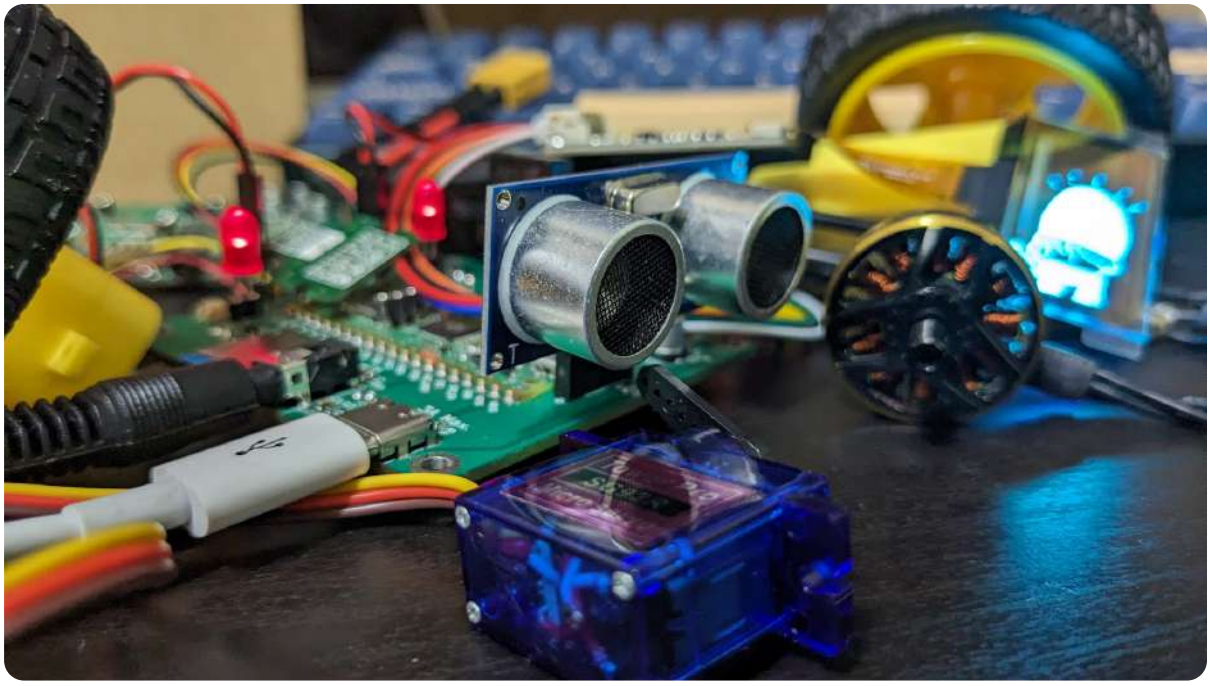
Turtle Pico Kit 仕様書



目次

1. キット概要

Turtle Pico Kitは、**いつまでも**使え、**いろんなもの**が作れる電子工作ツールです。

本仕様書では、キットのそれぞれの部品、及びそれらの制御プログラム例を全12種類ご紹介致します。

まず、キットには以下のものが同梱されています。

1.1. Turtle Pico Board (A)

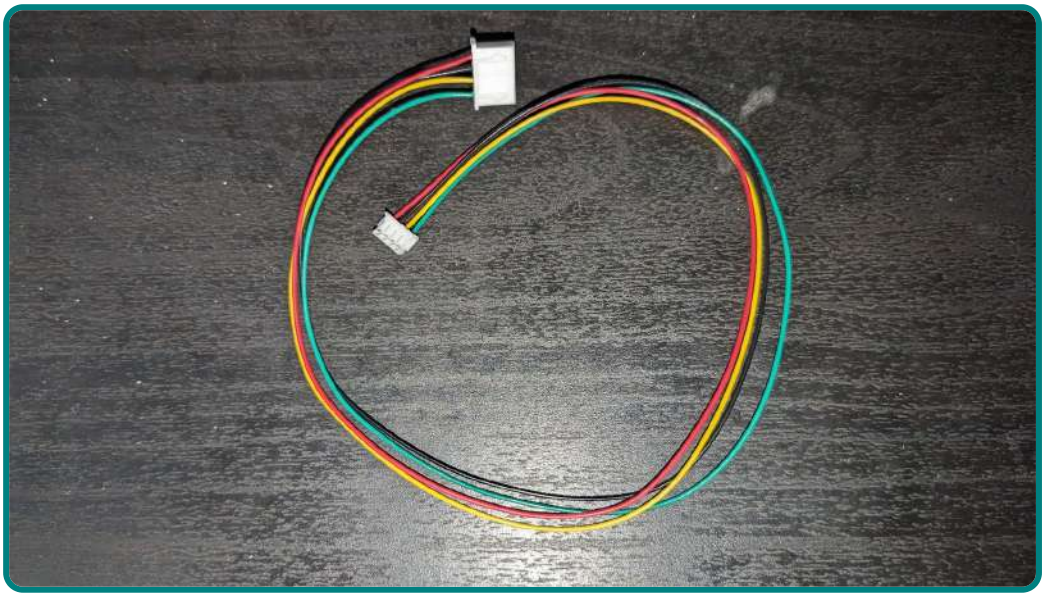
ボード本体です。Raspberry Pi Pico Wの**マイクロコンピュータ**が搭載されており、各種部品にコネクターを通して接続することが可能です。詳しい紹介は、2. Turtle Pico Board 概要にて行います。

1.2. USB Type-C ケーブル (A)

ボード本体とPCを接続するケーブルです。

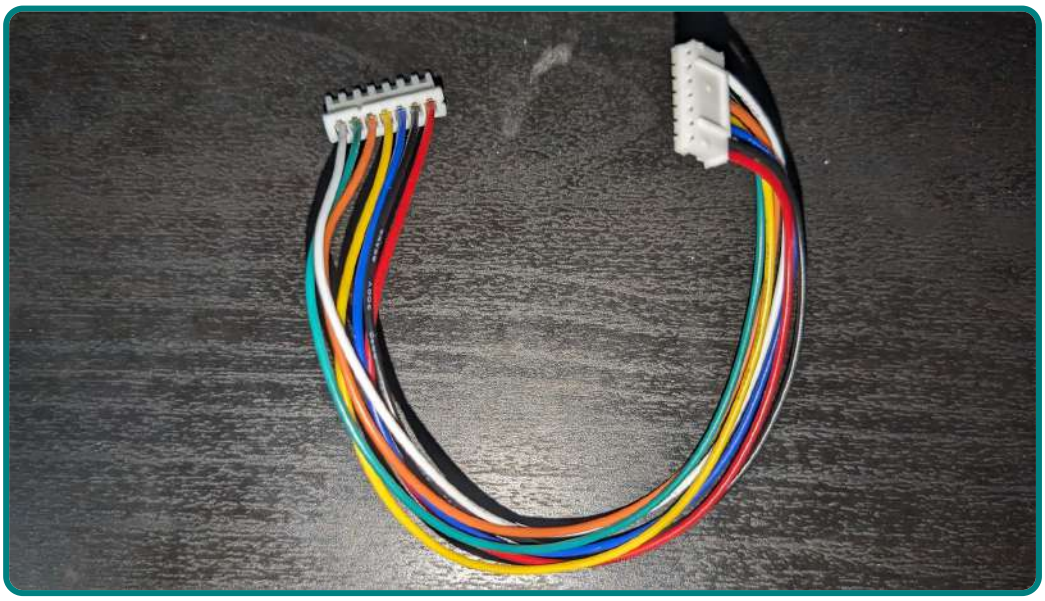
1.3. ZH4-XH4 ケーブル (J)

9軸センサー接続用のケーブルです。ZHがセンサー側、XHがボード側となっています。



1.4. PH7-XH7 ケーブル (G)

透明OLEDディスプレイと接続するためのケーブルです。PHがディスプレイ側、XHがボード側となっています。



1.5. LiPo バッテリー (B)

1.6. ESC (B)への電源供給用のバッテリーです。保管には十分注意してください。

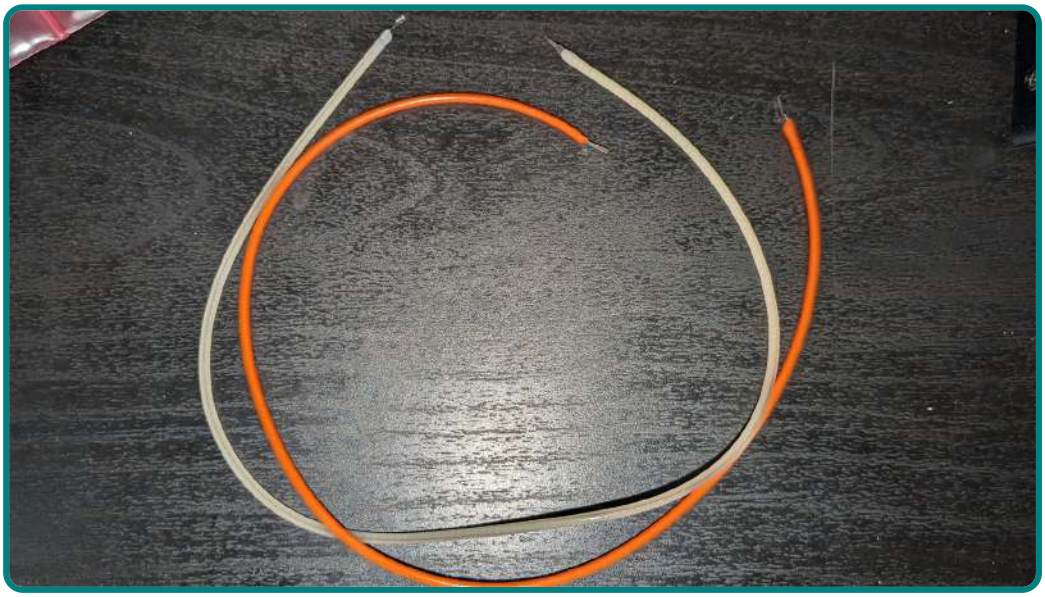
1.6. ESC (B)

