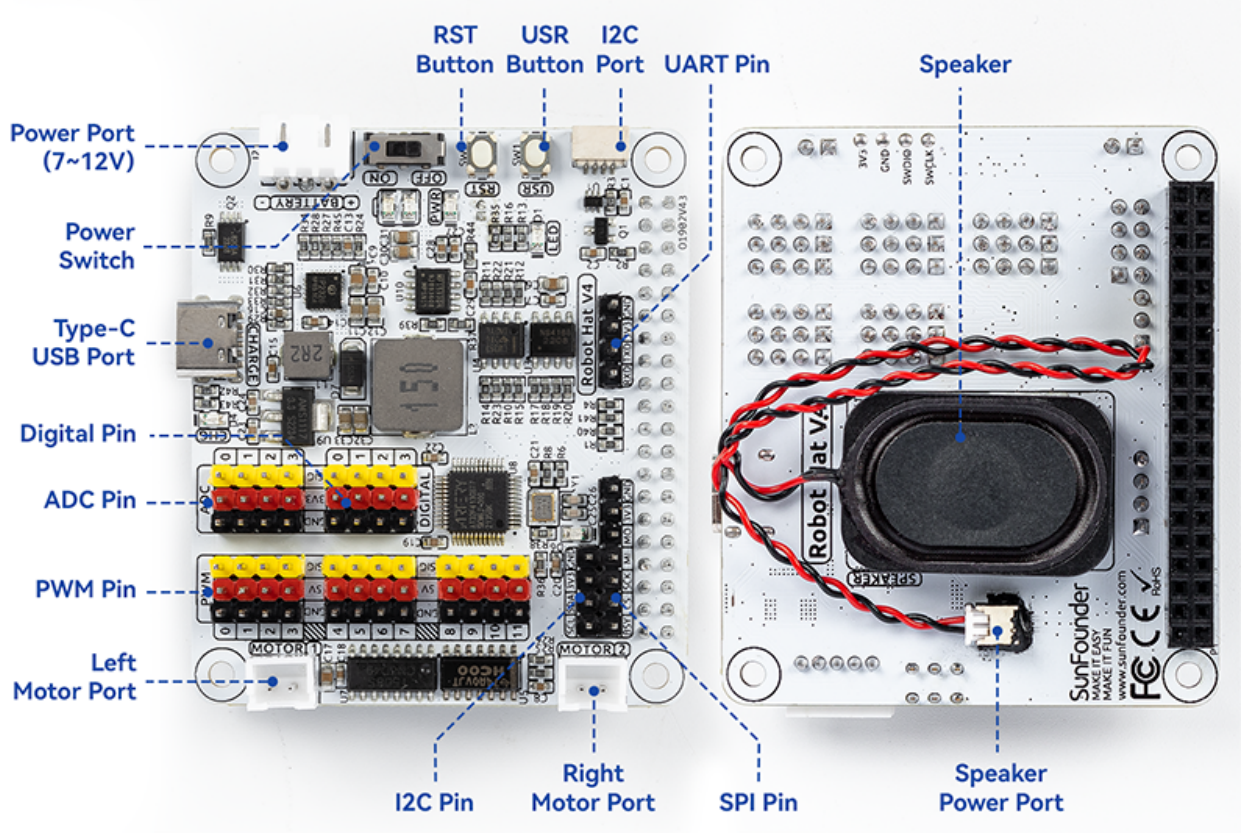
| Parameter: | Mindestwert: | Typischer Wert: | Höchstwert: | Einheit: |
|---|---|---|---|---|
| Eingangsspannung: | 4.25 | 5 | 8.4 | V |
| Batterie-Eingangsspannung: | 6.0 | 7.4 | 8.4 | V |
| Überladungsschutz (Batterie): | | 8.3 | | V |
| Eingangsunterspannungsschutz: | 4.15 | 4.25 | 4.35 | V |
| Eingangsüberspannungsschutz: | 8.3 | 8.4 | 8.5 | V |
| Ladestrom (5V): | | | 2.0 | A |
| Ausgangsstrom (5V): | | | 3.0 | A |
| Ausgangsspannung: | 5.166 | 5.246 | 5.327 | V |
| Überhitzungsschutz beim Laden: | 125 | 135 | 145 | °C |
| Überhitzungsschutz DC-DC: | 70 | 75 | 80 | °C |
| Motor-Ausgangsstrom: | | | 1.8 | A |

KAPITEL 2

# Hardware-Einführung

Der Robot Hat V4 verfügt über 2 Lithium-Batterieladungen, 5V/3A DC-DC-Entladung, I2S-Audioausgang und Lautsprecher, eine einfache Batteriestandanzeige, Mikrocontroller-basierte PWM- und ADC-Treiber sowie Motorsteuerungen.

## 2.1 Pinbelegung




**Power Port**

- 7-12V PH2.0 3-Pin-Stromeingang.
- Gleichzeitige Stromversorgung des Raspberry Pi und des Robot HAT.

**Power Switch**

- Den Strom des Robot HAT ein-/ausschalten.
- Wenn Sie den Strom an den Stromanschluss anschließen, wird der Raspberry Pi hochfahren. Sie müssen jedoch den Stromschalter auf ON stellen, um den Robot HAT zu aktivieren.

**Type-C USB Port**

- Stecken Sie das Type-C-Kabel ein, um die Batterie zu laden.
- Gleichzeitig leuchtet die Ladeanzeige in roter Farbe.
- Wenn die Batterie vollständig geladen ist, erlischt die Ladeanzeige.
- Wenn das USB-Kabel etwa 4 Stunden nach vollständiger Aufladung noch eingesteckt ist, blinkt die Ladeanzeige zur Erinnerung.

**Digital Pin**

- 4-Kanal digitale Pins, D0-D3.
- Pin: *Digitaler IO*.
- API: *Klasse Pin*.

**ADC Pin**

- 4-Kanal ADC-Pins, A0-A3.
- Pin: *ADC*.
- API: *Klasse ADC*.

**PWM Pin**

- 12-Kanal PWM-Pins, P0-P11.
- Pin: *PWM*.
- API: *Klasse PWM*.

**Left/Right Motor Port**

- 2-Kanal XH2.54 Motoranschlüsse.
- Pin: *Motoranschluss*.
- API: *Modul motor*, `1` für den linken Motoranschluss, `2` für den rechten Motoranschluss.

**I2C Pin und I2C Port**

- **I2C Pin**: P2.54 4-Pin-Schnittstelle.
- **I2C Port**: SH1.0 4-Pin-Schnittstelle, kompatibel mit QWIIC und STEMMA QT.
- Diese I2C-Schnittstellen sind über GPIO2 (SDA) und GPIO3 (SCL) mit der I2C-Schnittstelle des Raspberry Pi verbunden.
- Pin: *I2C*.
- API: *Klasse I2C*.

**SPI Pin**

- P2.54 7-Pin SPI-Schnittstelle.
- Pin: *SPI*.

**UART Pin**

- P2.54 4-Pin-Schnittstelle.
- Pin: *UART*.

**RST Button**

- Der RST-Knopf dient bei Verwendung von Ezblock als Knopf zum Neustarten des Ezblock-Programms.
- Wenn Ezblock nicht verwendet wird, hat der RST-Knopf keine vordefinierte Funktion und kann ganz nach Ihren Bedürfnissen angepasst werden.
- Pin: *Tasten*.
- API: *Klasse Pin*

**USR Button**

- Die Funktionen des USR-Knopfs können durch Ihre Programmierung festgelegt werden. (Herunterdrücken führt zu einem Eingang „0“; Loslassen erzeugt einen Eingang „1“.)
- API: *Klasse Pin*, Sie können `Pin("SW")` verwenden, um ihn zu definieren.
- Pin: *Tasten*.

**Battery Indicator**

- Zwei LEDs leuchten auf, wenn die Spannung höher als 7,6V ist.
- Eine LED leuchtet im Bereich von 7,15V bis 7,6V.
- Unter 7,15V schalten sich beide LEDs aus.
- *Batteriestandsanzeige*.

**Speaker and Speaker Port**

- **Speaker**: Dies ist ein 2030 Audio-Kammerlautsprecher.
- **Speaker Port**: Der Robot HAT ist mit einem Onboard-I2S-Audioausgang ausgestattet, zusammen mit einem 2030 Audio-Kammerlautsprecher, der einen Mono-Soundausgang bietet.
- Pin: *Lautsprecher und Lautsprecheranschluss*.
- API: *Klasse Music*

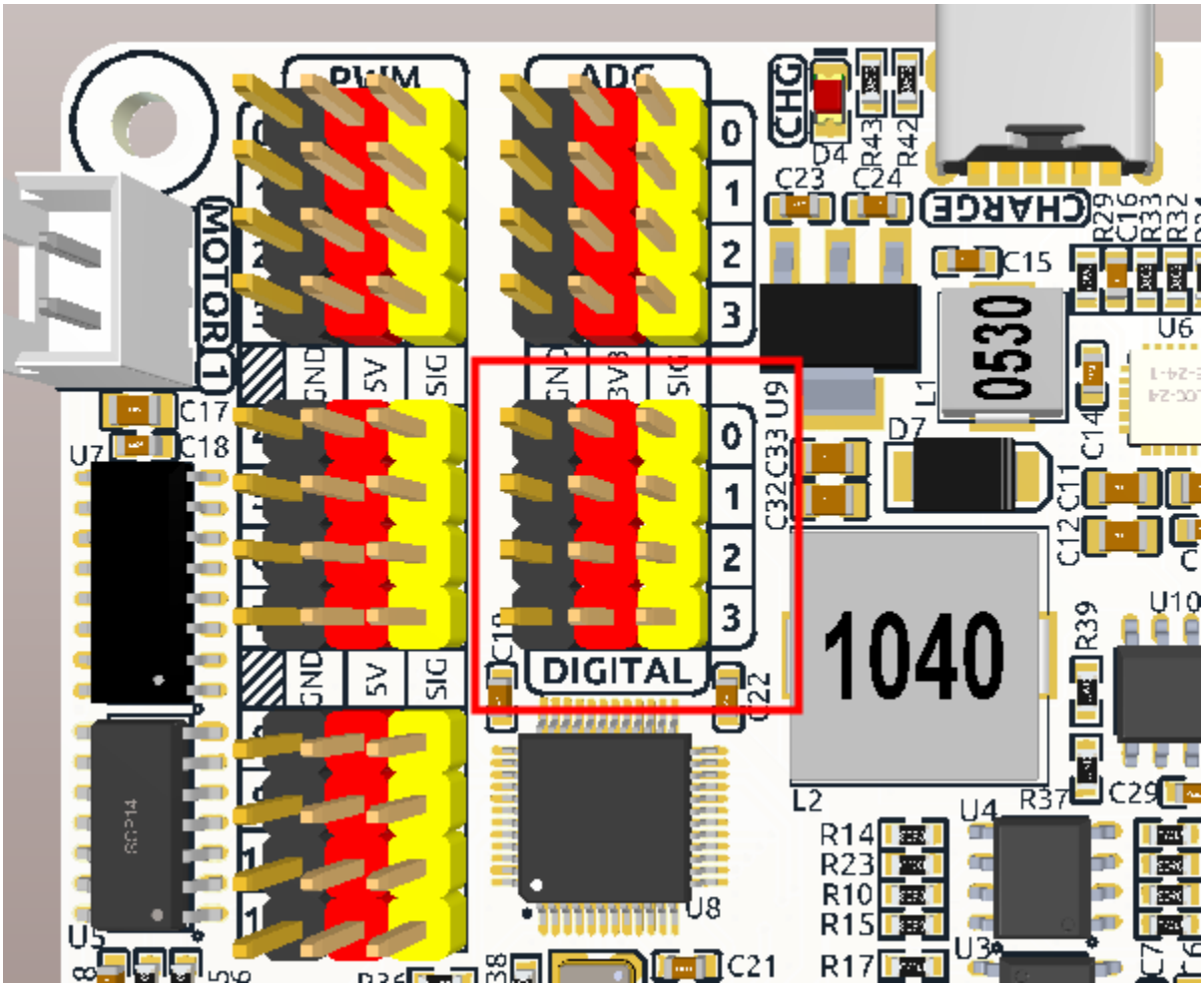
## 2.2 Pin-Belegung

Tab. 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 RICHTUNG | GPIO24 |
| SPI MOSI | MOSI | GND | GND |
| SPI MISO | MISO | USR-TASTE | 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 | Platinen-Identifikator 2 | GPIO12 |
| Platinen-Identifikator 1 | GPIO13 | GND | GND |
| I2S LRCLK | GPIO19 | RST-TASTE | GPIO16 |
| BENUTZER-LED | GPIO26 | NC | GPIO20 |
| GND | GND | I2S SDATA | GPIO21 |

## 2.3 Digitaler IO

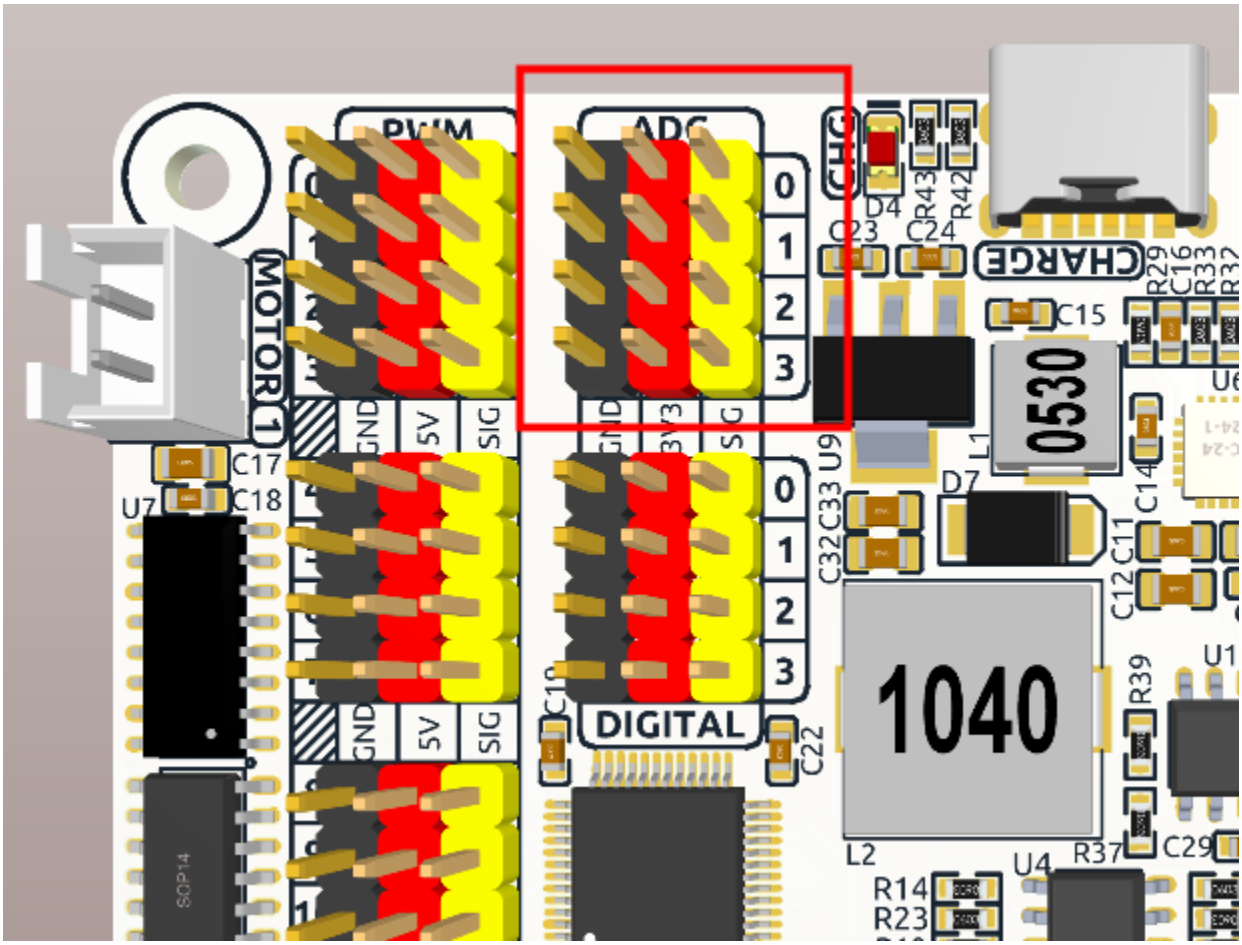
Der Robot HAT hat 4 Sätze von 3Pin digitalen Pins.



