Turtle Pico Specific

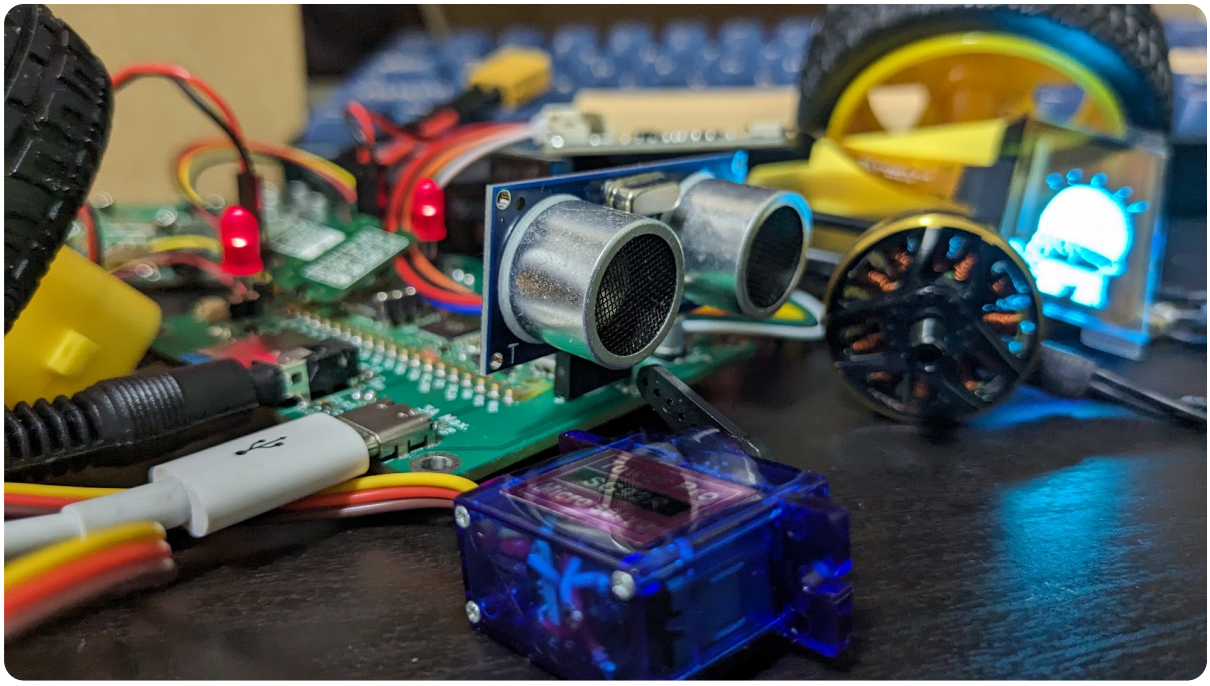


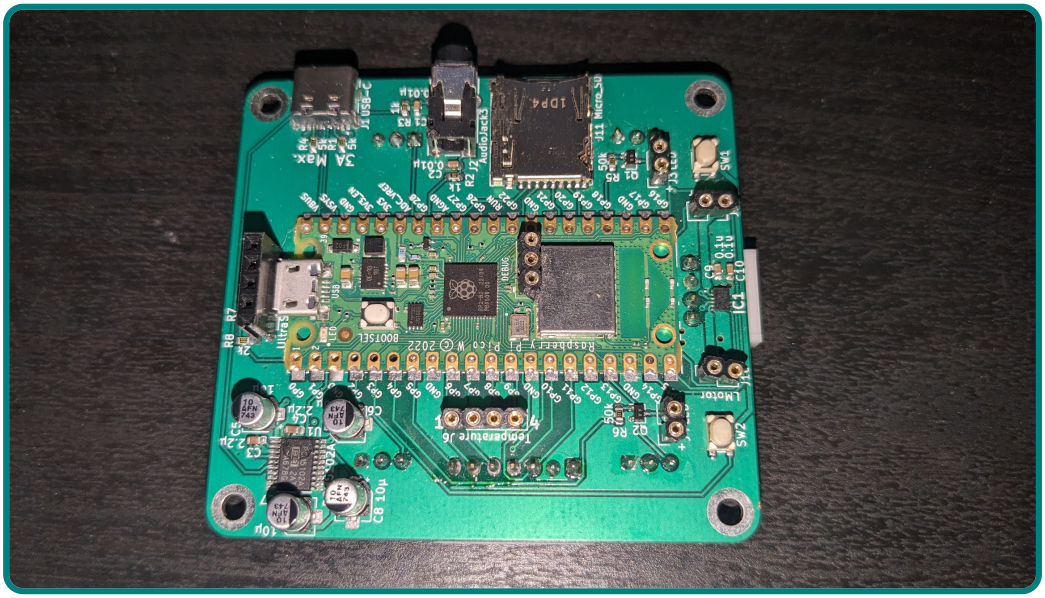
Table of Contents

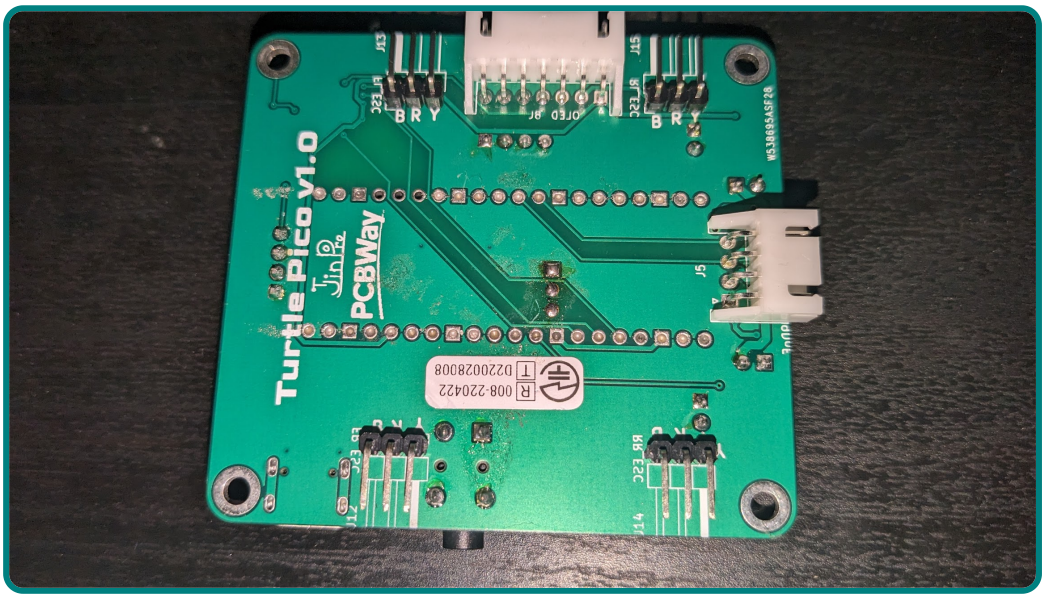
1. Turtle Pico Board Overview

This chapter provides an overview of the **Turtle Pico Board**.

1.1. appearance

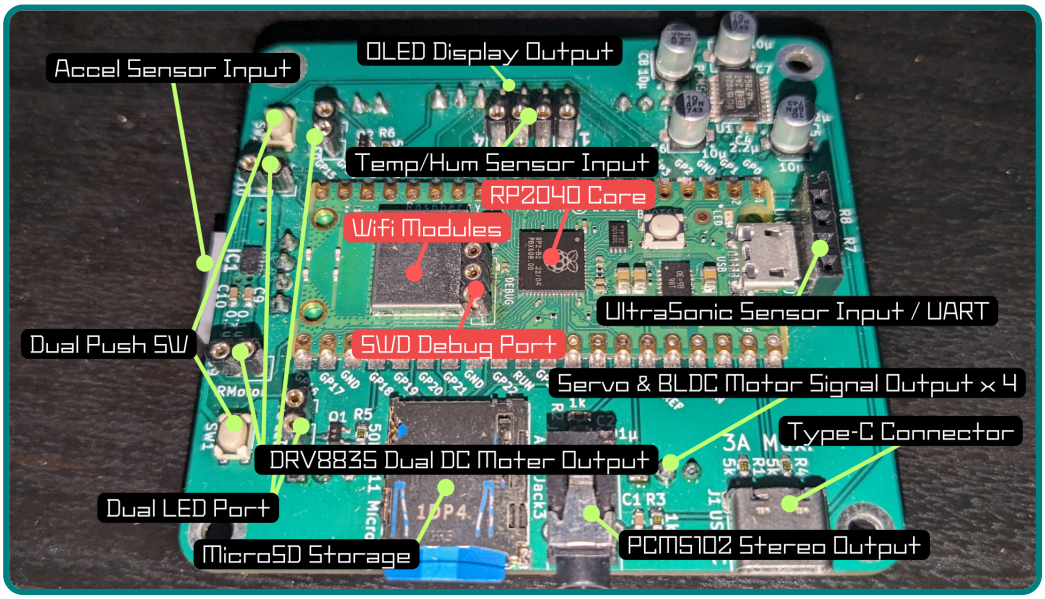
This is an exterior view of the Turtle Pico Board. The top image is the front and the bottom image is the back.





1.2. Interface/Function

An image summarizing the **interface** and **function** is shown below.



- **USB Type-C Connector**

  A **USB connector** to communicate with other devices. The current rating is **3A** due to the wiring width of the board. If more than this amount of current flows, the area near the connector on the board will heat up and the wiring may burn out.

- **MicroSD Card Slot (SPI)**

  Slot for connecting a Micro SD card. Connect via **SPI** communication.

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

  This is an **audio output port** with a 3.5mm stereo jack, directly connected to the PCM5102 output, which is connected to the PCM5102 via **I2S** communication. The sound is low with the output as it is, so if you need a loud output such as speakers, please connect an amplifier. If you want to listen through earphones, there is no need to do so.

- **OLED Display Port (SPI)**

  Port (back side) for connection to the transparent OLED control board(SSD1309). Connect via **SPI** communication.

- **Servo & BLDC Signal Output**

  Ports for sending control signals to **servo motors** and **brushless DC motors**.

- **DRV8835 Dual DCMotor Output**

  Two ports are provided for **DC motors** to be connected to the DRV8835 motor driver, with a maximum current rating of 1.5 A per port and 3 A total for the two ports.