ブラシレスDCモーターを制御するための部品です。XT60コネクタでLiPoから電源を供給し、赤茶黄の3ピンのQIコネクタでボード上の3ピンのピンヘッダに接続し、制御します。

1.7. フレキシブルLED (L)

くねくねと曲がるLEDです。型式はiPixelの**CY-FRX3002-M2001**で、赤色と青色のものが同梱されています。ボード上のオープンコレクタ出力のLEDポートに接続することで点灯が可能になります。

詳しくは4.2. LED点灯をご覧ください。



1.8. 透明有機ELディスプレイ (G)

透明な背景に青色のセルが点灯する、WAVESHARE**22545**の透明なOLEDディスプレイです。ボード上のXH7ピンのコネクタから、1.4. PH7-XH7 ケーブル (G)のケーブルを使用して接続します。SSD1306の互換品で**SSD1309**というドライバーが搭載されているので、SSD1306と同様にしてSPI通信での制御が可能です。

詳しくは4.4. 透明有機ELディスプレイをご覧ください。

1.9. Micro SDカード (C)

Turtle Picoのストレージとして使用する**SDカード**で、KIOXIAの**EXCERIA 16GB**です。ボード上のスロットに直接差し込んで使用します。

詳しくは4.5. SDカードをご覧ください。

1.10. サーボモーター (I)

角度の制御が可能なTower Proのマイクロサーボ**SG-92R**です。1.6. ESC (B)と同様のQIコネクタを、ボード上の3ピンのピンヘッダに接続することで簡単に制御が可能です。

詳しくは4.6. サーボモーターをご覧ください。

1.11. ブラシ付きDCモーター (D)

ギアーBOX付きのFEETECH**FT-DC90**というモーターが2個同梱されています。別売りでSwitchScienceで売られている専用のタイヤを使用すれば、簡単に床を走らせることが出来ます。ボード上のMotorと書かれたピンソケットのポートに接続して使用します。

詳しくは4.7. ブラシ付きDCモーターをご覧ください。

1.12. ブラシレスDCモーター (H)

ドローンで使用するブラシレスDCモーター TMOTOR **F2203.5** KV3550モデルです。1.6. ESC (B)の先に接続して使用します。ベースキットには一つしか同梱されていないので、ドローンを作成する場合は別にESCと本モーターを3セットご購入いただく必要があります。

詳しくは4.8. ブラシレスDCモーターをご覧ください。

1.13. 超音波センサー (E)

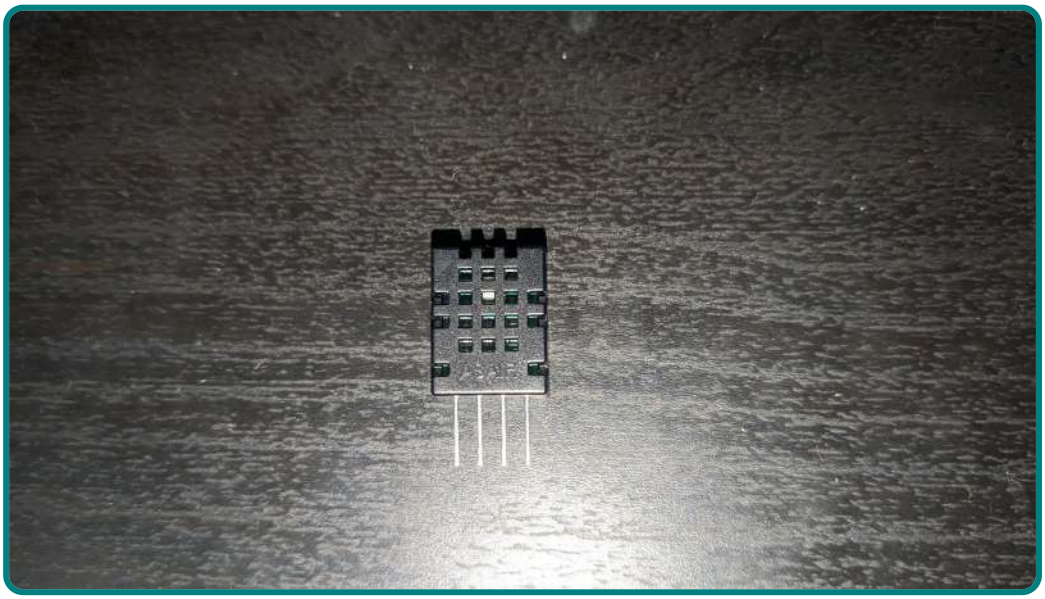
Rainbow E-TechnologyのHC-SR04です。超音波の応答時間を測定することで、障害物までの距離を測定できるセンサーです。Turtle Picoの文字通りの**目玉**となっています。ボード上正面にある4ピンのピンソケットに接続して使用します。

詳しくは4.9. 超音波センサーをご覧ください。

1.14. 温湿度センサー (F)

温度及び湿度を計測できるセンサー、Aosong Guangzhou ElectronicsのDHT20です。I2C通信により、計測値をマイコンにシリアル送信します。ボード上側面の4ピンのピンソケットに接続して使用します。

詳しくは4.10. 温湿度センサーをご覧ください。



1.15. 9軸センサー (J)

ドローンなどの制御に用いる姿勢を検出するセンサー、STMicroelectronicsのLSM9DS1です。具体的には、加速度、ジャイロ、地磁気がそれぞれ3軸分計測できるので9軸センサーです。ボード上裏面の4ピンのXHコネクタに1.3. ZH4-XH4 ケーブル (J)のケーブルを用いて接続し、使用します。

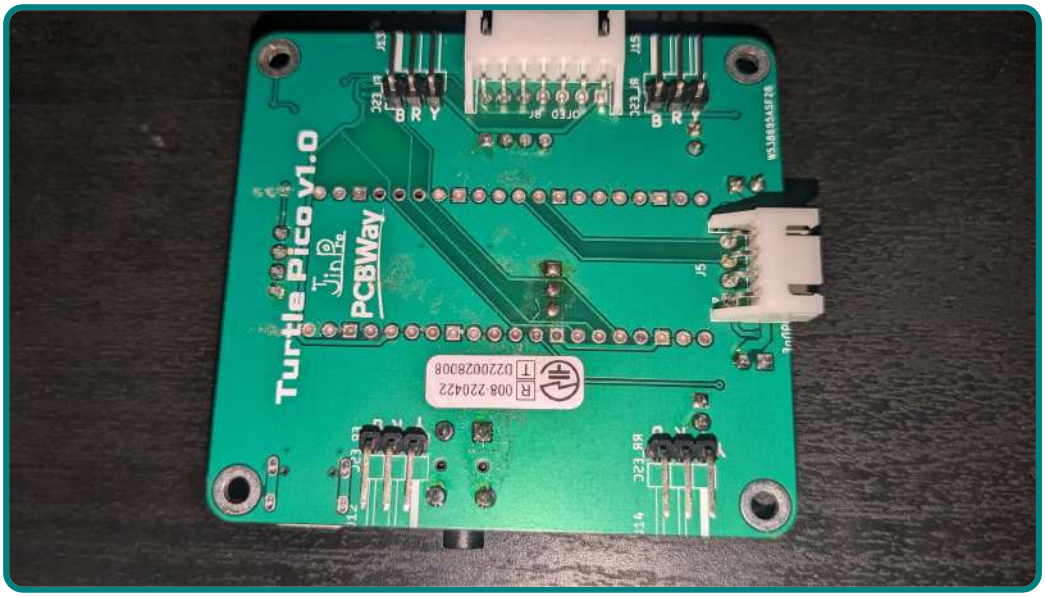
詳しくは4.11. 9軸センサーをご覧ください。

2. Turtle Pico Board 概要

本章では、**Turtle Pico Board**の概要を解説致します。

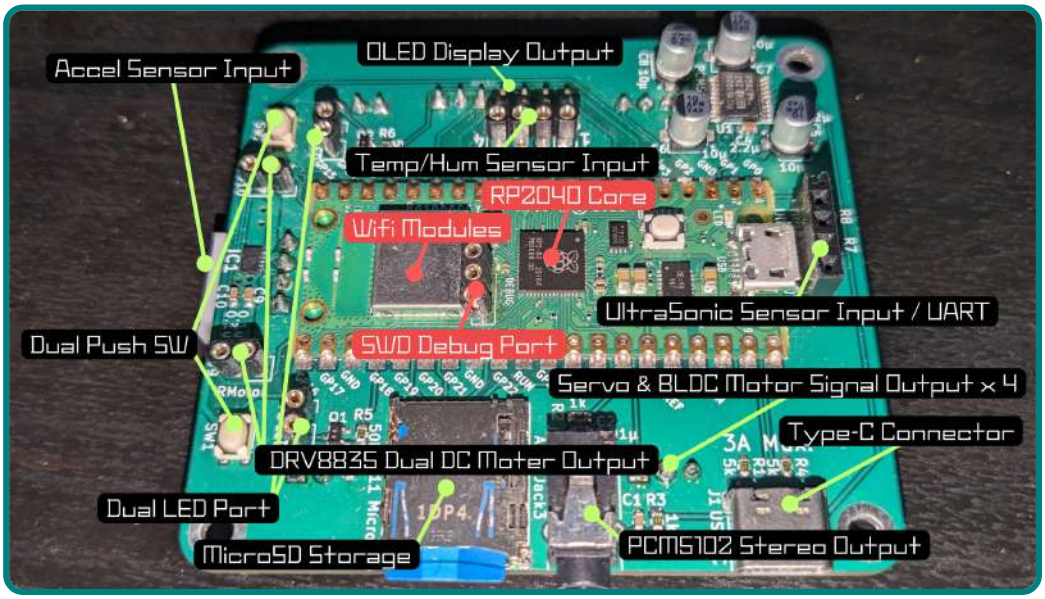
2.1. 外観

Turtle Pico Boardの外観です。上の画像が表、下の画像が裏の写真です。



2.2. インターフェース/機能

インターフェースと**機能**をまとめた画像を下に示します。



- **USB Type-C Connector**

  他のデバイスと通信するための**USBコネクタ**です。電流の定格は、基板の配線幅の関係で**3A**とします。これ以上の電流が流れると、基板上のコネクタ付近が熱を持ち、配線が焼ききれる可能性がありますので、この値は守ってご使用いただくようお願い致します。