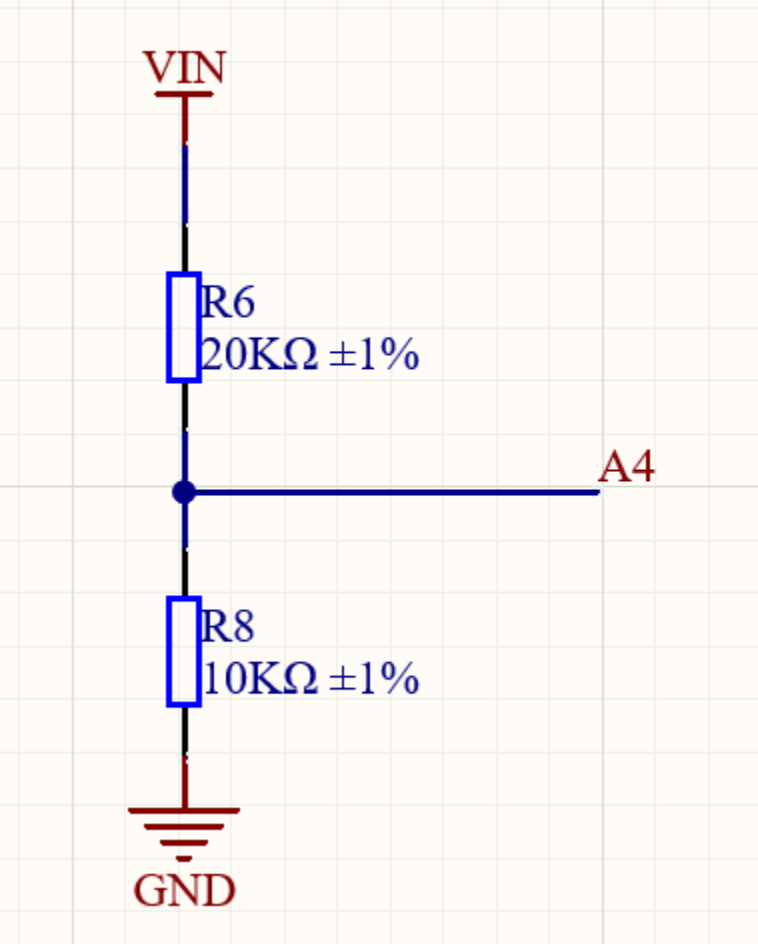

Tab. 2: Digitaler IO

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

## 2.4 ADC




Der Robot HAT verfügt über vier Sätze von 3Pin ADC (Analog-Digital-Wandler) Pins, jeweils im Abstand von 2,54 mm. Diese Pins arbeiten mit einer 3,3V Stromversorgung. Die ADC-Funktion, die eine 12-Bit-Präzision bietet, wird durch einen Mikrocontroller an Bord erleichtert. Detaillierte Anweisungen zum Lesen der ADC-Werte finden Sie im Abschnitt *On-Board-MCU*.

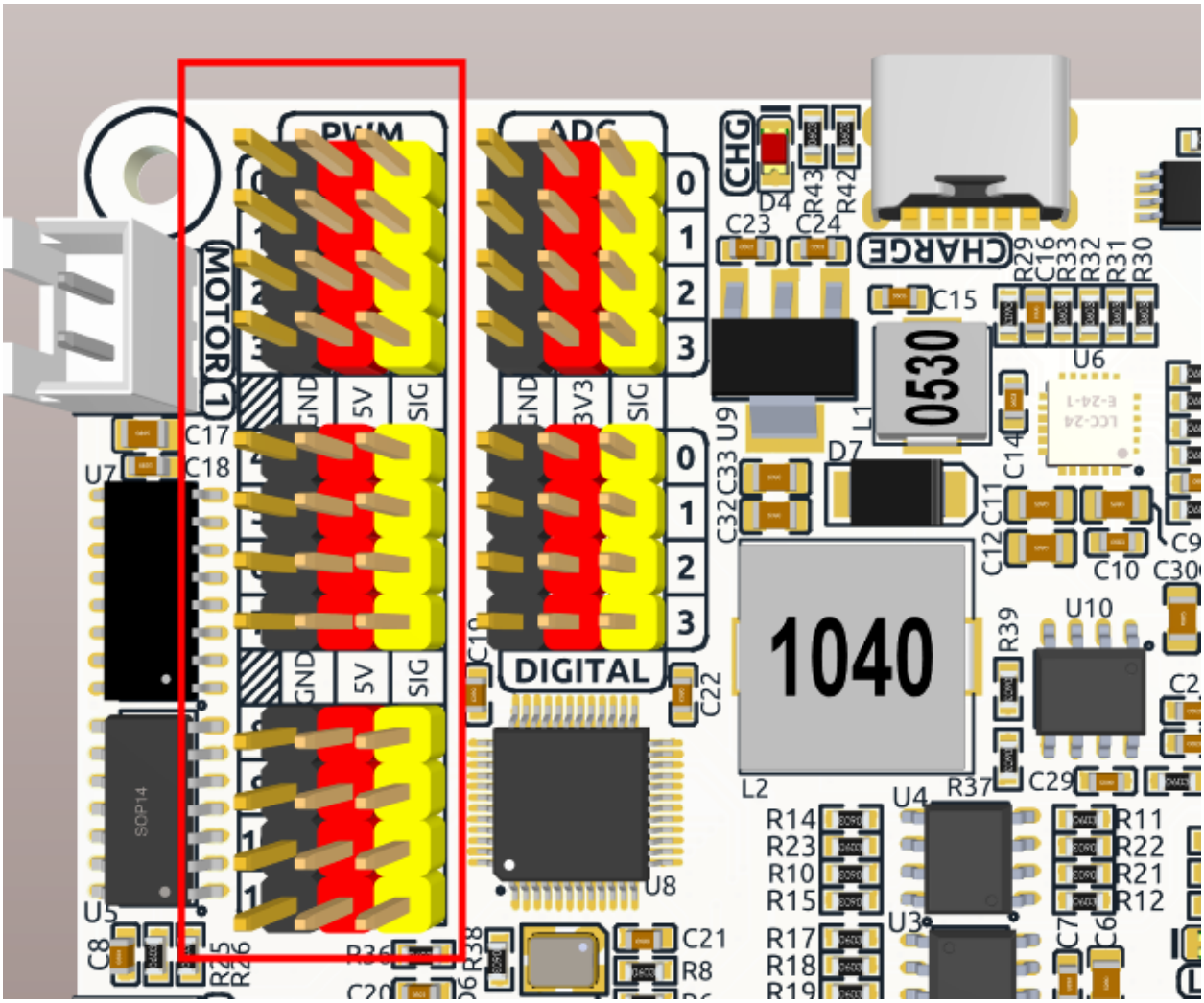



Außerdem ist der ADC-Kanal A4 über einen Spannungsteiler mit Widerständen mit der Batterie verbunden, der zur Messung der Batteriespannung zur Schätzung der ungefähren Batterieladung verwendet wird.

Das Verhältnis des Spannungsteilers beträgt 20K/10K, also:

- A4-Spannung (Va4) = Wert_A4 / 4095.0 * 3.3
- Batteriespannung (Vbat) = Va4*3
- Batteriespannung (Vbat) = Wert_A4 / 4095.0 * 3.3 * 3

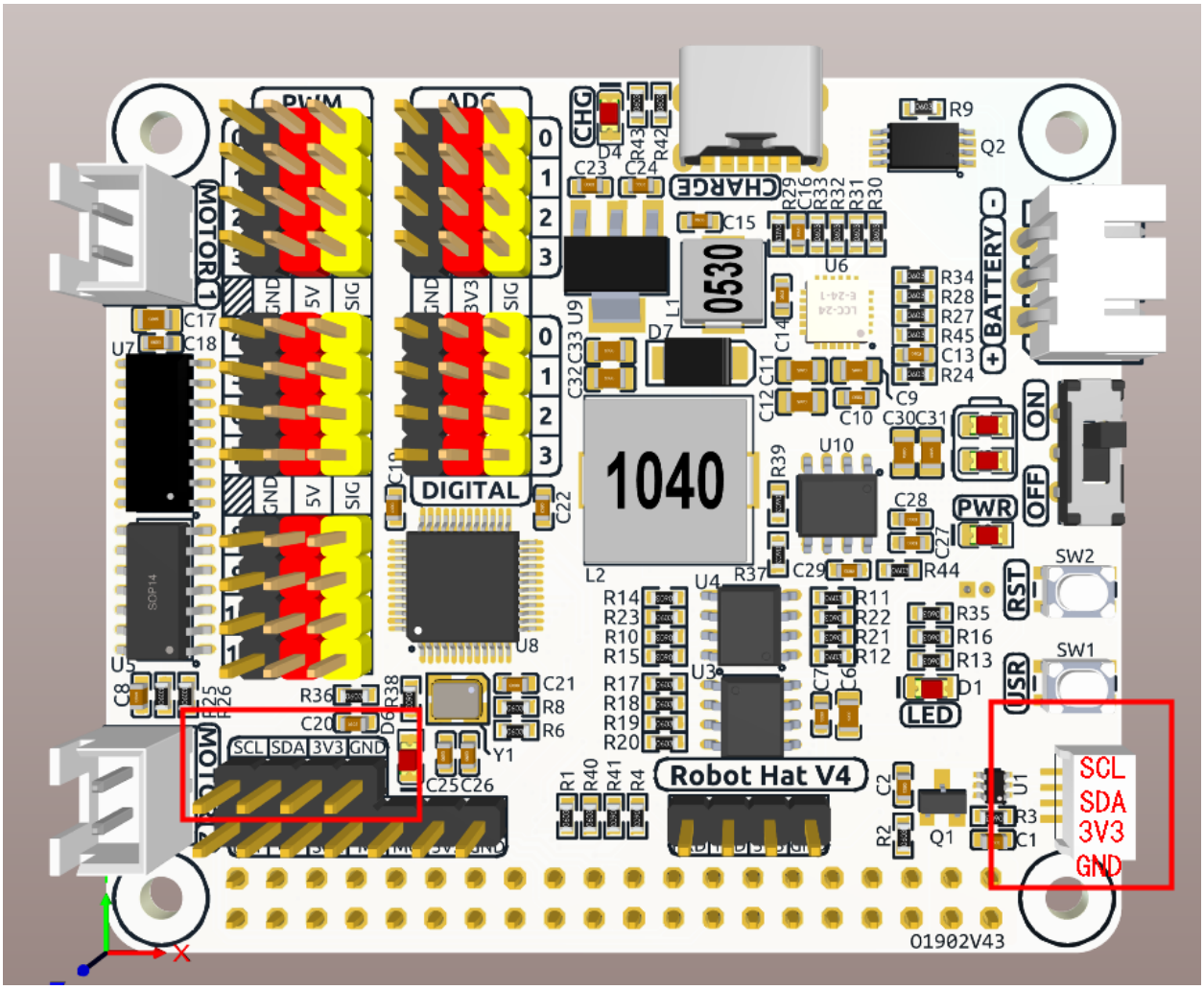
## 2.5 PWM




Der Robot HAT hat 4 Sätze von 3Pin PWM-Pins, jeweils im Abstand von 2,54 mm, und die Stromversorgung beträgt 5V. Die Methode zur Verwendung des PWM wird im Detail in *On-Board-MCU* beschrieben.

