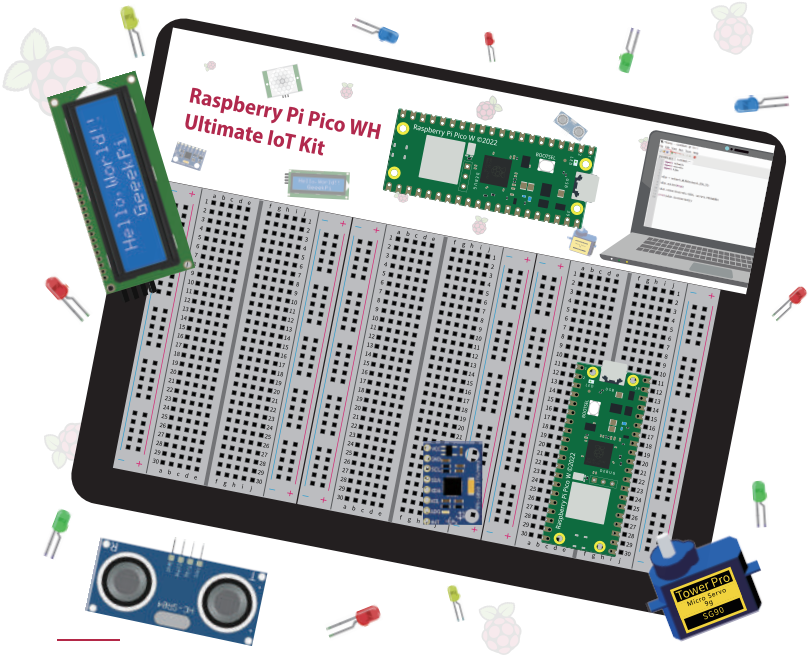
GeeekPi

Raspberry Pi Pico WH

IoT Starter Kit

KR-0006



Email: tech@52pi.net

DESCRIPTIONS

https://wiki.52pi.com

This kit is for makers, IoT engineer to build their own IoT projects, it contains Raspberry Pi Pico WH and Pico UPS module, OLED screen, temperature and humidity sensor, single channel relay and ultrasonic sensors, it can be combined to build different IoT projects, it contains a manual with several demo examples which is based MicroPython programming language

NOTE: Raspberry Pi Pico WH, **WH** means **wireless** and **with header pin soldered**

CONTENTS

FEATURES

- Easy to Understand
- Easy to assembling
- Mobile Power bank
- Various Sensors for IoT Projects
- Demo code and Examples

PACKAGE INCLUDES

- 1 x Raspberry Pi Pico WH
- 1 x breadboard experiment platform
- 1 x 0.96-inch OLED display
- 1 x DHT11 temperate and Humidity sensor
- 1 x Single Channel Relay
- 1 x LCD1602 display module
- 1 x Raindrops sensor
- 1 x PS2 Joystick Module
- 1 x MPU6050 Gyroscope module
- 1 x MicroUSB cable (Data Transfer)
- 20 x Male to male jumper wire
- 20 x Male to female jumper wire
- 1 x ULN2003AN driver board (drive stepper motor)
- 5 x Red LED
- 5 x Green LED
- 5 x Yellow LED
- 5 x Blue LED
- 20 x 220Ω Resistor
- 1 x Instructions
- 1 x 9g Servo
- 1 x Potentiometer
- 1 x Stepper motor
- 1 x PIR sensor
- 1 x Ultrasonic sensor

GETTING START

PREPARATION

Before you start to program, you need to download the firmware for your Raspberry Pi Pico WH, Please read following URLs carefully:
https://www.raspberrypi.com/products/raspberry-pi-pico/

Datasheet of RP2040 chip:
https://www.raspberrypi.com/documentation/microcontrollers/

Datasheet of RPI Pico W:
https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf

Raspberry Pi Pico W and Pico WH

Raspberry Pi Pico W adds on-board single-band 2.4GHz wireless interfaces (802.11n) using the Infineon CYW43439 while retaining the Pico form factor. The on-board 2.4GHz wireless interface has the following features:

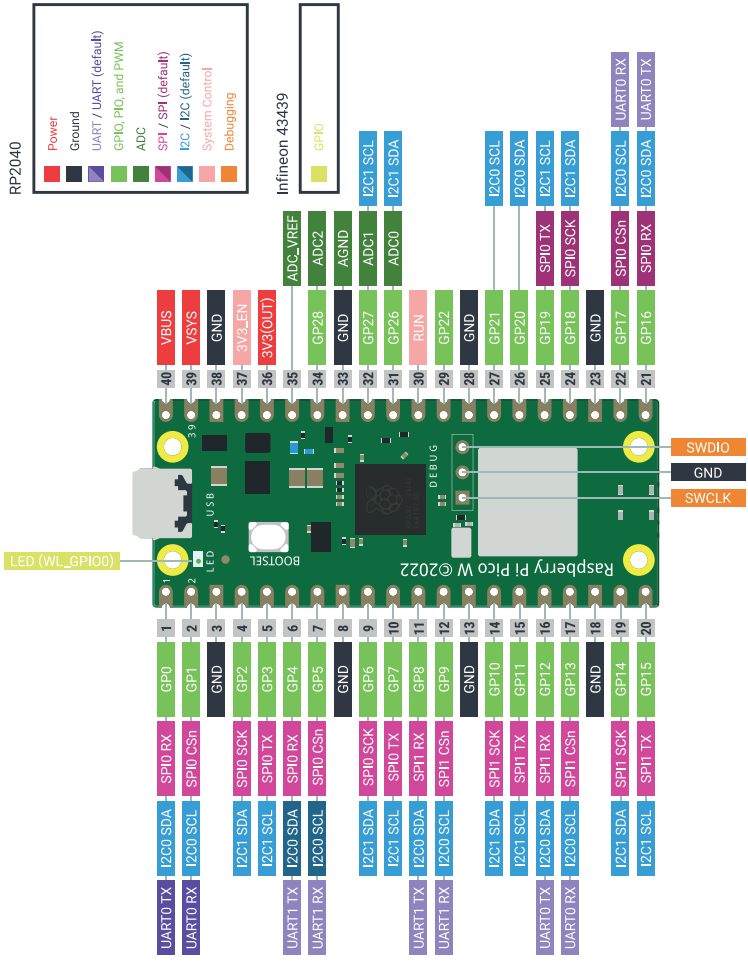
- Wireless (802.11n), single-band (2.4 GHz)
- WPA3
- Soft access point supporting up to four clients

The antenna is an onboard antenna licensed from ABRACON (formerly ProAnt). The wireless interface is connected via SPI to the RP2040 microcontroller.

Due to pin limitations, some of the wireless interface pins are shared. The CLK is shared with VSYS monitor, so only when there isn't an SPI transaction in progress can VSYS be read via the ADC. The Infineon CYW43439 DIN/DOUT and IRQ all share one pin on the RP2040. Only when an SPI transaction isn't in progress is it suitable to check for IRQs. The interface typically runs at 33MHz.

For best wireless performance, the antenna should be in free space. For instance, putting metal under or close by the antenna can reduce its performance both in terms of gain and bandwidth. Adding grounded metal to the sides of the antenna can improve the antenna's bandwidth.

Raspberry Pi Pico W and Pico WH



Debugging using another Raspberry Pi Pico

It is possible to use one Raspberry Pi Pico to debug another Pico. This is possible via picoprobe, an application that allows a Pico to act as a USB → SWD and UART converter. This makes it easy to use a Pico on non-Raspberry Pi platforms such as Windows, Mac, and Linux computers where you don't have GPIOs to connect directly to your Pico. Full instructions on how to use Picoprobe to do this are available in our 'getting started' documentation.

Download the UF2 file:
https://datasheets.raspberrypi.com/soft/picoprobe.uf2

Go to the Picoprobe Github repository:
https://github.com/raspberrypi/picoprobe

Resetting Flash memory

Pico's BOOTSEL mode lives in read-only memory inside the RP2040 chip, and can't be overwritten accidentally. No matter what, if you hold down the BOOTSEL button when you plug in your Pico, it will appear as a drive onto which you can drag a new UF2 file. There is no way to brick the board through software. However, there are some circumstances where you might want to make sure your Flash memory is empty. You can do this by dragging and dropping a special UF2 binary onto your Pico when it is in mass storage mode.

Download the UF2 file:
https://datasheets.raspberrypi.com/soft/flash_nuke.uf2

See the code on GitHub:
https://github.com/raspberrypi/pico-examples/blob/master/flash/nuke/nuke.c

WHAT IS MICROPYTHON?

MicroPython is a full implementation of the Python 3 programming language that runs directly on embedded hardware like Raspberry Pi Pico. You get an interactive prompt (the REPL) to execute commands immediately via USB Serial, and a built-in file system.