- **MicroSD Card Slot (SPI)**

  Micro SDカードを接続するためのスロットです。**SPI**通信で接続します。

- **PCM 5102 Stereo Output (I2S)**

  3.5mmステレオジャックによる**音声出力ポート**です。PCM5102による出力を直結しています。PCM5102とは**I2S**通信で接続します。
  そのままの出力では音が小さいため、スピーカー等大きな出力を必要とする場合はアンプを接続してください。イヤホン等で聞く分に関しては、その必要はありません。

- **OLED Display Port (SPI)**

  透明有機ELの制御基板(SSD1309)に接続するためのポート（裏面）です。**SPI**通信で接続します。

- **Servo & BLDC Signal Output**

  サーボモーターや**ブラシレスDCモーター**に制御信号を送信するためのポートです。

- **DRV8835 Dual DCMotor Output**

  DRV8835モータードライバーに接続する**DCモーター**用のポートが2つついています。DCモーターに流せる定格電流値は、1ポート最大で1.5A、2ポート合計で3Aです。

- **UltraSonic Sensor Input / UART**

  超音波センサーの入出力ポートです。超音波センサーを使用しない場合、中央2つのピンを使用して**UART**通信のポートとしても使用いただけます。

- **Temp/Hum Sensor Input (I2C)**

  温湿度センサーの入力ポートです。**I2C**通信で接続します。

- **Accel Sensor Input (I2C)**

  Accel Sensorとなっていますが実際には**9軸センサー**の入力ポート（裏面）です。**I2C**通信で接続します。

- **Dual Push SW**

  Raspi Pico W直結の**プッシュスイッチ**が2つついています。

- **Dual LED Port**

  トランジスターのコレクターに接続されている、オープンコレクター出力の**LEDポート**です。出力することができる電流は1ポート最大15mAほどで、PWMで明るさを制御することが可能です。

- **SWD Debug Port**

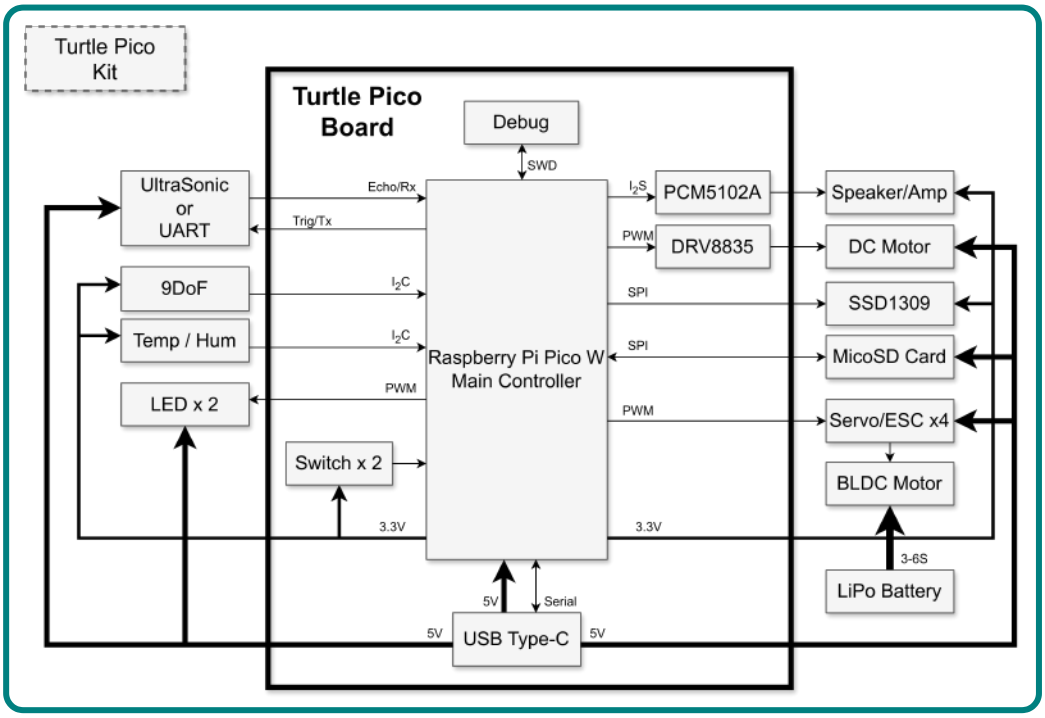
  Pico SDKやRTOSなどで使用する場合に**書き込み/デバッグ**に使用するポートです。**Pico Probe**などを使用して**OpenOCD**などでデバッグをします。

- **WiFi Module**

  Raspberry Pi Pico W内蔵の**WiFiモジュール**です。バンドは2.5GHzのみに対応しています。このモジュールを使用してWebに接続します。

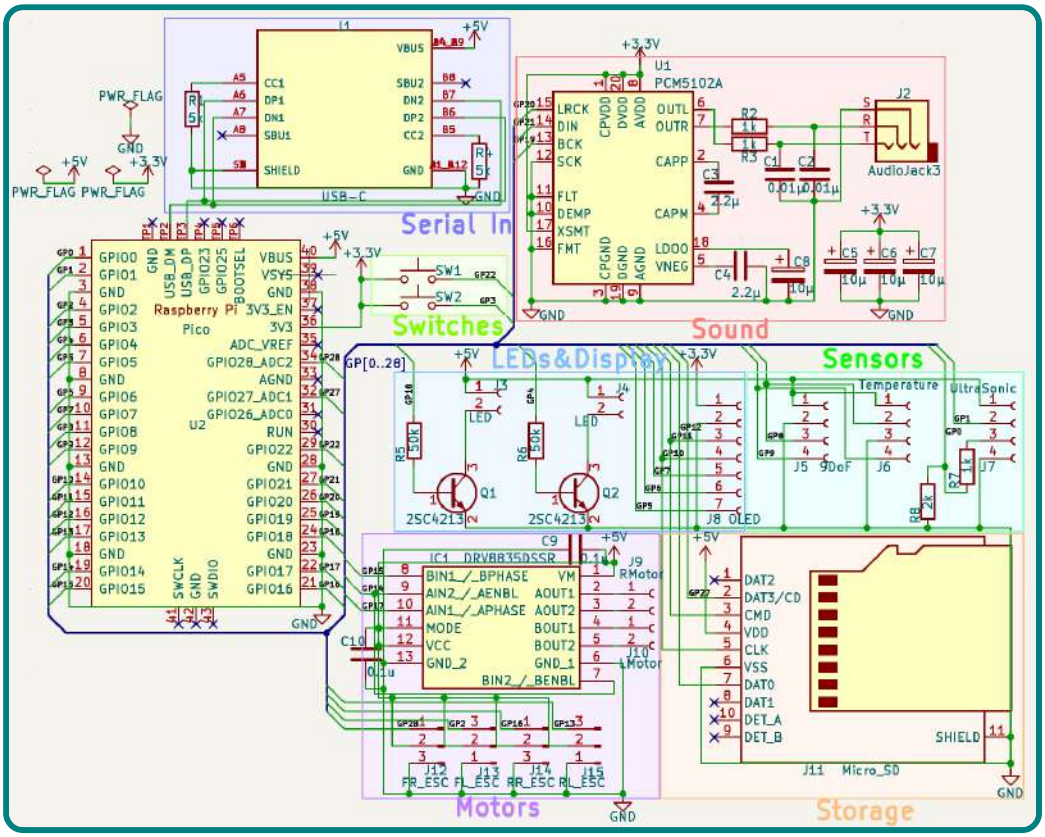
2.3. ブロック図

Turtle Pico Boardの**ブロック図**を以下に示します。



2.4. 回路図/ピンアサイン

Turtle Pico Boardの**回路図**及びRaspberry Pi Pico Wの対応する**ピンアサイン**を以下に示します。



GPIO	Funciton
0	Uart - **Rx** / Ultrasonic - **Echo**
1	Uart - **Tx** / Ultrasonic - **Trig**
2	ESC / Servo - **Signal**
3	Push **SW** (Left)
4	**LED** (Left)
5	OLED(SSD1306) - **RST**
6	OLED(SSD1306) - **DC**
7	OLED(SSD1306) - **CS**
8	I2C - **SDA**
9	I2C - **SCL**
10	SPI - **SCK**
11	SPI - **MOSI**
12	SPI - **MISO**
13	ESC / Servo - **Signal**
14	DRV8835 - **Enable**
15	DRV8835 - **Bin**
16	ESC / Servo - **Signal**
17	DRV8835 - **Ain**
18	**LED** (Right)
19	I2S - **BCLK**(SCK)
20	I2S - **LRCLK**(WS)
21	I2S - **SDATA**
22	Push **SW** (Right)
26	None
27	SD Card - **CS**
28	ESC / Servo - **Signal**