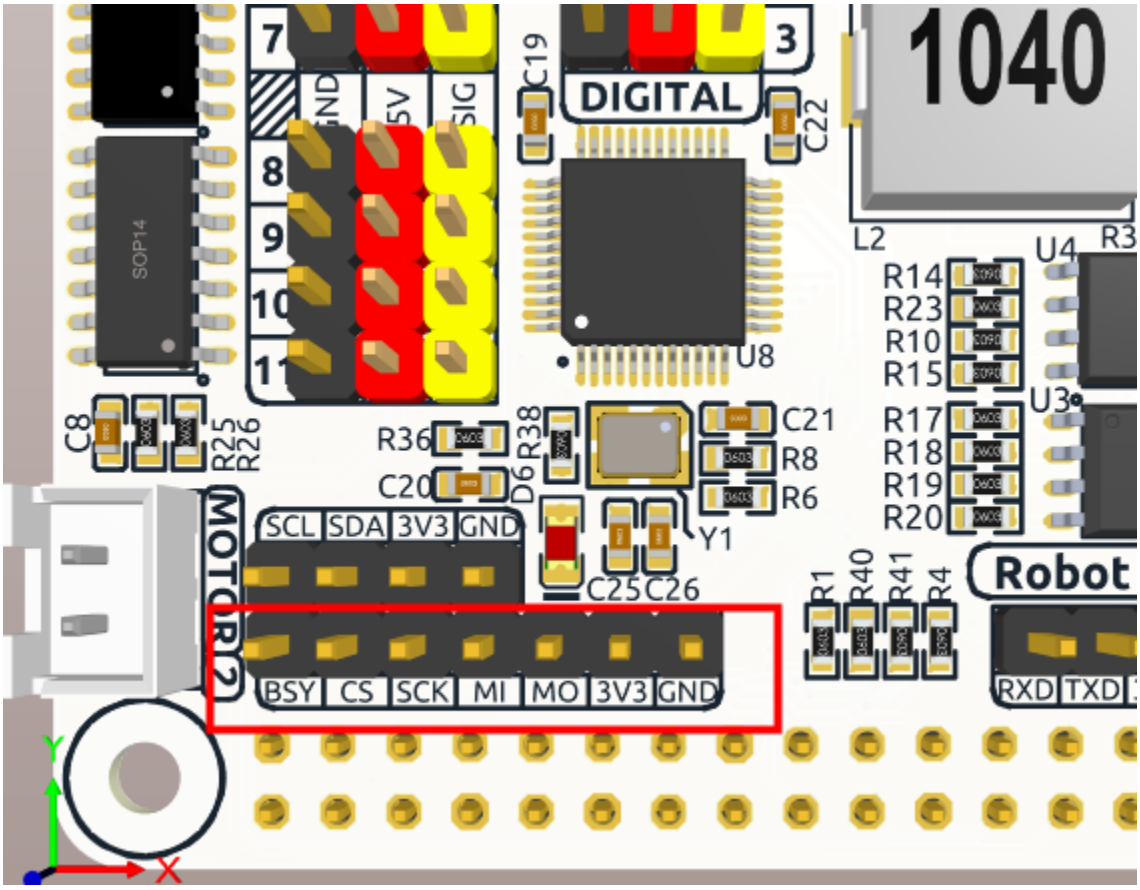
**Bemerkung:** Die Kanäle PWM13 & 14 werden für den Motorantrieb verwendet.

## 2.6 I2C




Der Robot HAT verfügt über zwei I2C-Schnittstellen. Eine davon ist die P2.54 4-Pin-Schnittstelle und die andere ist die SH1.0 4-Pin-Schnittstelle, die mit QWIIC und STEMMA QT kompatibel ist. Diese I2C-Schnittstellen sind über GPIO2 (SDA) und GPIO3 (SCL) mit der I2C-Schnittstelle des Raspberry Pi verbunden. Das Board verfügt auch über einen *On-Board-MCU*, und die beiden Signalleitungen haben 10K Pull-up-Widerstände.

## 2.7 SPI

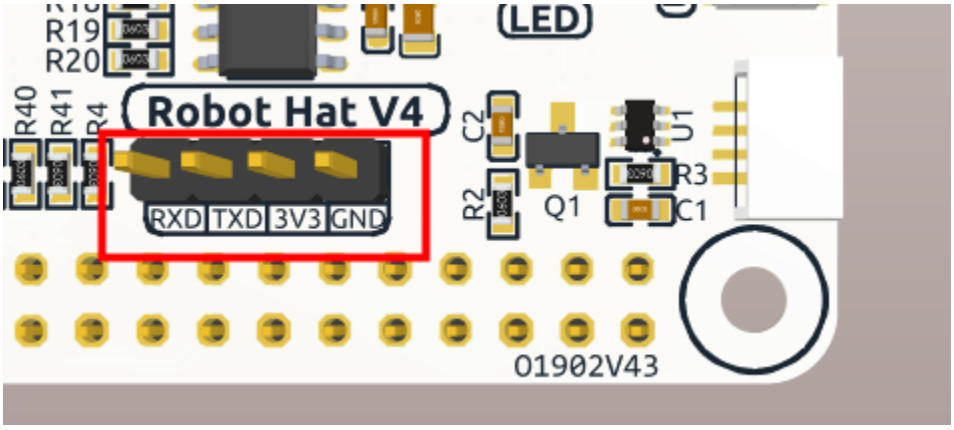



Die SPI-Schnittstelle des Robot HAT ist eine 7-Pin P2.54-Schnittstelle. Sie verbindet sich mit der SPI-Schnittstelle des Raspberry Pi und umfasst einen zusätzlichen I/O-Pin, der für Zwecke wie Interrupts oder Resets verwendet werden kann.

Tab. 3: SPI

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

## 2.8 UART




Die UART-Schnittstelle des Robot HAT ist eine 4-Pin P2.54-Schnittstelle. Sie verbindet sich mit den GPIO14 (TXD) und GPIO15 (RXD) Pins des Raspberry Pi.

## 2.9 Tasten

Der Robot HAT wird mit 1 LED und 2 Tasten geliefert, die alle direkt mit den GPIO-Pins des Raspberry Pi verbunden sind. Die RST-Taste dient bei der Verwendung von Ezblock als Taste zum Neustart des Ezblock-Programms. Wird Ezblock nicht verwendet, hat die RST-Taste keine vordefinierte Funktion und kann ganz nach Ihren Bedürfnissen angepasst werden.

Tab. 4: LED & Taste

| Robot Hat V4 | Raspberry Pi |
|---|---|
| LED | GPIO26 |
| USR | GPIO25 |
| RST | GPIO16 |

## 2.10 Lautsprecher und Lautsprecheranschluss

Der Robot HAT ist mit einem Onboard-I2S-Audioausgang sowie einem 2030 Audio-Kammerlautsprecher ausgestattet, der einen Monoklang-Ausgang bietet.

Tab. 5: I2S

| I2S | Raspberry Pi |
|---|---|
| LRCLK | GPIO19 |
| BCLK | GPIO18 |
| SDATA | GPIO21 |

## 2.11 Motoranschluss

Der Motortreiber des Robot HAT unterstützt 2 Kanäle und kann mit 2 digitalen Signalen zur Richtungssteuerung und 2 PWM-Signalen zur Geschwindigkeitsregelung gesteuert werden.

Tab. 6: Motortreiber

| Motor | IO |
|---|---|
| Motor1 Richtung | GPIO23 |
| Motor1 Leistung | PWM13 |
| Motor2 Richtung | GPIO24 |
| Motor2 Leistung | PWM12 |

## 2.12 Batteriestandsanzeige

Die Batteriestandsanzeige auf dem Robot HAT überwacht die Batteriespannung mithilfe einer Spannungsteiler-Methode und dient als Referenz zur Schätzung des Batteriestands. Die Beziehung zwischen der LED und der Spannung ist wie folgt:

Tab. 7: Batteriestand

| LED Batterie | Gesamtspannung |
|---|---|
| 2 LEDs an | Mehr als 7,6V |
| 1 LED an | Mehr als 7,15V |
| Beide LEDs aus | Weniger als 7,15V |

Wenn eine der Batterien 4,1V erreicht oder überschreitet, während die anderen darunter liegen, wird der Ladestrom dieser spezifischen Batterie reduziert.

KAPITEL 3

# Über den Akku

**Akku**


- **VCC**: Positiver Batteriepol, hier gibt es zwei Sätze von VCC und GND, um den Strom zu erhöhen und den Widerstand zu verringern.

- **Middle**: Um die Spannung zwischen den beiden Zellen auszugleichen und so den Akku zu schützen.
- **GND**: Negativer Batteriepol.

Dies ist ein benutzerdefinierter Akkupack von SunFounder, bestehend aus zwei 18650 Akkus mit einer Kapazität von 2000mAh. Der Anschluss ist PH2.0-3P, der direkt nach dem Einsetzen in das Shield geladen werden kann.

**Eigenschaften**

- Akkuladung: 5V/2A
- Akku-Ausgang: 5V/5A
- Akkukapazität: 3.7V 2000mAh x 2
- Akkulaufzeit: 90min
- Ladezeit des Akkus: 130min
- Anschluss: PH2.0, 3P

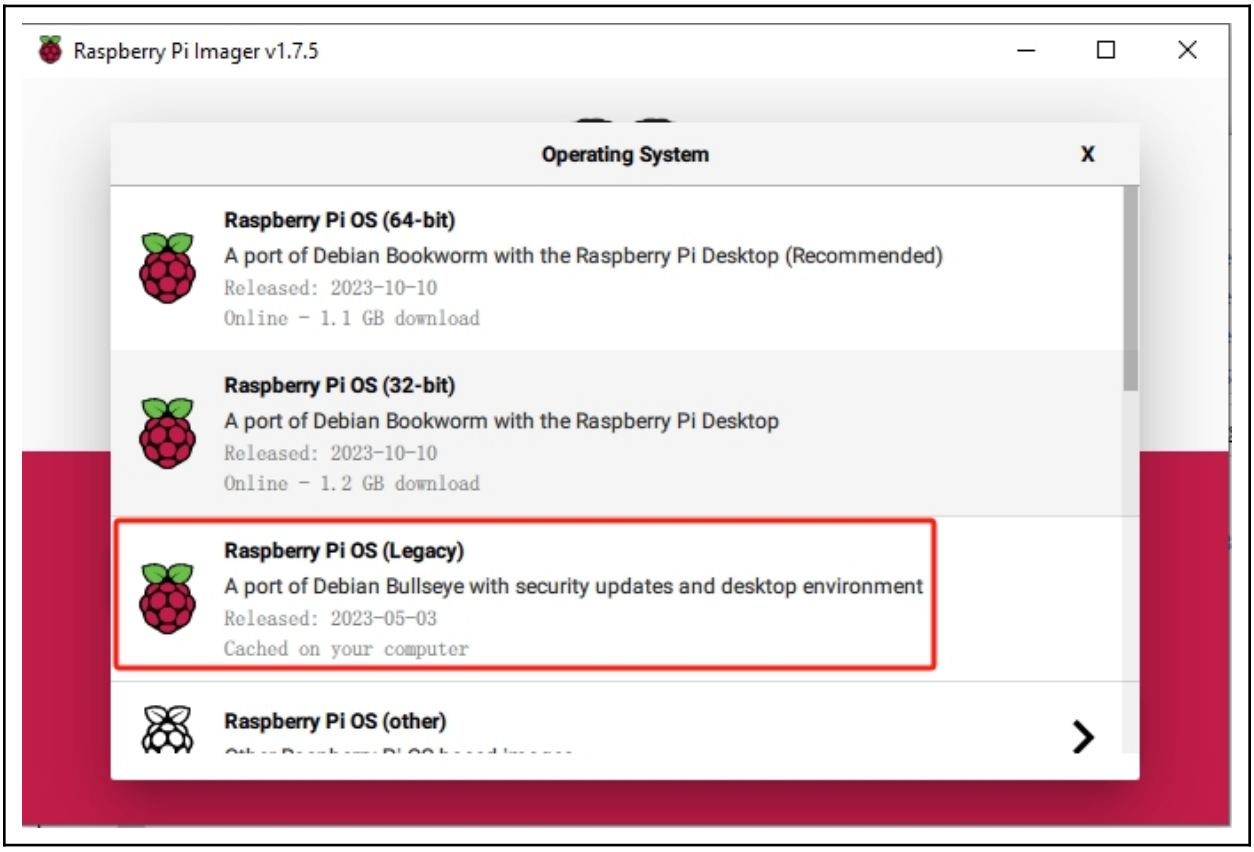
KAPITEL 4

# Installieren Sie das Modul `robot-hat`

`robot-hat` ist die unterstützte Bibliothek für den Robot HAT.

**Warnung:**

- Bei der Installation des Raspberry Pi OS verwenden Sie bitte die Version **Raspberry Pi OS (Legacy) - Debian Bullseye**.
- Wenn Sie die Version **Bookworm** installieren, funktioniert der **Speaker** möglicherweise nicht korrekt.




1. Update your system.

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

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

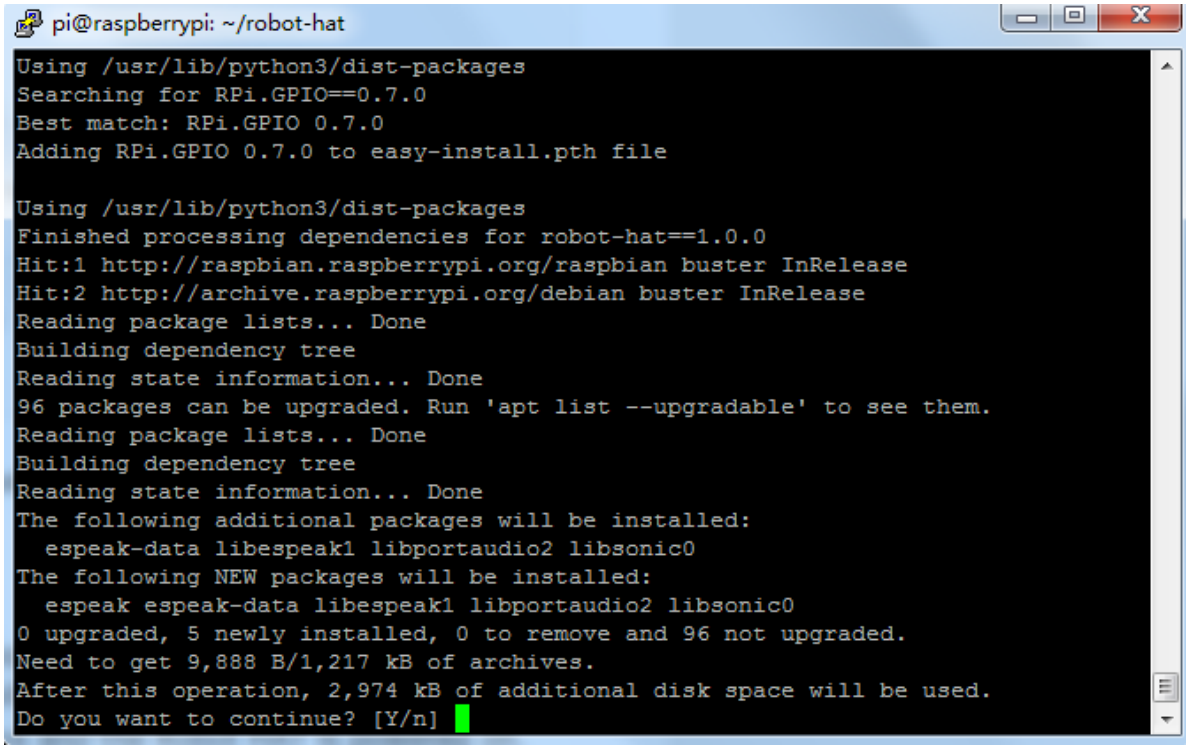
   **Bemerkung:** Python3 related packages must be installed if you are installing the **Lite** version OS.