- **UltraSonic Sensor Input / UART**

  Input/output port for **ultrasonic sensors**. When the ultrasonic sensor is not used, the center two pins can be used as a port for **UART** communication.

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

  This is the input port for the temperature/humidity sensor. Connect via **I2C** communication.

- **Accel Sensor Input (I2C)**

  Although it is labeled Accel Sensor, it is actually an input port (on the back) for a **9-axis sensor**. It is connected via **I2C** communication.

- **Dual Push SW**

  There are two **push switches** on the board that are directly connected to the Raspi Pico W.

- **Dual LED Port**

  An open collector output **LED port** connected to the collector of a transistor. The maximum current that can be output is about 15mA per port, and the brightness can be controlled by PWM.

- **SWD Debug Port**

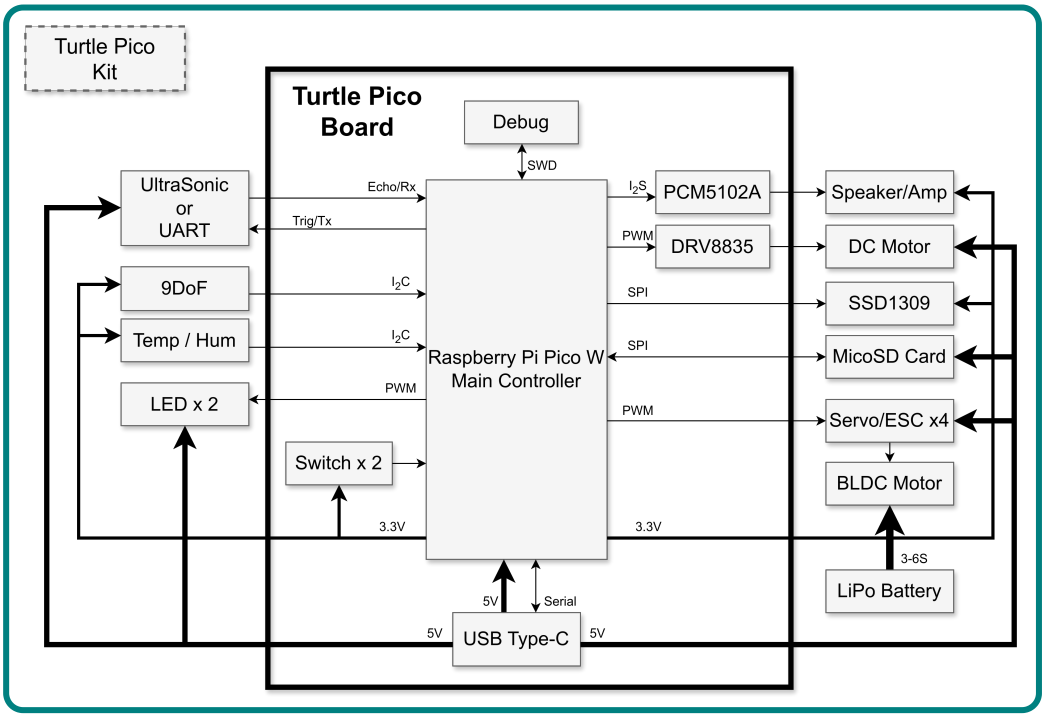
  This port is used for **writing/debugging** when using Pico SDK, RTOS, etc. **Pico Probe**, etc. for debugging with **OpenOCD**, etc.

- **WiFi Module**

  This is a **WiFi module** built into the Raspberry Pi Pico W. It supports only 2.5GHz band. Use this module to connect to the web.