3. 環境構築

Turtle Pico(RaspberryPi Pico)の**環境構築**には様々な方法があります。ここでは、代表的な以下の3つについて、簡単に紹介します。

1. **Terminal (REPL)**
2. **Thonny**
3. **VSCode (Micro Pico)**

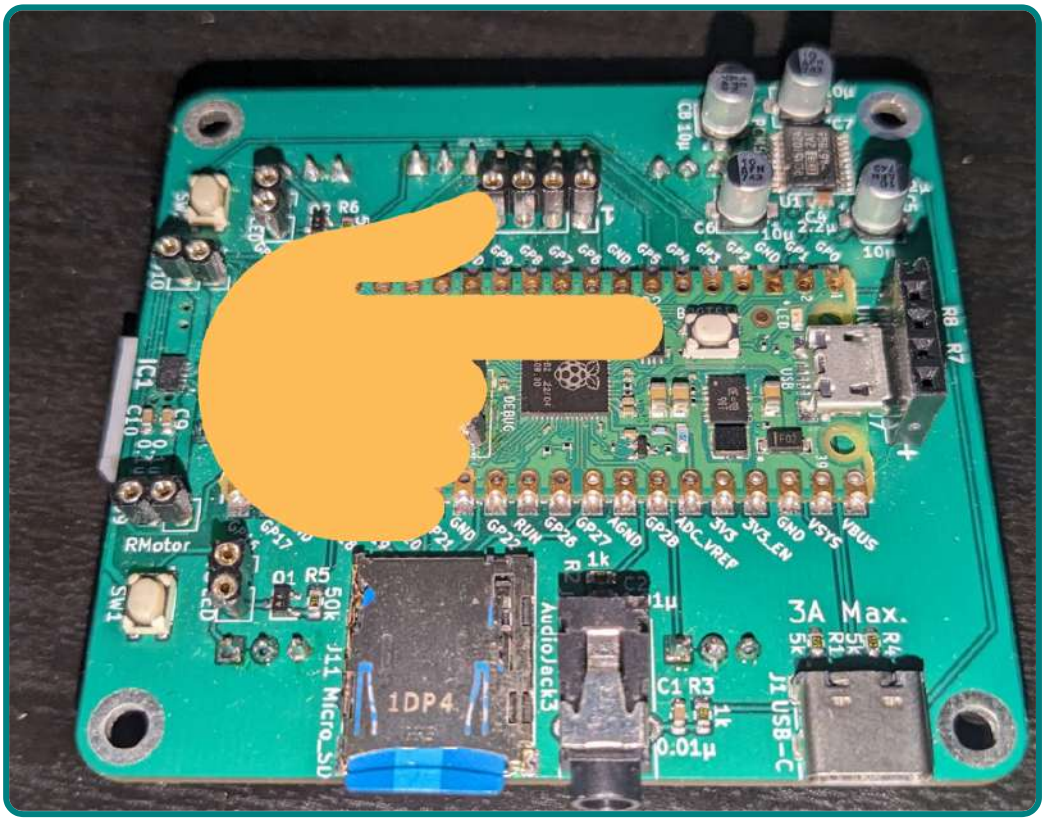
言語はいずれも**MicroPython**で、C言語での環境はここでは取り上げません。また、MicroPythonとはいいつつも、Pythonの環境をわざわざ構築する必要はなく、上記 3 つの方法で手軽に環境を構築することが可能です。

> ※厳密にはThonnyはPythonのIDEでPythonの環境が必要みたいですが、導入するときに自動で設定してくれているのか、なにか特別に導入する必要はありません。

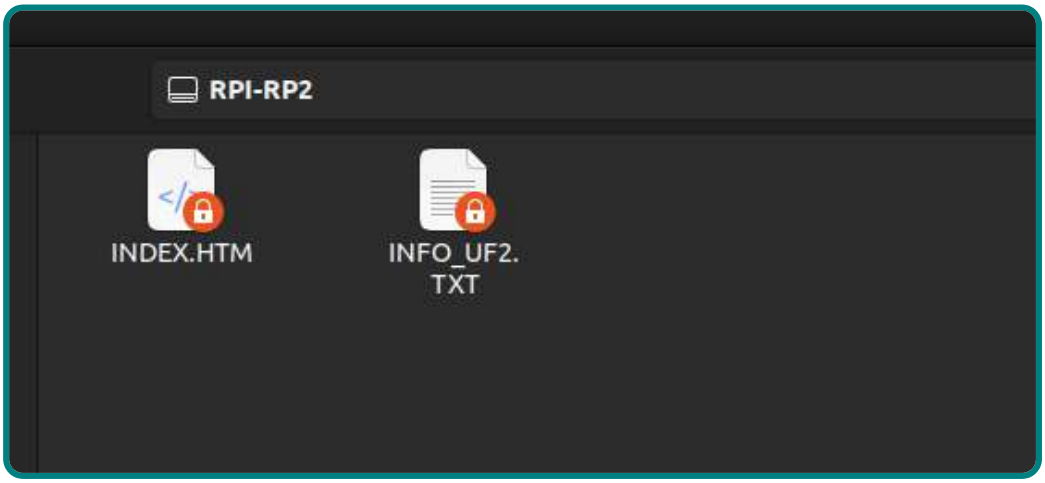
3.1. ファームウェアの書き込み (MicroPython)

ファームウェアとは、Raspberry Pi Picoにて**MicroPython**を実行するのに必要なソフトウェアのことで、拡張子は **.uf2** です。
ファームウェアに書き込むにはまず、Raspberry Pi Picoを**ストレージモード**でPCと接続する必要があります。

ストレージモードにするには、下の画像の位置にある**BOOTSEL**ボタンを押しながら、USB TypeCケーブルを使用してPCとTurtle Picoを接続するだけです。



ストレージモードで接続すると、その名の通り**ストレージ**としてアクセスできます。**RPI-RP2**という名前でマウントされるので、そのストレージを開きます。



ストレージを開くと、上の画像のようになっており、INFO_UF2.TXTファイルと、INDEX.HTMファイルの２つがあるのがわかります。このうち、**INDEX.HTM**ファイルを開き、Raspberrypi公式HPにアクセスしてください。

次に、一覧のメニューから**MicroPython**をクリックし、**Drag-and-Drop MicroPython**の中の**Download the correct MicroPython UF2 file for your board:** から**Raspberry Pi Pico W**を選択し、ファームウェア（.uf2ファイル）をダウンロードします。

あとはダウンロードしたuf2ファームウェアを、**ドラッグアンドドロップ**でRPI-RP2にコピーするだけで、ファームウェアが書き換わります。コピーが終わると、自動的に**シリアルデバイス**として認識するようになります。
※このとき自動的に認識しなければ、再度ケーブルを挿し直してください。

補足ですが、Linuxユーザーはシリアル通信するときには注意が必要です。というのも、Raspberry Pi Picoをシリアル通信で接続したときに、設定をしていないと**権限**の問題が発生してしまいます。

毎回`chnmod`などを使用して権限を設定しなおすのも面倒かと思いますので、

```
$ sudo adduser {username} dialout
```

のように`adduser`コマンドを使用し、**dialout**グループをユーザーに追加する方法を推奨します。

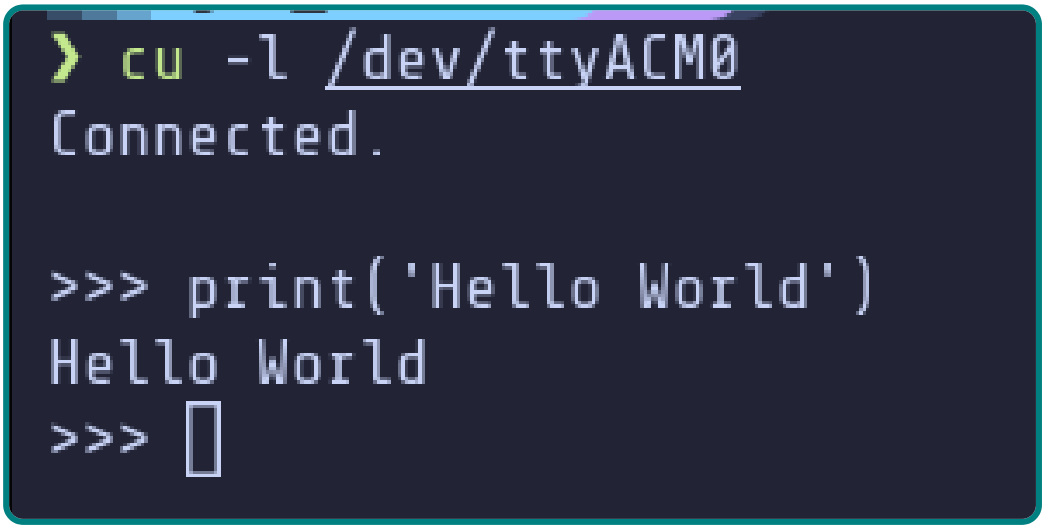
3.2. Terminal (REPL)

Windowsであれば**TeraTerm**、MacやLinux各種であれば、コマンドラインから`cu`や`minicom`コマンドを使用して3.1 ファームウェアの書き込み済みのTurtle Picoに**シリアル通信**で接続するだけで、対話形式でプログラムの実行が可能です。

ボーレートはどの設定でも基本繋がるようです（おそらく、USB-CDCデバイスの仕様）。
TeraTermの設定について詳しくはここで説明しませんが、シリアルポートでCOMデバイスを選択して接続すれば利用できます。

`cu`コマンドを使用する場合は、下図のように`-l`でデバイスを選択する必要があります。下記はLinuxでの例です。

```
$ cu -l /dev/ttyACM0
```