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

2. Geben Sie diesen Befehl im Terminal ein, um das `robot-hat` Paket zu installieren.

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

   **Bemerkung:** Führen Sie `setup.py` aus, um einige notwendige Komponenten herunterzuladen. Es könnte ein Netzwerkproblem auftreten und der Download fehlschlagen. In diesem Fall müssen Sie möglicherweise erneut herunterladen. Geben Sie in den folgenden Fällen `Y` ein und drücken Sie `Enter`, um den Prozess fortzusetzen.


pi@raspberrypi: ~/robot-hat
Using /usr/lib/python3/dist-packages
Searching for RPi.GPIO==0.7.0
Best match: RPi.GPIO 0.7.0
Adding RPi.GPIO 0.7.0 to easy-install.pth file
Using /usr/lib/python3/dist-packages
Finished processing dependencies for robot-hat==1.0.0
Hit:1 http://raspbian.raspberrypi.org/raspbian buster InRelease
Hit:2 http://archive.raspberrypi.org/debian buster InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
96 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
espeak-data libespeak1 libportaudio2 libsonic0
The following NEW packages will be installed:
espeak espeak-data libespeak1 libportaudio2 libsonic0
0 upgraded, 5 newly installed, 0 to remove and 96 not upgraded.
Need to get 9,888 B/1,217 kB of archives.
After this operation, 2,974 kB of additional disk space will be used.
Do you want to continue? [Y/n]

KAPITEL 5

# Installieren Sie `i2samp.sh` für den Lautsprecher

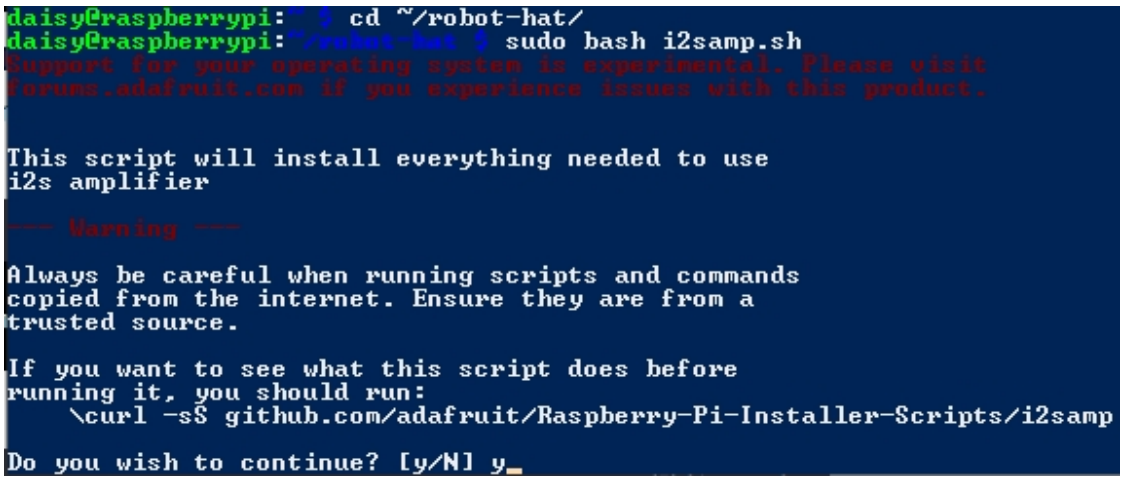
Das `i2samp.sh` ist ein ausgefeiltes Bash-Skript, das speziell für die Einrichtung und Konfiguration eines I2S (Inter-IC Sound) Verstärkers auf Raspberry Pi und ähnlichen Geräten entwickelt wurde. Unter der MIT-Lizenz lizenziert, gewährleistet es die Kompatibilität mit einer Vielzahl von Hardware- und Betriebssystemen und führt gründliche Überprüfungen durch, bevor mit irgendeiner Installation oder Konfiguration fortgefahren wird.

Wenn Sie möchten, dass Ihr Lautsprecher ordnungsgemäß funktioniert, müssen Sie dieses Skript definitiv installieren.

Die Schritte sind wie folgt:

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

Geben Sie `y` ein und drücken Sie `Enter`, um das Skript weiter auszuführen.




Geben Sie `y` ein und drücken Sie `Enter`, um `/dev/zero` im Hintergrund auszuführen.

```
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
```

Geben Sie `y` ein und drücken Sie `Enter`, um den Raspberry Pi neu zu starten.


```
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
```


**Warnung:** Wenn nach dem Neustart kein Ton vorhanden ist, müssen Sie möglicherweise das `i2samp.sh`-Skript mehrmals ausführen.

KAPITEL 6

# On-Board-MCU

Der Robot HAT ist mit einem AT32F415CBT7 Mikrocontroller von Artery ausgestattet. Es handelt sich um einen ARM Cortex-M4 Prozessor mit einer maximalen Taktfrequenz von 150MHz. Der Mikrocontroller verfügt über 256KB Flash-Speicher und 32KB SRAM.

Das Onboard-PWM und ADC werden vom Mikrocontroller gesteuert. Die Kommunikation zwischen dem Raspberry Pi und dem Mikrocontroller erfolgt über die I2C-Schnittstelle. Die für die Kommunikation verwendete I2C-Adresse ist 0x14 (7-Bit-Adressformat).

## 6.1 Einführung

Der RESET-Pin des Onboard-MCU ist mit dem Raspberry Pi GPIO 5 oder `MCURST` für *robot_hat.Pin* verbunden. Der MCU verwendet die 7-Bit-Adresse `0x14`.

## 6.2 ADC

Registeradressen sind 3 Byte, von 0x170000 bis 0x140000 sind es ADC-Kanäle 0 bis 3. Die ADC-Präzision beträgt 12 Bit, und der Wert liegt zwischen 0 und 4095. Weitere Einzelheiten finden Sie in *robot_hat.ADC*.

| Adresse | Beschreibung |
|---|---|
| `0x170000` | ADC-Kanal 0 |
| `0x160000` | ADC-Kanal 1 |
| `0x150000` | ADC-Kanal 2 |
| `0x140000` | ADC-Kanal 3 |
| `0x130000` | ADC-Kanal 4 (Batteriestand) |

**Beispiel:**

ADC-Wert des Kanals 0 lesen:

```
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 hat ein 1-Byte-Register mit 2-Byte-Werten.

### 6.3.1 Ändern der PWM-Frequenz

Die Frequenz wird mit Vorwähler und Periode definiert.

Um die Frequenz einzustellen, müssen Sie zuerst die gewünschte Periode definieren. Wie bei Arduino ist dies normalerweise 255, oder wie bei PCA9685 4095.

Die CPU-Taktfrequenz beträgt 72 MHz. Dann können Sie den Vorwähler aus Ihrer gewünschten Frequenz berechnen

> Vorwähler = 72MHz / (Periode + 1) / Frequenz - 1

Wenn Ihnen die Periode egal ist, gibt es eine Möglichkeit, sowohl die Periode als auch den Vorwähler aus der Frequenz zu berechnen. Siehe *robot_hat.PWM.freq()*.

### 6.3.2 Pulsbreite

Die Kontrolle der Pulsbreite des Kanals ist recht einfach, schreiben Sie einfach den Wert in das Register.

**Aber** was ist der Wert? Wenn Sie die PWM auf eine 50%-Pulsbreite einstellen möchten, müssen Sie genau wissen, was die Periode ist. Basierend auf der oben genannten Berechnung, wenn Sie die Periode auf 4095 einstellen, dann entspricht das Setzen des Puls-Wertes auf 2048 ungefähr einer 50%-Pulsbreite.

| Adresse | Beschreibung |
|---|---|
| `0x20` | PWM-Kanal 0 **Einschaltwert** einstellen |
| `0x21` | PWM-Kanal 1 **Einschaltwert** einstellen |
| `0x22` | PWM-Kanal 2 **Einschaltwert** einstellen |
| `0x23` | PWM-Kanal 3 **Einschaltwert** einstellen |
| `0x24` | PWM-Kanal 4 **Einschaltwert** einstellen |
| `0x25` | PWM-Kanal 5 **Einschaltwert** einstellen |
| `0x26` | PWM-Kanal 6 **Einschaltwert** einstellen |
| `0x27` | PWM-Kanal 7 **Einschaltwert** einstellen |
| `0x28` | PWM-Kanal 8 **Einschaltwert** einstellen |
| `0x29` | PWM-Kanal 9 **Einschaltwert** einstellen |
| `0x2A` | PWM-Kanal 10 **Einschaltwert** einstellen |
| `0x2B` | PWM-Kanal 11 **Einschaltwert** einstellen |
| `0x2C` | Geschwindigkeit von Motor 2 **Einschaltwert** einstellen |
| `0x2D` | Geschwindigkeit von Motor 1 **Einschaltwert** einstellen |

### 6.3.3 Vorteiler

Register ab 0x40 dienen zur Einstellung des PWM-Vorteilers. Der Bereich umfasst 0~65535. Es gibt nur 4 Timer für alle 14 Kanäle. Siehe *PWM Timer(WICHTIG)*

| Adresse | Beschreibung |
|---|---|
| `0x40` | Timer 0 **Vorteiler** einstellen |
| `0x41` | Timer 1 **Vorteiler** einstellen |
| `0x42` | Timer 2 **Vorteiler** einstellen |
| `0x43` | Timer 3 **Vorteiler** einstellen |

### 6.3.4 Periode

Register ab 0x44 dienen zur Einstellung der PWM-Periode. Der Bereich umfasst 0~65535. Es gibt nur 4 Timer für alle 14 Kanäle. Siehe *PWM Timer(WICHTIG)*

| Adresse | Beschreibung |
|---|---|
| `0x44` | Timer 0 **Periode** einstellen |
| `0x45` | Timer 1 **Periode** einstellen |
| `0x46` | Timer 2 **Periode** einstellen |
| `0x47` | Timer 3 **Periode** einstellen |

### 6.3.5 PWM-Timer(WICHTIG)

Was ist ein PWM-Timer? Der PWM-Timer ist ein Werkzeug, um den PWM-Kanal für Sie ein- und auszuschalten.

Der MCU hat nur 4 Timer für PWM: Das bedeutet, Sie können die Frequenz nicht für verschiedene Kanäle mit demselben Timer einstellen.

Beispiel: Wenn Sie die Frequenz auf Kanal 0 einstellen, werden die Kanäle 1, 2, 3 beeinflusst. Wenn Sie die Frequenz von Kanal 2 ändern, werden die Kanäle 0, 1, 3 überschrieben.

Dies geschieht beispielsweise, wenn Sie sowohl einen passiven Summer (der ständig die Frequenz ändert) als auch einen Servo (der eine feste Frequenz von 50 Hz benötigt) steuern möchten. Dann sollten Sie diese auf zwei verschiedene Timer aufteilen.

| Timer | PWM-Kanal |
|---|---|
| Timer 0 | 0, 1, 2, 3 |
| Timer 1 | 4, 5, 6, 7 |
| Timer 2 | 8, 9, 10, 11 |
| Timer 3 | 12, 13 (für Motoren) |

### 6.3.6 Beispiel

```
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 MCU zurücksetzen

Derzeit liest die Firmware einen festen 3-Byte-Wert, dann kann sie ADC-Werte zurückgeben oder PWM steuern. Deshalb benötigt das ADC-Register 3 Byte, wobei die letzten 2 Byte 0 sind.

Wenn Ihr Programm mitten in der Kommunikation unterbrochen wird, kann die Firmware hängen bleiben und die Daten verschieben. Selbst wenn wir ein Timeout beim Warten auf 3-Byte-Daten haben.

Wenn dies der Fall ist, müssen Sie den MCU zurücksetzen. Um ihn zurückzusetzen, können Sie den Befehl robot_hat verwenden:

```
robot_hat reset_mcu
```

Oder Sie können es in Ihrem Python-Code tun:

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

Oder Sie können einfach den Reset-Pin (GPIO 5) für 10 ms herunterziehen und dann für weitere 10 ms wieder hochziehen, denn das ist es, was `reset_mcu` macht.

```
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)
```

KAPITEL 7

# Referenz

Robot Hat Bibliothek

## 7.1 Klasse `Pin`

**Beispiel**

```
# 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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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*)

Basisklasse: *_Basic_class*

Pin-Manipulationsklasse

**OUT = 1**

Pin-Modus Ausgang

**IN = 2**

Pin-Modus Eingang

**PULL_UP = 17**

Pin interner Pull-Up

**PULL_DOWN = 18**

Pin interner Pull-Down

**PULL_NONE = None**

Pin ohne internen Pull

**IRQ_FALLING = 33**

Pin-Interrupt abfallend

**IRQ_RISING = 34**

Pin-Interrupt abfallend

**IRQ_RISING_FALLING = 35**

Pin-Interrupt steigend und fallend

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

Einen Pin initialisieren

**Parameter**

- **pin** (*int/str*) – Pin-Nummer des Raspberry Pi
- **mode** (*int*) – Pin-Modus (IN/OUT)
- **pull** (*int*) – Pin Pull-Up/Down (PUD_UP/PUD_DOWN/PUD_NONE)

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

Den Pin einrichten

**Parameter**

- **mode** (*int*) – Pin-Modus (IN/OUT)
- **pull** (*int*) – Pin Pull-Up/Down (PUD_UP/PUD_DOWN/PUD_NONE)

**dict**(*_dict=None*)

Das Pin-Wörterbuch setzen/erhalten

**Parameter**

**_dict** (*dict*) – Pin-Wörterbuch, lassen Sie es leer, um das Wörterbuch zu erhalten

**Rückgabe**
Pin-Wörterbuch

**Rückgabetyp**
dict

`__call__`(*value*)

Pin-Wert setzen/erhalten

**Parameter**
**value** (*int*) – Pin-Wert, lassen Sie es leer, um den Wert zu erhalten (0/1)

**Rückgabe**
Pin-Wert (0/1)

**Rückgabetyp**
int

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

Pin-Wert setzen/erhalten

**Parameter**
**value** (*int*) – Pin-Wert, lassen Sie es leer, um den Wert zu erhalten (0/1)

**Rückgabe**
Pin-Wert (0/1)

**Rückgabetyp**
int

`on`()

Pin einschalten (high)

**Rückgabe**
Pin-Wert (1)

**Rückgabetyp**
int

`off`()

Pin ausschalten (low)

**Rückgabe**
Pin-Wert (0)

**Rückgabetyp**
int

`high`()

Pin auf high setzen (1)

**Rückgabe**
Pin-Wert (1)

**Rückgabetyp**
int

`low`()

Pin auf low setzen (0)

**Rückgabe**
Pin-Wert (0)

**Rückgabetyp**
int

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

Pin-Interrupt setzen

**Parameter**

- **handler** (*function*) – Callback-Funktion des Interrupt-Handlers
- **trigger** (*int*) – Interrupt-Auslöser (RISING, FALLING, RISING_FALLING)
- **bouncetime** (*int*) – Interrupt-Entprellzeit in Millisekunden

**name**()

Pin-Name abrufen

**Rückgabe**
Pin-Name

**Rückgabetyp**
str

## 7.2 Klasse ADC

**Beispiel**