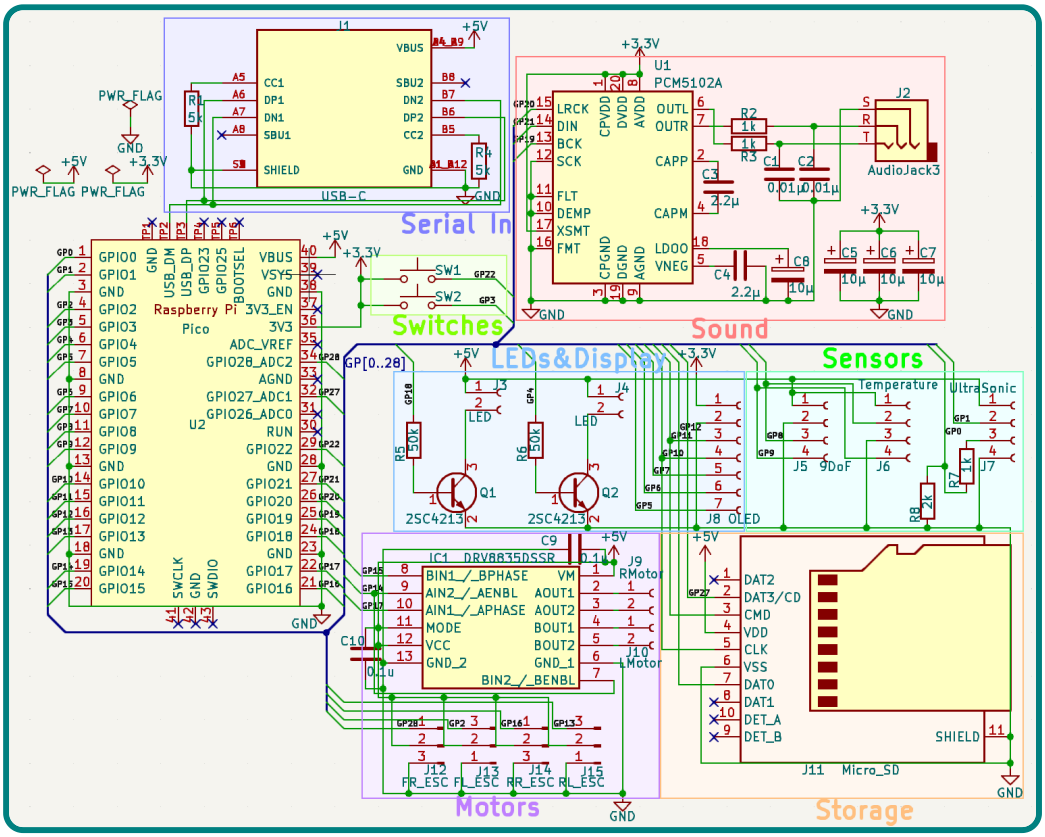
1.3. Block Diagram

A **block diagram** of the Turtle Pico Board is shown below.



1.4. Schematic/Pin Assignments

The **schematic** of the Turtle Pico Board and the corresponding **pin assignments** of the Raspberry Pi Pico W are shown below.



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**

2. Dnvironment Building

There are various ways to **build an environment** for the Turtle Pico (RaspberryPi Pico). Here, we will briefly introduce the following three typical methods.

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

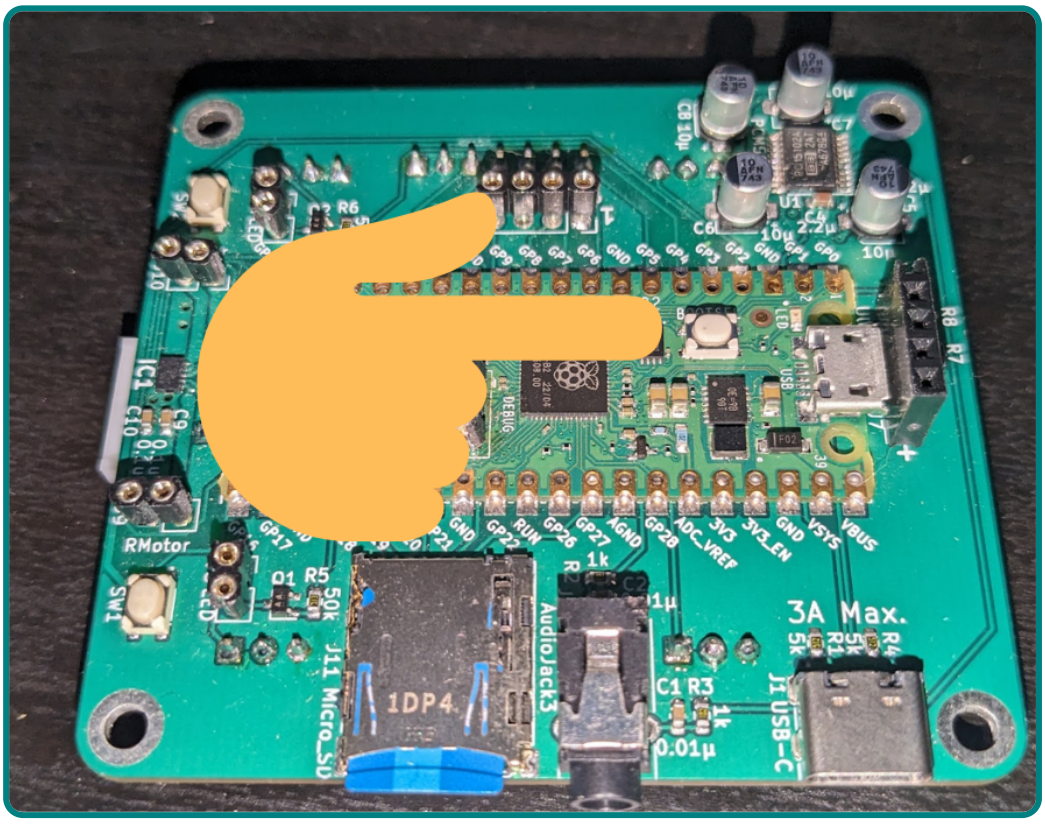
The languages are all **MicroPython**, and environments in C are not covered here. Although MicroPython is called “MicroPython,” it is not necessary to go to the trouble of building a Python environment, and the above three methods can be used to easily build an environment.

```
Strictly speaking, Thonny is a Python IDE and requires a Python
environment, but it seems to be set up automatically when it is
installed, so there is no need to install anything special.
```