Pythonの構文等の説明につきましては、ここでは割愛させていただきます。

3.3. Thonny

Thonnyを使用する方法は簡単です。導入しないといけないソフトウェアは１つだけで、**Thonny**というPythonのIDEのみです。

- **Thonny**

OSは**Windows**、**Mac**、**Linux**に対応しています。

3.3.1. Windows / Mac

公式サイトよりインストーラーをダウンロードし、インストールしてください。

3.3.2. Linux

下記のように`wget`コマンドを使用してシェルスクリプトを読み込ませるだけで、簡単にインストールできます。

```
$ wget -O - https://thonny.org/installer-for-linux
```

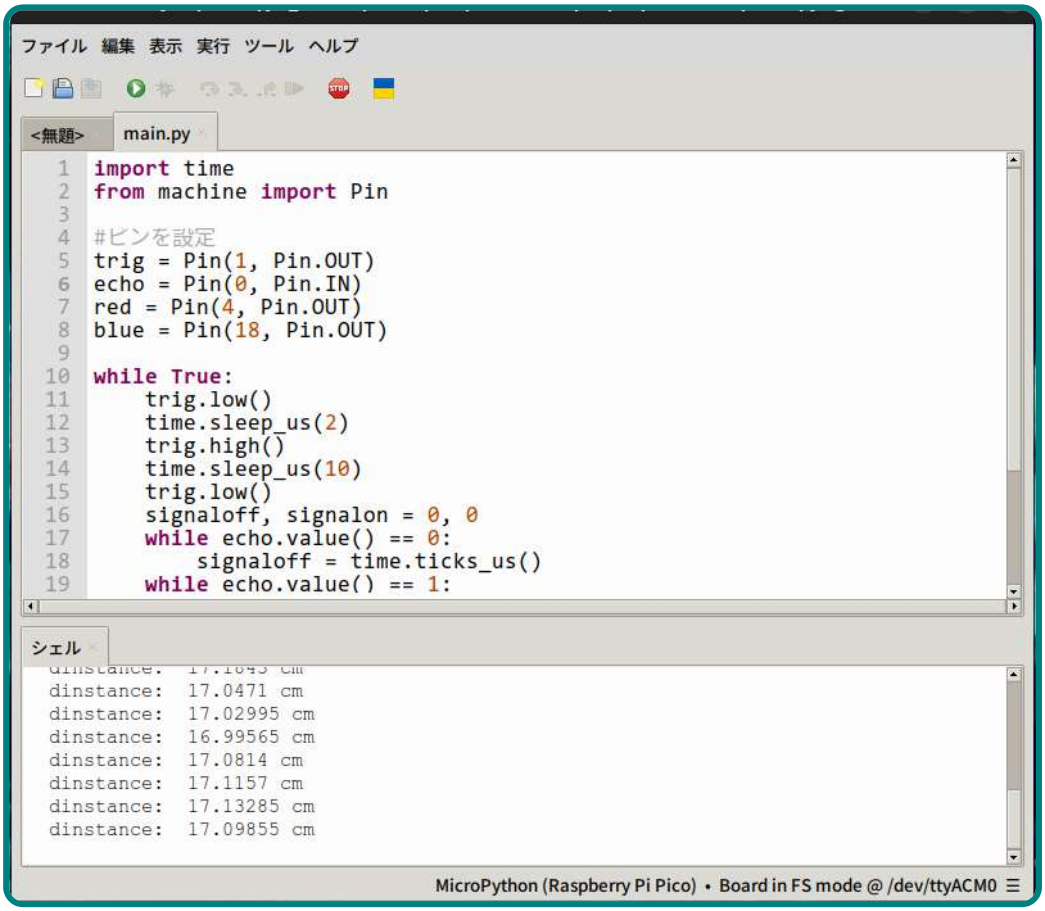
起動する時は、コマンドラインに`thonny`と入力するだけでアプリが使用できるようになります。

3.3.3. 全OS共通

`pip`コマンドで、各OS上のターミナルで下記のようにコマンドを入力し、インストールすることも可能です。

この方法では、`pip`のインストールが事前に必要ですが、それについてはここでは割愛させていただきます。

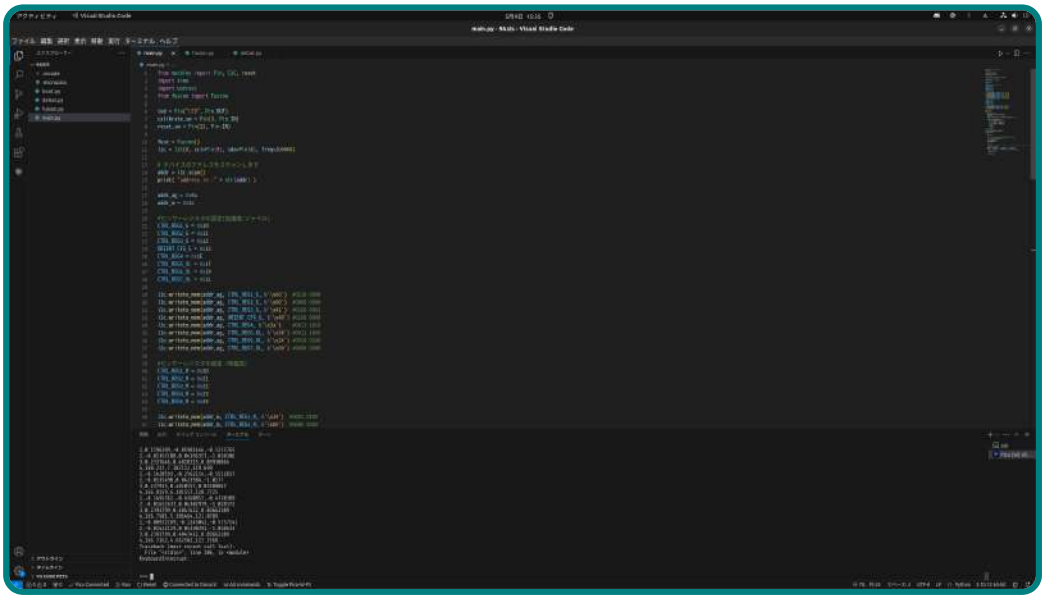
```
$ pip install thonny
```



画像はlinuxで実行したものになります。

Thonnyでの詳しい操作方法や導入方法等は、お手数ですが**公式Wiki**をご覧ください。

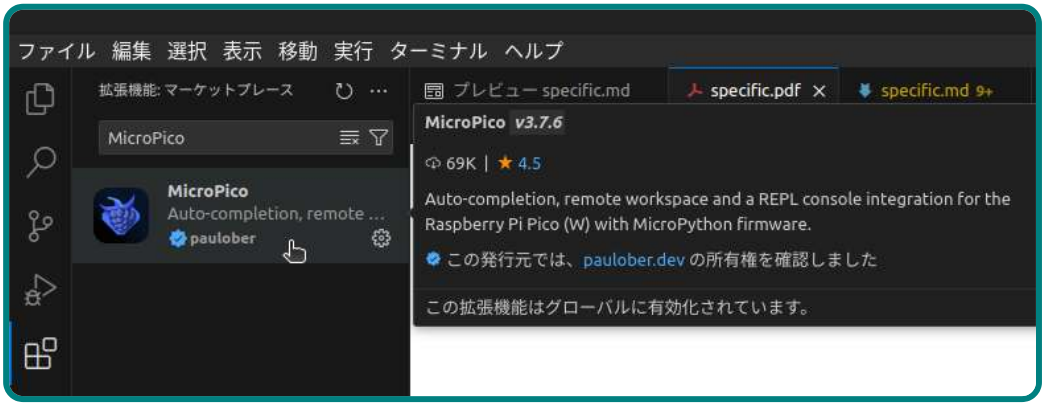
3.4. VSCode (Micro Pico)



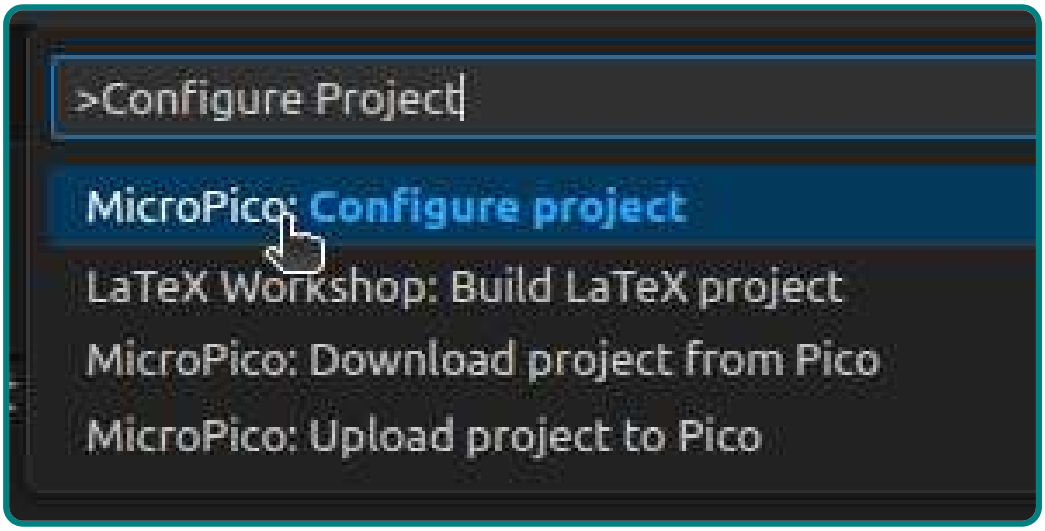
VSCodeというよく知られたテキストエディタを使用する方法もあります。導入しないといけないものが多いですが、こちらは非常に操作がしやすく、デザインもよくて分かりやすく、更にエディタ機能が充実しており、話題のAIなども**拡張機能**で導入できるのでおすすめです。