The Pico port of MicroPython includes modules for accessing low-level chip-specific hardware.

The MicroPython:
https://github.com/micropython/micropython/wiki

NOTE

If you're new to MicroPython, our official guide, "Get started with MicroPython on Raspberry Pi Pico", is a great place to start. Learn the basics of MicroPython and physical computing, connect your Pico to displays and sensors, build alarms, reaction games, and more.

Drag-and-Drop MicroPython

You can program your Pico by connecting it to a computer via USB, then dragging and dropping a file onto it so we' ve put together a downloadable UF2 file to let you install MicroPython more easily.

Download the correct MicroPython UF2 file for your board:
Raspberry Pi Pico W (with urequests and upip preinstalled):
https://micropython.org/download/rp2-pico-w/rp2-pico-w-latest.uf2

Then go ahead and:

Push and hold the BOOTSEL button and plug your Pico into the USB port of your Raspberry Pi or other computer. Release the BOOTSEL button after your Pico is connected.

It will mount as a Mass Storage Device called RPI-RP2.

Drag and drop the MicroPython UF2 file onto the RPI-RP2 volume. Your Pico will reboot. You are now running MicroPython.

You can access the REPL via USB Serial.

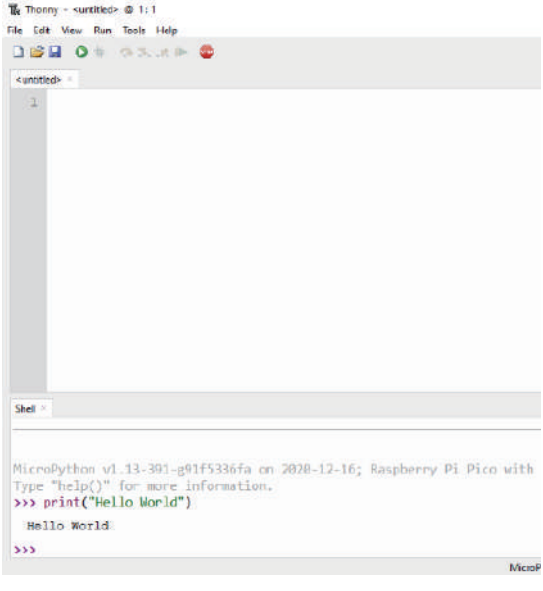
The Raspberry Pi Pico Python SDK book contains step-by-step instructions for connecting to your Pico and programming it in MicroPython using both the command line and the Thonny IDE. which you can download from:
https://thonny.org/

and then download raspberry pi pico python SDK manual PDF file:
https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-python-sdk.pdf

1. **Download and install Thonny** for your OS, if you don't already have it. You can grab it for free from the Thonny website.
https://thonny.org/

2. **Connect the Raspberry Pi Pico** to your computer and in **Thonny go to Tools > Options and click on the Interpreter tab**. From the interpreter dropdown list select MicroPython (Raspberry Pi Pico). The port dropdown menu can be left to automatically detect the Pico. **Click Ok** to close.
The Python Shell (also called REPL, Read, Eval, Print, Loop) will now update to show that the Pico is connected and working.

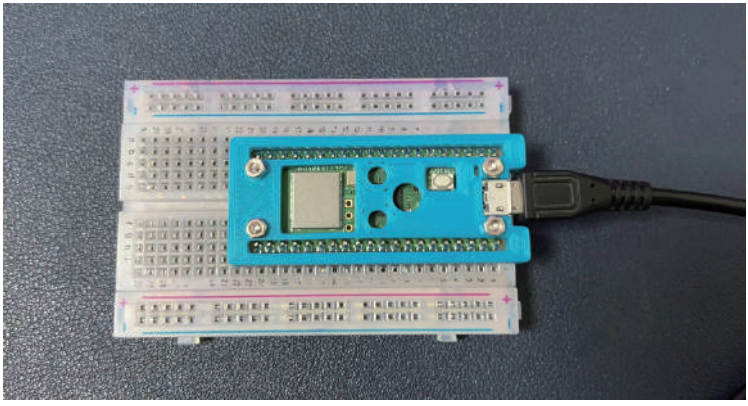
3. To test we can write a quick print function to say "Hello World." Press Enter to run the code.



DEMO 1 LIGHTS UP LED ONBOARD

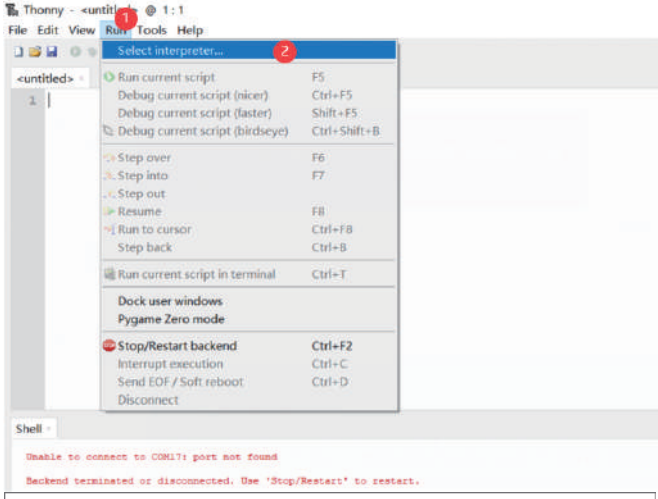
1. WIRING

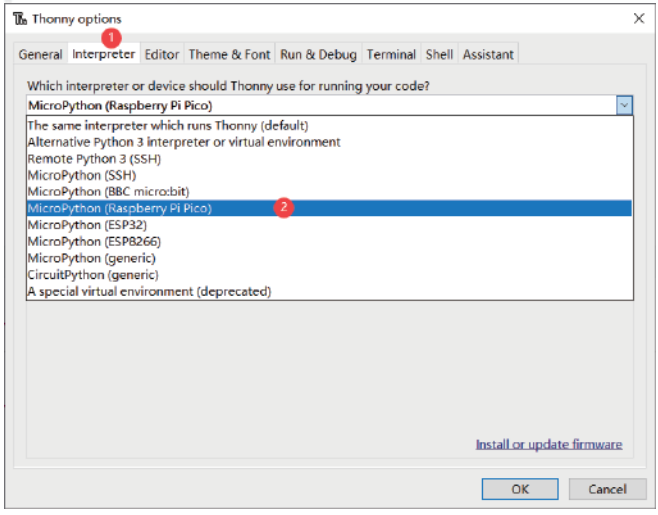
Do not need wiring, LED is onboard. Just connect the Raspberry Pi Pico to breadboard and connect the MicroUSB cable to USB port on your PC.

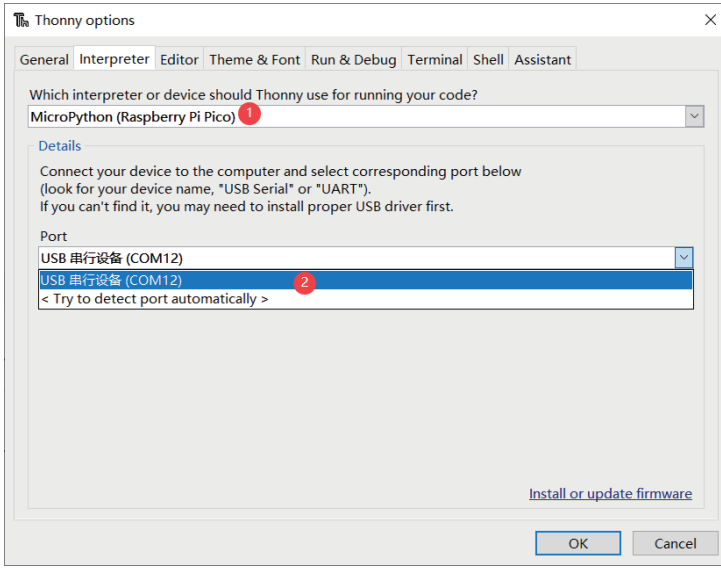


2. CODING

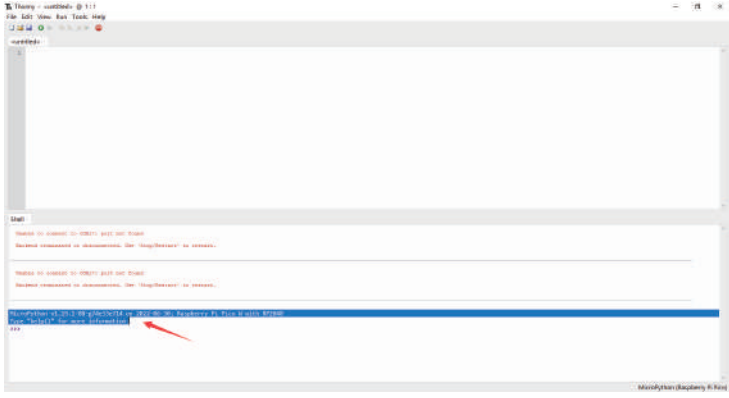
Open thonny IDE and connect the board to your PC or Raspberry Pi, and click "**run**"→ "**select interpreter**"→ "**Raspberry Pi Pico**"→ **select serial port** (please click the menu to select your serial port, such as COM1, or COMX, depends on your system. the Wi-Fi variant there are changes made in the MicroPython framework, to let the same code blink a led on any Raspberry Pi Pico, they gave the needed pin a name; "LED" , and added two functions, on and off.


