SunFounder Robot HAT

www.sunfounder.com

2024 年 04 月 02 日

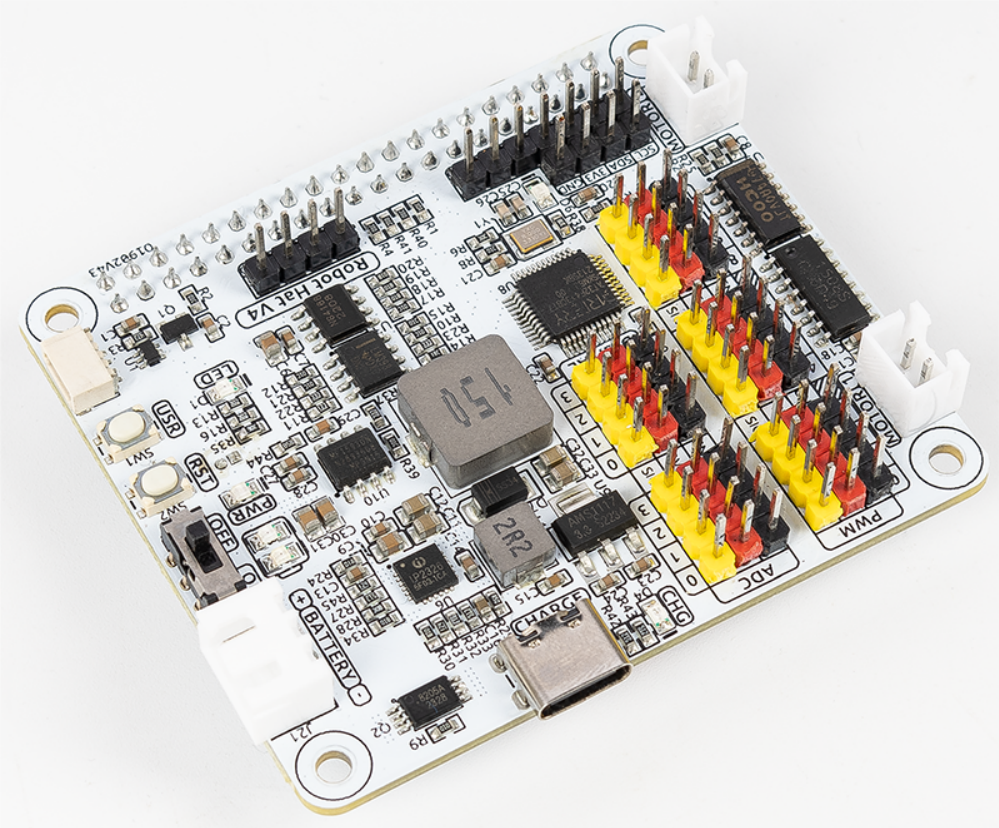
目次

SunFounder Robot HAT をお選びいただき、ありがとうございます。

注釈: このドキュメントは以下の言語で利用可能です。

-
-
-

ご希望の言語でドキュメントにアクセスするために、それぞれのリンクをクリックしてください。



Robot HAT は、Raspberry Pi を迅速にロボットに変換できる多機能拡張ボードです。MCU が搭載されており、Raspberry Pi の PWM 出力と ADC 入力を拡張するとともに、モータードライバーチップ、Bluetooth モジュール、I2S オーディオモジュール、モノラルスピーカー、そして Raspberry Pi 自体から出る GPIO も搭載しています。

また、背景音楽や効果音を再生し、TTS 機能を実装してプロジェクトをより魅力的にするためのスピーカーも付属しています。

7-12V PH2.0 2 ピン電源入力と 2 つの電源インジケータを受け入れます。このボードには、ユーザーが使用可能な LED と、いくつかの効果を迅速にテストするためのボタンも備えています。

この文書では、SunFounder が提供する Python `robot-hat` ライブラリを通じて、Robot HAT のインターフェイス

機能とこれらのインターフェイスの使用方法を完全に理解できます。

第 1 章

特徴

- シャットダウン電流：<0.5mA
- 電源入力：USB Type-C、5V/2A
- 充電電力：5V/2A 10W
- 出力電力：5V/3A
- 付属のバッテリー：2 x 3.7V 18650 リチウムイオンバッテリー、XH2.0 3P インターフェース
- バッテリー保護：逆極性保護
- 充電保護：入力低電圧保護、入力過電圧保護、充電バランス、過熱保護
- オンボード充電指示灯：CHG
- オンボード電源指示灯：PWR
- オンボード 2 つのバッテリーレベル指示 LED
- オンボードユーザー LED、2 つの触覚スイッチ
- モータードライバー：5V/1.8A x 2
- 4 チャンネル 12 ビット ADC
- 12 チャンネル PWM
- 4 チャンネルデジタル信号
- オンボード SPI インターフェース、UART インターフェース、I2C インターフェース
- モノスピーカー：8 Ω 1W

表 1　電気特性

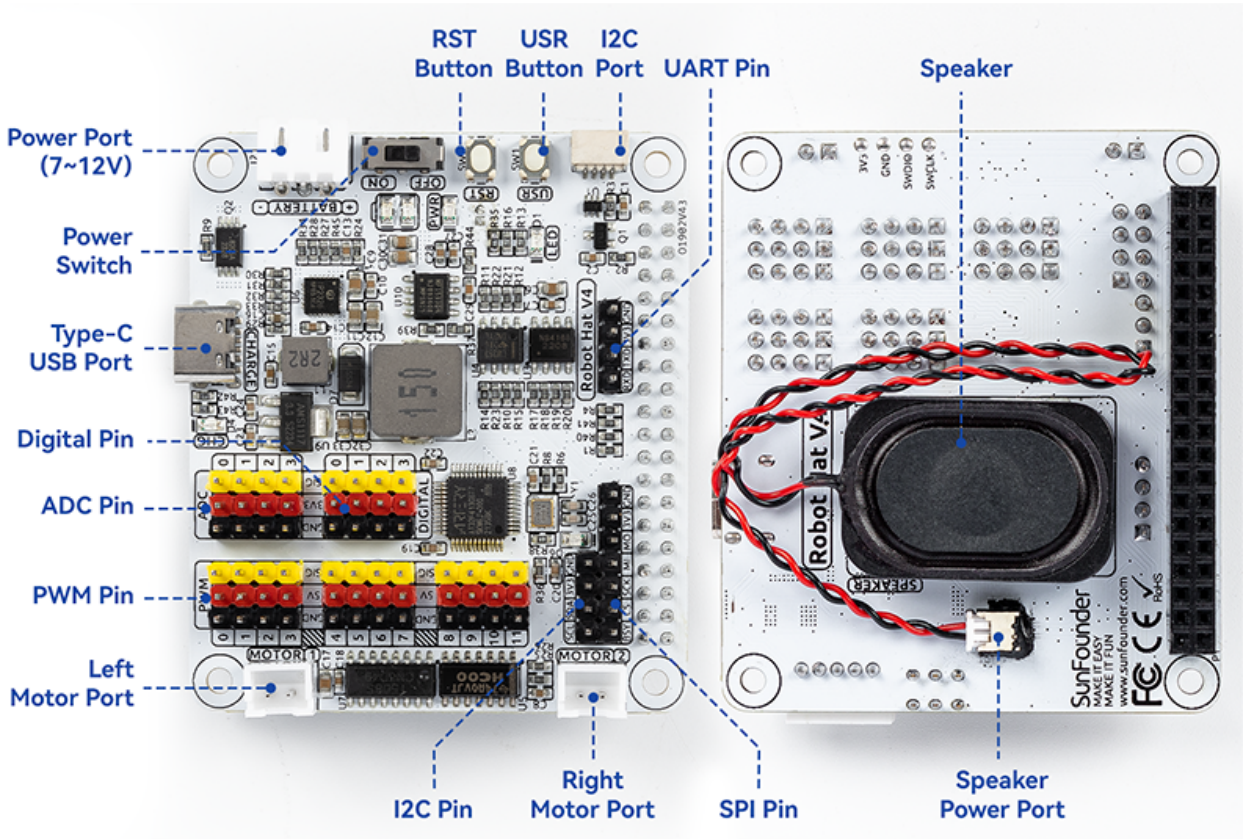
パラメータ：	最小値：	典型値：	最大値：	単位：
入力電圧：	4.25	5	8.4	V
バッテリー入力電圧：	6.0	7.4	8.4	V
過充電保護（バッテリー）：		8.3		V
入力低電圧保護：	4.15	4.25	4.35	V
入力過電圧保護：	8.3	8.4	8.5	V
充電電流（5V）：			2.0	A
出力電流（5V）：			3.0	A
出力電圧：	5.166	5.246	5.327	V
充電過熱保護：	125	135	145	° C
DC-DC 過熱保護：	70	75	80	° C
モーター出力電流：			1.8	A

第 2 章

ハードウェア紹介

Robot Hat V4 には、2 つのリチウムバッテリー充電、5V/3A DC-DC 放電、I2S オーディオ出力およびスピーカー、シンプルなバッテリーレベルインジケータ、マイクロコントローラベースの PWM および ADC ドライバ、モータードライバが特徴です。

2.1 ピン配置



Power Port

- 7-12V PH2.0 3 ピン電源入力。
- Raspberry Pi と Robot HAT を同時に給電します。

Power Switch

- Robot HAT の電源をオン/オフします。
- 電源ポートに電源を接続すると、Raspberry Pi が起動します。しかし、Robot HAT を有効にするためには、電源スイッチを ON に切り替える必要があります。

Type-C USB Port

- Type-C ケーブルを挿入してバッテリーを充電します。
- 同時に、充電インジケータが赤色で点灯します。
- バッテリーが完全に充電されると、充電インジケータが消灯します。
- バッテリーが完全に充電された後、約 4 時間 USB ケーブルがまだ接続されている場合、充電インジケータが点滅してプロンプト表示されます。

Digital Pin

- 4 チャンネルのデジタルピン、D0-D3。
- ピン： *Digital IO*。
- API： **クラス** *Pin*。

ADC Pin

- 4 チャンネル ADC ピン、A0-A3。
- ピン： *ADC*。
- API： **クラス** *ADC*。

PWM Pin

- 12 チャンネル PWM ピン、P0-P11。
- ピン： *PWM*。
- API： **クラス** *PWM*。

Left/Right Motor Port

- 2 チャンネル XH2.54 モーターポート。
- ピン： **モーターポート**。
- API： **モジュール** *motor*、 1 は左モーターポート用、 2 は右モーターポート用。

I2C Pin and I2C Port

- **I2C Pin**：P2.54 4 ピンインターフェース。
- **I2C Port**：SH1.0 4 ピンインターフェース、QWIIC および STEMMA QT と互換性があります。
- これらの I2C インターフェースは、GPIO2 (SDA) および GPIO3 (SCL) を介して Raspberry Pi の I2C インターフェースに接続されています。
- ピン： *I2C*。
- API： **クラス** *I2C*。

SPI Pin

- P2.54 7 ピン SPI インターフェース。
- ピン： *SPI*。

UART Pin

- P2.54 4 ピンインターフェース。
- ピン： *UART*。

RST Button

- Ezblock を使用しているとき、RST ボタンは Ezblock プログラムを再起動するためのボタンとして機能します。
- Ezblock を使用していない場合、RST ボタンにはあらかじめ定義された機能はなく、あなたのニーズに合わせて完全にカスタマイズすることができます。
- ピン： **ボタン**。
- API： **クラス** *Pin*

USR Button

- USR ボタンの機能はプログラミングによって設定することができます。(押すと入力「0」になり、放すと入力「1」になります。)
- API： **クラス** *Pin* 、 `Pin("SW")` を使用して定義することができます。
- ピン： **ボタン**。

Battery Indicator

- 電圧が 7.6V を超えると 2 つの LED が点灯します。
- 7.15V から 7.6V の範囲では 1 つの LED が点灯します。
- 7.15V 以下では両方の LED が消灯します。

- **バッテリーレベルインジケーター**。

Speaker and Speaker Port

- **Speaker**：これは 2030 オーディオチャンバースピーカーです。
- **Speaker Port**：Robot HAT は、2030 オーディオチャンバースピーカーと共に、オンボード I2S オーディオ出力を備え、モノサウンド出力を提供します。
- ピン： **スピーカーとスピーカーポート**。
- API： **クラス** *Music*

2.2 ピンマッピング

表 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	モーター 1 DIR	GPIO23
NC	3V3	モーター 2 DIR	GPIO24
SPI MOSI	MOSI	GND	GND
SPI MISO	MISO	USR ボタン	GPIO25
SPI SCLK	SCLK	SPI CE0	CE0
GND	GND	NC	CE1
NC	ID_SD	NC	ID_SC
MCU リセット	GPIO5	GND	GND
(SPI)BSY	GPIO6	ボード識別子 2	GPIO12
ボード識別子 1	GPIO13	GND	GND
I2S LRCLK	GPIO19	RST ボタン	GPIO16
ユーザー LED	GPIO26	NC	GPIO20
GND	GND	I2S SDATA	GPIO21