- **VSCode本体**
- VSCode拡張機能 - **MicroPico**
- VSCode拡張機能 - **Python**
- VSCode拡張機能 - **IntelliCode**
- VSCode拡張機能 - **Pylance**

VSCode本体については、**公式HP**より各OSのインストーラをダウンロードして、インストールするだけです。**VSCode拡張機能**については、VSCode内の拡張機能のメニューから、それぞれの名称で検索を掛けて導入するだけです。



MicroPicoを起動するには、フォルダを開いた状態で、**コマンドパレット** (ショートカット: **Ctrl + Shift + P**で開く) を開き、**MicroPico: Configure Project**を選択する必要があります。これによって設定されたMicroPicoフォルダをVSCodeで開くと、以降毎回自動でMicroPicoを起動してくれます。



MicroPicoのそれ以外の操作につきましては、ここでは割愛させていただきます。
お手数ですが、詳細は**公式リポジトリ**をご覧ください。

4. プログラム例

ここからは、各部品を制御する**テストプログラム**についての紹介と、各制御方法などをご紹介させていただきます。

ここで紹介するソースコードは、**Turtle Pico公式リポジトリ**に掲載しており、 **MITライセンス**で提供しています。ご自由にダウンロード/改変してお使いください。
※ここで紹介するものは、あくまで**テスト用**のものなので、そこまで高度な制御を提供するものではありませんので、その点だけ予めご了承ください。

4.1. ディレクトリ構成

Turtle Pico公式リポジトリに格納しているプログラム例の**ディレクトリ構成**は下記のようになっております。

Turtle-Pico/example/
├**9Axis/**
│　└**lib/**
├**BLDCMotor/**
│　└**lib/**
├**DCMotor/**
│　└**lib/**
├**I2S_Sound/**
│　└lib/
├**Ironman/**
│　└**lib/**
├**MusicPlayer/**
│　└**lib/**
├**SDCard/**
│　└**lib/**
├**Sensors_Web/**
│　└**lib/**
├**Servo/**
│　└**lib/**
├**SmartHomeo/**
│　└**lib/**
├**Temperature/**
│　└**lib/**
├**Ultrasonic/**
│　└**lib/**
├**flexibleLED/**
│　└**lib/**
├**flexibleLED_Web/**
│　└**lib/**
└**oled/**
　└**lib/**

各フォルダには、**lib**フォルダ、及び**main.py**と**boot.py**ファイル、**.micropicoファイル**の4つで成立しています。各ファイルの役割は、下記のとおりです。

- **lib**

  必要なライブラリのファイルを格納しています。使用する時は、

  ```
  from {fileName} import {className}
  ```

  として使用します。

- **.micropico**

  VSCode拡張機能のmicropicoを、該当のフォルダ内で有効にするファイルです。特に意味はありません。

- **main.py**

  MicroPythonのメインとなるファイルです。**boot.py**の次に実行され、このファイルの処理が終了するとPicoは**休止状態**となってしまいますので、繰り返し実行する必要がある場合は、`while`ループなどを使用して処理を続行させます。

- **boot.py**

  最初に実行されるファイルです。ボード上の使用する各パーツのセットアップなどを記述すると良いですが、**main.py**の冒頭に各フォルダごとにそれらを記述すればいいので、特に使用することはありません。

全ライブラリフォルダには、下記に示す**TurtlePico.py**というTurtlePicoボードを動かす上で必要な定義ファイルを配置しています。このクラスから、ピン情報を参照することが出来ます。

```
# Turtle Pico Pin Define File

class TurtlePico:

    LED_R:      int = 18
    LED_L:      int = 4

    SW_R:       int = 22
    SW_L:       int = 3

    MOTOR_EN:   int = 14
    MOTOR_R:    int = 17
    MOTOR_L:    int = 15

    SPI_ID:     int = 1
    SPI_SCK:    int = 10
    SPI_MOSI:   int = 11
    SPI_MISO:   int = 12

    OLED_CS:    int = 7
    OLED_DC:    int = 6
    OLED_RST:   int = 5

    SD_CS:      int = 27

    I2C_ID:     int = 0
    I2C_SCL:    int = 9
    I2C_SDA:    int = 8

    I2S_ID:     int = 0
    I2S_BCLK:   int = 19
    I2S_LRCLK:  int = 20
    I2S_SDATA:  int = 21

    ESC_SERVO_FR:int = 28
    ESC_SERVO_RR:int = 16
    ESC_SERVO_FL:int = 2
    ESC_SERVO_RL:int = 13

    TRIG_TX:    int = 1
    ECHO_RX:    int = 0
```

ピン情報の詳細は、**2.4. 回路図/ピンアサイン**を参照してください。
バージョンによってピンアサインが異なるので、ダウンロードするバージョンを間違えないように注意してください。

4.2. LED点灯

LEDを点灯させるには、マイコンの出力では電流が足りません。そこでTurtle Picoでは、マイコンからの出力をトランジスターのベース部分につなぎ、出力のピンソケットをVCCとコレクターの間に接続することで、**オープンコレクター**接続としています。

これにより、最大で15mAほどまでの電流が出力可能になります。この電流の値は、接続するLEDのVfがVccを超えない限り一定となります。

それではプログラムを解説していきます。まず、ライブラリの読み込み部分です。

```
import machine
from machine import PWM
import time
from lib.TurtlePico import TurtlePico
```

`import`はライブラリを読み込むための命令で、**machine**ライブラリは主にRaspberry Pi Picoの**ハードウェア**に関連するもの（タイマーやGPIOなど）で、2行目ではその中でも`PWM`クラスを読み込んでいます。

3行目の`time`ライブラリは時間管理をするためのもので、中でも遅延用の`sleep`関数などはよく使用します。

最後の行は、4.1 ディレクトリ構成で説明した`TurtlePico`ライブラリを読み込んでいます。

ピンは以下のようにして設定します。

```
blue = PWM(machine.Pin(TurtlePico.LED_R, machine.Pin.OUT))
red = PWM(machine.Pin(TurtlePico.LED_L, machine.Pin.OUT))
```

PWMはGOIOピンからHighとLowの出力を繰り返す矩形波を出力し、そのうちHigh時間の割合（**デューティ比**）を変化させることにより、平均的な電圧を上げたり下げたりする機能で、マイコンの基本的な機能の一つです。ボード右側のLEDを`blue`、左側のLEDを`red`として定義しています。

High(3.3V)とLow(0V)のみしか出力することが出来ないRaspberry Pi Picoでも、PWMを使用すれば0～3.3V間で連続的に電圧を変化させることができるため、本プログラムではこれを使用して**明るさ**を調節します。

PWMの該当するピンはトランジスターのベースにつながっている**抵抗器**に接続されているため、この電圧を変更すると**オームの法則**に従ってベースに流れる電流も変化し、オープンコレクタ出力の電流も同様に変化します。

```
blue.freq(1000)
red.freq(1000)
```

ここでは、各PWM出力の波形の周波数を調節しています。PWMの電圧を変化させるのに必要なのはデューティ比だけですが、周波数がおそすぎるとこの変化が目に見えてしまい、ただ点滅しているだけにしか見えないので、`freq`関数の引数に周波数を与え、周波数を目に見えない速度に設定する必要があります。

```
while True:
    if i > 65536:
        status = 1
    elif i < 0:
        status = 0
    red.duty_u16(i)
    blue.duty_u16(65536-i)
    print(i)
    if status == 0:
        i += 30
    else:
        i -= 30

    time.sleep(0.001)
```

aaa デューティ比は`duty_u16`関数によって、0~65535の間で設定する必要があり、これが0~100%に対応しています。このプログラムでは、`i`という変数を0~65536までの間で30ずつ増減させます。最初に0から65536まで向かう時は`status`という変数を0にして`i`が**増加中**だということを示し、65536から0に向かう時は`status`を1にして`i`が**減少中**ということを示します。

`status`がどのような値であるかで、`i`に30を加えるか/減じるかを`if`分岐によって処理しています。またそうして変化するデューティ比を、`red`ではそのまま、`blue`では65536から引くことによって出力し、`bule`と`red`で明るさを**反転**させています。

最後に、`print`でデューティ比をデバッグように出力し、また各ループごとに`sleep`関数で1ミリ秒の遅延を挟んでいます。

4.3. I2S音楽再生

音楽というものはアナログ値ですが、mp3やwavのような音楽データは全てディジタルにて表現されます。これらを再生するには、I2Sというオーディオ通信用のシリアル通信を用い、オーディオDACにてアナログのデータにする必要があります。

Turtle Picoに搭載されているオーディオDACはPCM5102Aで、アンプは搭載されていませんので、イヤホンジャックでの視聴がメインです。今回はWavPlayerというライブラリを使用してwavファイルを再生することとします。

それではプログラムを解説していきます。まず、ライブラリの読み込み部分です。

```
import os
from machine import Pin
from machine import SPI
from lib.wavplayer import WavPlayer
from lib.sdcard import SDCard
from lib.TurtlePico import TurtlePico
```

SDカード内の音楽データを読み取るための宣言（3,5行目）と先述の`wavPlayer`を読み取るための宣言（6行目）となっています。

```
cs = Pin(TurtlePico.SD_CS)

spi = SPI( 1,
           baudrate = 25_000_000,
           polarity=0,
           phase=0,
           bits=8,
           firstbit=SPI.MSB,
           sck  = Pin(TurtlePico.SPI_SCK),
           mosi = Pin(TurtlePico.SPI_MOSI),
           miso = Pin(TurtlePico.SPI_MISO))

sd = SDCard(spi, cs)
sd.init_spi(25_000_000)  # increase SPI bus speed to SD card
os.mount(sd, "/sd")
list = os.listdir("/sd")
print(list)
```