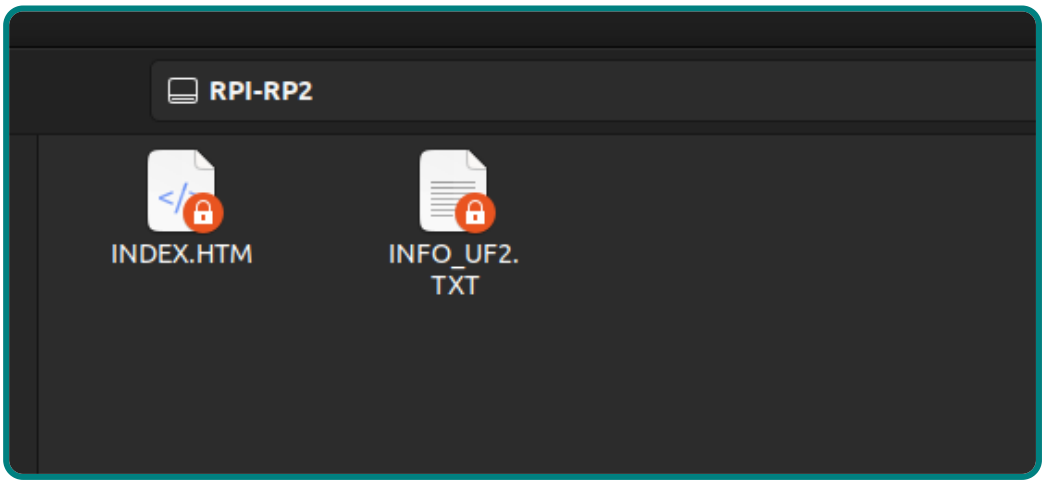
2.1. Writing Firmware (MicroPython)

Firmware is the software required to run **MicroPython** on the Raspberry Pi Pico and has a **.uf2** extension. To write to the firmware, you must first connect the Raspberry Pi Pico to your PC in **storage mode**.

To enter storage mode, simply connect the Turtle Pico to your PC with a USB TypeC cable while pressing the **BOOTSEL** button located in the image position below.

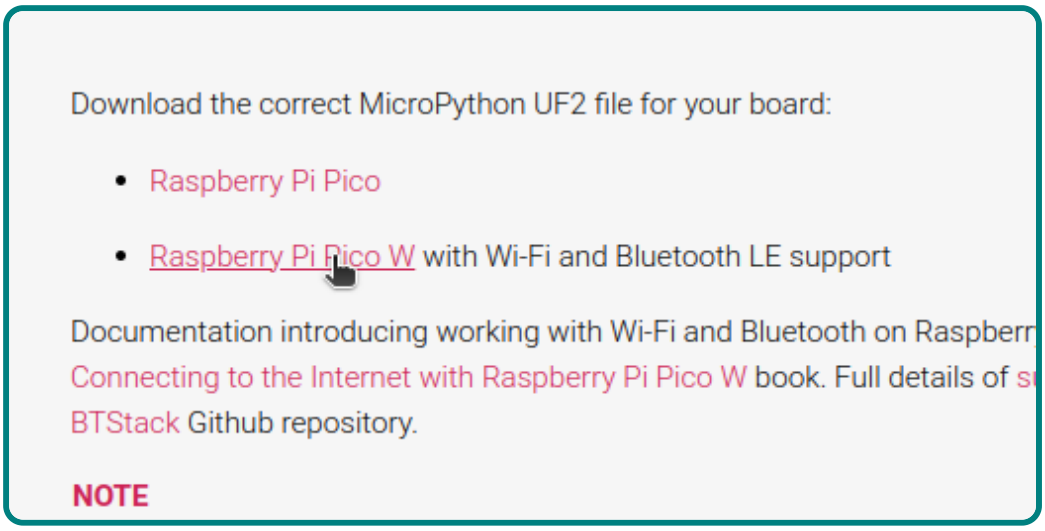


When you connect in storage mode, you can access it as **storage**, as the name implies. It will be mounted as **RPI-RP2**, so open that storage.



When you open the storage, you will see the image above, with the INFO_UF2.TXT file and the INDEX.HTM file. Open the **INDEX.HTM** file and access the Official HP of Raspberrypi.

Next, click on **MicroPython** from the list menu, and select **Raspberry Pi Pico W** from **Download the correct MicroPython UF2 file for your board:** in **Drag-and-Drop MicroPython**. This will download the firmware (.uf2 file).



Then simply copy the downloaded uf2 firmware to the RPI-RP2 by **drag and drop** and the firmware will be rewritten. Once the copy is complete, it will automatically be recognized as a **serial device**. If the cable is not recognized automatically at this time, reinsert the cable again.

As a side note, Linux users should be careful when communicating serially. This is because when the Raspberry Pi Pico is connected via serial communication, there will be **permission** issues if it is not configured.

I think it would be troublesome to re-set the permissions using `chnmod` and so on every time,

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

The recommended method is to use the `adduser` command to add the **dialout** group to the user.