```
# 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*)

Basisklasse: *I2C*

Analog-Digital-Wandler

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

Analog-Digital-Wandler

**Parameter**
**chn** (*int/str*) – Kanalnummer (0-7/A0-A7)

**read**()

ADC-Wert lesen

**Rückgabe**
ADC-Wert (0-4095)

**Rückgabetyp**
int

**read_voltage**()

ADC-Wert lesen und in Spannung umwandeln

**Rückgabe**
Spannungswert (0-3.3(V))

**Rückgabetyp**
float

## 7.3 Klasse PWM

**Beispiel**

```
# 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()}")
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# Set pulse width to 2048 which is also 50%
p1.pulse_width(2048)
```

**API**

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

Basisklasse: *I2C*

Pulsweitenmodulation (PWM)

**REG_CHN = 32**

Kanalregister-Präfix

**REG_PSC = 64**

Vorteiler-Register-Präfix

**REG_ARR = 68**

Periodenregister-Präfix

**CLOCK = 72000000.0**

Taktfrequenz

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

PWM initialisieren

**Parameter**
**channel** (*int/str*) – PWM-Kanalnummer (0-13/P0-P13)

**freq**(*freq=None*)

Frequenz setzen/abfragen, leer lassen, um die Frequenz zu erhalten

**Parameter**
**freq** (*float*) – Frequenz (0-65535) (Hz)

**Rückgabe**
Frequenz

**Rückgabetyp**
float

**prescaler**(*prescaler=None*)

Vorteiler setzen/abfragen, leer lassen, um den Vorteiler zu erhalten

**Parameter**
**prescaler** (*int*) – Vorteiler (0-65535)

**Rückgabe**
Vorteiler

**Rückgabetyp**
int

**period**(*arr=None*)

Periode setzen/abfragen, leer lassen, um die Periode zu erhalten

**Parameter**
**arr** (*int*) – Periode (0-65535)

**Rückgabe**
Periode

**Rückgabetyp**
int

**pulse_width**(*pulse_width=None*)

Pulsbreite setzen/abfragen, leer lassen, um die Pulsbreite zu erhalten

**Parameter**
**pulse_width** (*float*) – Pulsbreite (0-65535)

**Rückgabe**
Pulsbreite

**Rückgabetyp**
float

**pulse_width_percent**(*pulse_width_percent=None*)

Pulsbreitenprozentsatz setzen/abfragen, leer lassen, um den Pulsbreitenprozentsatz zu erhalten

**Parameter**
**pulse_width_percent** (*float*) – Pulsbreitenprozentsatz (0-100)

**Rückgabe**
Pulsbreitenprozentsatz

**Rückgabetyp**
float

## 7.4 Klasse Servo

**Beispiel**

```
# 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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# 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*)

Basisklasse: *PWM*

Servomotorklasse

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

Die Servomotorklasse initialisieren

**Parameter**
**channel** (*int/str*) – PWM-Kanalnummer (0-14/P0-P14)

**angle**(*angle*)

Den Winkel des Servomotors einstellen

**Parameter**
**angle** (*float*) – Winkel (-90~90)

**pulse_width_time**(*pulse_width_time*)

Die Pulsbreite des Servomotors einstellen

**Parameter**
**pulse_width_time** (*float*) – Pulsbreitenzeit (500~2500)

## 7.5 Modul motor

### 7.5.1 Klasse Motors

**Beispiel**

Initialisieren

```
# Import Motor class
from robot_hat import Motors

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

Direkte Steuerung eines Motors. Motor 1/2 entspricht der Kennzeichnung auf der Platine

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
motors[2].speed(-100)
# Stop all motors
motors.stop()
```

Einrichtung für hochrangige Steuerung, welche Funktionen von einfachen Bewegungen wie Vorwärts, Rückwärts, Links, Rechts, Stopp bis zu komplexeren wie Joysticksteuerung, Kalibrierung der Motorenrichtung usw. bereitstellt.

**Bemerkung:** All diese Einrichtungen müssen nur einmal ausgeführt werden und werden in einer Konfigurationsdatei gespeichert. Wenn Sie das nächste Mal die Klasse Motors laden, wird sie aus der Konfigurationsdatei geladen.

```
# 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
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)
```

Jetzt den Roboter steuern

```
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*)

Basisklasse: *_Basic_class*

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

Motoren mit robot_hat.motor.Motor initialisieren

**Parameter**

**db** (*str*) – Pfad zur Konfigurationsdatei

**__getitem__**(*key*)

Spezifischen Motor abrufen

**stop**()

Alle Motoren stoppen

property left

linker Motor

property right

rechter Motor

set_left_id(*id*)

Linke Motor-ID einstellen, diese Funktion muss nur einmal ausgeführt werden. Sie speichert die Motor-ID in der Konfigurationsdatei und lädt sie beim Initialisieren der Klasse.

**Parameter**
**id** (*int*) – Motor-ID (1 oder 2)

set_right_id(*id*)

Rechte Motor-ID einstellen, diese Funktion muss nur einmal ausgeführt werden. Sie speichert die Motor-ID in der Konfigurationsdatei und lädt sie beim Initialisieren der Klasse.

**Parameter**
**id** (*int*) – Motor-ID (1 oder 2)

set_left_reverse()

Linken Motor umkehren, diese Funktion muss nur einmal ausgeführt werden. Sie speichert den umgekehrten Status in der Konfigurationsdatei und lädt diesen beim Initialisieren der Klasse.

**Rückgabe**
ob derzeit umgekehrt ist

**Rückgabetyp**
bool

set_right_reverse()

Rechten Motor umkehren, diese Funktion muss nur einmal ausgeführt werden. Sie speichert den umgekehrten Status in der Konfigurationsdatei und lädt diesen beim Initialisieren der Klasse.

**Rückgabe**
ob derzeit umgekehrt ist

**Rückgabetyp**
bool

speed(*left_speed*, *right_speed*)

Motorgeschwindigkeit einstellen

**Parameter**

- **left_speed** (*float*) – Geschwindigkeit des linken Motors (-100,0 bis 100,0)
- **right_speed** (*float*) – Geschwindigkeit des rechten Motors (-100,0 bis 100,0)

forward(*speed*)

Vorwärts

**Parameter**
**speed** (*float*) – Motorgeschwindigkeit (-100,0 bis 100,0)

backward(*speed*)

Rückwärts

**Parameter**
**speed** (*float*) – Motorgeschwindigkeit (-100,0 bis 100,0)

**turn_left**(*speed*)

Links abbiegen

**Parameter**
**speed** (*float*) – Motorgeschwindigkeit (-100,0 bis 100,0)

**turn_right**(*speed*)

Rechts abbiegen

**Parameter**
**speed** (*float*) – Motorgeschwindigkeit (-100,0 bis 100,0)

### 7.5.2 Klasse Motor

**Beispiel**

```
# 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*)

Basisklasse: object

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

Einen Motor initialisieren

**Parameter**

- **pwm** (robot_hat.pwm.PWM) – PWM-Pin zur Geschwindigkeitssteuerung des Motors
- **dir** (robot_hat.pin.Pin) – Pin zur Steuerung der Motorenrichtung

**speed**(*speed=None*)

Motorgeschwindigkeit abrufen oder einstellen

**Parameter**
**speed** (*float*) – Motorgeschwindigkeit (-100,0 bis 100,0)

**set_is_reverse**(*is_reverse*)

Motor umkehren oder nicht

**Parameter**
**is_reverse** (*bool*) – Wahr oder Falsch

## 7.6 Modul modules

### 7.6.1 Klasse Ultrasonic

**Beispiel**

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

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

# 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 Klasse ADXL345

**Beispiel**

```
# 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*)

Basisklasse: *I2C*

ADXL345-Module

**X = 0**
X

**Y = 1**
Y

**Z = 2**
Z

**__init__**(**args*, *address: int = 83*, *bus: int = 1*, ***kwargs*)
ADXL345 initialisieren

**Parameter**
**address** (*int*) – Adresse des ADXL345

**read**(*axis: int = None*) → Union[float, List[float]]
Eine Achse des ADXL345 lesen

**Parameter**
**axis** (*int*) – Wert (g) einer Achse lesen, ADXL345.X, ADXL345.Y oder ADXL345.Z, None für alle Achsen

**Rückgabe**
Wert der Achse oder Liste aller Achsen

**Rückgabetyp**
float/list

### 7.6.3 Klasse RGB_LED

**Beispiel**

```
# 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*)
Einfache 3-Pin-RGB-LED

**ANODE = 1**
Gemeinsame Anode

**CATHODE = 0**
Gemeinsame Kathode

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

RGB-LED initialisieren

**Parameter**

- **r_pin** (robot_hat.PWM) – PWM-Objekt für Rot
- **g_pin** (robot_hat.PWM) – PWM-Objekt für Grün
- **b_pin** (robot_hat.PWM) – PWM-Objekt für Blau
- **common** (*int*) – RGB_LED.ANODE oder RGB_LED.CATHODE, Standard ist ANODE

**Verursacht**

- **ValueError** – wenn gemeinsam weder ANODE noch CATHODE ist
- **TypeError** – wenn r_pin, g_pin oder b_pin kein PWM-Objekt ist

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

Farbe an RGB-LED schreiben

**Parameter**

**color** (*str/int/tuple/list*) – Zu schreibende Farbe, Hex-String beginnend mit „#“, 24-Bit-Integer oder Tupel aus (Rot, Grün, Blau)

### 7.6.4 Klasse Buzzer

**Beispiel**

Importieren und Initialisieren

```
# 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"))
```

Aktives Summer-Signal

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

Passiver Summer Einfache Nutzung

```
# Play a Tone for 1 second
p_buzzer.play(music.note("C3"), duration=1)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# 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))
```

Passiver Summer Manuelle Steuerung

```
# Play a tone
p_buzzer.play(music.note("C4"))
# Pause for 1 second
time.sleep(1)
# Play another tone
p_buzzer.play(music.note("C5"))
# Pause for 1 second
time.sleep(1)
# Stop playing
p_buzzer.off()
```

Ein Lied spielen! Babyhai!

```
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*]*)

Summer initialisieren

**Parameter**

**pwm** (*robot_hat.PWM/robot_hat.Pin*) – PWM-Objekt für passiven Summer oder Pin-Objekt für aktiven Summer

**on**()

Summer einschalten

**off**()

Summer ausschalten

**freq**(*freq: float*)

Frequenz des passiven Summers einstellen

**Parameter**

**freq** (*int/float*) – Frequenz des Summers, verwenden Sie Music.NOTES, um die Frequenz der Note zu erhalten

**Verursacht**

**TypeError** – wenn auf aktiven Summer eingestellt

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

Frequenz abspielen

**Parameter**

- **freq** (*float*) – abzuspielende Frequenz, Sie können Music.note() verwenden, um die Frequenz der Note zu erhalten
- **duration** (*float*) – Dauer jeder Note in Sekunden, None bedeutet kontinuierliches Abspielen

**Verursacht**

**TypeError** – wenn auf aktiven Summer eingestellt

### 7.6.5 Klasse Grayscale_Module

**Beispiel**

```
# 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}")
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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-Kanal-Graustufen-Modul

**LEFT = 0**

Linker Kanal

**MIDDLE = 1**

Mittlerer Kanal

**RIGHT = 2**

Rechter Kanal

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

Graustufen-Modul initialisieren

**Parameter**

- **pin0** (*robot_hat.ADC/int*) – ADC-Objekt oder Integer für Kanal 0
- **pin1** (*robot_hat.ADC/int*) – ADC-Objekt oder Integer für Kanal 1
- **pin2** (*robot_hat.ADC/int*) – ADC-Objekt oder Integer für Kanal 2
- **reference** (*1*3 list, [int, int, int]*) – Referenzspannung

**reference**(*ref: list = None*) → list

Referenzwert abrufen oder einstellen

**Parameter**

**ref** (*list*) – Referenzwert, None um den Referenzwert abzurufen

**Rückgabe**

Referenzwert

**Rückgabetyp**

list

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

Zeilenstatus lesen

**Parameter**

**datas** (*list*) – Liste der Graustufendaten, falls None, vom Sensor lesen

**Rückgabe**

Liste des Zeilenstatus, 0 für Weiß, 1 für Schwarz

**Rückgabetyp**

list

**read**(*channel: int = None*) → list

einen Kanal oder alle Daten lesen

**Parameter**

**channel** (*int/None*) – zu lesender Kanal, leer lassen, um alle zu lesen. 0, 1, 2 oder Grayscale_Module.LEFT, Grayscale_Module.MIDDLE, Grayscale_Module.RIGHT

**Rückgabe**

Liste der Graustufendaten

**Rückgabetyp**
list

## 7.7 Klasse Robot

**Beispiel**

```
# 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"] = [
    [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*)

Basisklasse: *_Basic_class*

Robot-Klasse

Diese Klasse dient zum Erstellen eines Servo-Roboters mit dem Robot HAT

Es gibt eine Servo-Initialisierung, alle Servos bewegen sich mit einer spezifischen Geschwindigkeit. Servo-Offset und so weiter. Es erleichtert das Erstellen eines Roboters. Alle Pi-Serie Roboter von SunFounder verwenden diese Klasse. Schauen Sie sich diese für weitere Details an.

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

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

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

**move_list = {}**

Voreingestellte Aktionen

**max_dps = 428**

Servo maximale Grad pro Sekunde

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

Die Robot-Klasse initialisieren

**Parameter**

- **pin_list** (*list*) – Liste der Pin-Nummern [0-11]
- **db** (*str*) – Pfad zur Konfigurationsdatei
- **name** (*str*) – Robotername
- **init_angles** (*list*) – Liste der Anfangswinkel
- **init_order** (*list*) – Liste der Initialisierungsreihenfolge (Servos werden einzeln initialisiert, falls es zu einem plötzlichen starken Strom kommt, der die Spannung der Stromversorgung herunterzieht. Die Standardreihenfolge ist die Pin-Liste. In einigen Fällen benötigen Sie eine andere Reihenfolge, verwenden Sie diesen Parameter, um sie festzulegen.)