2.3 Digital IO

Robot HAT には 4 セットの 3 ピンデジタルピンがあります。

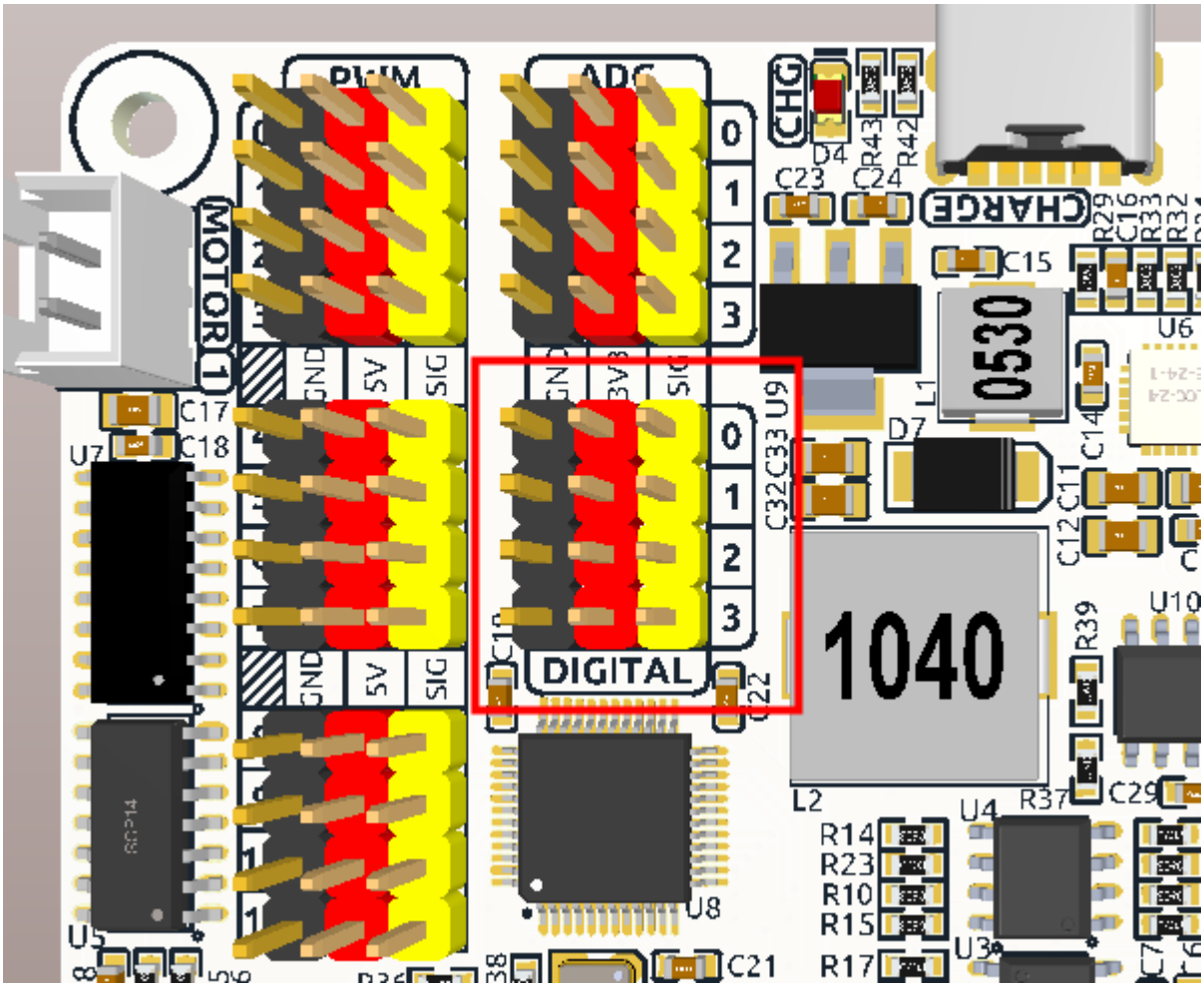
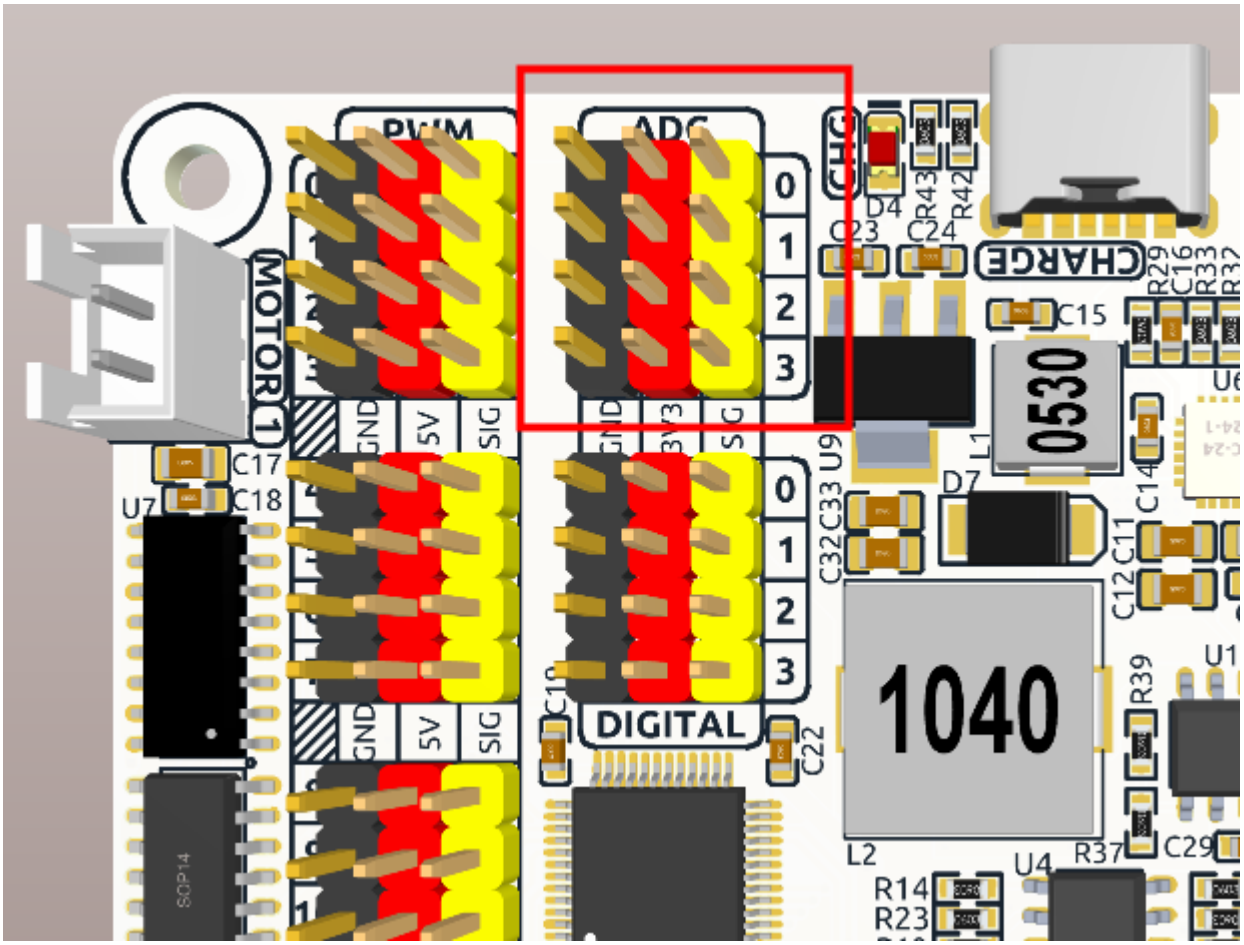


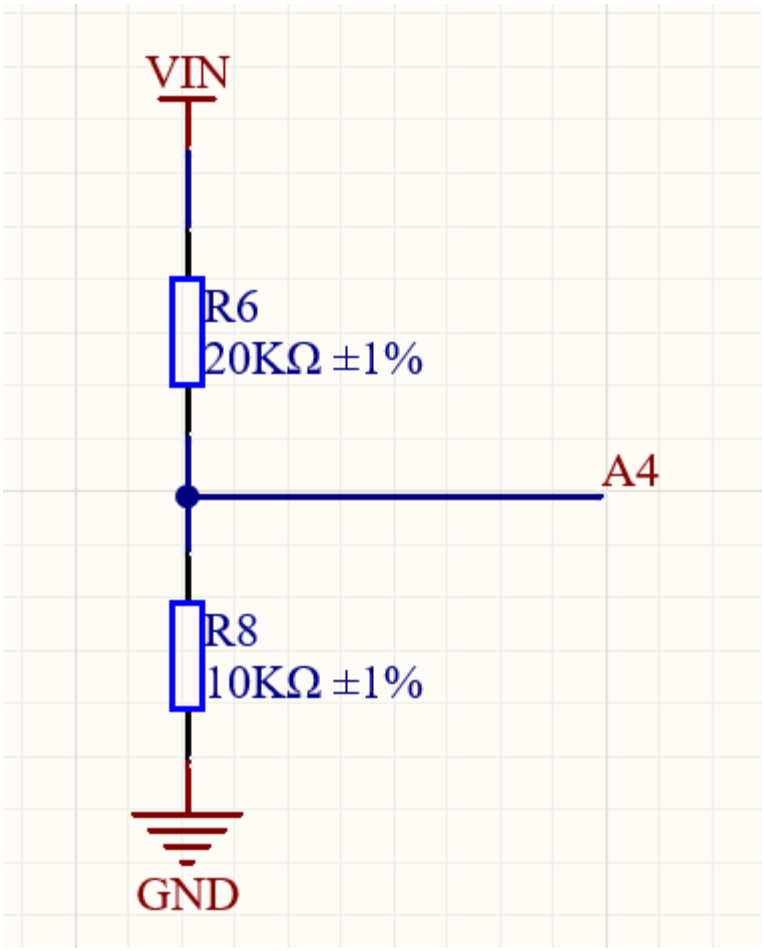
表 2　Digital IO

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

2.4 ADC



Robot HAT は、4 セットの 3 ピン ADC（アナログ・デジタルコンバータ）ピンを特長としており、それぞれが 2.54mm 間隔で配置されています。これらのピンは 3.3V の電源で動作します。12 ビットの精度を提供する ADC 機能は、オンボードのマイクロコントローラによって容易になっています。ADC 値の読み取り方法の詳細は、 **オンボード** *MCU* セクションで提供されています。

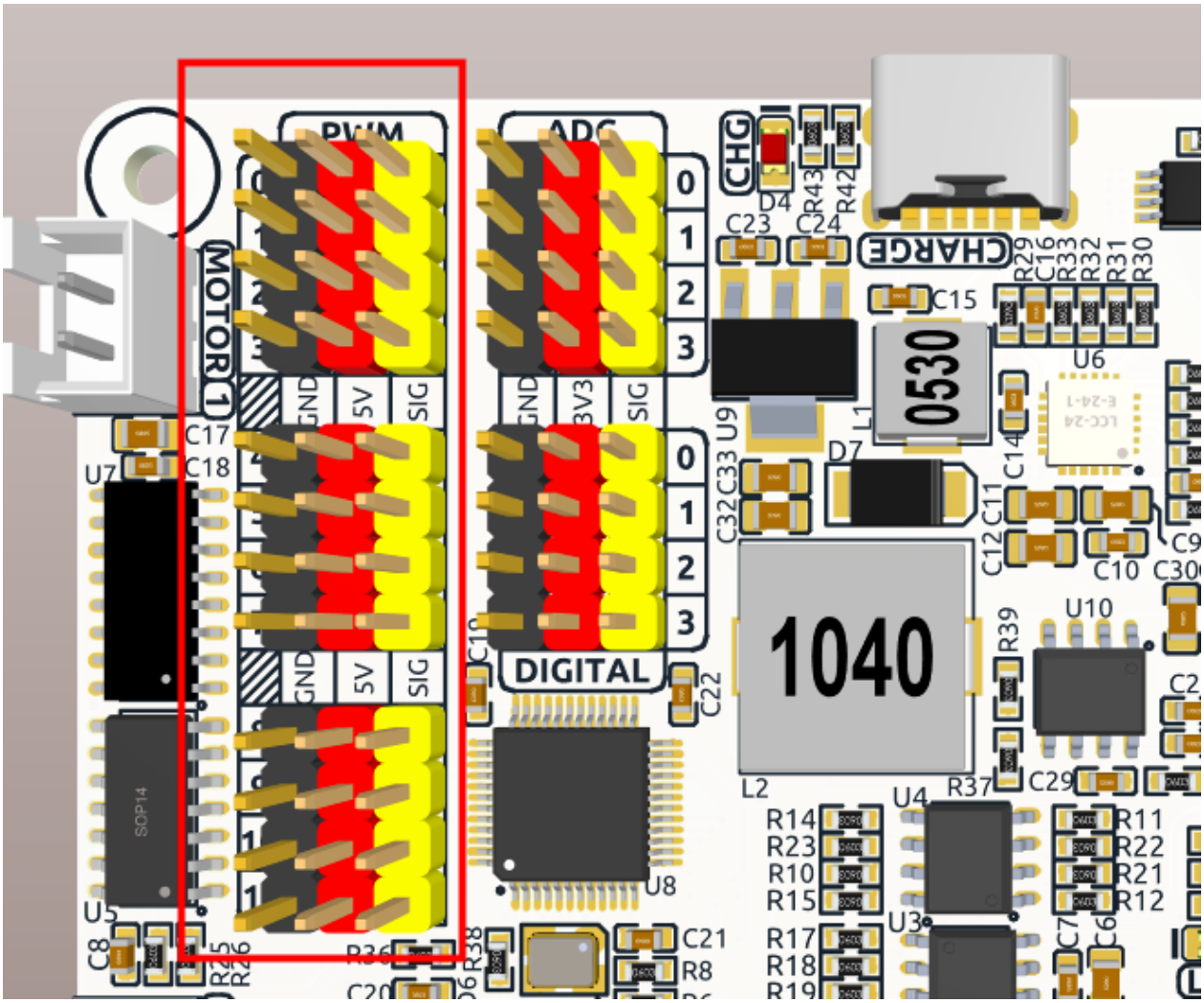


また、ADC チャネル A4 は抵抗を使用した電圧分割器を介してバッテリーに接続されており、バッテリー電圧を測定しておおよそのバッテリー残量を推定するために使用されます。

電圧分割比は 20K/10K なので：

- A4 電圧 (Va4) = value_A4 / 4095.0 * 3.3
- バッテリー電圧 (Vbat) = Va4*3
- バッテリー電圧 (Vbat) = value_A4 / 4095.0 * 3.3 * 3