Thonny - <untitled> @ 1 : 1
File Edit View Run Tools Help
1
2
Select interpreter...
Run current script F5
Debug current script (nicer) Ctrl+F5
Debug current script (faster) Shift+F5
Debug current script (birdseye) Ctrl+Shift+B
Step over F6
Step into F7
Step out
Resume F8
Run to cursor Ctrl+F8
Step back Ctrl+B
Run current script in terminal Ctrl+T
Dock user windows
Pygame Zero mode
Stop/Restart backend Ctrl+F2
Interrupt execution Ctrl+C
Send EOF / Soft reboot Ctrl+D
Disconnect
<untitled>
Shell
Unable to connect to COM17: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.


Thonny options
1
General Interpreter Editor Theme & Font Run & Debug Terminal Shell Assistant
Which interpreter or device should Thonny use for running your code?
MicroPython (Raspberry Pi Pico)
The same interpreter which runs Thonny (default)
Alternative Python 3 interpreter or virtual environment
Remote Python 3 (SSH)
MicroPython (SSH)
MicroPython (BBC micro:bit)
MicroPython (Raspberry Pi Pico)
2
MicroPython (ESP32)
MicroPython (ESP8266)
MicroPython (generic)
CircuitPython (generic)
A special virtual environment (deprecated)
Install or update firmware
OK
Cancel



Before you coding, make sure that your MicroPython version is:



NOTE:
MicroPython v1.19.1-88-g74e33e714 on 2022-06-30; Raspberry Pi Pico W with RP2040

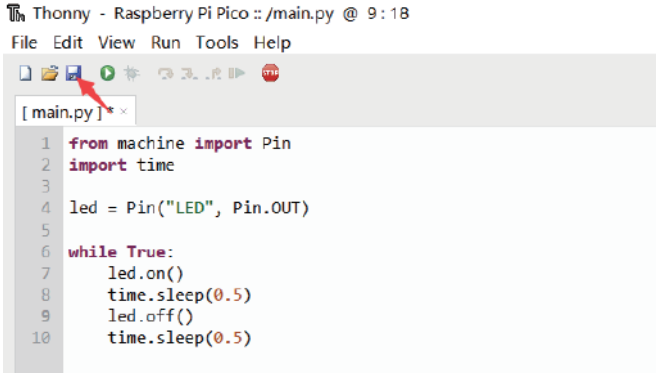
Type "help()" for more information.

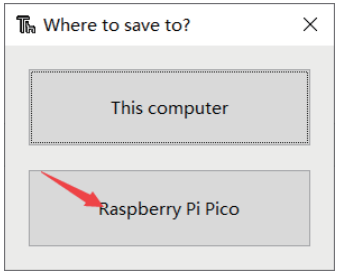
Create new file and input:

```
from machine import Pin
import time

led = Pin("LED", Pin.OUT)

while True:
    led.on()  # a method instead of setting the value
    time.sleep(0.5)
    led.off()  #  turn it off again.
    time.sleep(0.5)
```

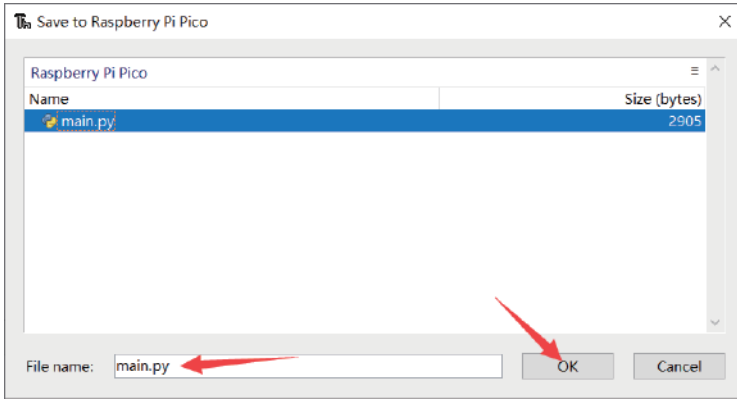
3. TESTING

Click "file"→Save as →"select Raspberry Pi Pico" and then click "play button" you will see the LED is blinking.





and named it as "main.py"

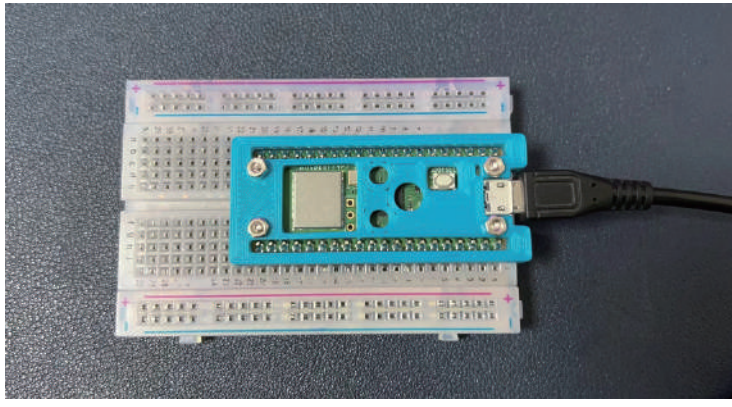


and you will see the LED onboard is blinking.

DEMO 2 CONNECT TO LOCAL NETWORK

1. WIRING

Do not need wiring, LED is onboard. Just connect the Raspberry Pi Pico to breadboard and connect the MicroUSB cable to USB port on your PC.



2. CODING

1. Setup your Raspberry Pi Pico W
You will need to install MicroPython on your Pico W before you can proceed further.

2. Open the Thonny editor to a blank document.

3. Create an object called SSID and in it store the SSID of your Wi-Fi access point.

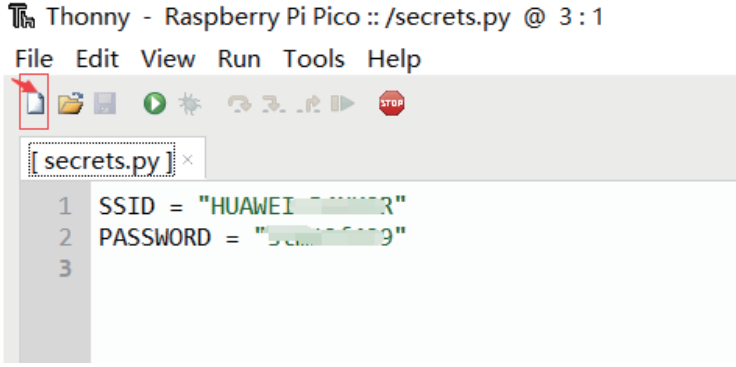
```
SSID = "YOUR WIFI AP"
```

4. **Create an object called PASSWORD and store your Wi-Fi password.**

```
PASSWORD = "YOUR_PASSWORD"
```

5. **Save the file to the Raspberry Pi Pico W as secrets.py** By storing our sensitive details in a secrets file, we can freely share the project code with friends, or online. Just remember not to share the secrets file too.

6. **Click on New File** to create a new blank document.