**new_list**(*default_value*)

Erstellen Sie eine Liste von Servowinkeln mit Standardwert

**Parameter**
**default_value** (*int or float*) – Standardwert der Servowinkel

**Rückgabe**
Liste der Servowinkel

**Rückgabetyp**
list

**servo_write_raw**(*angle_list*)

Servowinkel auf spezifische Rohwinkel einstellen

**Parameter**
**angle_list** (*list*) – Liste der Servowinkel

**servo_write_all**(*angles*)

Servowinkel mit ursprünglichem Winkel und Offset auf spezifische Winkel einstellen

**Parameter**
**angles** (*list*) – Liste der Servowinkel

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

Servo mit Geschwindigkeit oder bpm auf spezifische Winkel bewegen

**Parameter**

- **targets** (*list*) – Liste der Servowinkel
- **speed** (*int or float*) – Geschwindigkeit der Servobewegung
- **bpm** (*int or float*) – Schläge pro Minute

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

Führen Sie eine vordefinierte Aktion mit Bewegungsname, Schritt und Geschwindigkeit aus

**Parameter**

- **motion_name** (*str*) – Bewegung
- **step** (*int*) – Schritt der Bewegung
- **speed** (*int or float*) – Geschwindigkeit der Bewegung

**set_offset**(*offset_list*)

Offset der Servowinkel einstellen

**Parameter**
**offset_list** (*list*) – Liste der Servowinkel

**calibration**()
Alle Servos in die Ausgangsposition bewegen

**reset**()
Servo in die ursprüngliche Position zurücksetzen

## 7.8 Klasse Music

**Warnung:**

- Sie müssen sudo beim Ausführen dieses Skripts hinzufügen, falls der Lautsprecher nicht funktioniert.
- *F3: Warum kommt kein Ton aus dem Lautsprecher?*.

**Beispiel**

Initialisieren

```
# Import Music class
from robot_hat import Music

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

Töne abspielen

```
# 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))
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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))
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))
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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))
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))
```

Ton abspielen

```
# 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")
```

Musik abspielen

```
# 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**

Basisklasse: *_Basic_class*

Musik, Soundeffekte und Notenkontrolle abspielen

**NOTE_BASE_FREQ = 440**

Basisnotenfrequenz für die Berechnung (A4)

**NOTE_BASE_INDEX = 69**

Basisnotenindex für die Berechnung (A4), MIDI-kompatibel

**NOTES = [None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, 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']**

Notennamen, MIDI-kompatibel

**__init__**()

Die Basisklasse initialisieren

**Parameter**
**debug_level** (*str/int*) – Debug-Level, 0(kritisch), 1(Fehler), 2(Warnung), 3(Info) oder 4(Debug)

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

Taktart einstellen/abrufen

**Parameter**

- **top** (*int*) – Oberzahl der Taktart
- **bottom** (*int*) – Unterzahl der Taktart

**Rückgabe**
Taktart

**Rückgabetyp**
tuple

**key_signature**(*key: int = None*)

Tonart einstellen/abrufen

**Parameter**
**key** (*int/str*) – Tonart verwenden KEY_XX_MAJOR oder String „#“, „##“, „bbb“, „bbbb“

**Rückgabe**
Tonart

**Rückgabetyp**
int

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

Tempo einstellen/abrufen (Schläge pro Minute, bpm)

**Parameter**

- **tempo** (*float*) – Tempo
- **note_value** – Notenwert (1, 1/2, Music.HALF_NOTE usw.)

**Rückgabe**
Tempo

**Rückgabetyp**
int

**beat**(*beat*)

Berechnen der Schlagverzögerung in Sekunden aus dem Tempo

**Parameter**
**beat** (*float*) – Schlagindex

**Rückgabe**
Schlagverzögerung

**Rückgabetyp**
float

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

Frequenz einer Note abrufen

**Parameter**

- **note_name** (*string*) – Notenname (siehe NOTES)
- **natural** (*bool*) – ob natürliche Note

**Rückgabe**
Frequenz der Note

**Rückgabetyp**
float

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

Soundeffekt-Datei abspielen

**Parameter**
**filename** (*str*) – Name der Soundeffekt-Datei

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

Soundeffekt im Hintergrund abspielen

**Parameter**

- **filename** (*str*) – Name der Soundeffekt-Datei
- **volume** (*int*) – Lautstärke 0-100, leer lassen, um die Lautstärke nicht zu ändern

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

Musikdatei abspielen

**Parameter**

- **filename** (*str*) – Name der Sounddatei
- **loops** (*int*) – Anzahl der Wiederholungen, 0: endlos wiederholen, 1: einmal abspielen, 2: zweimal abspielen, …
- **start** (*float*) – Startzeit in Sekunden
- **volume** (*int*) – Lautstärke 0-100, leer lassen, um die Lautstärke nicht zu ändern

music_set_volume(*value*)

Musiklautstärke einstellen

**Parameter**
**value** (*int*) – Lautstärke 0-100

music_stop()

Musik stoppen

music_pause()

Musik pausieren

music_resume()

Musik fortsetzen

music_unpause()

Musikwiedergabe fortsetzen

sound_length(*filename*)

Länge des Soundeffekts in Sekunden abrufen

**Parameter**
**filename** (*str*) – Name der Soundeffekt-Datei

**Rückgabe**
Länge in Sekunden

**Rückgabetyp**
float

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

Ton-Daten zum Abspielen abrufen

**Parameter**

- **freq** (*float*) – Frequenz
- **duration** (*float*) – Dauer in Sekunden

**Rückgabe**
Ton-Daten

**Rückgabetyp**
list

play_tone_for(*freq*, *duration*)

Ton für angegebene Dauer abspielen

**Parameter**

- **freq** (*float*) – Frequenz, Sie können NOTES verwenden, um die Frequenz zu erhalten
- **duration** (*float*) – Dauer in Sekunden

## 7.9 Klasse TTS

**Warnung:**

- Sie müssen sudo hinzufügen, wenn Sie dieses Skript ausführen, falls der Lautsprecher nicht funktioniert.
- *F3: Warum kommt kein Ton aus dem Lautsprecher?*.

**Beispiel**

```
# 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*)

Basisklasse: *_Basic_class*

Text-zu-Sprache-Klasse

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

Unterstützte TTS-Sprache für pico2wave

**ESPEAK = 'espeak'**

espeak TTS-Engine

**PICO2WAVE = 'pico2wave'**

pico2wave TTS-Engine

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

TTS-Klasse initialisieren.

**Parameter**
**engine** (*str*) – TTS-Engine, TTS.PICO2WAVE oder TTS.ESPEAK

**say**(*words*)

Worte aussprechen.

**Parameter**
**words** (*str*) – Zu sprechende Worte.

**espeak**(*words*)

Worte mit espeak sprechen.

**Parameter**
**words** (*str*) – Zu sprechende Worte.

**pico2wave**(*words*)

Worte mit pico2wave sprechen.

**Parameter**
**words** (*str*) – Zu sprechende Worte.

**lang**(**value*)

Sprache einstellen/abfragen. Leer lassen, um die aktuelle Sprache zu erhalten.

**Parameter**
**value** (*str*) – Sprache.

**supported_lang**()

Unterstützte Sprache abrufen.

**Rückgabe**
Unterstützte Sprache.

**Rückgabetyp**
list

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

Espeak-Parameter einstellen.

**Parameter**

- **amp** (*int*) – Amplitude.
- **speed** (*int*) – Geschwindigkeit.
- **gap** (*int*) – Lücke.
- **pitch** (*int*) – Tonhöhe.

## 7.10 Modul utils

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

Lautstärke einstellen

**Parameter**
**value** (*int*) – Lautstärke(0~100)

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

Befehl ausführen und Status sowie Ausgabe zurückgeben

**Parameter**
**cmd** (*str*) – auszuführender Befehl

**Rückgabe**
Status, Ausgabe

**Rückgabetyp**
tuple

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

Überprüfen, ob Befehl installiert ist

**Parameter**
**cmd** (*str*) – zu überprüfender Befehl

**Rückgabe**
Wahr, wenn installiert

**Rückgabetyp**
bool

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

Wert von einem Bereich in einen anderen Bereich abbilden

**Parameter**

- **x** (*float/int*) – abzubildender Wert
- **in_min** (*float/int*) – Eingangsminimum
- **in_max** (*float/int*) – Eingangsmaximum
- **out_min** (*float/int*) – Ausgangsminimum
- **out_max** (*float/int*) – Ausgangsmaximum

**Rückgabe**
Abgebildeter Wert

**Rückgabetyp**
float/int

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

IP-Adresse abrufen

**Parameter**
**ifaces** (*list*) – Zu überprüfende Schnittstellen

**Rückgabe**
IP-Adresse oder False, wenn nicht gefunden

**Rückgabetyp**
str/False

robot_hat.utils.**reset_mcu**()

MCU auf dem Robot Hat zurücksetzen.

Dies ist hilfreich, wenn der MCU irgendwie in einer I2C-Datenübertragungsschleife steckt und der Raspberry Pi einen IOError beim Lesen des ADC, Manipulieren des PWM usw. erhält.

robot_hat.utils.**get_battery_voltage**()

Batteriespannung abrufen

**Rückgabe**
Batteriespannung (V)

**Rückgabetyp**
float

## 7.11 Klasse `fileDB`

**Beispiel**

```
# Import fileDB class
from robot_hat import fileDB

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# 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"))

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

Jetzt können Sie die Konfigurationsdatei `config` in Bash überprüfen.

```
cat config
```

**API**

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

Basen: `object`

Eine dateibasierte Datenbank.

Eine dateibasierte Datenbank zum Lesen und Schreiben von Argumenten in einer bestimmten Datei.

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

Initialisiere db_file als Datei zum Speichern der Daten.

**Parameter**

- **db** (*str*) – Die Datei zum Speichern der Daten.
- **mode** (*str*) – Der Modus der Datei.
- **owner** (*str*) – Der Besitzer der Datei.

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

Überprüfen, ob die Datei existiert, ansonsten eine erstellen.

**Parameter**

- **file_path** (*str*) – Die zu überprüfende Datei
- **mode** (*str*) – Der Modus der Datei.
- **owner** (*str*) – Der Besitzer der Datei.

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

Wert mit dem Namen der Daten abrufen

**Parameter**

- **name** (*str*) – Der Name des Arguments
- **default_value** (*str*) – Der Standardwert des Arguments

**Rückgabe**

Der Wert des Arguments

**Rückgabetyp**
str

**set**(*name*, *value*)

Wert anhand des Namens setzen. Oder eines erstellen, falls das Argument nicht existiert

**Parameter**

- **name** (*str*) – Der Name des Arguments
- **value** (*str*) – Der Wert des Arguments

## 7.12 Klasse I2C

**Beispiel**

```
# 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)
```

Für weitere Informationen zum I2C-Protokoll siehe adc.py und pwm.py

**API**

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

Basisklasse: *_Basic_class*

I2C-Bus Lese-/Schreibfunktionen

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

Initialisierung des I2C-Busses

**Parameter**

- **address** (*int*) – I2C-Geräteadresse
- **bus** (*int*) – I2C-Busnummer

**scan**()

I2C-Bus nach Geräten scannen

**Rückgabe**
Liste der gefundenen I2C-Adressen von Geräten

**Rückgabetyp**
list

`write`(*data*)

Daten an das I2C-Gerät schreiben

**Parameter**
**data** (*int/list/bytearray*) – Zu schreibende Daten

**Wirft**
ValueError, falls die Schreiboperation kein int, keine Liste oder kein Bytearray ist

`read`(*length=1*)

Daten vom I2C-Gerät lesen

**Parameter**
**length** (*int*) – Anzahl der zu empfangenden Bytes

**Rückgabe**
Empfangene Daten

**Rückgabetyp**
list

`mem_write`(*data*, *memaddr*)

Daten an eine spezifische Registeradresse senden

**Parameter**

- **data** (*int/list/bytearray*) – Zu sendende Daten, int, Liste oder Bytearray
- **memaddr** (*int*) – Registeradresse

**Verursacht**
**ValueError** – Falls die Daten kein int, keine Liste oder kein Bytearray sind

`mem_read`(*length*, *memaddr*)

Daten von einer spezifischen Registeradresse lesen

**Parameter**

- **length** (*int*) – Anzahl der zu empfangenden Bytes
- **memaddr** (*int*) – Registeradresse

**Rückgabe**
Empfangene Bytearray-Daten oder False bei einem Fehler

**Rückgabetyp**
list/False

`is_avaliable`()

Überprüfen, ob das I2C-Gerät verfügbar ist

**Rückgabe**
True, wenn das I2C-Gerät verfügbar ist, andernfalls False

**Rückgabetyp**
bool

## 7.13 Klasse _Basic_class

_Basic_class ist eine Logger-Klasse für alle Klassen zur Protokollierung. Wenn Sie Protokolle einer Klasse sehen möchten, fügen Sie einfach ein Debug-Argument hinzu.