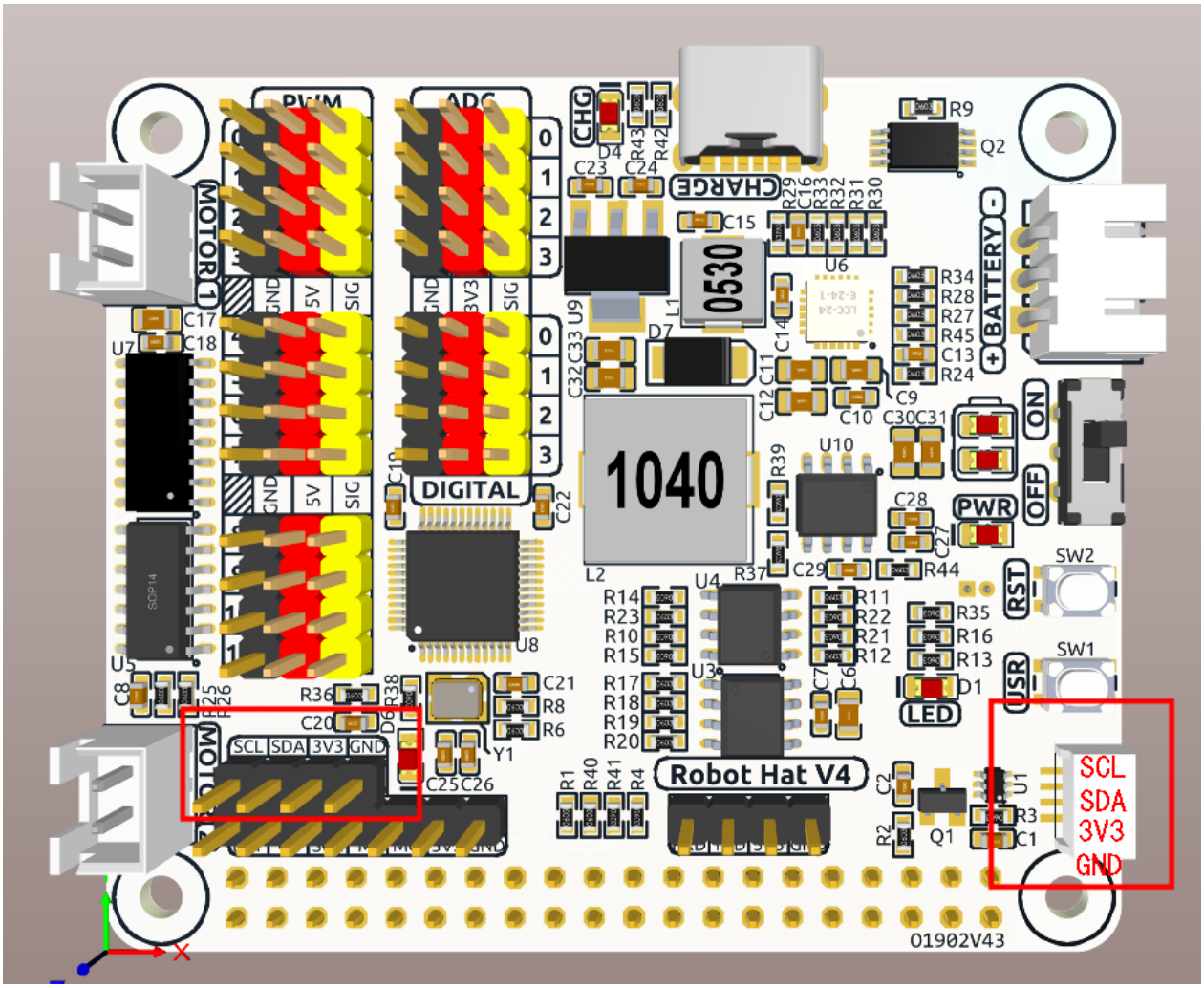
2.5 PWM



Robot HAT には、2.54mm 間隔で配置された 4 セットの 3 ピン PWM ピンがあり、電源は 5V です。PWM の使用方法は、 **オンボード** *MCU* で詳しく説明されています。

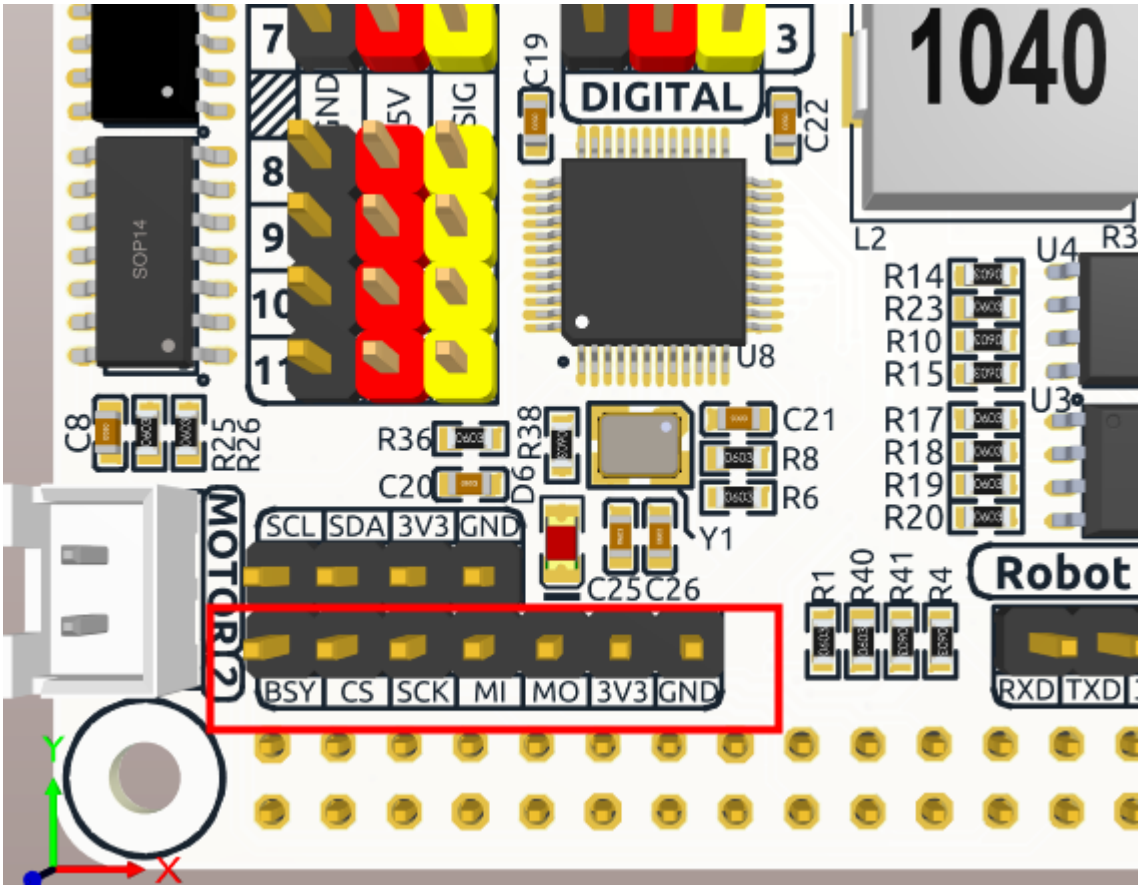
注釈: PWM13 および 14 チャンネルはモータードライブ用に使用されます。

2.6 I2C



Robot HAT には 2 つの I2C インターフェイスがあります。一つは P2.54 4 ピンインターフェイス、もう一つは QWIIC および STEMMA QT と互換性のある SH1.0 4 ピンインターフェイスです。これらの I2C インターフェイスは GPIO2 (SDA) および GPIO3 (SCL) を介して Raspberry Pi の I2C インターフェイスに接続されています。ボードには **オンボード** *MCU* も搭載されており、2 つの信号線には 10K のプルアップ抵抗があります。

2.7 SPI

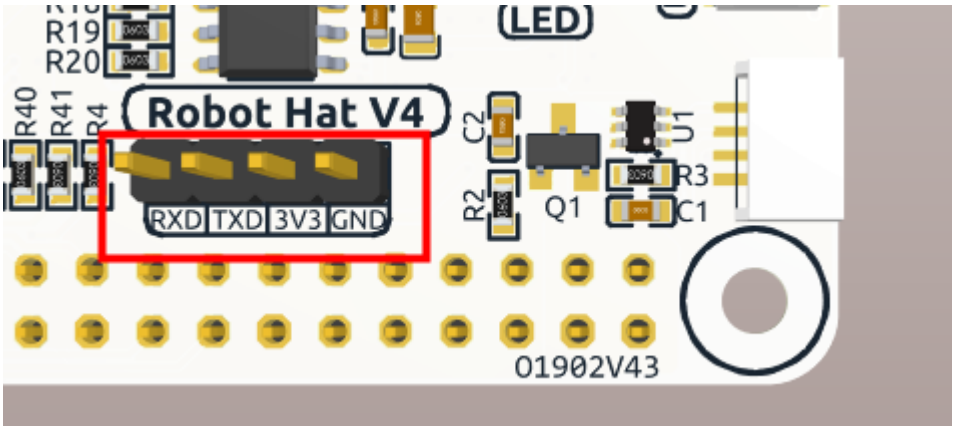


Robot HAT の SPI インターフェイスは 7 ピンの P2.54 インターフェイスです。これは Raspberry Pi の SPI インターフェイスに接続され、割り込みやリセットなどの目的に使用できる追加の I/O ピンを含んでいます。

表 3　SPI

Robot Hat V4	Raspberry Pi
BSY	GPIO6
CS	CE0(GPIO8)
SCK	SCLK(GPIO11)
MI	MISO(GPIO9)
MO	MOSI(GPIO10)
3V3	3.3V 電源
GND	グラウンド

2.8 UART



Robot HAT の UART インターフェイスは 4 ピンの P2.54 インターフェイスです。これは Raspberry Pi の GPIO14 (TXD) と GPIO15 (RXD) ピンに接続されます。

2.9 ボタン

Robot HAT には 1 つの LED と 2 つのボタンがあり、すべて Raspberry Pi の GPIO ピンに直接接続されています。Ezblock を使用している場合、RST ボタンは Ezblock プログラムを再起動するためのボタンとして機能します。Ezblock を使用していない場合、RST ボタンには事前に定義された機能はなく、必要に応じて完全にカスタマイズすることができます。

表 4　LED &ボタン

Robot Hat V4	Raspberry Pi
LED	GPIO26
USR	GPIO25
RST	GPIO16

2.10 スピーカーとスピーカーポート

Robot HAT にはオンボードの I2S オーディオ出力が搭載されており、2030 オーディオチャンバースピーカーと共にモノラルサウンド出力を提供します。

表 5　I2S

I2S	Raspberry Pi
LRCLK	GPIO19
BCLK	GPIO18
SDATA	GPIO21

2.11 モーターポート

Robot HAT のモータードライバーは 2 チャンネルをサポートしており、2 つのデジタル信号を使用して方向を制御し、2 つの PWM 信号を使用して速度を制御することができます。

表 6　モータードライバー

モーター	IO
モーター 1 方向	GPIO23
モーター 1 電源	PWM13
モーター 2 方向	GPIO24
モーター 2 電源	PWM12

2.12 バッテリーレベルインジケーター

Robot HAT のバッテリーレベルインジケーターは、電圧分割器法を使用してバッテリー電圧を監視し、バッテリーレベルの推定のための参考として機能します。LED と電圧の関係は以下の通りです：

表 7　バッテリーレベル

LED **バッテリー**	**合計電圧**
2 つの LED 点灯	7.6V 以上
1 つの LED 点灯	7.15V 以上
両方の LED 消灯	7.15V 未満

バッテリーのうちの 1 つが 4.1V に達するかそれを超えると、他のバッテリーがその閾値以下の場合、その特定のバッテリーの充電電流が減少します。

第 3 章

バッテリーについて

バッテリー

- **VCC**: バッテリーの正極端子。ここには VCC と GND の 2 組があり、電流を増やし抵抗を減らすためです。
- **Middle**: 二つのセル間の電圧をバランスさせ、バッテリーを保護します。
- **GND**: バッテリーの負極端子。

これは SunFounder によって製造されたカスタムバッテリーパックで、2000mAh の容量を持つ 18650 バッテリー

2 個で構成されています。コネクターは PH2.0-3P で、シールドに挿入後、直接充電が可能です。

特徴

- バッテリー充電: 5V/2A
- バッテリー出力: 5V/5A
- バッテリー容量: 3.7V 2000mAh x 2
- バッテリー寿命: 90 分
- バッテリー充電時間: 130 分
- コネクター: PH2.0, 3P