7. **Import three modules of code, network, secrets and time.** These three modules enable our Pico to connect to a Wi-Fi network, use the data stored in secrets.py and to add a pause to the code.

```
import network
import secrets
import time
```

8. **Create an object, wlan, to create a connection from our code to the Pico W wireless chip.**
We use this connection to issue commands that will connect and check our Wi-Fi connection.

```
wlan = network.WLAN(network.STA_IF)
```

9. **Turn on the Raspberry Pi Pico W's Wi-Fi.**

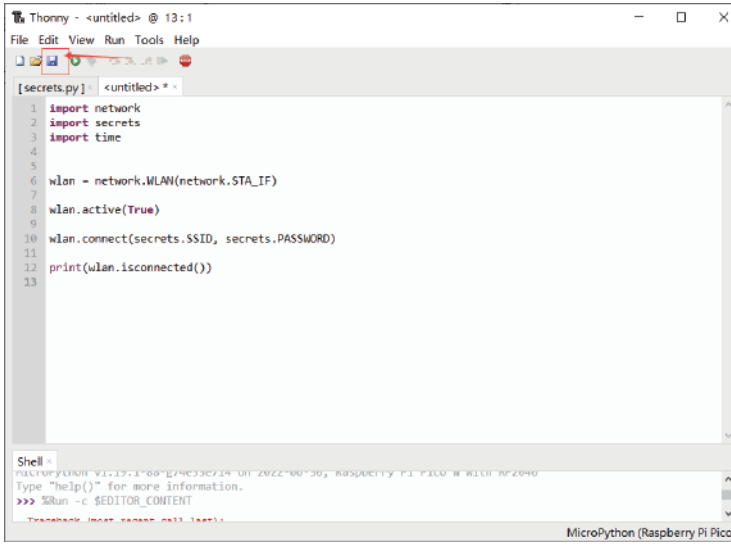
```
wlan.active(True)
```

10. Connect to your router using the SSID and PASSWORD stored in the secrets.py file.

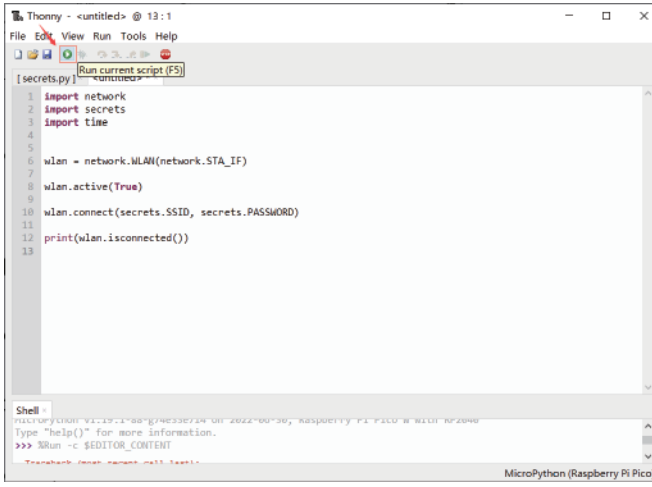
```
wlan.connect(secrets.SSID, secrets.PASSWORD)
```

11. Print the connection status to the Python shell. This will print True if connected, and False if the connection failed.

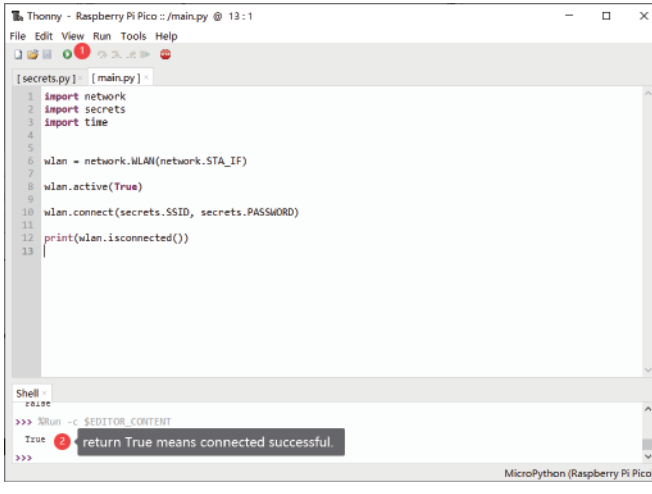
12. Click Save and then select “Raspberry Pi Pico” . Save the file as Wi-Fi.py to the Raspberry Pi Pico W.



13. Click on Run to start the code.



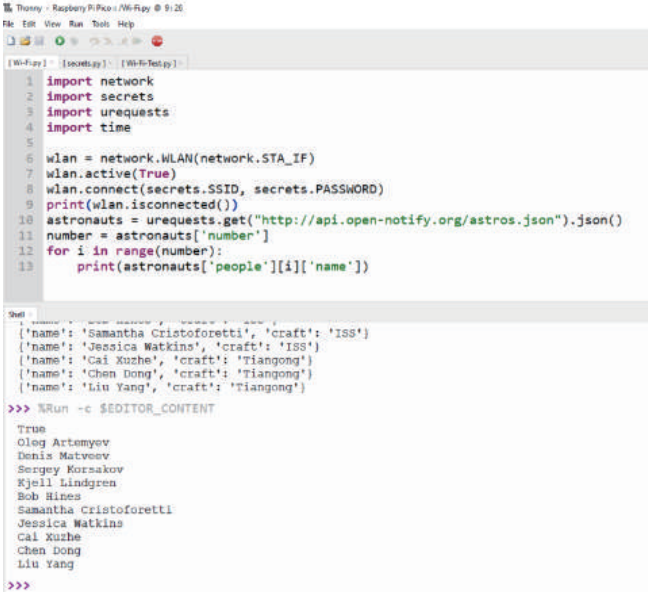
14. Look in the Python Shell for True or False. True means we are connected.



3. COMPLETE CODE LISTING

```
import network
import secrets
import time
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(secrets.SSID, secrets.PASSWORD)
print(wlan.isconnected())
```

Using the Raspberry Pi Pico W with External Data



1. Add a line after "import time" and import the urequests module.
This module enables us to work with network requests such as HTTP and JSON.

```
import urequests
```

2. After print(wlan.isconnected()) add a new line which creates an object "astronauts" and then uses urequests to get the information in a JSON format.
JavaScript Object Notation is an open standard file format which bears a striking resemblance to Python's Dictionary which uses keys (names) to retrieve values from the object.
http://api.open-notify.org


```
{
   message: "success",
 - people: [
     - {
           name: "Cai Xuzhe",
           craft: "Tiangong"
       },
     - {
           name: "Chen Dong",
           craft: "Tiangong"
       },
     - {
           name: "Liu Yang",
           craft: "Tiangong"
       },
     - {
           name: "Sergey Prokopyev",
           craft: "ISS"
       },
     - {
           name: "Dmitry Petelin",
           craft: "ISS"
       },
     - {
           name: "Frank Rubio",
           craft: "ISS"
       },
     - {
           name: "Nicole Mann",
           craft: "ISS"
       },
     - {
           name: "Josh Cassada",
           craft: "ISS"
       },
     - {
           name: "Koichi Wakata",
```

```
astronauts = urequests.get("http://api.open-notify.org/astros.json").json()
```

3. **Create an object, number, which will open the astronauts object, and look for the key 'number'.** The value linked to that key is then stored in the number object.

```
number = astronauts['number']
```

4. **Create a for loop that will iterate for the number of people on the International Space Station.** This value could change as astronauts come and go, so rather than hard coding a value we use the live data.

```
for i in range(number):
```

5. **Print the name of each astronaut on the International Space Station using a series of keys that target the specific data.** Our dictionary 'astronauts' has many keys, but we are interested in the 'people' , the value of "i" will increment each time the loop goes round, and it selects each person from a list embedded in the dataset. We then use another key, 'name' to get the name of that astronaut.

```
  print(astronauts['people'][i]['name'])
```

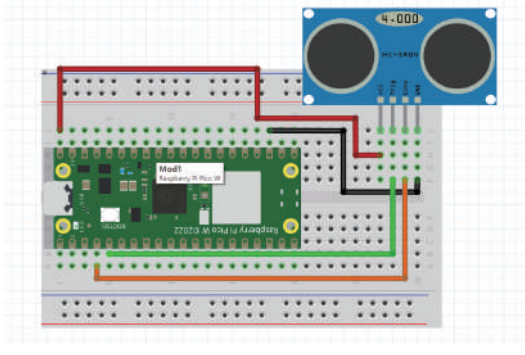
6. **Save the code and when ready click on Run to start the code.**

7. **The names of all the astronauts on the International Space Station will appear in the Python Shell.** Note that "True" still appears, confirming that our Internet connection is established.

DEMO 3 MEASURE DISTANCE AND SHOW THE DATA ON WEB

1. WIRING

Insert the ultrasonic sensor into the breadboard.
Connect the 3V3 pin of the Raspberry Pi Pico to the VCC pin of the ultrasonic sensor using a male to male jumper wire.
Connect a GND pin on the Raspberry Pi Pico to the GND pin of the ultrasonic sensor using a jumper wire.
Connect the Trigger pin of the ultrasonic sensor to GPIO pin 3 of the Raspberry Pi Pico.
Connect the Echo pin from the ultrasonic sensor to GPIO pin 2 of the Raspberry Pi Pico.



2. CODING

Software Setup of Ultrasonic Sensor on Raspberry Pi Pico

With the circuit built, connect your Raspberry Pi Pico and open the Thonny application.