SDカードの設定です。SDカードの詳細は、4.5. SDカードにて詳述することとし、ここでは割愛させていただきます。

```
WAV_FILE = "12-Inner-Universe.wav"

wp = WavPlayer(
    id=TurtlePico.I2S_ID,
    sck_pin=TurtlePico.I2S_BCLK,
    ws_pin=TurtlePico.I2S_LRCLK,
    sd_pin=TurtlePico.I2S_SDATA,
    ibuf=40000,
    volume=-2
)
```

まず最初に、読み込みたいファイル名を指定しています。

wp変数は、WavPlayerクラスのインスタンスで、コンストラクタに引き渡す変数は主にi2sに関するものです。i2sに関するピンやIDの情報は、全てTurtlePicoライブラリに格納していますので、それを使用してインスタンスを生成しています。ibufは内部のバッファ長です。

また、volume変数が追加されていますが、こちらは既存のwavplayerライブラリから新規に追加した要素で、音量を設定しています。

```
print("==========  START PLAYBACK ==========")
try:
    wp.play(WAV_FILE)
except (KeyboardInterrupt, Exception) as e:
    print("caught exception {} {}".format(type(e).__name__, e))

while wp.isplaying():
    pass
```

再生部分はこのようになっています。wp.playにてファイル名を指定すれば簡単に再生が可能です。再生中の処理はwp.isplayingをwhileで指定してあげればループさせることができます。

この他にも、wp.pauseやwp.resumeなど、WavPlayerライブラリには基本的な再生機能が備わっています。また、基本的に今のところはwavのみの対応ですが、mp3を流せるように今後検討していきたいと考えています。

4.4. 透明有機ELディスプレイ

透明有機ELディスプレイが同梱ですが、基本的にssd1309またはssd1306の互換のものであれば同様に制御が可能です。 なお、ここではwavshareのssd1309搭載透明有機ELディスプレイを前提で解説させていただきます。

```
from machine import Pin, SPI, I2C
import framebuf, time
from lib.ssd1306 import SSD1306_SPI, SSD1306_I2C
from lib.img_lib import img_lib
from lib.TurtlePico import TurtlePico
```

基本的には他のものと同じものを読み込んでいますが、SPIまたはI2Cにてシリアル通信でSSD1309と通信するため、`machine`モジュール内のSPIやI2Cを読み込んでセッティングします。

ディスプレイ関連で使用するライブラリは、`SSD1306_SPI`（SPIの場合）、`SSD1306_I2C`（I2Cの場合）です。また、4行目で読み込んでいる`img_lib`にはいくつか画像のバイナリデータを格納しています。 2行目の`framebuf`ライブラリは、先述の`img_lib`に格納したバイナリデータをディスプレイに表示させるためのものです。

```
SW_Up = Pin(TurtlePico.SW_L, Pin.IN, Pin.PULL_DOWN)
SW_Down = Pin(TurtlePico.SW_R, Pin.IN, Pin.PULL_DOWN)
#中略
item_selected = 0
item_sel_previous = 0
item_sel_next = 0

button_up_clicked = False
button_down_clicked = True

num_items = len(img_lib.menu_item_fb)
```

メイン部で`img_lib`に格納したアイコンをメニュー表示するために、使用する各種変数です。`item_sel~`は現在、過去、未来に選択されるメニューの番号、`button_~_clicked`はメニューをアップさせるか、ダウンさせるかを制御する変数です。`num_items`はアイテムの個数を格納します。

```
spi = SPI( TurtlePico.SPI_ID, baudrate = 100000, sck =
Pin(TurtlePico.SPI_SCK), mosi = Pin(TurtlePico.SPI_MOSI))

oled_cs = Pin(TurtlePico.OLED_CS,Pin.OUT)
dc = Pin(TurtlePico.OLED_DC,Pin.OUT)
rst = Pin(TurtlePico.OLED_RST,Pin.OUT)
display = SSD1306_SPI(128, 64, spi, dc, rst, oled_cs)
```

SPI通信をする場合は、1行目のように宣言します。基本的にSPIポートはTurtlePicoライブラリ内にピンやID情報を格納しているので、それを使用して宣言する形です。ssd1309はSPIの他に、csやdc、rstのピンを参照する必要がありますが、これらも同様にTurtlePicoライブラリから参照します。

display変数は、これらの情報を使用して、SSD1306_~をインスタンス化したものです。

```
while(True):
    if(SW_Up.value() == 1 and not button_up_clicked):
        item_selected = item_selected - 1
        button_up_clicked = True
        if(item_selected < 0):
            item_selected = num_items - 1
    if(SW_Down.value() == 1 and not button_down_clicked):
        item_selected = item_selected + 1
        button_down_clicked = True
        if(item_selected >= num_items):
            item_selected = 0

    if(SW_Up.value() == 0 and button_up_clicked):
        button_up_clicked = False
    if(SW_Down.value() == 0 and button_down_clicked):
        button_down_clicked = False

    item_sel_previous = item_selected - 1
    if(item_sel_previous < 0):
        item_sel_previous = num_items - 1
    item_sel_next = item_selected + 1
    if(item_sel_next >= num_items):
        item_sel_next = 0

    #backgroundを表示
    display.fill(0)
    display.blit(img_lib.item_sel_background 0, 22)
    display.blit(img_lib.scrollbar_background, 125, 0)

    #previous item
    display.blit(img_lib.menu_item_fb[item_sel_previous], 5, 2)
    display.text(img_lib.menu_item_char[item_sel_previous], 29, 7, 1)

    #selected item
    display.blit(img_lib.menu_item_fb[item_selected], 5, 24)
    display.text(img_lib.menu_item_char[item_selected], 29, 27, 1)

    #next item
    display.blit(img_lib.menu_item_fb[item_sel_next], 5, 46)
    display.text(img_lib.menu_item_char[item_sel_next], 29, 50, 1)

    display.show()
```

while文の内部です。最初のif文郡は、スイッチによる切り替え処理です。先述の変数を使用して分岐にて制御しています。`display.fill`は画面全体を塗りつぶせるので、色に0を指定して画面をリセットします。 `display.blit`で`img_lib`内に用意した`framebuf`を表示させることができます。変数には座標の情報が必要になります。

`display.text`は単純にテキストを表示させるものです。同様に、座標と色の情報が必要です。これらをセッティングし、`display.show`を実行することで実際に表示させることができます。

成功すると、oledには以下のようなメニューが表示されます。

4.5. SDカード

SDカードには大きなデータを格納するストレージとして機能させるのがちょうど良いでしょう。画像のバイナリデータ等も多くなってくるとこちらに移行するのが良差そうです。

```
from machine import Pin, SPI
import machine
import os
from lib.sdcard import SDCard
from lib.TurtlePico import TurtlePico
```

SPIを使用するので、1行目では`machine`モジュールから読み込んでいます。ディレクトリなどの制御をするための、`os`ライブラリを3行目で読み込んでいます。

また、SDカードのセッティングをするための`SDCard`ライブラリを4行目で読み込んでいます。SDカードはSPIで接続が可能ですが、電源投入直後ではMMCというモードで起動します。そのため、ライブラリ内のSPIとして接続するための基本機能が必要です。

```
cs = Pin(TurtlePico.SD_CS)

spi = SPI( TurtlePico.SPI_ID,
           baudrate = 100000,
           sck  = machine.Pin(TurtlePico.SPI_SCK),
           mosi = machine.Pin(TurtlePico.SPI_MOSI),
           miso = machine.Pin(TurtlePico.SPI_MISO))

sd = SDCard(spi, cs)
```

最初にcs（チップセレクト）を`TurtlePico`ライブラリ内の該当するピンで設定し、SPIインスタンスを`spi`変数として宣言します。その後、これを`SDCard`クラスに渡し、`sd`としてインスタンス化しています。

```
os.mount(sd, '/sd')
os.chdir('sd')

list = os.listdir("/sd")
print(list)
```

この`sd`変数を使用し、`os`ライブラリに渡すことで、1行目のようにsdカードをマウントできます。2行目の`os.chdir`でカレントディレクトリをSDカード内に移し、その後`os.listdir`でディレクトリ内のファイル名を取得し、`print`で表示しています。

4.6. サーボモーター

サーボモーターはLEDのときと同じように、PWM似て制御します。マイクロサーボのSG-92では、デューティ比によって角度が変わり、2.5%で-90°、12%で90°となり、この間で角度を自由に設定可能です。

まず、ライブラリを読み込みますが、これは4.2. LED点灯と全く同じです。

```
from machine import PWM, Pin
import time
from lib.TurtlePico import TurtlePico
```

次に、先述のデューティ比と角度の関係を変数（変更しないため定数として使用）で設定します。

```
servo1 = PWM(Pin(TurtlePico.ESC_SERVO_FR))
servo1.freq(50)

max_duty = 65535
dig_min = 0.025   #-90°
dig_max = 0.12     #90°

i = 0
status = 0
```

サーボモーターに送る周波数も50Hzと決められているので、2行目のようになっています。`max_duty`は16bitで設定するので、65535がmaxです。`i`や`status`は-90°と90°を往復するために必要な変数です。

```
while True:
    if i > 180:
        status = 1
    elif i < 0:
        status = 0
    deg = dig_min + i * (dig_max - dig_min) / 180
    print(i)
    if status > 0:
        i -= 1
    else:
        i += 1
    servo1.duty_u16(int(deg * max_duty))
    time.sleep_ms(10)
```

`while`文の中身です。先述の`i`や`status`を使用して、`deg`の角度を決定し、デバッグ用に`print`しています。これらの値をサーボモーターに適用するには、`dutyu16`で設定する必要があります。