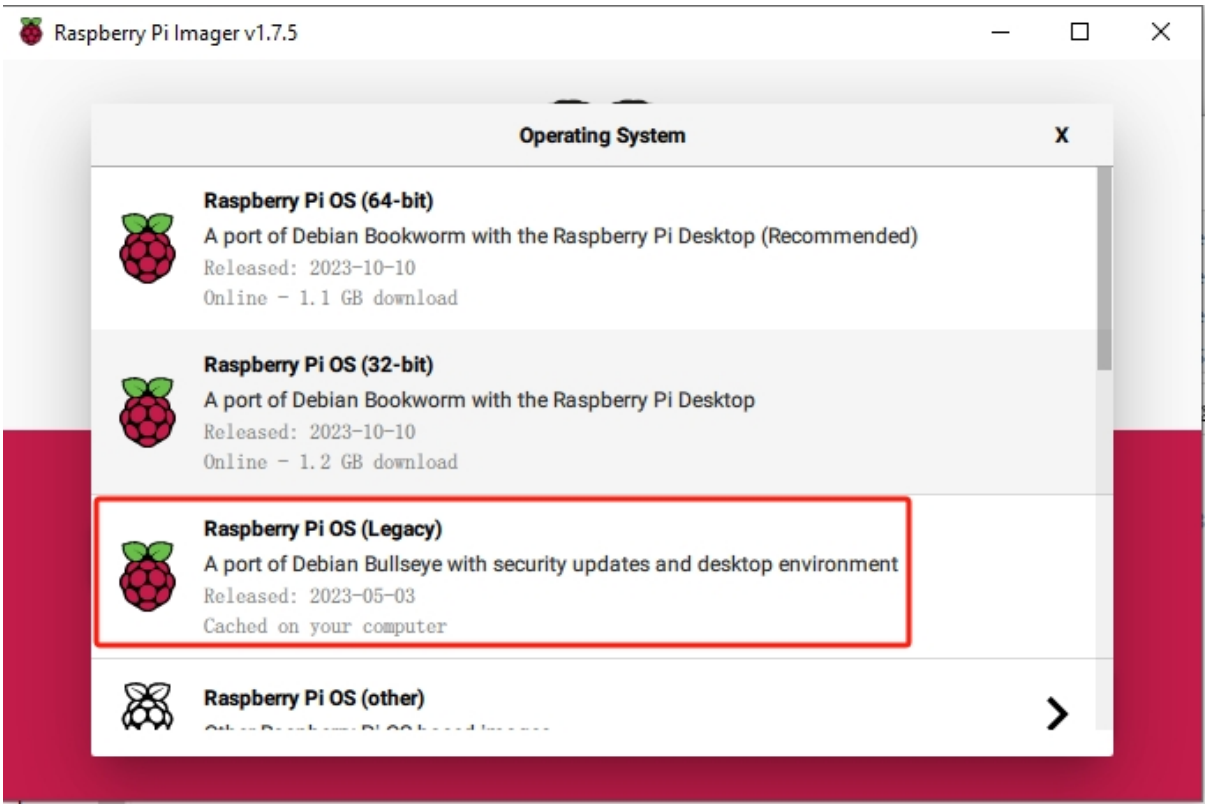
第 4 章

robot-hat モジュールのインストール

robot-hat は Robot HAT に対応したライブラリです。

警告:

- Raspberry Pi OS をインストールする際には、**Raspberry Pi OS (Legacy)** バージョンの **Debian Bullseye** を使用してください。
- インストールするバージョンが **Bookworm** の場合、 **Speaker** が正しく機能しないことがあります。



1. システムをアップデートしてください。

   インターネットに接続していることを確認し、システムをアップデートしてください:

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

   注釈: OS の **Lite** バージョンをインストールする場合、Python3 関連のパッケージをインストールする必要があります。

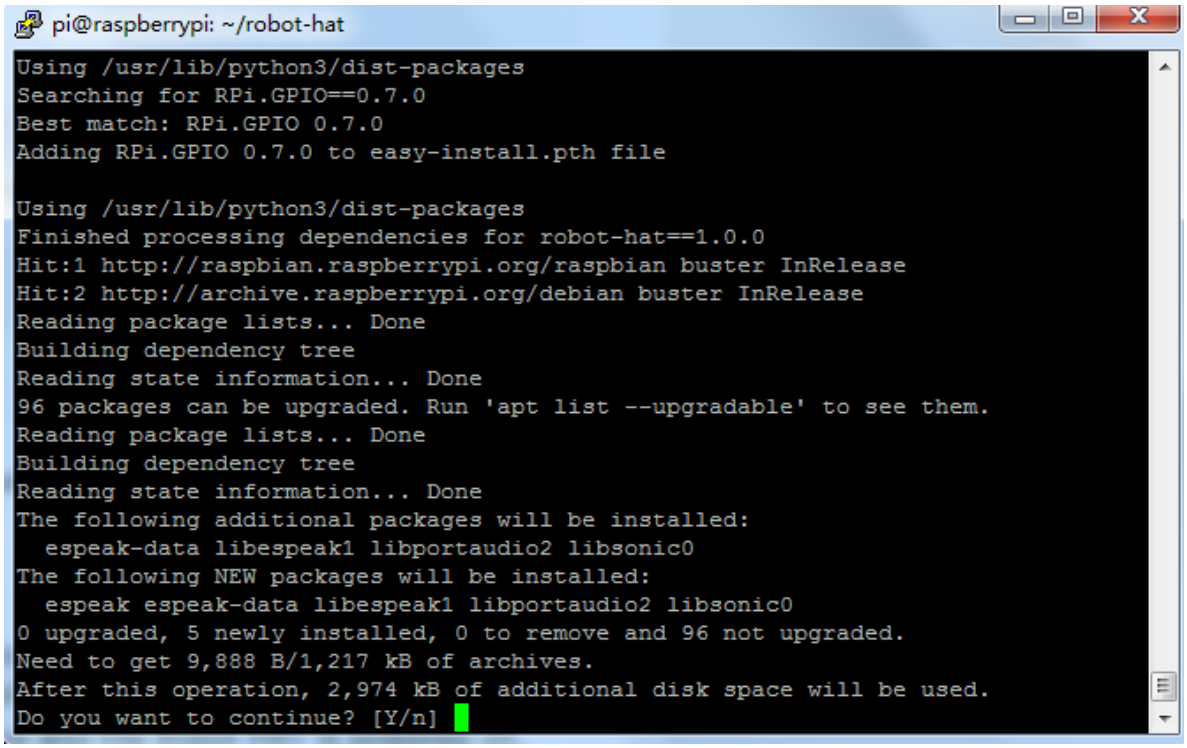
   ```
   sudo apt install git python3-pip python3-setuptools python3-smbus
   ```

   ---

2. このコマンドをターミナルに入力して、robot-hat パッケージをインストールします。

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

   注釈: setup.py を実行して必要なコンポーネントをダウンロードします。ネットワークの問題でダウンロードが失敗することがあります。その場合は、再度ダウンロードが必要になるかもしれません。次のような場合には、 Y と入力し Enter を押してプロセスを続行してください。


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]

第５章

スピーカー用の i2samp.sh をインストールする

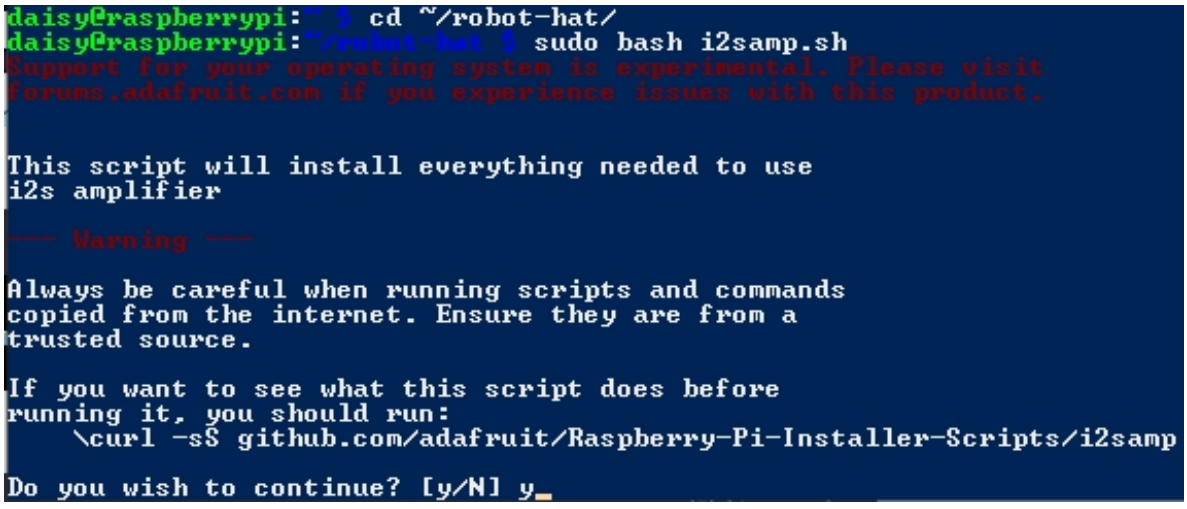
i2samp.sh は、Raspberry Pi や類似のデバイスで I2S（Inter-IC Sound）アンプを設定し、構成するために特別に設計された高度な Bash スクリプトです。MIT ライセンスのもとで、さまざまなハードウェアとオペレーティングシステムとの互換性を保証し、インストールや構成を進める前に徹底的なチェックを行います。

スピーカーを適切に動作させたい場合は、このスクリプトのインストールが絶対に必要です。

手順は以下の通りです：

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

y と入力し enter を押して、スクリプトの実行を続けます。



y と入力し enter を押して、バックグラウンドで /dev/zero を実行します。

```
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_
```

y と入力し enter を押して、Raspberry Pi を再起動します。

```
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_
```

警告: 再起動後に音が出ない場合は、 i2samp.sh スクリプトを数回実行する必要があるかもしれません。

第 6 章

オンボード MCU

Robot HAT は、Artery の AT32F415CBT7 マイクロコントローラーを搭載しています。これは、最大クロック周波数が 150MHz の ARM Cortex-M4 プロセッサーです。マイクロコントローラーには、256KB のフラッシュメモリと 32KB の SRAM があります。

オンボードの PWM と ADC は、マイクロコントローラによって駆動されます。Raspberry Pi とマイクロコントローラ間の通信は、I2C インターフェイスを通じて確立されます。通信に使用される I2C アドレスは 0x14（7 ビットアドレス形式）です。

6.1 紹介

オンボード MCU の RESET ピンは、Raspberry Pi の GPIO 5、または *robot_hat.Pin* の `MCURST` に接続されています。MCU は 7 ビットアドレスの `0x14` を使用しています。

6.2 ADC

レジスタアドレスは 3 バイトで、0x170000 から 0x140000 までが ADC チャンネル 0 から 3 です。ADC の精度は 12 ビットで、値は 0 から 4095 までです。*robot_hat.ADC* で詳細を確認してください。

アドレス	説明
`0x170000`	ADC チャンネル 0
`0x160000`	ADC チャンネル 1
`0x150000`	ADC チャンネル 2
`0x140000`	ADC チャンネル 3
`0x130000`	ADC チャンネル 4（バッテリーレベル）

例:

チャンネル 0 の ADC 値を読む:

```
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 は 1 バイトのレジスタと 2 バイトの値を持っています。

6.3.1 PWM 周波数の変更

周波数はプリスケーラと周期で定義されます。

周波数を設定するには、まず設定したい周期を定義する必要があります。Arduino では通常 255、PCA9685 では 4095 です。

CPU クロックは 72MHz です。そこから、希望の周波数に応じたプリスケーラを計算できます

> プリスケーラ = 72MHz / (周期 + 1) / 周波数 - 1

周期にこだわらない場合は、周波数から周期とプリスケーラの両方を計算する方法があります。 *robot_hat.PWM.freq()* を参照してください。

6.3.2 パルス幅

チャンネルのパルス幅を制御するのは簡単で、値をレジスタに書き込むだけです。

しかし 値とは何か？ PWM を 50% のパルス幅に設定したい場合、周期が正確に何であるかを知る必要があります。上記の計算に基づき、周期を 4095 に設定した場合、パルス値を 2048 に設定すると、約 50% のパルス幅になります。

アドレス	説明
0x20	PWM チャンネル 0 **オン値** を設定
0x21	PWM チャンネル 1 **オン値** を設定
0x22	PWM チャンネル 2 の **オン値** を設定
0x23	PWM チャンネル 3 の **オン値** を設定
0x24	PWM チャンネル 4 の **オン値** を設定
0x25	PWM チャンネル 5 の **オン値** を設定
0x26	PWM チャンネル 6 の **オン値** を設定
0x27	PWM チャンネル 7 の **オン値** を設定
0x28	PWM チャンネル 8 の **オン値** を設定
0x29	PWM チャンネル 9 の **オン値** を設定
0x2A	PWM チャンネル 10 の **オン値** を設定
0x2B	PWM チャンネル 11 の **オン値** を設定
0x2C	モーター 2 の速度 **オン値** を設定
0x2D	モーター 1 の速度 **オン値** を設定

6.3.3 プリスケーラー

0x40 から始まるレジスタは PWM プリスケーラーを設定するためのもので、範囲は 0〜65535 です。全 14 チャンネルに対してタイマーは 4 つのみです。 *PWM* **タイマー** *(***重要***)* を参照してください。

アドレス	説明
0x40	タイマー 0 の **プリスケーラー** を設定
0x41	タイマー 1 の **プリスケーラー** を設定
0x42	タイマー 2 の **プリスケーラー** を設定
0x43	タイマー 3 の **プリスケーラー** を設定

6.3.4 周期

0x44 から始まるレジスタは PWM 周期を設定するためのもので、範囲は 0〜65535 です。全 14 チャンネルに対してタイマーは 4 つのみです。 *PWM* **タイマー** *(***重要***)* を参照してください。

アドレス	説明
0x44	タイマー 0 の **周期** を設定
0x45	タイマー 1 の **周期** を設定
0x46	タイマー 2 の **周期** を設定
0x47	タイマー 3 の **周期** を設定

6.3.5 PWM タイマー（重要）

PWM タイマーとは何ですか？ PWM タイマーは、PWM チャンネルをオン・オフするためのツールです。

MCU には PWM 用のタイマーが 4 つしかありません：つまり、同じタイマーで異なるチャンネルの周波数を設定することはできません。

例：チャンネル 0 に周波数を設定すると、チャンネル 1、2、3 に影響が及びます。チャンネル 2 の周波数を変更すると、チャンネル 0、1、3 が上書きされます。

これは、常に周波数を変更するパッシブブザーと固定周波数 50Hz が必要なサーボを同時に制御したい場合のような状況です。この場合、それらを 2 つの異なるタイマーに分けるべきです。

タイマー	PWM チャンネル
タイマー 0	0, 1, 2, 3
タイマー 1	4, 5, 6, 7
タイマー 2	8, 9, 10, 11
タイマー 3	12, 13（モーター用）

6.3.6 例

```
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 をリセット

現在のファームウェアは固定 3 バイトの値を読み取り、その後 ADC 値を返すか PWM を制御できます。そのため ADC レジスタは最後の 2 バイトが 0 の 3 バイトが必要です。