**Beispiel**

```
# See PWM log
from robot_hat import PWM

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

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

**API**

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

Grundklasse für alle Klassen

mit Debug-Funktion

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

Debug-Level

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

Namen der Debug-Levels

**__init__**(*debug_level='warning'*)

Initialisierung der Grundklasse

**Parameter**

**debug_level** (*str/int*) – Debug-Level, 0(kritisch), 1(Fehler), 2(Warnung), 3(Info) oder 4(Debug)

**property debug_level**

Debug-Level

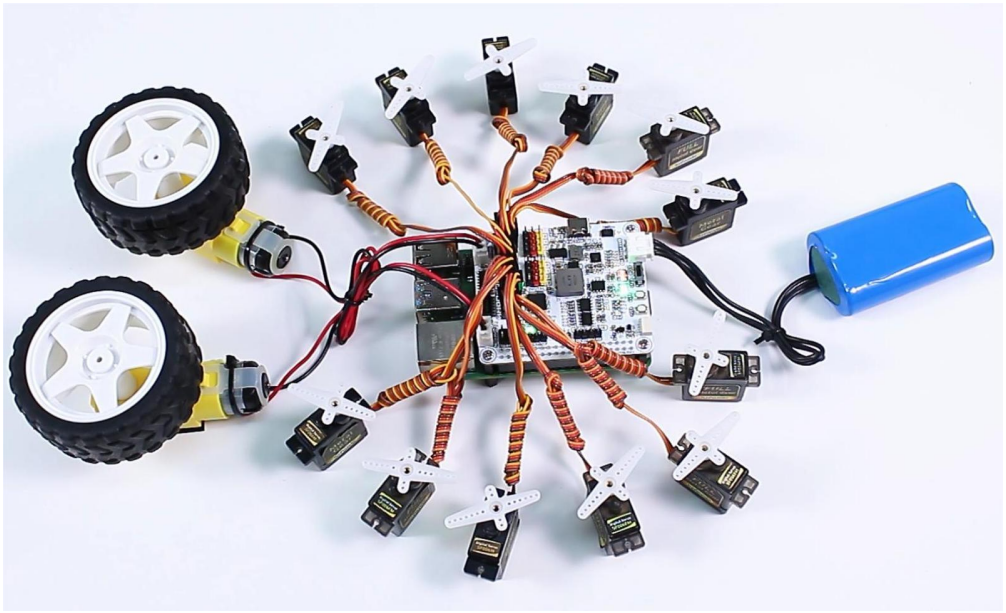
KAPITEL 8

# Einige Projekte

Hier finden Sie eine Sammlung faszinierender Projekte, alle umgesetzt mit dem Robot HAT. Wir stellen Ihnen detaillierten Code zur Verfügung, der Ihnen die Möglichkeit bietet, diese Projekte selbst auszuprobieren.

## 8.1 Servos und Motoren steuern

In diesem Projekt haben wir 12 Servos und zwei Motoren, die gleichzeitig arbeiten.



Es ist jedoch wichtig zu beachten, dass, wenn Ihre Servos und Motoren einen hohen Anlaufstrom haben, es empfohlen wird, sie separat zu starten, um unzureichenden Stromversorgungsstrom zu vermeiden, der zum Neustart des Raspberry Pi führen könnte.

**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()

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.")
```

(Fortsetzung auf der nächsten Seite)

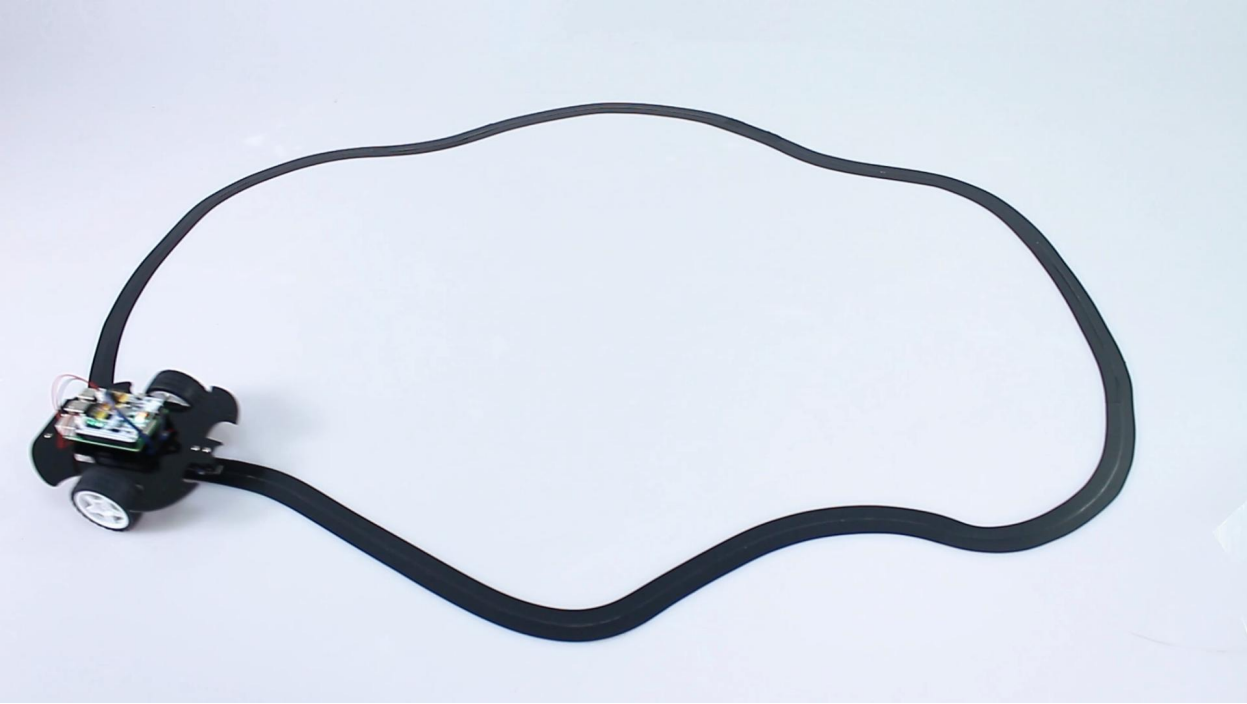
(Fortsetzung der vorherigen Seite)

```
# Initialize servos to their initial position
initialize_servos()

# Control motors and servos
control_motors_and_servos()
```

## 8.2 DIY-Auto

Neben der Eignung für einfache Experimente ist der Robot HAT ideal als zentraler Controller in der Robotik, wie zum Beispiel für intelligente Autos, geeignet.

In diesem Projekt haben wir ein einfaches linienfolgendes Auto gebaut.



**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)
```

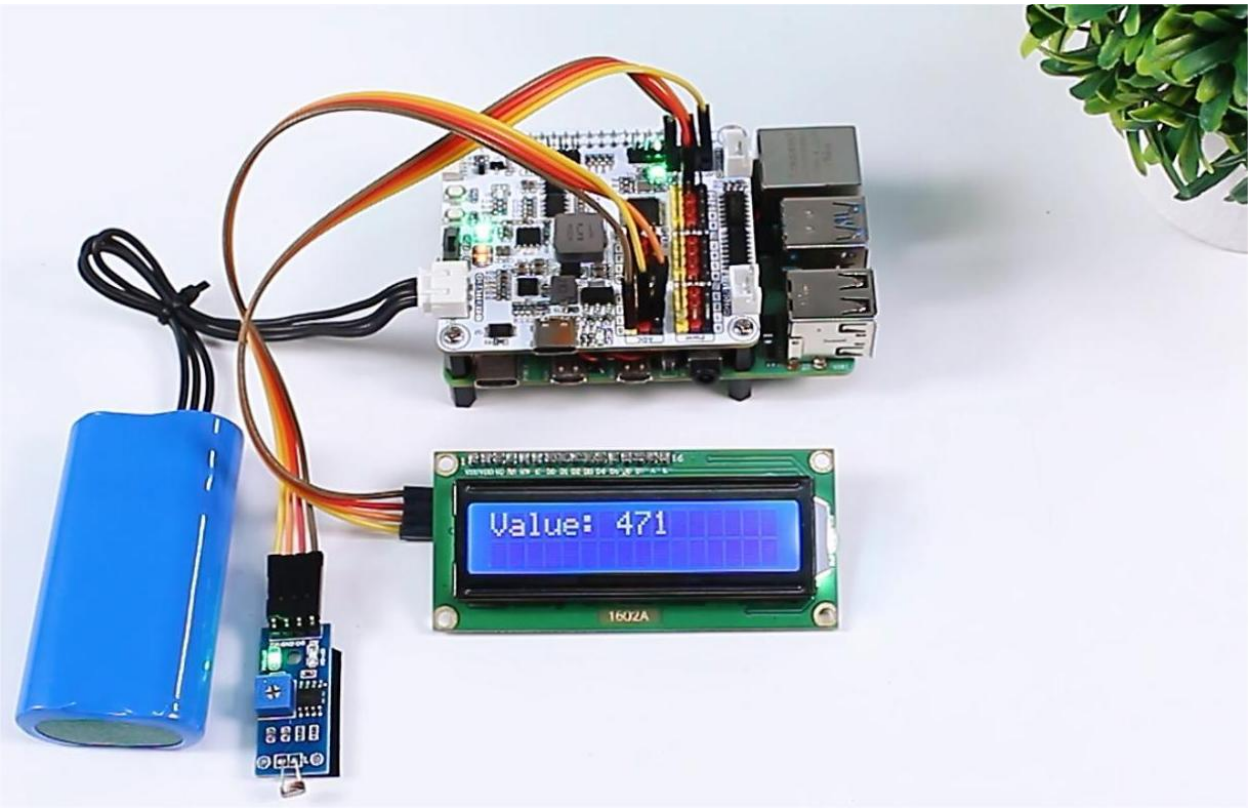
(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        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
            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 Lesen vom Fotowiderstandsmodul

In diesem Projekt erfassen wir die Lichtintensität und zeigen sie auf dem I2C LCD1602 an.



**Schritte**

1. In diesem Projekt wird ein I2C LCD1602 verwendet, daher ist es notwendig, die relevanten Bibliotheken herunterzuladen, damit es funktioniert.

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

2. Installieren Sie `smbus2` für I2C.

```
sudo pip3 install smbus2
```

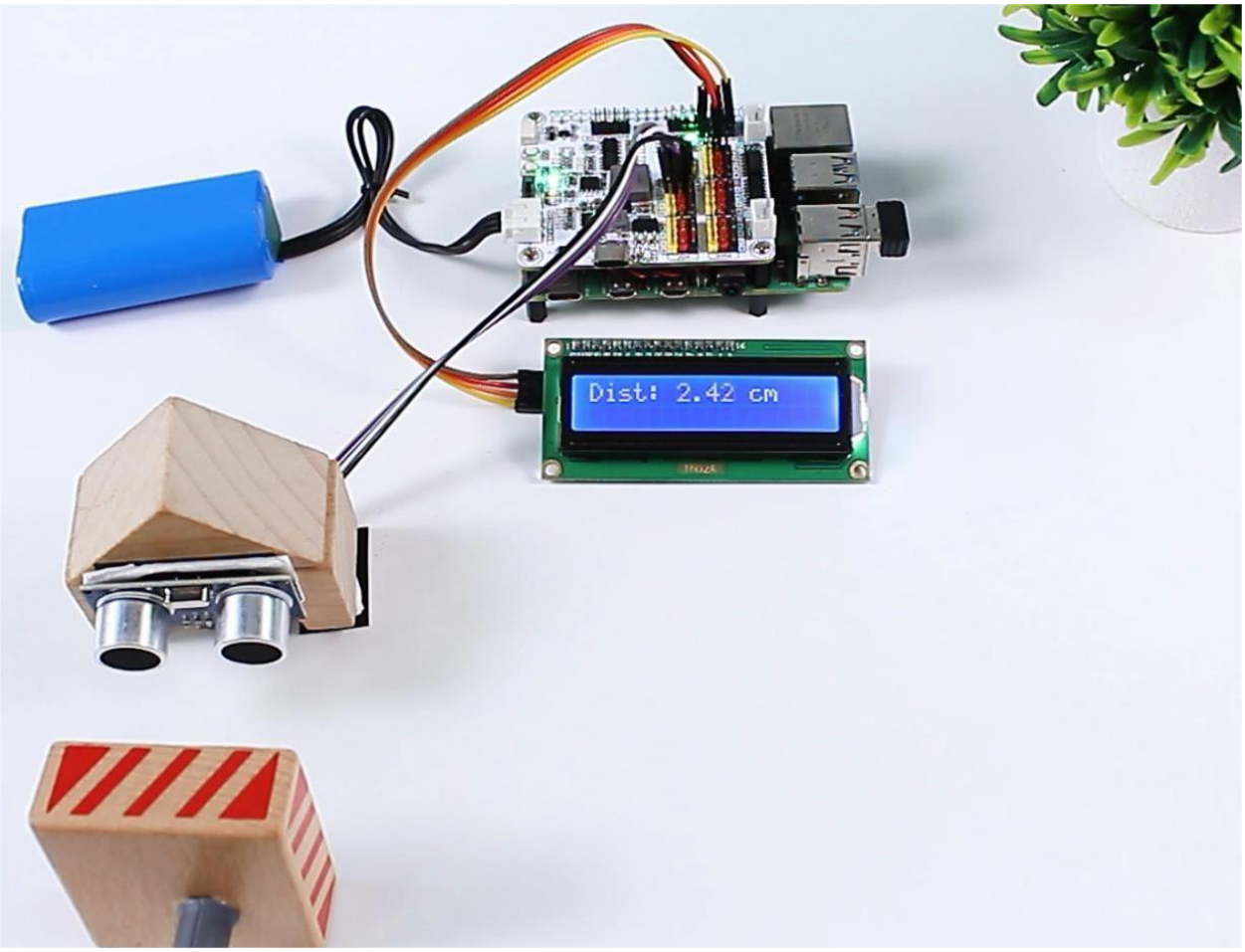
3. Speichern Sie den folgenden Code auf Ihrem Raspberry Pi und geben Sie ihm einen Namen, zum Beispiel `photoresistor.ty`.

```
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. Verwenden Sie den Befehl `sudo python3 photoresistor.ty`, um diesen Code auszuführen.

## 8.4 Lesen vom Ultraschallmodul

In diesem Projekt verwenden wir Ultraschallsensoren, um Entfernungen zu messen und die Messwerte auf dem I2C LCD1602 anzuzeigen.




**Schritte**

1. In diesem Projekt wird ein I2C LCD1602 verwendet, daher ist es notwendig, die relevanten Bibliotheken herunterzuladen, um es zum Laufen zu bringen.