1. Import the Pin class from the machine library and then import the utime library. The former is used to control GPIO pins, the latter is a library of time based functions.

```
from machine import Pin
import utime
```

2. **Create two new objects, trigger and echo**. These objects configure the GPIO pins of the Pico to be used with the ultrasonic sensor. For example, our trigger pin is used to send a pulse of current, as such it is an output pin. The echo pin receives the reflected pulse, so echo is an input.

```
trigger = Pin(3, Pin.OUT)
echo = Pin(2, Pin.IN)
```

3. **Create a function, ultra()**, which will contain the code necessary to take a reading.

```
def ultra():
```

4. **Pull the trigger pin low,** to ensure that it is not active, then pause for two microseconds.

```
  trigger.low()
  utime.sleep_us(2)
```

5. **Pull the trigger pin high for five microseconds before pulling the trigger pin low**. This will send a short pulse from the ultrasonic sensor and then turn off the pulse.

```
  trigger.high()
  utime.sleep_us(5)
  trigger.low()
```

6. **Create a while loop to check the echo pin**. If no echo pulse is received, update a variable, signaloff so that it contains a timestamp in microseconds.

```
  while echo.value() == 0:
     signaloff = utime.ticks_us()
```

7. **Create another while loop, this time to check if an echo has been received**. This will store the current timestamp in microseconds to the signalon variable.

```
  while echo.value() == 1:
     signalon = utime.ticks_us()
```

8. **Create a new variable, timepassed**, which will store the value total time taken for the pulse to leave the sensor, hit the object and return back to the sensor as an echo.

```
  timepassed = signalon - signaloff
```

9. **Create a new variable, distance**. This variable will store the answer of the equation. We multiply the journey time (timepassed) by the speed of sound (343.2 m/s, which is

0.0343 cm per microsecond) the product of that equation is divided by two as we do not need the total journey distance, just the distance from the object to the sensor.

```
distance = (timepassed * 0.0343) / 2
```

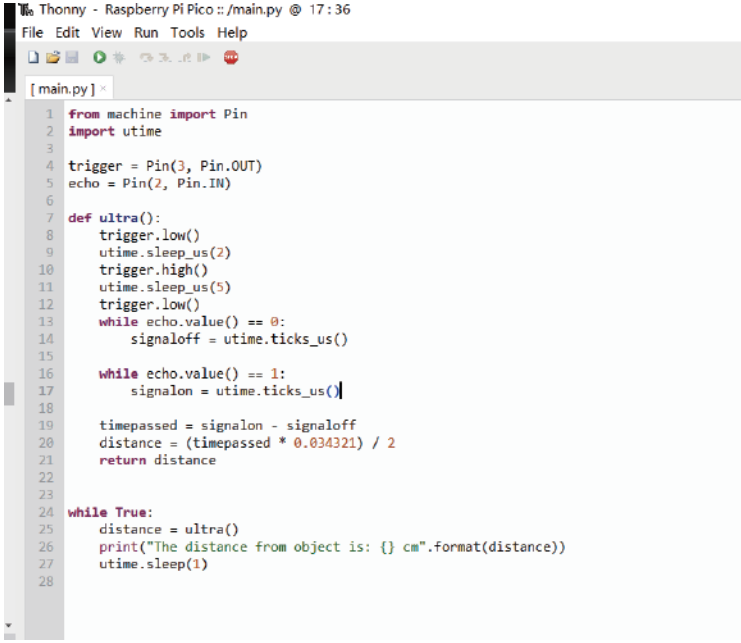
10. Print a message to the Python Shell showing the distance.

```
return distance
```

11. Moving out of the function we now create a loop that will run the function every second.

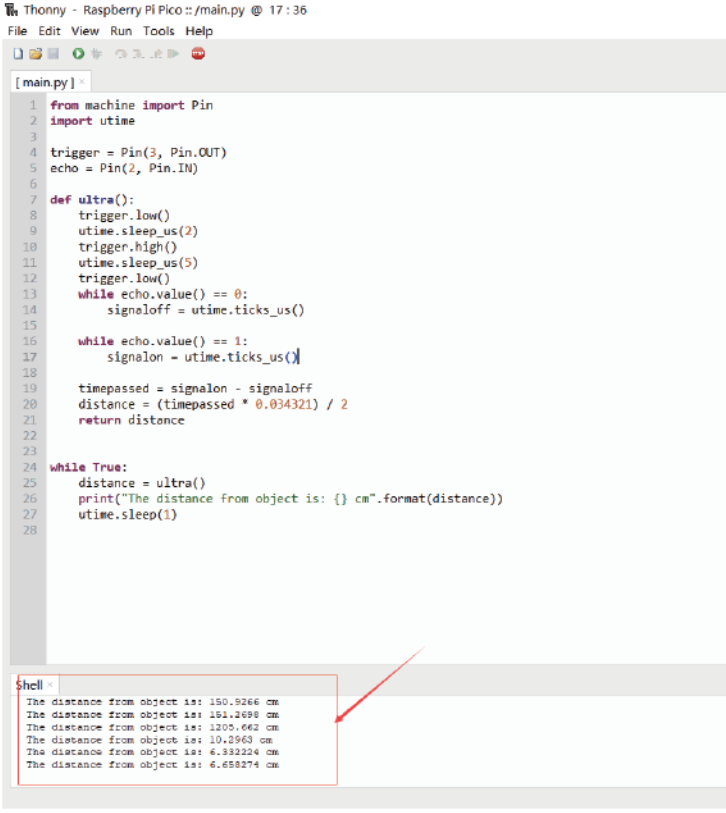
```
while True:
  distance = ultra()
  print("The distance from object is: {} cm".format(distance))
  utime.sleep(1)
```

In Thonny IDE will be like:



3. TESTING

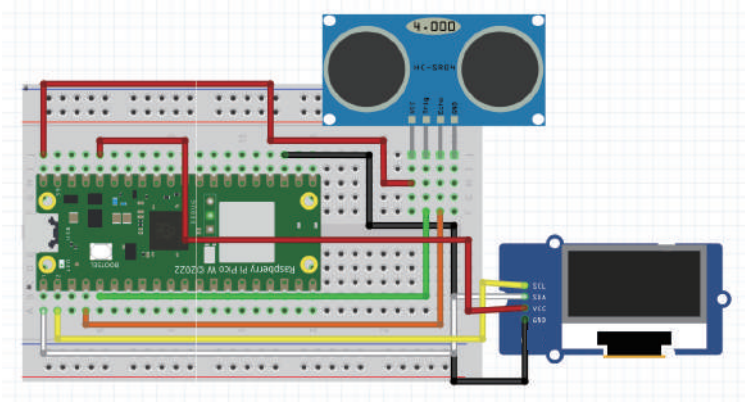
save it to main.py and click run button.
you will see:



DEMO 4 MEASURE DISTANCE AND SHOW THE DATA ON 0.96-INCH OLED

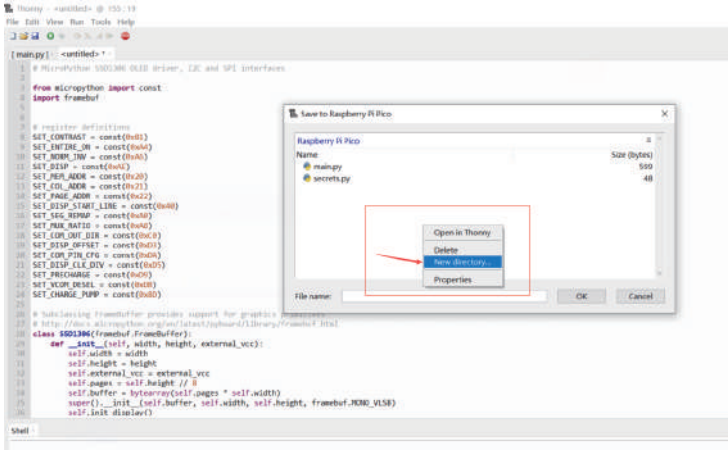
1. WIRING

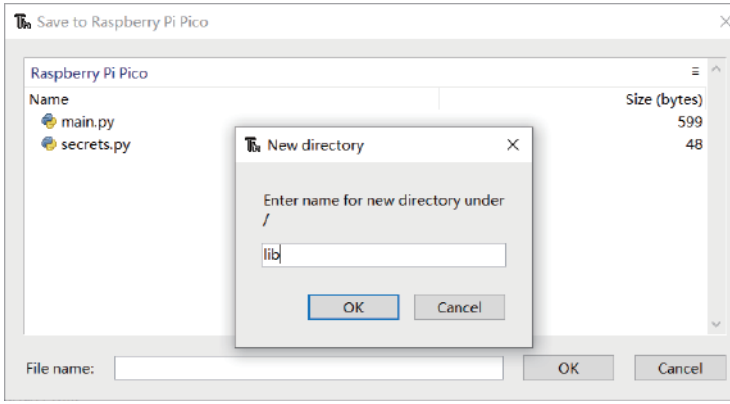
Connect the SDA pin of the OLED to GPIO pin 0 of the Raspberry Pi Pico.
Connect the SCL pin from the OLED to GPIO pin 1 of the Raspberry Pi Pico.
Connect the VCC pin from the OLED to 3V3(OUT) of the Raspberry Pi Pico.
Connect the GND pin from the OLED to GND of the Raspberry Pi Pico.