そして、通信の途中でプログラムが中断されると、ファームウェアが固まってデータがずれる可能性があります。3 バイトのデータ待ちにはタイムアウトも設定しています。

そのような場合は、MCU をリセットする必要があります。リセットするには、robot_hat コマンドを使用できます：

```
robot_hat reset_mcu
```

または、Python コードで行うこともできます：

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

あるいは、リセットピン（GPIO 5）を 10ms 引き下げてから、さらに 10ms 引き上げるだけで、それが `reset_mcu` が行うことです。

```
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)
```

第 7 章

参照

Robot Hat ライブラリ

7.1 クラス Pin

例

```
# 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()
```

(次のページに続く)

(前のページからの続き)

```
# 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*)

ベースクラス: *_Basic_class*

ピン操作クラス

OUT = 1

ピンモード出力

IN = 2

ピンモード入力

PULL_UP = 17

ピン内部プルアップ

PULL_DOWN = 18

ピン内部プルダウン

PULL_NONE = None

ピン内部プルなし

IRQ_FALLING = 33

ピン割り込みフォーリング

IRQ_RISING = 34

ピン割り込みフォーリング

IRQ_RISING_FALLING = 35

ピン割り込みライジングとフォーリングの両方

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

ピンの初期化

パラメータ

- **pin** (*int/str*) -- Raspberry Pi のピン番号

- **mode** (*int*) -- ピンモード (IN/OUT)
- **pull** (*int*) -- ピンプルアップ/ダウン (PUD_UP/PUD_DOWN/PUD_NONE)

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

ピンのセットアップ

パラメータ

- **mode** (*int*) -- ピンモード (IN/OUT)
- **pull** (*int*) -- ピンプルアップ/ダウン (PUD_UP/PUD_DOWN/PUD_NONE)

dict(*_dict=None*)

ピン辞書の設定/取得

パラメータ

_dict (*dict*) -- ピン辞書、辞書を取得するには空のままにしてください

戻り値

ピン辞書

戻り値の型

dict

__call__(*value*)

ピン値の設定/取得

パラメータ

value (*int*) -- ピン値、値を取得する場合は空にしておく (0/1)

戻り値

ピン値 (0/1)

戻り値の型

int

value(*value: bool = None*)

ピン値の設定/取得

パラメータ

value (*int*) -- ピン値、値を取得する場合は空にしておく (0/1)

戻り値

ピン値 (0/1)

戻り値の型

int

on()

ピンをオン (ハイ) に設定

戻り値

ピン値 (1)

戻り値の型

int

off()

ピンをオフ (ロー) に設定

戻り値

ピン値 (0)

戻り値の型

int

high()

ピンをハイ (1) に設定

戻り値

ピン値 (1)

戻り値の型

int

low()

ピンをロー (0) に設定

戻り値

ピン値 (0)

戻り値の型

int

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

ピン割り込みを設定

パラメータ

- **handler** (*function*) -- 割り込みハンドラコールバック関数
- **trigger** (*int*) -- 割り込みトリガー (RISING, FALLING, RISING_FALLING)
- **bouncetime** (*int*) -- 割り込みバウンスタイム（ミリ秒）

name()

ピン名を取得

戻り値

ピン名

戻り値の型

str

7.2 クラス ADC

例

```
# 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*)

ベースクラス: *I2C*

アナログからデジタルへの変換器

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

アナログからデジタルへの変換器

パラメータ

chn (*int/str*) -- チャンネル番号 (0-7/A0-A7)

read()

ADC 値を読む

戻り値

ADC 値 (0-4095)

戻り値の型

int

read_voltage()

ADC 値を読み取り電圧に変換する

戻り値 電

圧値 (0-3.3(V))

戻り値の型

float

7.3 クラス PWM

例

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

(次のページに続く)

(前のページからの続き)

```
# 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*)

ベースクラス: *I2C*

パルス幅変調（PWM）

REG_CHN = 32

チャンネルレジスタの接頭辞

REG_PSC = 64

プリスケーラーレジスタの接頭辞

REG_ARR = 68

周期レジスタの接頭辞

CLOCK = 72000000.0

クロック周波数

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

PWM の初期化

パラメータ

channel (*int/str*) -- PWM チャンネル番号 (0-13/P0-P13)

freq(*freq=None*)

周波数の設定/取得、周波数を取得するには空白にしておく

パラメータ

freq (*float*) -- 周波数 (0-65535)(Hz)

戻り値 周

波数

戻り値の型

float

prescaler(*prescaler=None*)

プリスケーラーを設定/取得、プリスケーラーを取得するには空欄にしておく

パラメータ

prescaler (*int*) -- プリスケーラー (0-65535)

戻り値 プ

リスケーラー

戻り値の型

int

period(*arr=None*)

周期を設定/取得、周期を取得するには空欄にしておく

パラメータ

arr (*int*) -- 周期 (0-65535)

戻り値 周

期

戻り値の型

int

pulse_width(*pulse_width=None*)

パルス幅を設定/取得、パルス幅を取得するには空欄にしておく

パラメータ

pulse_width (*float*) -- パルス幅 (0-65535)

戻り値 パ

ルス幅

戻り値の型

float

pulse_width_percent(*pulse_width_percent=None*)

パルス幅のパーセンテージを設定/取得、パルス幅のパーセンテージを取得するには空欄にしておく

パラメータ

pulse_width_percent (*float*) -- パルス幅のパーセンテージ (0-100)

戻り値

パルス幅のパーセンテージ

戻り値の型

float

7.4 クラス Servo

例

```
# 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
```

(次のページに続く)

(前のページからの続き)

```
# 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*)

ベースクラス: *PWM*

サーボモータークラス

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

サーボモータークラスを初期化する

パラメータ

channel (*int/str*) -- PWM チャンネル番号 (0-14/P0-P14)

angle(*angle*)

サーボモーターの角度を設定する

パラメータ

angle (*float*) -- 角度 (-90~90)

pulse_width_time(*pulse_width_time*)

サーボモーターのパルス幅を設定する

パラメータ

pulse_width_time (*float*) -- パルス幅時間 (500~2500)

7.5 モジュール motor

7.5.1 クラス Motors

例

初期化

```
# Import Motor class
from robot_hat import Motors
```

(次のページに続く)

(前のページからの続き)

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

モーターを直接制御します。モーター 1/2 は PCB のマークに従います

```
# 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()
```

ハイレベル制御の設定を行います。ハイレベル制御は、単純な前進、後退、左、右、停止から、ジョイスティック制御、モーター方向のキャリブレーションなどの複雑な機能を提供します。

注釈: これらの設定は一度だけ実行する必要があり、設定ファイルに保存されます。次に Motors クラスをロードするときは、設定ファイルからロードされます。

```
# 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)
```

これでロボットを制御できます

```
import time

motors.forward(100)
time.sleep(1)
motors.backward(100)
```

(次のページに続く)

(前のページからの続き)

```
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*)

ベースクラス: *_Basic_class*

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

robot_hat.motor.Motor でモーターを初期化

パラメータ

db (*str*) -- 設定ファイルのパス

__getitem__(*key*)

特定のモーターを取得する

stop()

すべてのモーターを停止する

property left

左のモーター

property right

右のモーター

set_left_id(*id*)

左のモーター ID を設定します。この機能は一度だけ実行する必要があり、モーター ID を設定ファイルに保存し、クラスが初期化されるときにモーター ID をロードします。

パラメータ

id (*int*) -- モーター ID（1 または 2）

set_right_id(*id*)

右のモーター ID を設定します。この機能は一度だけ実行する必要があり、モーター ID を設定ファイルに保存し、クラスが初期化されるときにモーター ID をロードします。

パラメータ

id (*int*) -- モーター ID（1 または 2）

set_left_reverse()

左のモーターを逆回転に設定します。この機能は一度だけ実行する必要があり、逆回転の状態を設定ファイルに保存し、クラスが初期化されるときに逆回転の状態をロードします。

戻り値 現

在逆回転しているかどうか

戻り値の型

bool

set_right_reverse()

右のモーターを逆回転に設定します。この機能は一度だけ実行する必要があり、逆回転の状態を設定ファイルに保存し、クラスが初期化されるときに逆回転の状態をロードします。

戻り値 現

在逆回転しているかどうか

戻り値の型

bool

speed(*left_speed*, *right_speed*)

モーター速度を設定する

パラメータ

- **left_speed** (*float*) -- 左モーターの速度 (-100.0~100.0)
- **right_speed** (*float*) -- 右モーターの速度 (-100.0~100.0)

forward(*speed*)

前進

パラメータ

speed (*float*) -- モーター速度 (-100.0~100.0)

backward(*speed*)

後退

パラメータ

speed (*float*) -- モーター速度 (-100.0~100.0)

turn_left(*speed*)

左折

パラメータ

speed (*float*) -- モーター速度 (-100.0~100.0)

turn_right(*speed*)

右折

パラメータ

speed (*float*) -- モーター速度 (-100.0~100.0)

7.5.2 クラス Motor

例

```
# 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*)

ベースクラス: object

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

モーターを初期化する

パラメータ

- **pwm** (robot_hat.pwm.PWM) -- モーター速度制御用 PWM ピン
- **dir** (robot_hat.pin.Pin) -- モーター方向制御ピン

speed(*speed=None*)

モーター速度の取得または設定

パラメータ

speed (*float*) -- モーター速度 (-100.0~100.0)

set_is_reverse(*is_reverse*)

モーターの逆転を設定するかどうか

パラメータ

is_reverse (*bool*) -- True または False

7.6 モジュール modules

7.6.1 クラス Ultrasonic

例

```
# 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 クラス ADXL345

例

```
# Import ADXL345 class
from robot_hat import ADXL345

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

(次のページに続く)

(前のページからの続き)

```
# 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*)

ベースクラス: *I2C*

ADXL345 モジュール

X = 0

X

Y = 1

Y

Z = 2

Z

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

ADXL345 を初期化する

パラメータ

address (*int*) -- ADXL345 のアドレス

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

ADXL345 から軸を読み取る

パラメータ

axis (*int*) -- 軸の値 (g) を読み取る、ADXL345.X、ADXL345.Y または ADXL345.Z、すべての軸の場合は None

戻り値 軸の値、またはすべての軸のリスト

戻り値の型

float/list

7.6.3 クラス RGB_LED

例

```
# 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*)

シンプルな 3 ピン RGB LED

ANODE = 1

共通アノード

CATHODE = 0

共通カソード

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

RGB LED を初期化する

パラメータ

- **r_pin** (robot_hat.PWM) -- 赤用の PWM オブジェクト
- **g_pin** (robot_hat.PWM) -- 緑用の PWM オブジェクト
- **b_pin** (robot_hat.PWM) -- 青用の PWM オブジェクト
- **common** (*int*) -- RGB_LED.ANODE または RGB_LED.CATHODE、デフォルトは ANODE

例外

- **ValueError** -- 共通が ANODE または CATHODE でない場合
- **TypeError** -- r_pin、g_pin、または b_pin が PWM オブジェクトでない場合

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

RGB LED に色を書き込む

パラメータ

color (*str/int/tuple/list*) -- 書き込む色、"#"で始まる 16 進数文字列、24 ビット整数、または (red, green, blue) のタプル

7.6.4 クラス Buzzer

例

インポートと初期化

```
# 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"))
```

アクティブブザーのビープ音

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

パッシブブザーの簡単な使用方法

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

(次のページに続く)

(前のページからの続き)

```
# 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))
```

パッシブブザーの手動制御

```
# 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()
```

曲を演奏する！ベイビーシャーク！

```
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")
```

(次のページに続く)

(前のページからの続き)

```
    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*]*)

ブザーを初期化する

パラメータ

pwm (*robot_hat.PWM/robot_hat.Pin*) -- パッシブブザー用の PWM オブジェクトまたはアクティブブザー用のピンオブジェクト

on()

ブザーをオンにする

off()

ブザーをオフにする

freq(*freq: float*)

パッシブブザーの周波数を設定する

パラメータ

freq (*int/float*) -- ブザーの周波数、Music.NOTES を使用して音符の周波数を取得する

例外

TypeError -- アクティブブザーに設定した場合

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

周波数を演奏する

パラメータ

- **freq** (*float*) -- 演奏する周波数、Music.note() を使用して音符の周波数を取得できる
- **duration** (*float*) -- 各音符の持続時間（秒）、None は連続して演奏することを意味する

例外

TypeError -- アクティブブザーに設定した場合

7.6.5 クラス Grayscale_Module

例

```
# 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 チャンネルグレースケールモジュール

LEFT = 0

左チャンネル

MIDDLE = 1

中央チャンネル

RIGHT = 2

右チャンネル

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

グレースケールモジュールを初期化する

パラメータ

- **pin0** (*robot_hat.ADC/int*) -- チャンネル 0 用の ADC オブジェクトまたは整数
- **pin1** (*robot_hat.ADC/int*) -- チャンネル 1 用の ADC オブジェクトまたは整数
- **pin2** (*robot_hat.ADC/int*) -- チャンネル 2 用の ADC オブジェクトまたは整数
- **reference** (*1*3 list, [int, int, int]*) -- 基準電圧

reference(*ref: list = None*) → list

基準値の取得と設定

パラメータ

ref (*list*) -- 基準値、基準値を取得する場合は None

戻り値

基準値

戻り値の型

list

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

ライン状態の読み取り

パラメータ

datas (*list*) -- グレースケールデータのリスト、None の場合はセンサーから読み取り

戻り値

ライン状態のリスト、0 は白、1 は黒

戻り値の型

list

read(*channel: int = None*) → list

チャンネルまたはすべてのデータを読み取る

パラメータ

channel (*int/None*) -- 読み取るチャンネル、すべてを読み取るには空のままにする。0、1、2 または Grayscale_Module.LEFT、Grayscale_Module.CENTER、Grayscale_Module.RIGHT