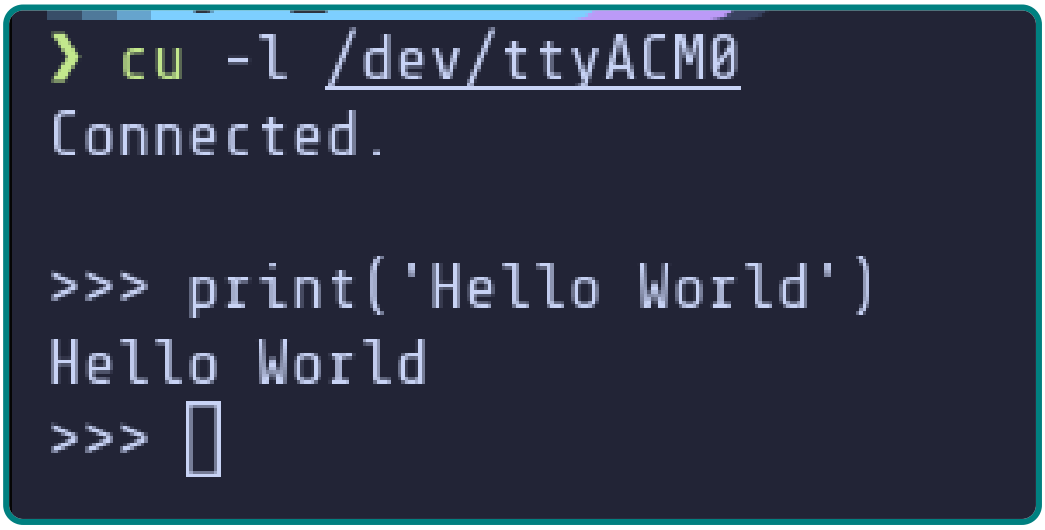
2.2. Terminal (REPL)

You can use **TeraTerm** on Windows, or `cu` or `minicom` command from the command line on Mac or various Linux platforms, to run the program interactively on the Turtle Simply connect to the Pico via **serial communication** to run the program interactively.

The basic connection seems to be established at any baud rate setting (probably a specification of the USB-CDC device).
We will not go into the details of setting up TeraTerm here, but you can use it by selecting and connecting a COM device on the serial port.

When using the `cu` command, you must select the device with `-l` as shown in the figure below. Below is an example on Linux.

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



We will not go into the syntax and other details of Python here.

2.3. Thonny

Using **Thonny** is simple. There is only one software that needs to be installed, a Python IDE called **Thonny**.

- **Thonny**

Operating systems supported are **Windows**, **Mac**, and **Linux**.

2.3.1. Windows / Mac

Download and install the installer from **Official Site**.

2.3.2. Linux

Simply load the shell script using the `wget` command as shown below.

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

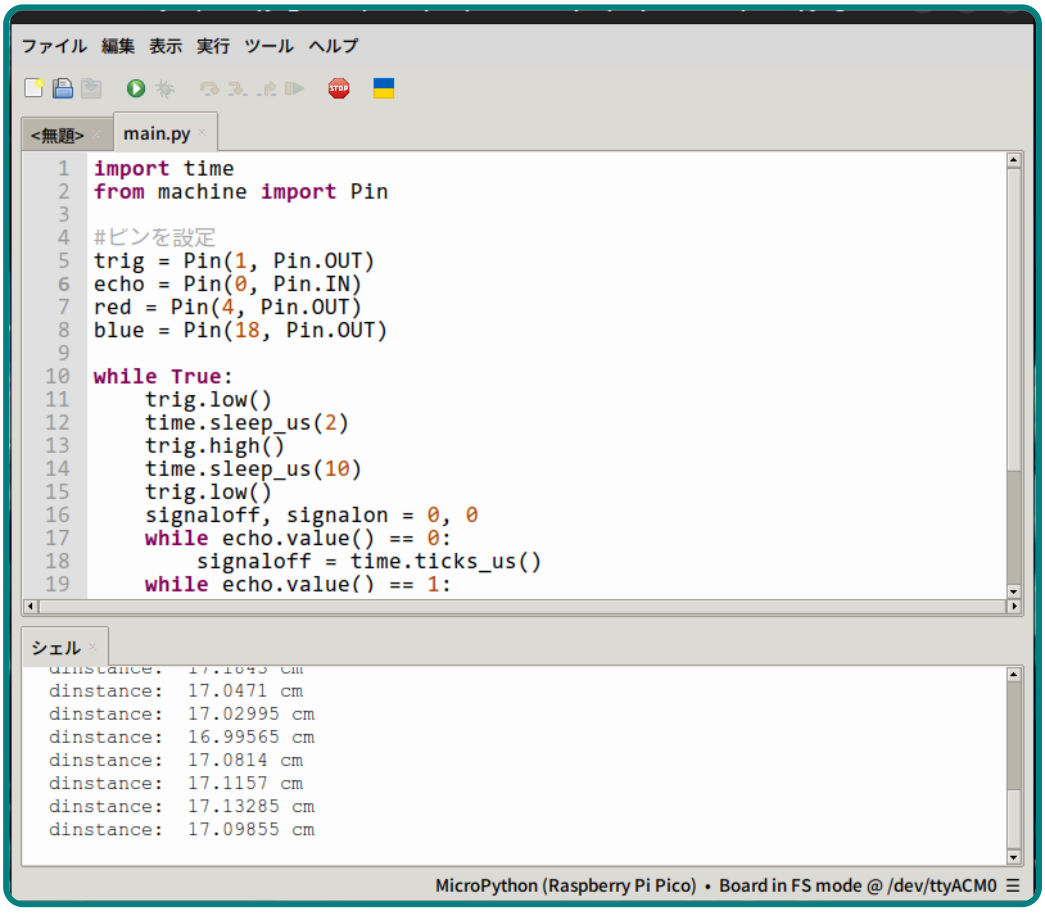
To start the application, simply type `thonny` in the command line.

2.3.3. Common to all OS

You can also use the `pip` command to install the software by typing the following command in a terminal on each OS.

This method requires prior installation of `pip`, but we will not discuss that here.