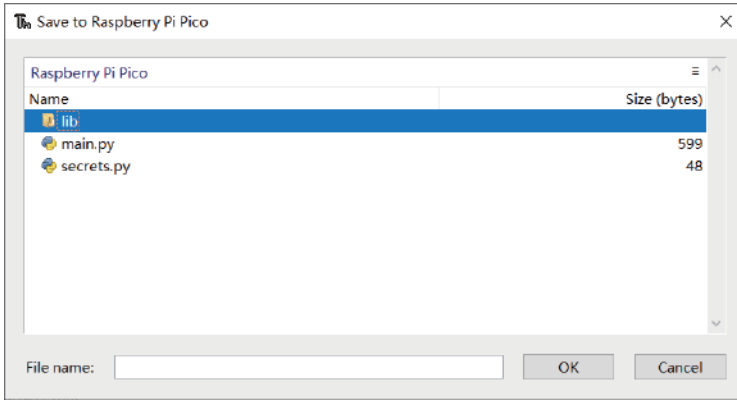


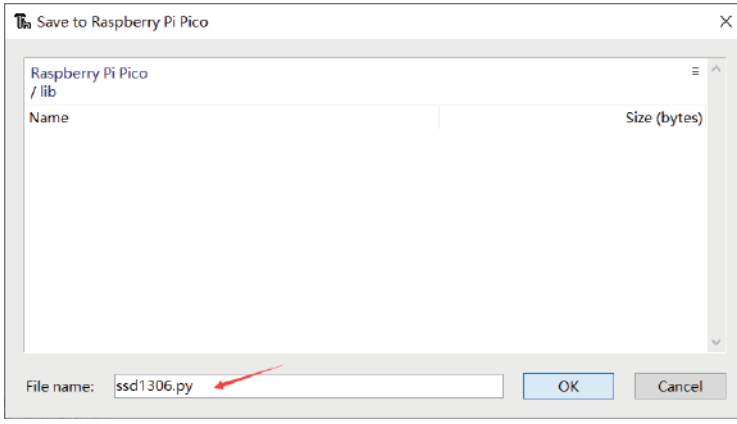
2. CODING

At first, you need to create a new file called: ssd1306.py on Thonny IDE, and paste following code to it. and save it to Raspberry Pi Pico' s lib folder, if lib folder does not exist, Create one by right click the blank.


Thonny - <untitled> @ 155 : 19
File Edit View Run Tools Help
[main.py] [<untitled> *]
# MicroPython SSD1306 OLED driver, I2C and SPI interfaces
from micropython import const
import framebuf
# register definitions
SET_CONTRAST = const(0x81)
SET_ENTIRE_ON = const(0xA4)
SET_NORM_INV = const(0xA6)
SET_DISP = const(0xAE)
SET_MEM_ADDR = const(0x20)
SET_COL_ADDR = const(0x21)
SET_PAGE_ADDR = const(0x22)
SET_DISP_START_LINE = const(0x40)
SET_SEG_REMAP = const(0xA0)
SET_MUX_RATIO = const(0xA8)
SET_COM_OUT_DIR = const(0xC0)
SET_DISP_OFFSET = const(0xD3)
SET_COM_PIN_CFG = const(0xDA)
SET_DISP_CLK_DIV = const(0xD5)
SET_PRECHARGE = const(0xD9)
SET_VCOM_DESEL = const(0xDB)
SET_CHARGE_PUMP = const(0x8D)
# Subclassing FrameBuffer provides support for graphics primitives
# http://docs.micropython.org/en/latest/pyboard/library/framebuf.html
class SSD1306(framebuf.FrameBuffer):
def __init__(self, width, height, external_vcc):
self.width = width
self.height = height
self.external_vcc = external_vcc
self.pages = self.height // 8
self.buffer = bytearray(self.pages * self.width)
super().__init__(self.buffer, self.width, self.height, framebuf.MONO_VLSB)
self.init_display()
Shell
Save to Raspberry Pi Pico
Raspberry Pi Pico
Name
Size (bytes)
main.py
599
secrets.py
48
Open in Thonny
Delete
New directory...
Properties
File name:
OK
Cancel


Save to Raspberry Pi Pico
Raspberry Pi Pico
Name
Size (bytes)
main.py
599
secrets.py
48
New directory
Enter name for new directory under
/
lib
OK
Cancel
File name:
OK
Cancel





ssd1306.py file content:
MicroPython SSD1306 OLED driver, I2C and SPI interfaces

```
from micropython import const
import framebuf
```

```
# register definitions
SET_CONTRAST = const(0x81)
SET_ENTIRE_ON = const(0xA4)
SET_NORM_INV = const(0xA6)
SET_DISP = const(0xAE)
SET_MEM_ADDR = const(0x20)
SET_COL_ADDR = const(0x21)
SET_PAGE_ADDR = const(0x22)
SET_DISP_START_LINE = const(0x40)
SET_SEG_REMAP = const(0xA0)
SET_MUX_RATIO = const(0xA8)
SET_COM_OUT_DIR = const(0xC0)
SET_DISP_OFFSET = const(0xD3)
SET_COM_PIN_CFG = const(0xDA)
SET_DISP_CLK_DIV = const(0xD5)
SET_PRECHARGE = const(0xD9)
SET_VCOM_DESEL = const(0xDB)
SET_CHARGE_PUMP = const(0x8D)

# Subclassing FrameBuffer provides support for graphics primitives
# http://docs.micropython.org/en/latest/pyboard/library/framebuf.html
class SSD1306(framebuf.FrameBuffer):
    def __init__(self, width, height, external_vcc):
        self.width = width
        self.height = height
        self.external_vcc = external_vcc
        self.pages = self.height // 8
        self.buffer = bytearray(self.pages * self.width)
        super().__init__(self.buffer, self.width, self.height, framebuf.MONO_VLSB)
        self.init_display()

    def init_display(self):
        for cmd in (
            SET_DISP | 0x00,  # off
            # address setting
            SET_MEM_ADDR,
            0x00,  # horizontal
            # resolution and layout
            SET_DISP_START_LINE | 0x00,
```

```
        SET_SEG_REMAP | 0x01,  # column addr 127 mapped to SEG0
        SET_MUX_RATIO,
        self.height - 1,
        SET_COM_OUT_DIR | 0x08,  # scan from COM[N] to COM0
        SET_DISP_OFFSET,
        0x00,
        SET_COM_PIN_CFG,
        0x02 if self.width > 2 * self.height else 0x12,
        # timing and driving scheme
        SET_DISP_CLK_DIV,
        0x80,
        SET_PRECHARGE,
        0x22 if self.external_vcc else 0xF1,
        SET_VCOM_DESEL,
        0x30,  # 0.83*Vcc
        # display
        SET_CONTRAST,
        0xFF,  # maximum
        SET_ENTIRE_ON,  # output follows RAM contents
        SET_NORM_INV,  # not inverted
        # charge pump
        SET_CHARGE_PUMP,
        0x10 if self.external_vcc else 0x14,
        SET_DISP | 0x01,
    ):  # on
        self.write_cmd(cmd)
    self.fill(0)
    self.show()

def poweroff(self):
    self.write_cmd(SET_DISP | 0x00)

def poweron(self):
    self.write_cmd(SET_DISP | 0x01)

def contrast(self, contrast):
    self.write_cmd(SET_CONTRAST)
    self.write_cmd(contrast)
```

```
    def invert(self, invert):
        self.write_cmd(SET_NORM_INV | (invert & 1))

    def show(self):
        x0 = 0
        x1 = self.width - 1
        if self.width == 64:
            # displays with width of 64 pixels are shifted by 32
            x0 += 32
            x1 += 32
        self.write_cmd(SET_COL_ADDR)
        self.write_cmd(x0)
        self.write_cmd(x1)
        self.write_cmd(SET_PAGE_ADDR)
        self.write_cmd(0)
        self.write_cmd(self.pages - 1)
        self.write_data(self.buffer)


class SSD1306_I2C(SSD1306):
    def __init__(self, width, height, i2c, addr=0x3C, external_vcc=False):
        self.i2c = i2c
        self.addr = addr
        self.temp = bytearray(2)
        self.write_list = [b"\x40", None]  # Co=0, D/C#=1
        super().__init__(width, height, external_vcc)

    def write_cmd(self, cmd):
        self.temp[0] = 0x80  # Co=1, D/C#=0
        self.temp[1] = cmd
        self.i2c.writeto(self.addr, self.temp)

    def write_data(self, buf):
        self.write_list[1] = buf
        self.i2c.writevto(self.addr, self.write_list)


class SSD1306_SPI(SSD1306):
    def __init__(self, width, height, spi, dc, res, cs, external_vcc=False):
```

```
        self.rate = 10 * 1024 * 1024
        dc.init(dc.OUT, value=0)
        res.init(res.OUT, value=0)
        cs.init(cs.OUT, value=1)
        self.spi = spi
        self.dc = dc
        self.res = res
        self.cs = cs
        import time

        self.res(1)
        time.sleep_ms(1)
        self.res(0)
        time.sleep_ms(10)
        self.res(1)
        super().__init__(width, height, external_vcc)

    def write_cmd(self, cmd):
        self.spi.init(baudrate=self.rate, polarity=0, phase=0)
        self.cs(1)
        self.dc(0)
        self.cs(0)
        self.spi.write(bytearray([cmd]))
        self.cs(1)

    def write_data(self, buf):
        self.spi.init(baudrate=self.rate, polarity=0, phase=0)
        self.cs(1)
        self.dc(1)
        self.cs(0)
        self.spi.write(buf)
        self.cs(1)
```