戻り値

グレースケールデータのリスト

戻り値の型

list

7.7 クラス Robot

例

```
# 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*)

ベースクラス: *_Basic_class*

ロボットクラス

このクラスは Robot HAT を使ってサーボロボットを作るためのものです

サーボの初期化、全サーボが特定の速度で動くこと、サーボのオフセットなどがあり、ロボットを作りやすくなっています。SunFounder の Pi シリーズロボットはすべてこのクラスを使用しています。詳細はそれらをチェックしてください。

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

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

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

move_list = {}

プリセットアクション

max_dps = 428

サーボの最大度/秒

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

ロボットクラスを初期化する

パラメータ

- **pin_list** (*list*) -- ピン番号のリスト [0-11]
- **db** (*str*) -- 設定ファイルのパス
- **name** (*str*) -- ロボットの名前
- **init_angles** (*list*) -- 初期角度のリスト
- **init_order** (*list*) -- 初期化の順序のリスト（突然の大電流の場合、サーボは 1 つずつ初期化され、電源電圧が下がることがあります。デフォルトの順序はピンリストです。場合によっては異なる順序が必要です。このパラメータを使用して設定してください。）

new_list(*default_value*)

デフォルト値でサーボ角度のリストを作成する

パラメータ

default_value (*int or float*) -- サーボ角度のデフォルト値

戻り値

サーボ角度のリスト

戻り値の型

list

servo_write_raw(*angle_list*)

サーボ角度を特定の生の角度に設定する

パラメータ

angle_list (*list*) -- サーボ角度のリスト

servo_write_all(*angles*)

元の角度とオフセットを使ってサーボ角度を特定の角度に設定する

パラメータ

angles (*list*) -- サーボ角度のリスト

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

速度または BPM でサーボを特定の角度に動かす

パラメータ

- **targets** (*list*) -- サーボ角度のリスト
- **speed** (*int or float*) -- サーボ移動の速度
- **bpm** (*int or float*) -- 分あたりの拍数

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

動作名とステップと速度を使ってプレフィックスアクションを実行する

パラメータ

- **motion_name** (*str*) -- 動作
- **step** (*int*) -- 動作のステップ
- **speed** (*int or float*) -- 動作の速度

set_offset(*offset_list*)

サーボ角度のオフセットを設定する

パラメータ

offset_list (*list*) -- サーボ角度のリスト

calibration()

全てのサーボをホームポジションに動かす

reset()

サーボを元の位置にリセットする

7.8 クラス Music

警告:

- スピーカーが動作しない場合、このスクリプトを実行する際には sudo を追加する必要があります。
- *Q3:* **スピーカーから音が出ないのはなぜですか？**.

例

初期化

```
# Import Music class
from robot_hat import Music

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

トーンを演奏する

```
# 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))
```

(次のページに続く)

(前のページからの続き)

```
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))
```

(次のページに続く)

(前のページからの続き)

```
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))
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))
```

サウンドを演奏する

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

(次のページに続く)

(前のページからの続き)

```
# Get sound length
music.sound_length("file.wav")
```

音楽を演奏する

```
# 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**

ベースクラス: *_Basic_class*

音楽、サウンドエフェクト、音符の制御を演奏する

NOTE_BASE_FREQ = 440

計算用の基準音の周波数（A4）

NOTE_BASE_INDEX = 69

計算用の基準音のインデックス（A4）MIDI 互換

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']

音符の名前、MIDI 互換

__init__()

基本クラスを初期化する

パラメータ

debug_level (*str/int*) -- デバッグレベル、0（クリティカル）、1（エラー）、2（警告）、3（情報）または 4（デバッグ）

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

拍子記号の設定/取得

パラメータ

- **top** (*int*) -- 拍子記号の上部の数字
- **bottom** (*int*) -- 拍子記号の下部の数字

戻り値

拍子記号

戻り値の型

tuple

key_signature(*key: int = None*)

キー記号の設定/取得

パラメータ

key (*int/str*) -- キー記号は KEY_XX_MAJOR または「#」、「##」、「bbb」、「bbbb」の文字列を使用

戻り値

キー記号

戻り値の型

int

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

テンポ（分あたりの拍数）の設定/取得

パラメータ

- **tempo** (*float*) -- テンポ
- **note_value** -- 音価（1、1/2、Music.HALF_NOTE など）

戻り値

テンポ

戻り値の型

int

beat(*beat*)

テンポから拍子の遅延を秒単位で計算する

パラメータ

beat (*float*) -- 拍子インデックス

戻り値

拍子の遅延

戻り値の型

float

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

音符の周波数を取得する

パラメータ

- **note_name** (*string*) -- 音符の名前（NOTES 参照）
- **natural** (*bool*) -- ナチュラルノートの場合

戻り値

音符の周波数

戻り値の型

float

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

サウンドエフェクトファイルを再生する

パラメータ

filename (*str*) -- サウンドエフェクトファイル名

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

サウンドエフェクトをスレッドで再生する（バックグラウンドで）

パラメータ

- **filename** (*str*) -- サウンドエフェクトファイル名
- **volume** (*int*) -- 音量 0-100、空欄にすると音量は変わらない

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

音楽ファイルを再生する

パラメータ

- **filename** (*str*) -- サウンドファイル名
- **loops** (*int*) -- ループ回数、0: 無限ループ、1:1 回再生、2:2 回再生、...
- **start** (*float*) -- 開始時間（秒）
- **volume** (*int*) -- 音量 0-100、空欄にすると音量は変わらない

music_set_volume(*value*)

音楽の音量を設定する

パラメータ

value (*int*) -- 音量 0-100

music_stop()

音楽を停止する

music_pause()

音楽を一時停止する

music_resume()

音楽を再開する

music_unpause()

音楽の一時停止を解除する（音楽を再開する）

sound_length(*filename*)

サウンドエフェクトの長さを秒で取得する

パラメータ

filename (*str*) -- サウンドエフェクトファイル名

戻り値

長さ（秒）

戻り値の型

float

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

演奏用のトーンデータを取得する

パラメータ

- **freq** (*float*) -- 周波数
- **duration** (*float*) -- 持続時間（秒）

戻り値

トーンデータ

戻り値の型

list

play_tone_for(*freq*, *duration*)

指定された持続時間のトーンを演奏する

パラメータ

- **freq** (*float*) -- 周波数、NOTES を使用して周波数を取得できる
- **duration** (*float*) -- 持続時間（秒）

7.9 クラス TTS

警告:

- スピーカーが動作しない場合、このスクリプトを実行する際には sudo を追加する必要があります。
- *Q3:* **スピーカーから音が出ないのはなぜですか？**.

例

```
# Import TTS class
from robot_hat import TTS

# Initialize TTS class
tts = TTS(lang='en-US')
# Speak text
```

(次のページに続く)

(前のページからの続き)

```
tts.say("Hello World")
# show all supported languages
print(tts.supported_lang())
```

API

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

ベースクラス: *_Basic_class*

テキストから音声への変換クラス

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

pico2wave のためのサポートされている TTS 言語

ESPEAK = 'espeak'

espeak TTS エンジン

PICO2WAVE = 'pico2wave'

pico2wave TTS エンジン

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

TTS クラスを初期化する。

パラメータ

engine (*str*) -- TTS エンジン、TTS.PICO2WAVE または TTS.ESPEAK

say(*words*)

言葉を話す。

パラメータ

words (*str*) -- 話す言葉。

espeak(*words*)

espeak で言葉を話す。

パラメータ

words (*str*) -- 話す言葉。

pico2wave(*words*)

pico2wave で言葉を話す。

パラメータ

words (*str*) -- 話す言葉。

lang(**value*)

言語を設定/取得する。現在の言語を取得するために空のままにする。

パラメータ

value (*str*) -- 言語。

supported_lang()

サポートされている言語を取得する。

戻り値

サポートされている言語。

戻り値の型

list

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

espeak のパラメータを設定する。

パラメータ

- **amp** (*int*) -- 振幅。
- **speed** (*int*) -- 速度。
- **gap** (*int*) -- ギャップ。
- **pitch** (*int*) -- ピッチ。

7.10 モジュール utils

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

音量を設定する

パラメータ

value (*int*) -- 音量（0〜100）

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

コマンドを実行し、状態と出力を返す

パラメータ

cmd (*str*) -- 実行するコマンド

戻り値

状態、出力

戻り値の型

tuple

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

コマンドがインストールされているかどうかを確認する

パラメータ

cmd (*str*) -- 確認するコマンド

戻り値

インストールされている場合は True

戻り値の型

bool

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

ある範囲の値を別の範囲にマップする

パラメータ

- **x** (*float/int*) -- マップする値
- **in_min** (*float/int*) -- 入力最小値
- **in_max** (*float/int*) -- 入力最大値
- **out_min** (*float/int*) -- 出力最小値
- **out_max** (*float/int*) -- 出力最大値

戻り値

マップされた値

戻り値の型

float/int

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

IP アドレスを取得する

パラメータ

ifaces (*list*) -- チェックするインターフェース

戻り値

見つかった場合は IP アドレス、見つからない場合は False

戻り値の型

str/False

robot_hat.utils.**reset_mcu**()

Robot Hat 上の MCU をリセットする。

これは、MCU が何らかの理由で I2C データ転送ループに固まり、Raspberry Pi が ADC の読み取り、PWM の操作などで IOError を取得している場合に役立ちます。

robot_hat.utils.**get_battery_voltage**()

バッテリー電圧を取得する

戻り値

バッテリー電圧（V）

戻り値の型

float

7.11 クラス FileDB

例

```
# 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"))

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

これで bash で設定ファイル config を確認できます。

```
cat config
```

API

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

ベースクラス: object

ファイルベースのデータベース。

特定のファイルで引数を読み書きするファイルベースのデータベースです。

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

db_file を初期化すると、データを保存するファイルになります。

パラメータ

- **db** (*str*) -- データを保存するファイル。
- **mode** (*str*) -- ファイルのモード。
- **owner** (*str*) -- ファイルの所有者。

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

ファイルが存在するかチェックし、存在しない場合は作成する。

パラメータ

- **file_path** (*str*) -- チェックするファイル
- **mode** (*str*) -- ファイルのモード。
- **owner** (*str*) -- ファイルの所有者。

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

データの名前で値を取得する

パラメータ

- **name** (*str*) -- 引数の名前
- **default_value** (*str*) -- 引数のデフォルト値

戻り値

引数の値

戻り値の型

str

set(*name*, *value*)

名前で値を設定する。または、引数が存在しない場合は作成する

パラメータ

- **name** (*str*) -- 引数の名前
- **value** (*str*) -- 引数の値

7.12 クラス I2C

例

```
# 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)
```

I2C プロトコルの詳細については、adc.py と pwm.py をご覧ください

API

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

ベースクラス: *_Basic_class*

I2C バスの読み書き機能

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

I2C バスを初期化する

パラメータ

- **address** (*int*) -- I2C デバイスアドレス

- **bus** (*int*) -- I2C バス番号

scan()

I2C バスをスキャンしてデバイスを検出する

戻り値

見つかったデバイスの I2C アドレスのリスト

戻り値の型

list

write(*data*)

I2C デバイスにデータを書き込む

パラメータ

data (*int/list/bytearray*) -- 書き込むデータ

例外

書き込みが int、リスト、またはバイト配列でない場合は ValueError

read(*length=1*)

I2C デバイスからデータを読み取る

パラメータ

length (*int*) -- 受信するバイト数

戻り値

受信したデータ

戻り値の型

list

mem_write(*data*, *memaddr*)

特定のレジスタアドレスにデータを送信する

パラメータ

- **data** (*int/list/bytearray*) -- 送信するデータ、int、リスト、またはバイト配列
- **memaddr** (*int*) -- レジスタアドレス

例外

ValueError -- データが int、リスト、またはバイト配列でない場合

mem_read(*length*, *memaddr*)

特定のレジスタアドレスからデータを読み取る

パラメータ

- **length** (*int*) -- 受信するバイト数
- **memaddr** (*int*) -- レジスタアドレス

戻り値

エラーがない場合は受信したバイト配列データ、エラーがある場合は False

戻り値の型

list/False

is_avaliable()

I2C デバイスが利用可能かどうかを確認する

戻り値

I2C デバイスが利用可能な場合は True、そうでない場合は False

戻り値の型

bool

7.13 クラス _Basic_class

_Basic_class はすべてのクラスのロガークラスで、特定のクラスのログを見たい場合は、デバッグ引数を追加するだけです。

例

```
# 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'*)

すべてのクラスの基本クラス

デバッグ機能付き

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

デバッグレベル

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

デバッグレベルの名称

__init__(*debug_level='warning'*)

基本クラスを初期化する

パラメータ

debug_level (*str/int*) -- デバッグレベル、0（重大）、1（エラー）、2（警告）、3（情報）または 4（デバッグ）

property debug_level

デバッグレベル

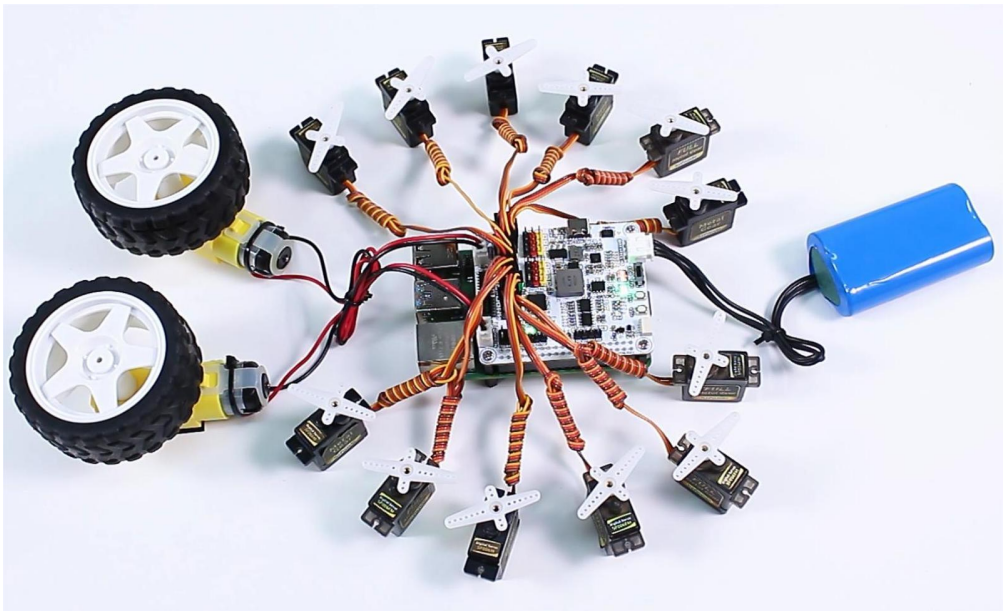
第 8 章

いくつかのプロジェクト

ここでは、Robot HAT を使用して実装された魅力的なプロジェクトのコレクションを紹介します。詳細なコードを提供し、これらのプロジェクトを自分で試す機会を提供します。

8.1 サーボとモーターの制御

このプロジェクトでは、12 個のサーボと 2 つのモーターが同時に動作します。



ただし、サーボやモーターの始動電流が高い場合は、それらを個別に起動して、電源電流不足を避けることをお勧めします。電流不足は Raspberry Pi の再起動につながる可能性があります。

コード