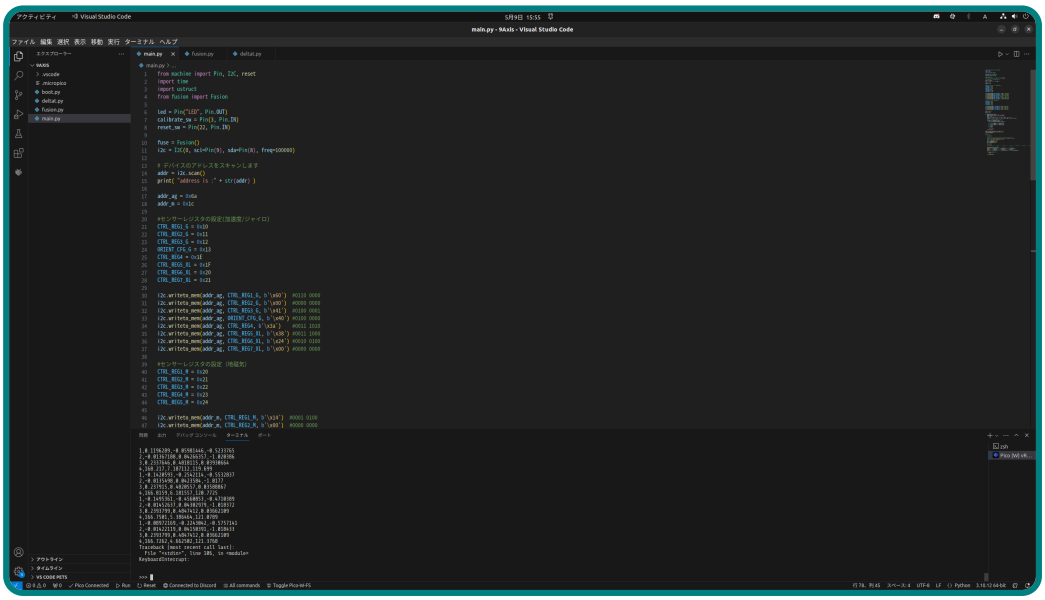
```
$ pip install thonny
```



The image will be the one executed on linux.

Please refer to **Official Wiki** for detailed information on how to operate and install the software in Thonny.

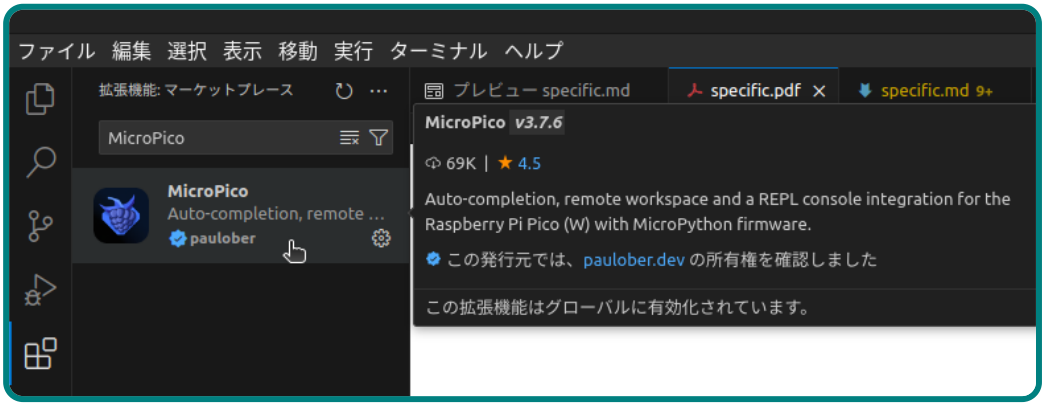
2.4. VSCode (Micro Pico)



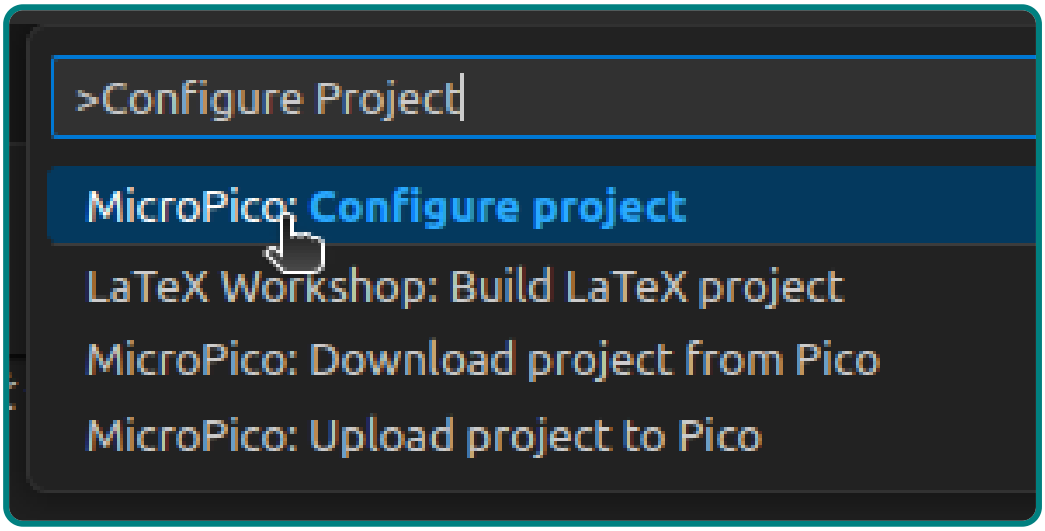
Another method is to use a well-known text editor called **VSCode**. There are many things you need to install, but this one is very easy to use, well designed, easy to understand, and has a full range of editor functions, including the popular AI with **extensions**, so it is recommended.