4.7. ブラシ付きDCモーター

ブラシ付きDCモーターはモータードライバーDRV8835を制御するだけなので特段難しい制御は行いません。

```
from machine import Pin
import time
from lib.TurtlePico import TurtlePico

enable = Pin(TurtlePico.MOTOR_EN, Pin.OUT)
left = Pin(TurtlePico.MOTOR_L, Pin.OUT)
right = Pin(TurtlePico.MOTOR_R, Pin.OUT)
```

基本的な宣言です。各ピンは`TurtlePico`ライブラリより参照します。

```
while True:

    time.sleep(5)
    enable.value(1)
    left.value(1)
    right.value(1)
    print("forward")

    time.sleep(5)
    left.value(0)
    right.value(1)
    print("turn right")

    time.sleep(5)
    left.value(1)
    right.value(0)
    print("forward")

    time.sleep(5)
    left.value(0)
    right.value(0)
    print("backward")

    time.sleep(5)
    enable.value(0)
    print("stop")
```

`while`文の中身です。`left`ピンに1を出力すれば、左のモーターが正回転、逆に0を出力すれば逆回転します。これは`right`でも同様です。また、`enable`ピンの出力を切り替えることで、発進/停止が制御できます。

4.8. ブラシレスDCモーター

ブラシレスDCモーターはサーボモーターと同様のポートを使用して、制御しますので、基本的には同じ作用です。

```
from machine import Pin, PWM
import time
import lib.TurtlePico as TurtlePico

r_sw = Pin(TurtlePico.SW_R, Pin.IN)
l_sw = Pin(TurtlePico.SW_L, Pin.IN, Pin.PULL_DOWN)

#duty cycle
Max_duty = 0.1 * 65536
Min_duty = 0.05 * 65536
```

PWMに必要な基本的なライブラリを宣言し、スイッチを宣言しています。`Max_duty`や`Min_duty`は今回使用したESCのデフォルト値です。この間を変化させることで、回転速度を変えることができます。

```
#calibration
def calibration():
    duty = Max_duty
    status = 1
    print("Calibration start")
    for i in range(0, 3):
        for j in range(0, 10):
            if status == 1:
                duty -= 0.005 * 65536
            else:
                duty += 0.005 * 65536
            print(duty / 65536)
            BLDC.duty_u16(int(duty))
            time.sleep(0.1)
        status = status * -1
        time.sleep(3)
    print("Calibration done")
```

キャリブレーション用の関数です。擬似的にデューティ比をスロットルマックスの状態からゼロの状態に数回往復してキャリブレーションを取っています。

```
BLDC = PWM(Pin(TurtlePico.ESC_SERVO_FR))
BLDC.freq(50)

duty = Max_duty
BLDC.duty_u16(int(duty))

isCalibrated = False
```

基本的なPWMの宣言を行っています。サーボモーターのときと同様に、今回使用している同梱のESCでも50Hzで設定することが必要です。

キャリブレーション用にも、PWMの初期値には最大のデューティ比を出力しておきます。最後の行にあるisCalibrated変数はキャリブレーションを行っているかを格納し、キャリブレーションを必須にするようにしています。

```
while True:
    if r_sw.value() == 1:
        if isCalibrated == False:
            calibration()
            isCalibrated = True
            duty = Min_duty
        else:
            if duty < Max_duty:
                duty += 0.001 * 65536
            print(duty / 65536)
            BLDC.duty_u16(int(duty))
            time.sleep(0.5)
    elif l_sw.value() == 1:
        if isCalibrated == True:
            if duty > Min_duty:
                duty -= 0.001 * 65536
            print(duty / 65536)
            BLDC.duty_u16(int(duty))
            time.sleep(0.5)
```

while文の中身です。右側のスイッチを押すことで、キャリブレーションをできるようにしています。電源を投入すれば、キャリブレーションをするまでずっと待機状態となります。

キャリブレーションは、今回同梱しているESCのものだと、**BLDCモーターをつないだESCに電源投入後、しばらく待って「ドーレーミーファーソー」の音がなった後**に右のスイッチを押して行ってください。

上記のように待機しないまま行うと、上手くできず、急にモーターが回りだすので非常に危険です。

キャリブレーション後、ボード上の右のスイッチで回転速度を上昇、左で回転速度を下降できるようにしています。0.001ずつデューティ比を変化させ、duty_u16にて16bitに変換し、出力しています。

4.9. 超音波センサー

超音波センサーは、2つの円筒がついていますが、片方の円筒から超音波を発し、もう片方の円筒でその反射をキャッチするのですが、この時発してから帰ってくるまでの時間差によって、障害物までの距離を測定するというものになっています。

まずは、以下のように基本的なライブラリを読み込みます。

```
import time
from machine import Pin
from lib.TurtlePico import TurtlePico

#ピンを設定
trig = Pin(TurtlePico.TRIG_TX, Pin.OUT)
echo = Pin(TurtlePico.ECHO_RX, Pin.IN)
red = Pin(TurtlePico.LED_L, Pin.OUT)
blue = Pin(TurtlePico.LED_R, Pin.OUT)
```

trig、echoがそれぞれ、片方の円筒から超音波を出力する用のピンと、もう片方の円筒に返ってきたら1を出力するピンです。trigに1をいれて、echoに1が返ってくるまで時間をカウントする、といった流れで距離を計測します。

```
while True:
    trig.low()
    time.sleep_us(2)
    trig.high()
    time.sleep_us(10)
    trig.low()
    signaloff, signalon = 0, 0
    while echo.value() == 0:
        signaloff = time.ticks_us()
    while echo.value() == 1:
        signalon = time.ticks_us()
    timepassed = signalon - signaloff
    distance = (timepassed * 0.0343) / 2
    print("dinstance: ",distance,"cm")
    blue.value(1)
    time.sleep(0.1)
```

`while`文の中身ですが、`echo.value`が1になるまで`while`で待機し、`tick_us`の差分で経過時間である`timepassed`を設定しているのがわかります。距離は、これに音速をかけて、往復分となるためそこから2で割った値（12行目）として計算しています。

4.10. 温湿度センサー

I2Cにて接続された温湿度センサーの値を読み込む簡易的なプログラムです。まずは、ライブラリの宣言ですが、I2C接続に必要なライブラリ以外、特別なものは読み込んでいません。

```
from machine import Pin, I2C
import time
from lib.TurtlePico import TurtlePico

i2c = I2C(TurtlePico.I2C_ID, scl=Pin(TurtlePico.I2C_SCL),
sda=Pin(TurtlePico.I2C_SDA), freq=100000)

# デバイスのアドレスをスキャンします
addr = i2c.scan()
print( "address is :" + str(addr) )

addr_temp = 56
```

8行目のアドレススキャン部分は主にデバッグ出力用のものです。もともと10進数で56と決まっている（11行目で宣言）ので、ここでは特に何もしていません、

```
while True:
    #測定コマンドを送信
    i2c.writeto(addr_temp, b'\xAC\x33\x00')

    #測定結果を読み込む
    data = i2c.readfrom(addr_temp, 6)
    status = data[0] >> 7
    while status == 1:

        #測定中は待機
        data = i2c.readfrom(addr_temp, 6)
        status = data[0] >> 7
        time.sleep_ms(10)

    #温湿度データを取得
    hum = data[1] << 12 | data[2] << 4 | (data[3] & 0xF0) >> 4
    temp = ((data[3] & 0x0F) << 16) | data[4] << 8 | data[5]

    hum = hum / 2**20 * 100
    temp = temp / 2**20 * 200 - 50

    print("hum: " + str(hum))
    print("temp: " + str(temp))
```

`while`文の中身です。センサに測定コマンドを送信後、受信したパケットを、データシートのパケット情報に基づき、シフト演算にて情報を取り出しています。また、センサーの値は圧縮されているので、19，20行目のように計算して適正な値に戻しています。

4.11. 9軸センサー

9軸センサーはmicropythonのちょうど良いライブラリが見つからなかったため、コードに設定などを直書きしています。ここで紹介するのは、メイン部分のみとさせていただきます。

```
from machine import Pin, I2C, reset
import time
import ustruct
from lib.fusion import Fusion
from lib.TurtlePico import TurtlePico
```

`reset`というのはPico自体をリセットできるライブラリで、リセットボタンを割り当てているためこれを読み込んでいます。

`ustruct`や`Fusion`は9DoFデータ受信用の関数で使用しており、ここでは詳しい解説は省きますが、`Fusion`というのはジャイロなどの誤差を修正し、正確な値を取得するためのものです。

```
i2c = I2C(TurtlePico.I2C_ID, scl=Pin(TurtlePico.I2C_SCL),
sda=Pin(TurtlePico.I2C_SDA), freq=100000)

# デバイスのアドレスをスキャンします
addr = i2c.scan()
print( "address is :" + str(addr) )

addr_ag = 0x6a
addr_m = 0x1c
```

例のごとくi2cを宣言しています。i2cのアドレスは、加速度・ジャイロと地磁気で別々になっているので、各アドレスを宣言しています。

```
while True:

    accel = get9DofData("accel")
    gyro = get9DofData("gyro")
    mag = get9DofData("mag")

    fuse.update(accel, gyro, mag)
    print("1," + str(gyro[0]) + "," + str(gyro[1]) + "," +
str(gyro[2]))
    print("2," + str(accel[0]) + "," + str(accel[1]) + "," +
str(accel[2]))
    print("3," + str(mag[0]) + "," + str(mag[1]) + "," + str(mag[2]))
    print("4," + str(fuse.roll) + "," + str(fuse.pitch) + "," +
str(fuse.heading))

    if reset_sw.value() == 1:
        reset()

    time.sleep_ms(20)
```

`get9DofData`で各データを取得します。メインコード内に記述していますが、ライブラリとして取り扱い、ここでは詳述しませんが、この関数を使用してデータを取得できるようにしています。

その後、`Fusion`ライブラリで座標を修正し、`print`で各データをコンソールに出力しています。その下には、リセットボタンの記述があります。