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

2. Installieren Sie `smbus2` für I2C.

```
sudo pip3 install smbus2
```

3. Speichern Sie den folgenden Code auf Ihrem Raspberry Pi und geben Sie ihm einen Namen, zum Beispiel `ultrasonic.ty`.

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

(Fortsetzung auf der nächsten Seite)

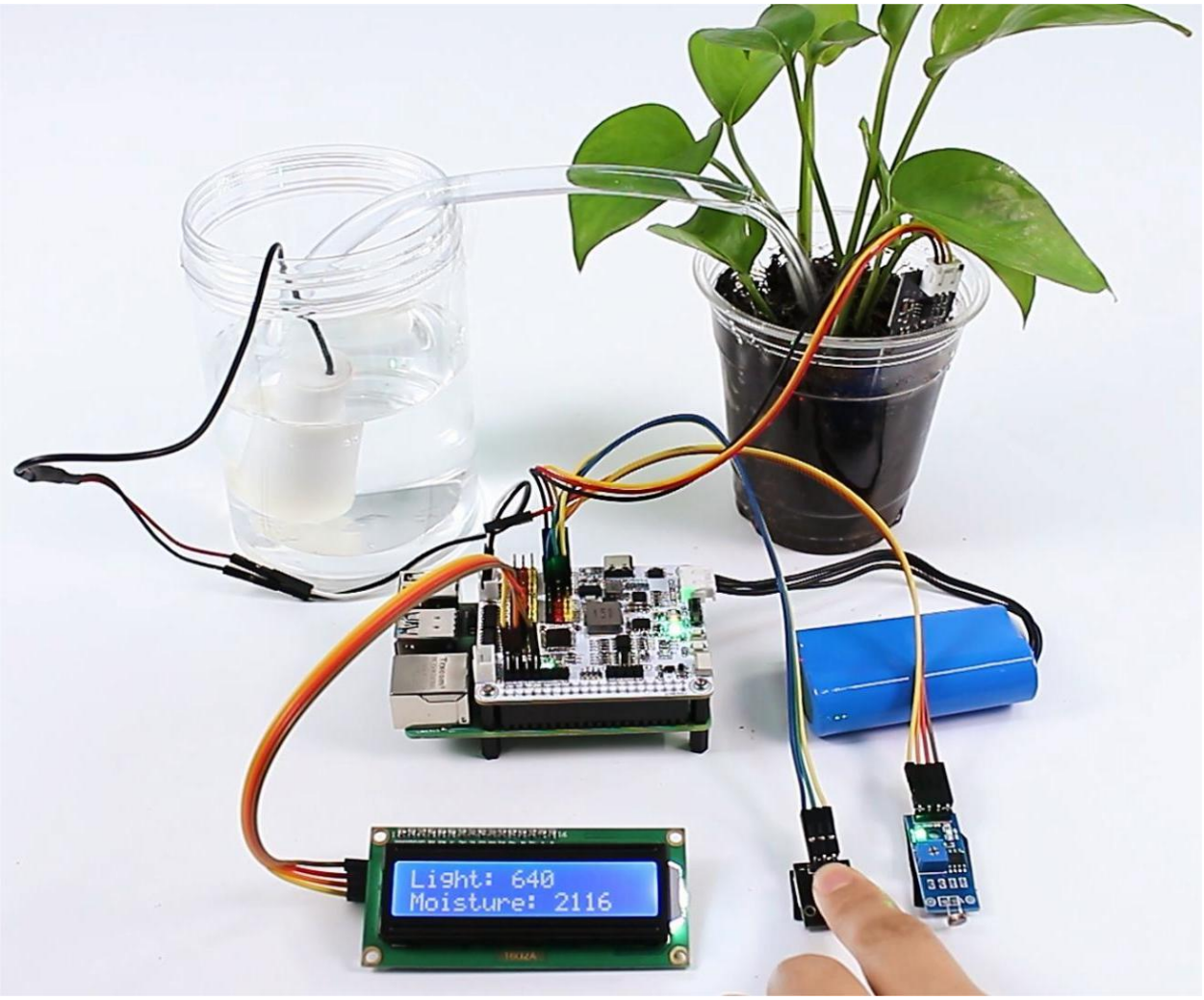
(Fortsetzung der vorherigen Seite)

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

# 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. Verwenden Sie den Befehl `sudo python3 ultrasonic.ty`, um diesen Code auszuführen.

## 8.5 Pflanzenmonitor

In diesem Projekt erfassen wir sowohl die Lichtintensität als auch den Feuchtigkeitsgehalt des Bodens und zeigen diese auf dem I2C LCD1602 an. Wenn Sie denken, dass die Bodenfeuchtigkeit unzureichend ist, können Sie das Tastenmodul drücken, um die Topfpflanze zu bewässern.




**Schritte**

1. In diesem Projekt wird ein I2C LCD1602 verwendet, daher ist es notwendig, die relevanten Bibliotheken herunterzuladen, um es zum Laufen zu bringen.

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

2. Installieren Sie `smbus2` für I2C.

```
sudo pip3 install smbus2
```

3. Speichern Sie den folgenden Code auf Ihrem Raspberry Pi und geben Sie ihm einen Namen, zum Beispiel `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()
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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    LCD1602.clear()

def loop():
    global running
    while running:
        light_value, moisture_value = read_sensors()
        update_lcd(light_value, moisture_value)
        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. Verwenden Sie den Befehl `sudo python3 plant_monitor.ty`, um diesen Code auszuführen.

## 8.6 Sage etwas

In diesem Abschnitt lernen Sie, wie Sie Text in Sprache umwandeln und vom Robot HAT laut aussprechen lassen.

**Schritte**

1. Wir holen Text von der Kommandozeile ab, damit der Robot HAT ihn artikulieren kann. Um dies zu erreichen, speichern Sie den folgenden Code als eine `.py`-Datei, wie zum Beispiel `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')
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# Read the text
tts.say(text_to_say)

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

2. Um den Robot HAT eine bestimmte Satz aussprechen zu lassen, können Sie den folgenden Befehl verwenden: `sudo python3 tts.py „any text“` - ersetzen Sie einfach `„any text“` durch den gewünschten Satz.

---

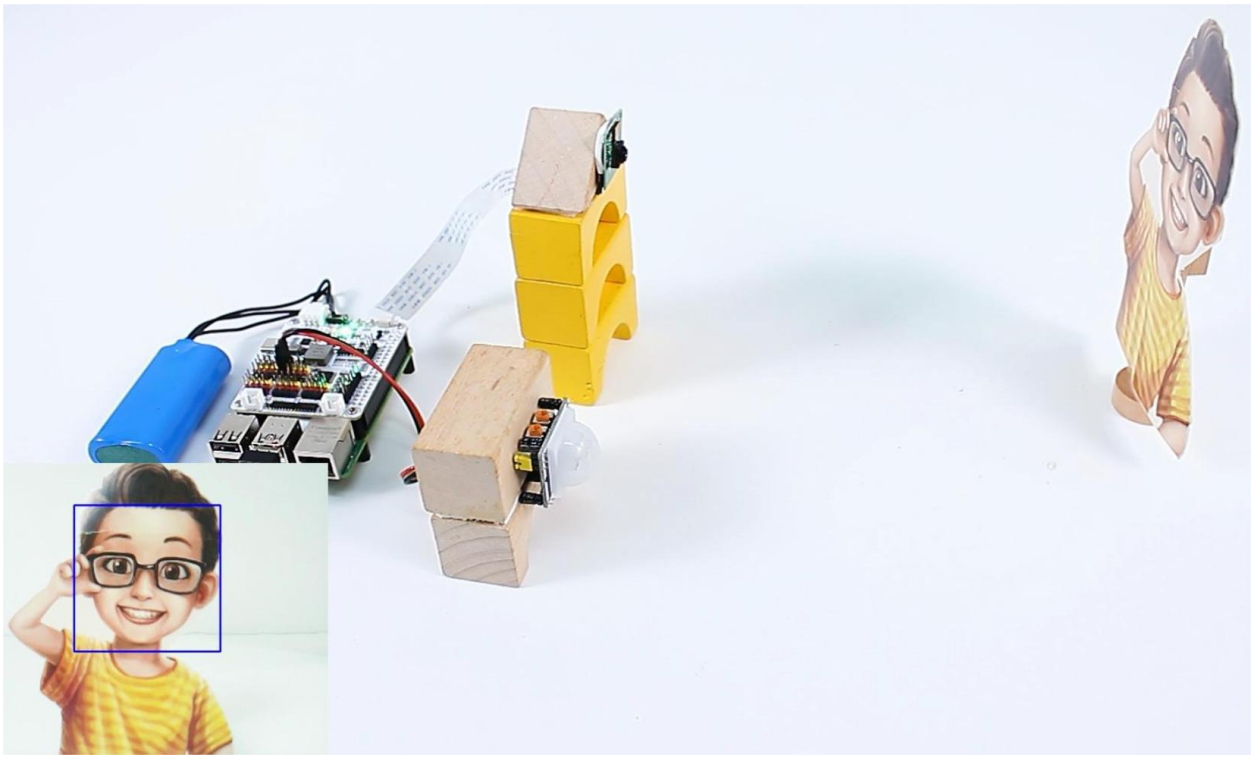
**Bemerkung:**

- *F3: Warum kommt kein Ton aus dem Lautsprecher?*

---

## 8.7 Sicherheitssystem

In diesem Projekt haben wir ein einfaches Sicherheitssystem erstellt. Der PIR-Sensor erkennt, ob jemand vorbeigeht, und dann wird die Kamera aktiviert. Wenn ein Gesicht erkannt wird, macht sie ein Bild und liefert gleichzeitig eine Warnmeldung.



**Schritte**

1. Installieren Sie die `vilib`-Bibliothek zur Gesichtserkennung.

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
cd vilib
sudo python3 install.py
```

2. Speichern Sie den folgenden Code auf Ihrem Raspberry Pi und geben Sie ihm einen Namen, zum Beispiel `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
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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

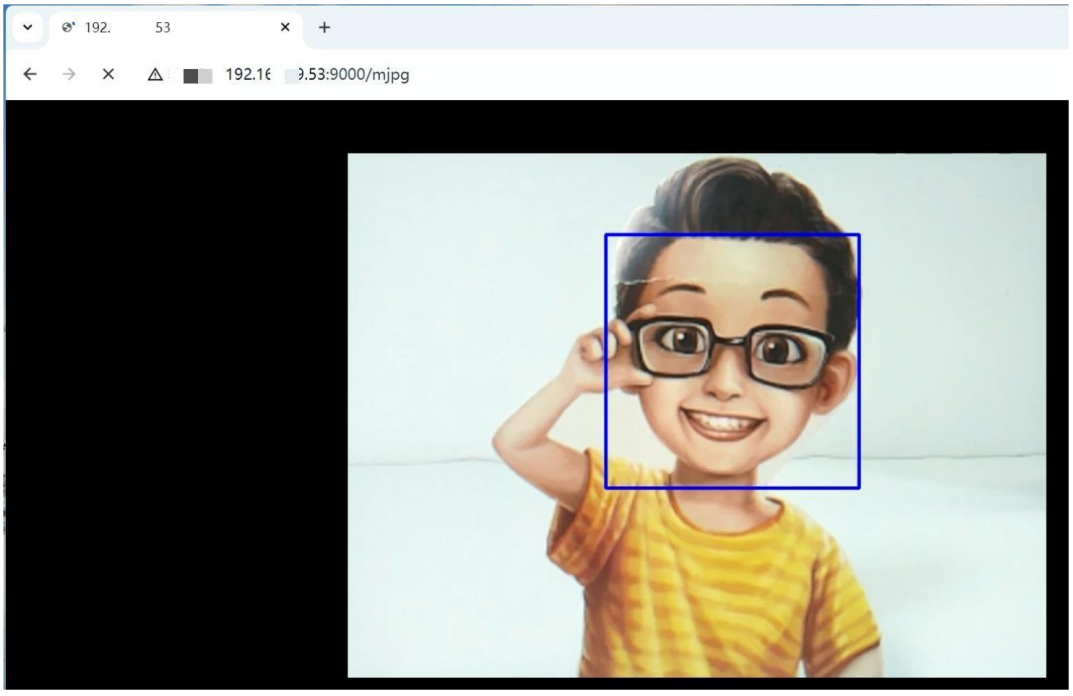
```
        # If no motion is detected, turn off the camera
        elif motion_detected:
            print("No motion detected. Finalizing camera...")
            Vilib.camera_close()
            motion_detected = False
            sleep(2)  # Delay before re-enabling motion detection

        sleep(0.1)  # Short delay to prevent CPU overuse

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

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

3. **Verwenden Sie den Befehl sudo `python3 security.py`, um diesen Code auszuführen.**

   **Bemerkung:**

   - *F3: Warum kommt kein Ton aus dem Lautsprecher?*

4. Öffnen Sie einen Webbrowser und geben Sie `http://rpi_ip:9000/mjpg` ein, um das aufgenommene Filmmaterial anzusehen. Zusätzlich finden Sie die aufgenommenen Gesichtsbilder in `/home/{Benutzername}/Bilder/`.



## 8.8 Community-Tutorials

•

Dieses Dokument fasst den SunFounder Raspberry Pi Robot HAT zusammen und beschreibt seinen Zweck, Kompatibilität, Spezifikationen und Testverfahren:

- **Einführung**: Erklärt die Rolle des Robot HAT bei der Vereinfachung der Steuerung für DIY-Roboterprojekte basierend auf dem Raspberry Pi.
- **Spezifikationen**: Beschreibt die technischen Daten, einschließlich Stromzufuhr, Batterieangaben, Anschlüsse und Merkmale des Motortreibers.
- **Übersicht der Anschlüsse**: Beschreibt verschiedene Anschlüsse wie Strom, Digital, Analog, PWM, I2C, SPI, UART und Motoranschlüsse.
- **Zusätzliche Komponenten**: Hebt zusätzliche Komponenten wie Tasten, LED und Lautsprecher hervor, mit Zuordnung zu den Raspberry Pi-Pins.
- **Einrichtung und Testen**: Anleitungen zum Montieren des Robot HAT, zu den notwendigen Komponenten und zu Testverfahren für Funktionen wie LED und Servomotoren.

KAPITEL 9

# FAQ

## 9.1 F1: Kann die Batterie angeschlossen werden, während gleichzeitig Strom an den Raspberry Pi geliefert wird?

A: Ja, der Robot HAT hat eine eingebaute Anti-Rückflussdiode, die verhindert, dass die Stromversorgung des Raspberry Pi zurück in den Robot HAT fließt.

## 9.2 F2: Kann der Robot HAT während des Ladens verwendet werden?

A: Ja, der Robot HAT kann während des Ladens verwendet werden. Beim Laden wird die Eingangsleistung durch den Ladungschip verstärkt, um die Batterien zu laden, während gleichzeitig Strom für die DC-DC-Herabsetzung für den externen Gebrauch bereitgestellt wird. Die Ladeleistung beträgt etwa 10W. Wenn der externe Stromverbrauch über einen längeren Zeitraum zu hoch ist, können die Batterien die Stromversorgung ergänzen, ähnlich wie ein Mobiltelefon beim Gebrauch aufgeladen wird. Es ist jedoch wichtig, auf die Kapazität der Batterie zu achten, um ein vollständiges Entleeren während des gleichzeitigen Ladens und Gebrauchs zu vermeiden.

## 9.3 F3: Warum kommt kein Ton aus dem Lautsprecher?

Wenn Ihr Skript läuft, aber der Lautsprecher keinen Ton erzeugt, könnte es mehrere Gründe geben:

1. Überprüfen Sie, ob das Skript `i2samp.sh` installiert wurde. Für detaillierte Anweisungen siehe: *Installieren Sie i2samp.sh für den Lautsprecher*.
2. Beim Ausführen von Skripten, die mit Lautsprechern zusammenhängen, ist es notwendig, `sudo` hinzuzufügen, um administrative Rechte zu erhalten. Zum Beispiel `sudo python3 tts.py`.
3. Verwenden Sie keine integrierten Programmierwerkzeuge des Raspberry Pi, wie Geany, um lautsprecherbezogene Skripte auszuführen. Diese Werkzeuge laufen mit Standardbenutzerrechten, während die Hardwaresteuerung, wie die Verwaltung von Lautsprechern, oft höhere Berechtigungen erfordert.

# Python-Modulindex

## r

# Stichwortverzeichnis

## Sonderzeichen



## A



## B



## C



## D



## E



## F



## G



## H



## I

## K



## L



## M



## N



## O



## P



## R



## S

## T



## U



## V



## W



## X



## Y



## Z