```
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)
```

(次のページに続く)

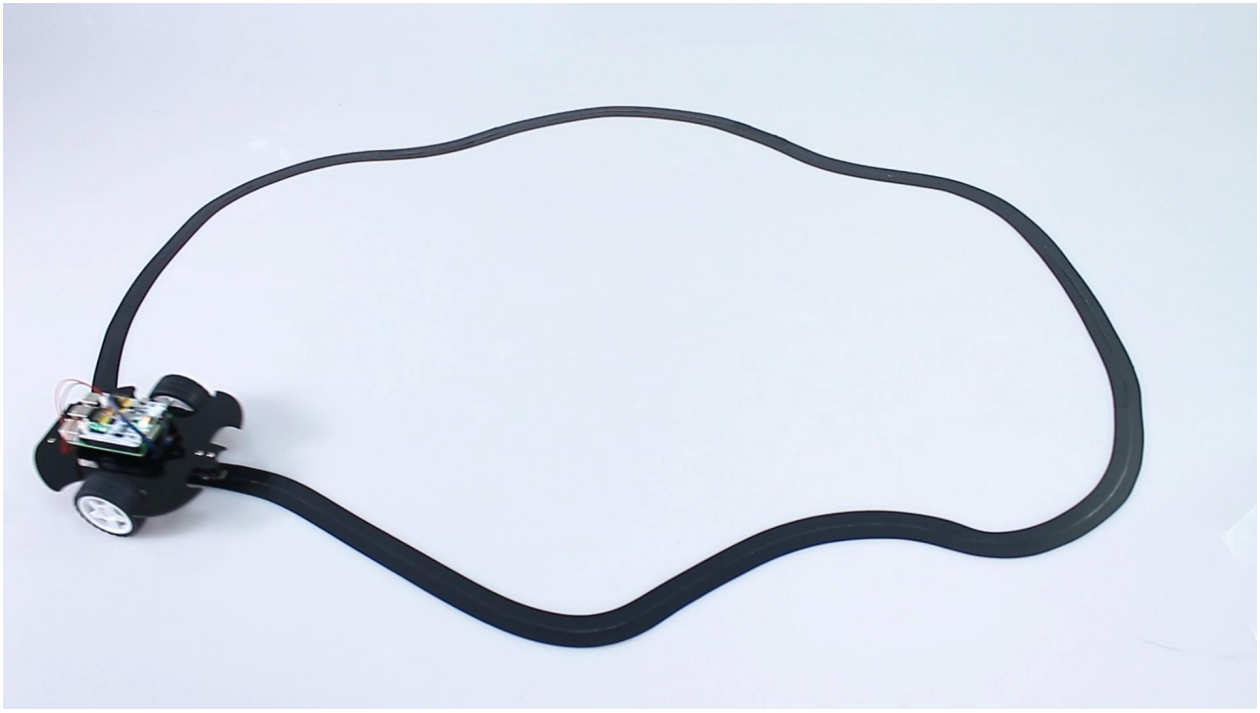
(前のページからの続き)

```
            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 カー

シンプルな実験に適しているだけでなく、スマートカーなどのロボティクスにおける中央コントローラとしても、Robot HAT は理想的です。

このプロジェクトでは、シンプルなラインフォローイングカーを作りました。



コード

```
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
```

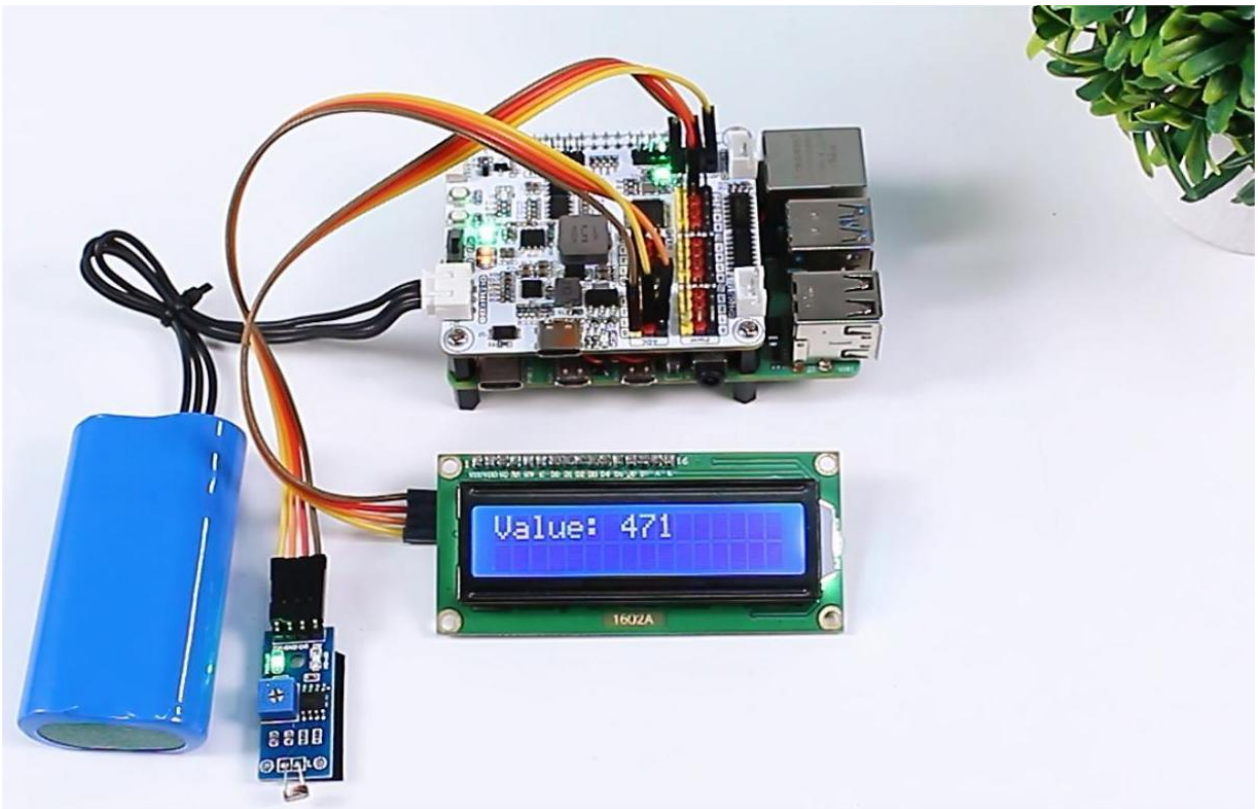
(次のページに続く)

(前のページからの続き)

```
            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 フォトレジスタモジュールから読み取る

このプロジェクトでは、光の強度を検出し、I2C LCD1602 に表示します。



手順

1. このプロジェクトでは I2C LCD1602 を使用しているため、関連するライブラリをダウンロードして機能させる必要があります。

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

2. I2C 用に smbus2 をインストールします。

```
sudo pip3 install smbus2
```

3. 以下のコードを Raspberry Pi に保存し、例えば photoresistor.ty のような名前を付けます。

```
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()
```

(次のページに続く)

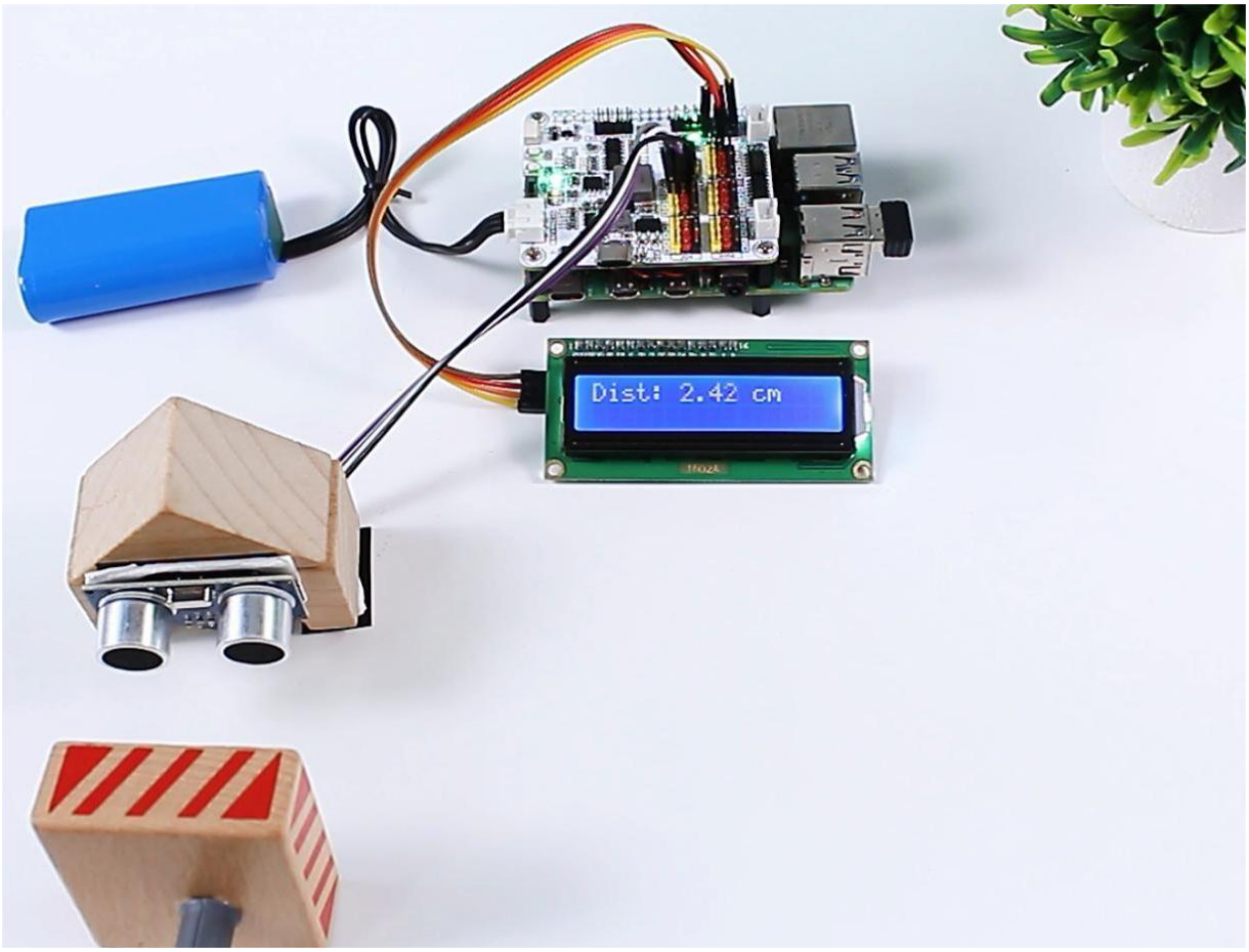
(前のページからの続き)

```
except KeyboardInterrupt:
    destroy()
except Exception as e:
    # Clear the LCD and print error message in case of an exception
    destroy()
    print("Error:", e)
```

4. このコードを実行するには、コマンド sudo python3 photoresistor.ty を使用します。

8.4 超音波モジュールからの読み取り

このプロジェクトでは、超音波センサーを使用して距離を測定し、その読み取り値を I2C LCD1602 に表示します。



手順

1. このプロジェクトでは I2C LCD1602 を使用しているため、関連するライブラリをダウンロードして機能させる必要があります。

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

2. I2C 用に smbus2 をインストールします。

```
sudo pip3 install smbus2
```

3. 以下のコードを Raspberry Pi に保存し、例えば ultrasonic.ty という名前を付けます。

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

# 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
```

(次のページに続く)