- **VSCode itself**
- VSCode Extensions - **MicroPico**
- VSCode Extensions - **Python**
- VSCode Extensions - **IntelliCode**
- VSCode Extensions - **Pylance**

For **VSCode itself**, simply download the installer for each OS from **Official HP** and install it. For **VSCode Extensions**, simply search for the name of the extension from the Extensions menu in VSCode and install it.



To start MicroPico, you must open the **Command Palette** (shortcut: open with **Ctrl + Shift + P**) with the folder open and select **MicroPico: Configure Project**. When you open the MicroPico folder configured in this way in VSCode, MicroPico will be launched automatically every time thereafter.



We will not discuss MicroPico's other operations here.
Please visit the **Official Repository** for more information.

3. Program Example

From here, we will introduce you to the **test program** that controls each component and how each is controlled.

The source code presented here is available in the **Official Turtle Pico Repository** and is provided under the **MIT License**. Feel free to download/modify it and use it as you like.
*Please note that what is presented here is only for **testing** and does not provide that much advanced control.

3.1. Directory Structure

The **directory structure** of the example program stored in the **Official Turtle Pico Repository** is as follows

IoT-IdeaKit/kit/src/
├**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/**

Each folder consists of a **lib** folder, **main.py** and **boot.py** files, and a **.micropico file**. The role of each file is as follows

- **lib**

  It contains the necessary library files. When used, it is used as follows

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

- **.micropico**

  This file enables micropico, a VSCode extension, in the corresponding folder. It has no particular meaning.

- **main.py**

  This is the main MicroPython file. It is executed after **boot.py**, and Pico goes into a **hibernation** state when this file finishes processing. So if you need to run it repeatedly, use a `while` loop or similar to keep the process going.

- **boot.py**

  This is the first file to be executed. It is good to describe the setup of each part to be used on the board, etc., but you can just describe them for each folder at the beginning of **main.py**, so there is no need to use this file in particular.

In the All Libraries folder, we have placed a definition file necessary to run the TurtlePico board called **TurtlePico.py** shown below. From this class, you can refer to pin information.

```
# 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
```

Refer to **1.4. Schematic/Pin Assignments** for pin information details.
Be careful not to download the wrong version, as the pin assignment differs depending on the version.

3.2. LED Lighting

The output of the microcontroller is not enough current to light the LEDs. Therefore, Turtle Pico connects the output from the microcontroller to the base of the transistor, and connects the output pin socket between VCC and the collector, making an **open collector** connection.

This allows a maximum current output of up to about 15 mA. The value of this current is constant as long as the Vf of the connected LED does not exceed Vcc.

Let us now explain the program. First, let's start with the library loading part.

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

The `import` is an instruction to load libraries. The **machine** libraries are mainly related to the Raspberry Pi Pico's **hardware** (timers, GPIOs, etc.), and the second line loads the `PWM` class among them.

The `time` library in the third line is for time management, among which the `sleep` function for delay is often used.

The last line loads the `TurtlePico` library described in 3.1 Directory Structure

The pins are set as follows

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

PWM is one of the basic functions of the microcontroller, which outputs a square wave that repeats High and Low outputs from the GOIO pin and raises or lowers the average voltage by varying the percentage of High time (**Duty Ratio**) of it. The LED on the right side of the board is defined as `blue` and the LED on the left side as `red`.

Even the Raspberry Pi Pico, which can only output High (3.3V) and Low (0V), can continuously change the voltage between 0 and 3.3V by using PWM, so this program uses this to adjust the **brightness**.

The corresponding pin of the PWM is connected to a **resistor** connected to the base of the transistor, so changing this voltage will also change the current flowing through the base according to **Ohm's Law**, and the current at the open collector output will change as well.

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

Here we are adjusting the frequency of the waveform of each PWM output: the duty ratio is all that is needed to change the voltage of the PWM, but if the frequency is too slow, this change will be visible and it will just look like a blink, so we need to give the frequency as an argument to the `freq` function and set the frequency to an invisible Therefore, the frequency must be set to an invisible speed by giving the frequency as an argument to the `freq` function.

```
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)
```

The duty ratio must be set between 0~65535 by the `duty_u16` function, which corresponds to 0~100%. In this program, the variable `i` is increased or decreased by 30 between 0~65536. When first going from 0 to 65536, the variable `status` is set to 0 to indicate that `i` is **increasing**, and when going from 65536 to 0, `status` is set to 1 to indicate that `i` is **decreasing**.

Depending on the value of `status`, an `if` branch is used to determine whether to add or subtract 30 to or from `i`. The output is then output by subtracting the duty ratio from 65536 for `blue` and leaving it as it is for `red`, and **inverting** the brightness for `bule` and `red`.

Finally, the duty ratio is output as debugging with `print` and also interspersed with a 1 ms delay with the `sleep` function for each loop.