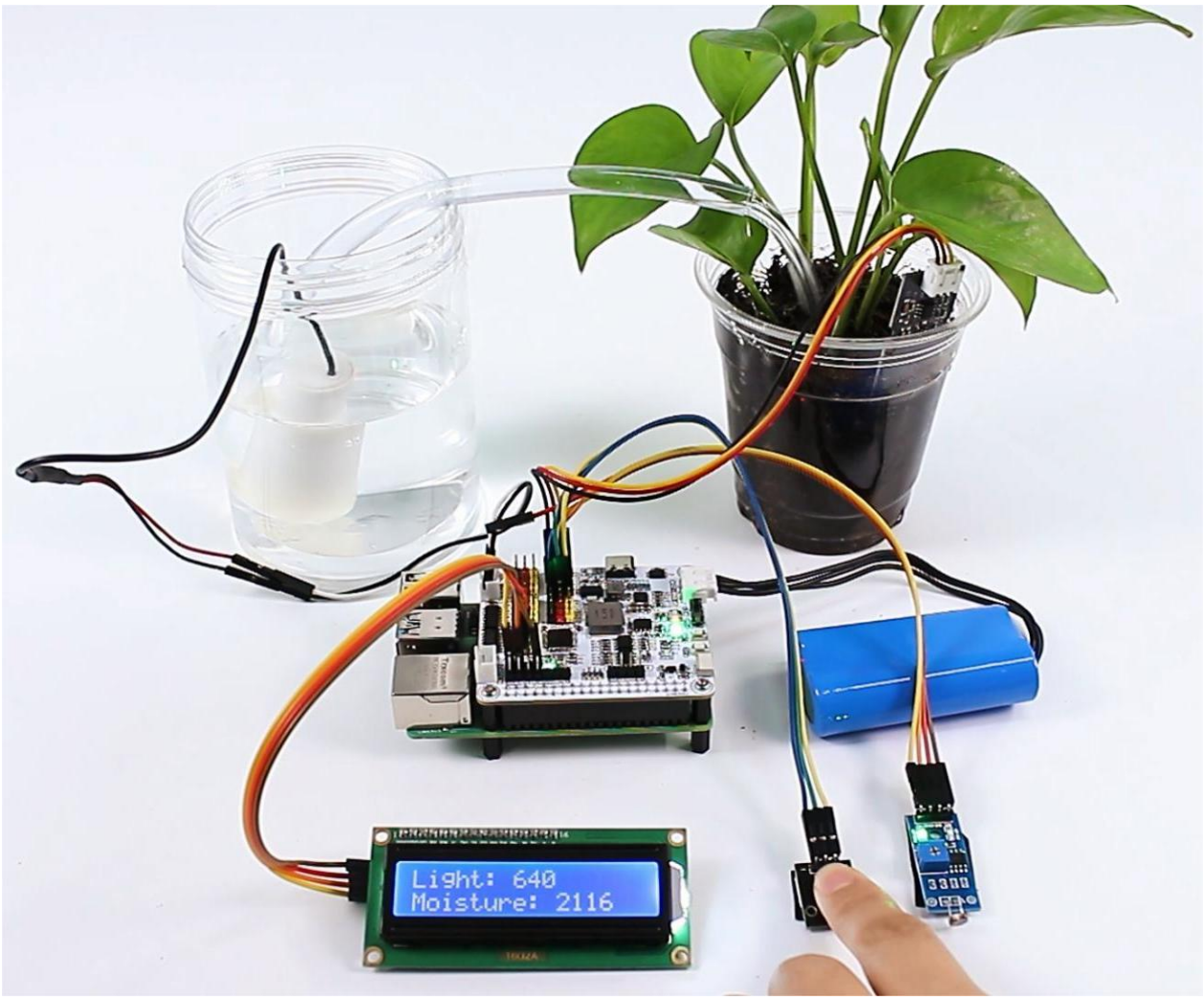
(前のページからの続き)

```
        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. このコードを実行するには、 `sudo python3 ultrasonic.ty` コマンドを使用します。

8.5 プラントモニター

このプロジェクトでは、光の強度と土壌の水分レベルの両方を検出し、I2C LCD1602 に表示します。土壌の水分が不足していると感じたら、ボタンモジュールを押して鉢植えに水をやることができます。



手順

1. このプロジェクトでは I2C LCD1602 を使用しているため、関連するライブラリをダウンロードして機能させる必要があります。

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

2. I2C 用に smbus2 をインストールします。

```
sudo pip3 install smbus2
```

3. 以下のコードを Raspberry Pi に保存し、例えば plant_monitor.ty という名前を付けます。

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

(次のページに続く)

(前のページからの続き)

```
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)
```

(次のページに続く)

(前のページからの続き)

```
            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)
        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. このコードを実行するには、 `sudo python3 plant_monitor.ty` コマンドを使用します。

8.6 何かを話す

このセクションでは、テキストを音声に変換して、Robot HAT に大声で話させる方法を学びます。

手順

1. コマンドラインからテキストを取得して Robot HAT がそれを話すようにします。これを実現するために、以下のコードを .py ファイルとして保存します。例えば tts.py などです。

```
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())
```

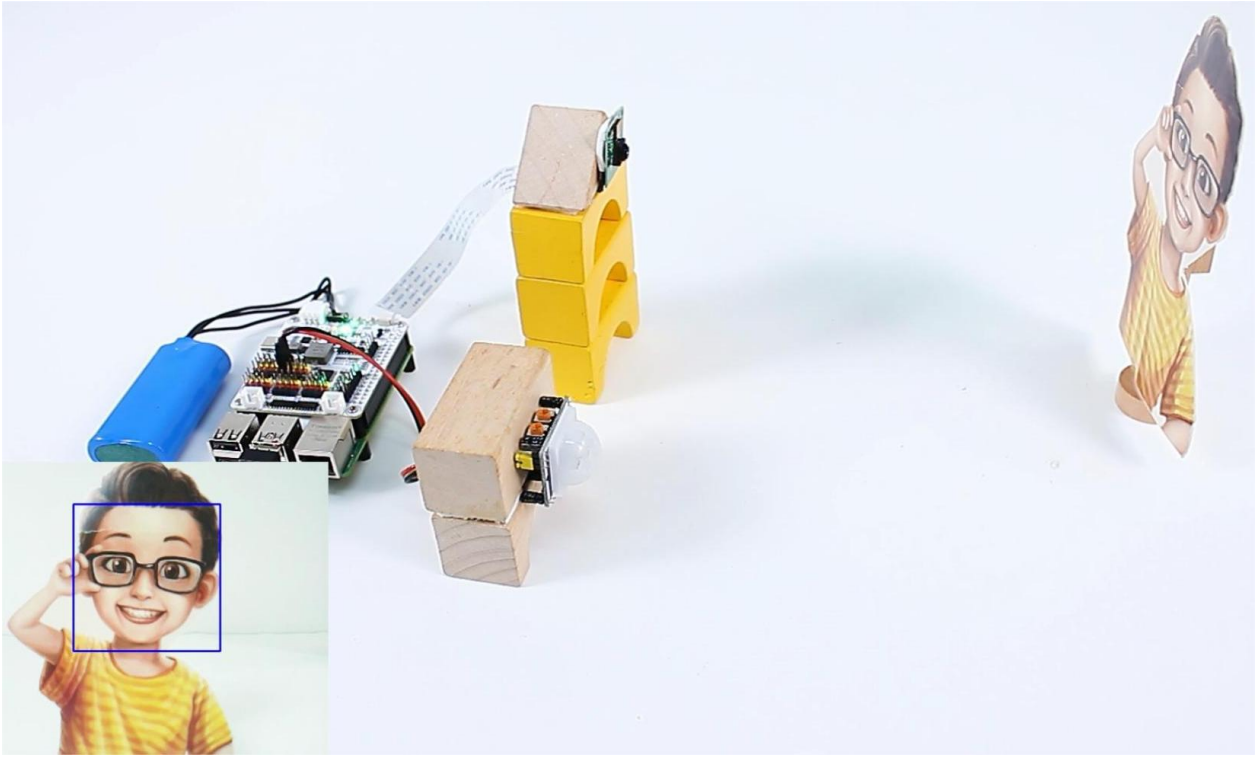
2. Robot HAT に特定の文章を発声させるには、次のコマンドを使用します： sudo python3 tts.py "**任意のテキスト**" - 単に "**任意のテキスト**" を希望のフレーズに置き換えてください。

注釈:

- *Q3:* **スピーカーから音が出ないのはなぜですか？**

8.7 セキュリティシステム

このプロジェクトでは、シンプルなセキュリティシステムを作成しました。PIR センサーが人の動きを検出すると、カメラが起動します。顔が検出されると、写真を撮り、同時に警告メッセージを発信します。



手順

1. 顔検出のための vilib ライブラリをインストールします。

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

2. 以下のコードを Raspberry Pi に保存し、例えば security.ty という名前を付けます。

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



# Initialize the TTS class
```

(次のページに続く)

(前のページからの続き)

```
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
```

(次のページに続く)

(前のページからの続き)

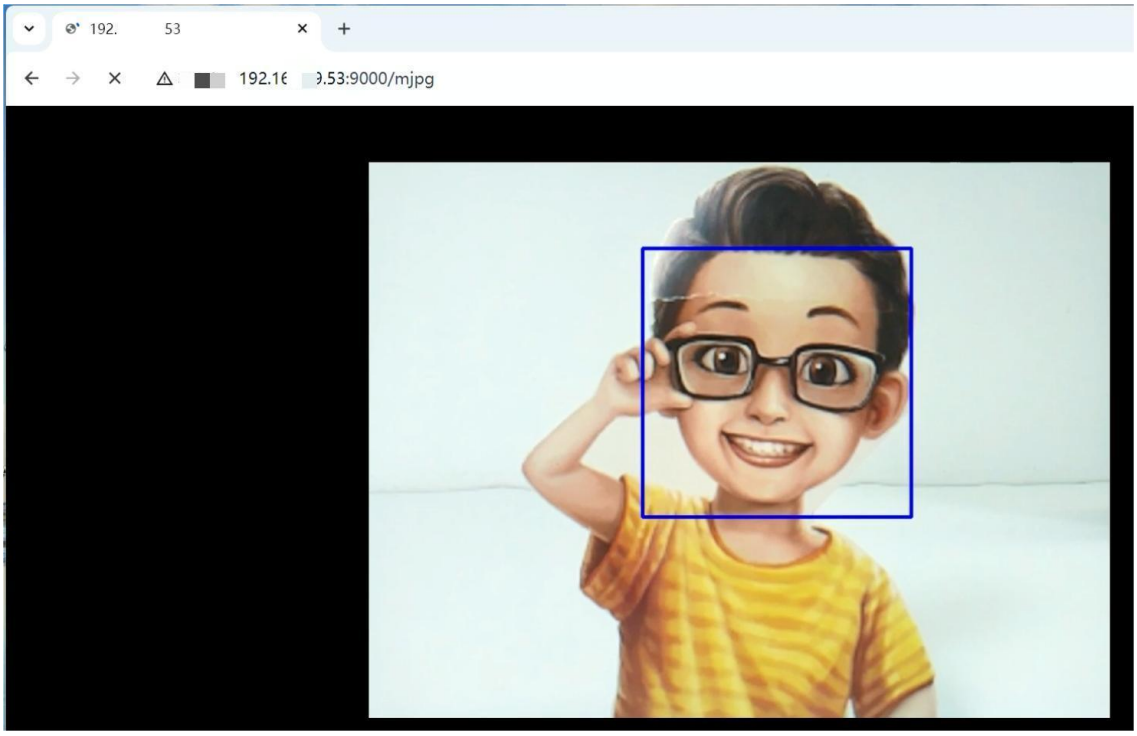
```
        # If no motion is detected, turn off the camera
        elif motion_detected:
            print("No motion detected. Finalizing camera...")
            Vilib.camera_close()
            motion_detected = False
            sleep(2)  # Delay before re-enabling motion detection

        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**このコードを実行するには、** `sudo python3 security.py` **コマンドを使用します。**

注釈:

- *Q3:* **スピーカーから音が出ないのはなぜですか？**

4. ウェブブラウザを開いて `http://rpi_ip:9000/mjpg` にアクセスし、キャプチャされた映像を視聴できます。さらに、キャプチャされた顔の画像は `/home/{username}/Pictures/` で見つけることができます。



8.8 コミュニティチュートリアル

•

この文書は、SunFounder Raspberry Pi Robot HAT に関する概要を説明し、その目的、互換性、仕様、テストについて述べています:

- **はじめに**: Raspberry Pi ベースの DIY ロボットプロジェクトの制御を簡素化する Robot HAT の役割について説明します。
- **仕様**: 電源入力、バッテリーの詳細、ポート、モータードライバーの機能など、技術的な仕様について詳述します。
- **ポートの概要**: 電源、デジタル、アナログ、PWM、I2C、SPI、UART、モーターポートなど、各種ポートについて説明します。
- **追加コンポーネント**: ボタン、LED、スピーカーなどの追加コンポーネントと、Raspberry Pi の PIN 配置を強調します。
- **セットアップとテスト**: Robot HAT の取り付け、必要なコンポーネント、LED やサーボモーターなどの機能のテスト手順について案内します。

第 9 章

よくある質問

9.1 Q1: バッテリーを接続しながら、同時に Raspberry Pi に電力を供給することは可能ですか？

A: はい、Robot HAT には逆流防止ダイオードが内蔵されており、Raspberry Pi の電力が Robot HAT に逆流するのを防ぎます。

9.2 Q2: 充電中に Robot HAT を使用することは可能ですか？

A: はい、充電中でも Robot HAT を使用できます。充電中、入力電力は充電チップによって増幅され、バッテリーを充電しながら外部使用のための DC-DC 降圧にも電力を供給します。充電電力は約 10W です。外部電力消費が長期間にわたって高すぎる場合、バッテリーが電力を補うことがあります。これは、使用中の携帯電話が充電されるのと同様です。ただし、同時に充電と使用を行う際には、バッテリーの容量に注意し、完全に消耗させないようにすることが重要です。

9.3 Q3: スピーカーから音が出ないのはなぜですか？

スクリプトが実行されているにもかかわらずスピーカーから音が出ない場合、いくつかの理由が考えられます：

1. `i2samp.sh` スクリプトがインストールされているか確認してください。詳細な指示については、 **スピーカー用の *i2samp.sh* をインストールする** を参照してください。
2. スピーカー関連のスクリプトを実行する際には、管理者権限を得るために sudo を追加する必要があります。例えば、 `sudo python3 tts.py` とします。
3. スピーカー関連のスクリプトを実行する際に、Raspberry Pi の組み込みプログラミングツール（例えば Geany）を使用しないでください。これらのツールは標準ユーザー権限で実行されますが、スピーカーの管理などのハードウェア制御にはより高い権限がしばしば必要です。

Python モジュール索引

r

索引