3. CODING AND TESTING

1. Import the Pin class and I2C class from the machine library and then import the SSD1306_I2C class from the library, import framebuf library, and setting the variables of the OLED display.

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import framebuf
import utime


WIDTH = 128
HEIGHT = 64
```

2. Create instance of the bus of I2C, named it i2c, and create an instance, named oled, after that, clear the screen with fill method.

```
i2c = I2C(0, scl=Pin(1), sda=Pin(0), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)

# clear screen
oled.fill(0)
```

3. Measure the distance with previous demo' s code and using text method of oled to display the distance information.

```
trigger = Pin(3, Pin.OUT)
echo = Pin(2, Pin.IN)

def ultra():
    trigger.low()
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(5)
    trigger.low()
    while echo.value() == 0:
        signaloff = utime.ticks_us()

    while echo.value() == 1:
        signalon = utime.ticks_us()

    timepassed = signalon - signaloff
    distance = (timepassed * 0.034321) / 2
    return distance

while True:
```

```
distance = ultra()
print("The distance from object is: {} cm".format(distance))
oled.text(str(distance), 0, 10)
utime.sleep(1)
```

[main.py] <untitled> *

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import framebuf
import utime


WIDTH = 128
HEIGHT = 64

i2c = I2C(0, scl=Pin(1), sda=Pin(0), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
oled.fill(0)

trigger = Pin(3, Pin.OUT)
echo = Pin(2, Pin.IN)

def ultra():
    trigger.low()
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(5)
    trigger.low()
    while echo.value() == 0:
        signaloff = utime.ticks_us()

    while echo.value() == 1:
        signalon = utime.ticks_us()

    timepassed = signalon - signaloff
    distance = (timepassed * 0.034321) / 2
    return distance

while True:
    distance = ultra()
    print("The distance from object is: {} cm".format(distance))
    oled.text(distance, 0, 10)
```

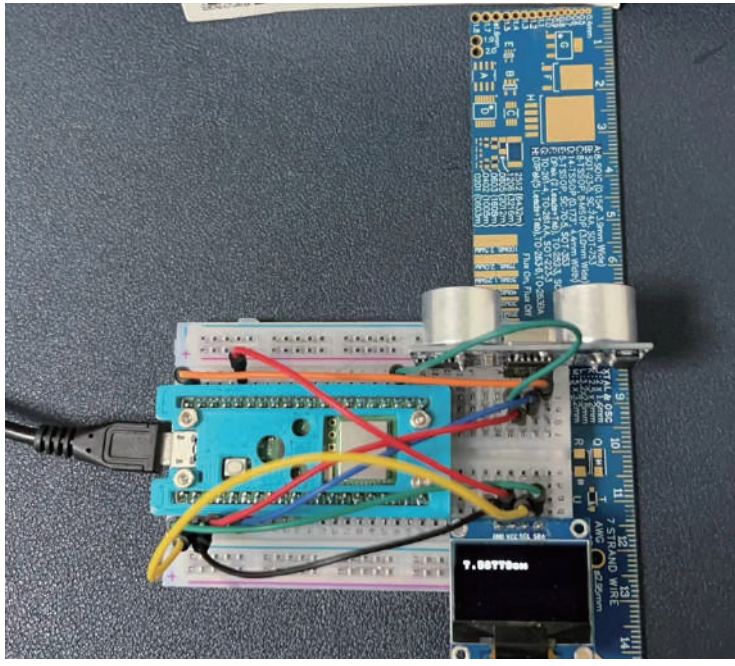
Here, you may need to convert the distance to string type by using str() method, and we can modify the while loop as following figure:

```
while True:
    distance = ultra()
    print("The distance from object is: {} cm".format(distance))
    oled.text(str(distance)+" cm", 0, 10)
    oled.show()
    utime.sleep(1)
    oled.fill(0)
```

1. oled.text("strings", x-positon, y-position)
str(distance) means to change the float type to string type so that we can put it into the text() method, and 0 is x position, 10 is y position.

2. Do remember let oled execute show() method to display the result on screen.

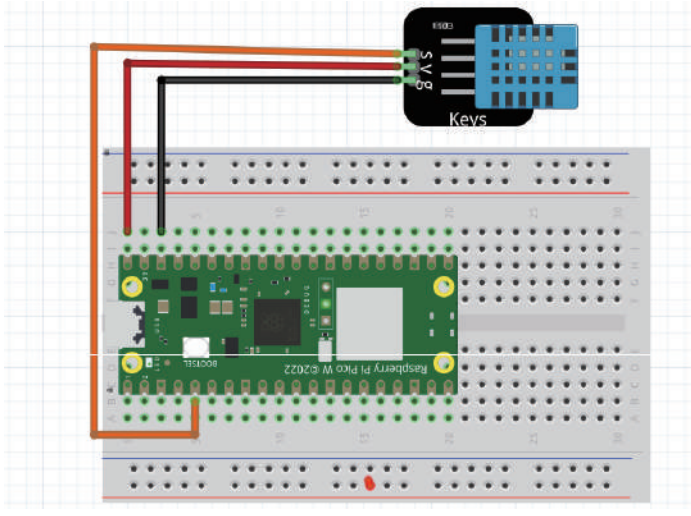
3. After a while, we need to clear screen so that we can see the result clearly.



DEMO 5 MEASURE TEMPERATURE AND HUMIDITY, SHOW THE DATA ON WEB

1. WIRING

Connect the DATA (S) Pin from DHT11 temperature and humidity sensor to GPIO Pin 2 (GP2) on Raspberry Pi Pico.
Connect the GND(g)Pin from DHT11 temperature and humidity sensor to GPIO Pin 2 (GP2) on Raspberry Pi Pico.
Connect the 3V3(V)Pin from DHT11 temperature and humidity sensor to 3V3(OUT) on Raspberry Pi Pico.



2. CODING

Open Thonny IDE and then create a new file called: dht.py and input following code:

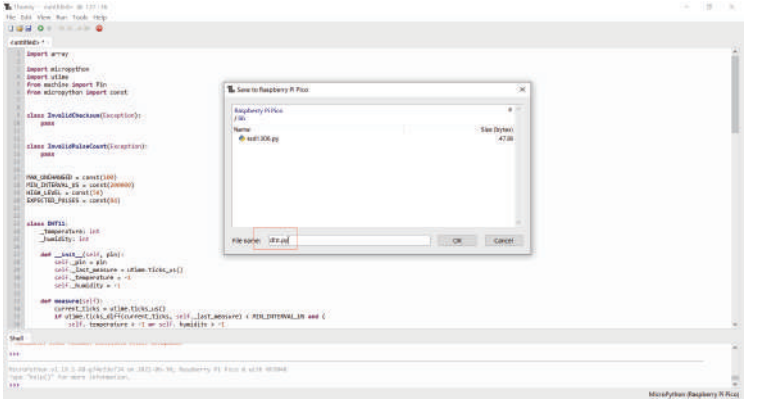
```
import array

import micropython
import utime
from machine import Pin
from micropython import const


class InvalidChecksum(Exception):
    pass


class InvalidPulseCount(Exception):
    pass


MAX_UNCHANGED = const(100)
MIN_INTERVAL_US = const(200000)
HIGH_LEVEL = const(50)
EXPECTED_PULSES = const(84)

class DHT11:
    _temperature: int
    _humidity: int

    def __init__(self, pin):
        self._pin = pin
        self._last_measure = utime.ticks_us()
        self._temperature = -1
        self._humidity = -1

    def measure(self):
        current_ticks = utime.ticks_us()
        if utime.ticks_diff(current_ticks, self._last_measure) < MIN_INTERVAL_US and (
```

```
        self._temperature > -1 or self._humidity > -1
    ):
        # Less than a second since last read, which is too soon according
        # to the datasheet
        return

    self._send_init_signal()
    pulses = self._capture_pulses()
    buffer = self._convert_pulses_to_buffer(pulses)
    self._verify_checksum(buffer)

    self._humidity = buffer[0] + buffer[1] / 10
    self._temperature = buffer[2] + buffer[3] / 10
    self._last_measure = utime.ticks_us()

@property
def humidity(self):
    self.measure()
    return self._humidity

@property
def temperature(self):
    self.measure()
    return self._temperature

def _send_init_signal(self):
    self._pin.init(Pin.OUT, Pin.PULL_DOWN)
    self._pin.value(1)
    utime.sleep_ms(50)
    self._pin.value(0)
    utime.sleep_ms(18)

@micropython.native
def _capture_pulses(self):
    pin = self._pin
    pin.init(Pin.IN, Pin.PULL_UP)

    val = 1
    idx = 0
```

```
    transitions = bytearray(EXPECTED_PULSES)
    unchanged = 0
    timestamp = utime.ticks_us()

    while unchanged < MAX_UNCHANGED:
      if val != pin.value():
        if idx >= EXPECTED_PULSES:
          raise InvalidPulseCount(
            "Got more than {} pulses".format(EXPECTED_PULSES)
          )
        now = utime.ticks_us()
        transitions[idx] = now - timestamp
        timestamp = now
        idx += 1

        val = 1 - val
        unchanged = 0
      else:
        unchanged += 1
    pin.init(Pin.OUT, Pin.PULL_DOWN)
    if idx != EXPECTED_PULSES:
      raise InvalidPulseCount(
        "Expected {} but got {} pulses".format(EXPECTED_PULSES, idx)
      )
    return transitions[4:]

  def _convert_pulses_to_buffer(self, pulses):
    """Convert a list of 80 pulses into a 5 byte buffer
    The resulting 5 bytes in the buffer will be:
      0: Integral relative humidity data
      1: Decimal relative humidity data
      2: Integral temperature data
      3: Decimal temperature data
      4: Checksum
    """
# Convert the pulses to 40 bits
    binary = 0
    for idx in range(0, len(pulses), 2):
      binary = binary << 1 | int(pulses[idx] > HIGH_LEVEL)
```

```
    # Split into 5 bytes
    buffer = array.array("B")
    for shift in range(4, -1, -1):
        buffer.append(binary >> shift * 8 & 0xFF)
    return buffer

def _verify_checksum(self, buffer):
    # Calculate checksum
    checksum = 0
    for buf in buffer[0:4]:
        checksum += buf
    if checksum & 0xFF != buffer[4]:
        raise InvalidChecksum()
```

save it.



2. Create a new file and input following code:
NOTE: here we import DHT11 class from dht library which we have just created it.

1. import DHT11 class and some other libraries.

```
from machine import Pin
from dht import DHT11
import utime
```

2. Define the DHT data pin, we need setting the pin as input pin and pull down the level.

```
DHTPin = Pin(3,Pin.OUT, Pin.PULL_DOWN)
```

3. Read the sensor's data and print it out.

```
while True:
    utime.sleep(1)
    sensor = DHT11(DHTPin)
    sensor.measure()

    t = (sensor.temperature)
    h = (sensor.humidity)

    print("Temperature: " + str(t))
    print("Humidity: " + str(h))
```

save to main.py and execute it by clicking the run button.

Next, we are going to put the result to a web page, and make Raspberry Pi Pico as a web server. when we open a browser and access Pico web server will refresh the page and show the temperature and humidity value. with a little MicroPython code, and some HTML, we can serve basic, static web pages from a Pico W.

here are two parts to this project. The HTML and the MicroPython code. The HTML is what our browser will see, and MicroPython acts as the means to serve the code.

Finally, we will serve the content to the world by learning how to forward external requests to our Raspberry Pi Pico W.
Open Thonny IDE and create a new file.

1. import libraries and make sure network library and socket library imported properly, once import those module, you can click run button to check if the library has been imported properly, if not, you may check your MicroPython' s version number, you may need to www.micropython.org and download the latest version.

```
from machine import Pin
from dht import DHT11
import network
import socket
import utime
import machine
```

2. Define the variables for the network connection.

```
ssid = 'your_2.4GHz_wifi-ssid'
password = 'your_password'
```

3. Create a function to connect the Raspberry Pi Pico to wireless access point.

```
def connect():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, password)
    while wlan.isconnected() == False:
        print("waiting for connections…")
        utime.sleep(1)

    ip = wlan.ifconfig()[0]
    print(f'Connect to IP: {ip}')
    return ip
```

4. Create a function to open the socket, so that client can access to Raspberry Pi Pico and retrieve the html page.

```
def open_socket(ip):
    addr = (ip, 80)
    connection = socket.socket()
    connection.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    connection.bind(addr)
    connection.listen(1)
    print(f'Binding the port 80 to {ip}')
    return connection
```

5. Create a function to hold web page.

```
def webpage(reading):
    html = f"""
            <!DOCTYPE html>
            <html>
            <head>
            <title> Pico W Temperature and humidity station</title>
            <meta http-equiv="refresh" content="10">
            </head>
            <body>
            <p>{reading}</p>
            </body>
            </html>
            """
    return str(html)
```

6. Create a function to serve the webpage, listen to the port and waiting for client coming.

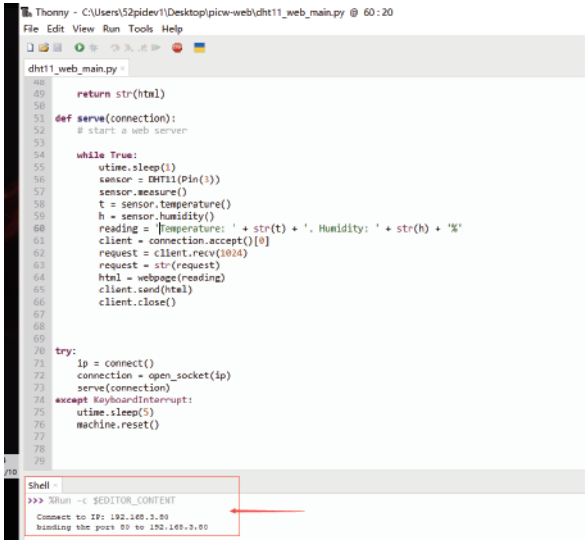
```
def serve(connection):
    # Start a web server
    while True:
        utime.sleep(1)
        sensor = DHT11(Pin(3))
        sensor.measure()
        t = sensor.temperature()
        h = sensor.humidity()
        reading = 'Temperature: ' + str(t) + '. Humidity: ' + str(h) + '%'
        client = connection.accept()[0]
```

```
    request = client.recv[1024]
    request = str(request)
    html = webpage(reading)
    client.send(html)
    client.close()
```

7. Finally, we are trying to connect the network and get the IP address, and binding to socket service and serve a web on the connection.

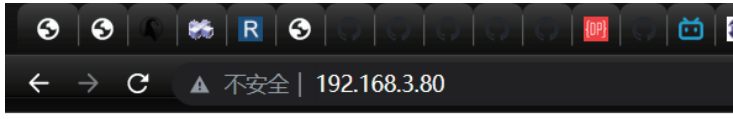
```
try:
    ip = connect()
    connection = open_socket(ip)
    serve(connection)
except KeyboardInterrupt:
    utime.sleep(3)
    machine.reset()
```

save it and run it in Thonny IDE:



3. TESTING

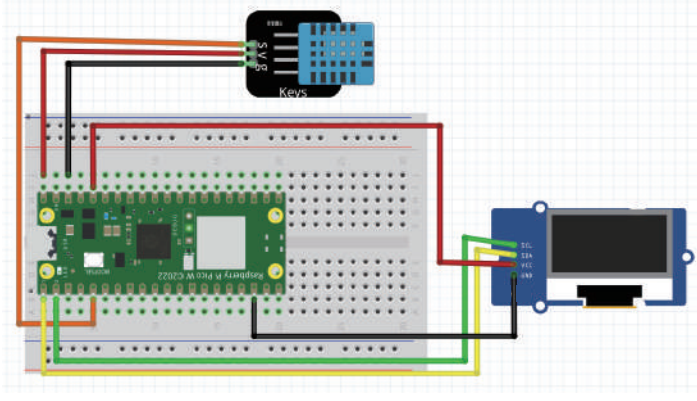
Open a browser and typing the IP address, you will see the temperature and humidity information on the web page.



DEMO 6 MEASURE TEMPERATURE AND HUMIDITY, SHOW THE DATA ON 0.96-INCH OLED

1. WIRING

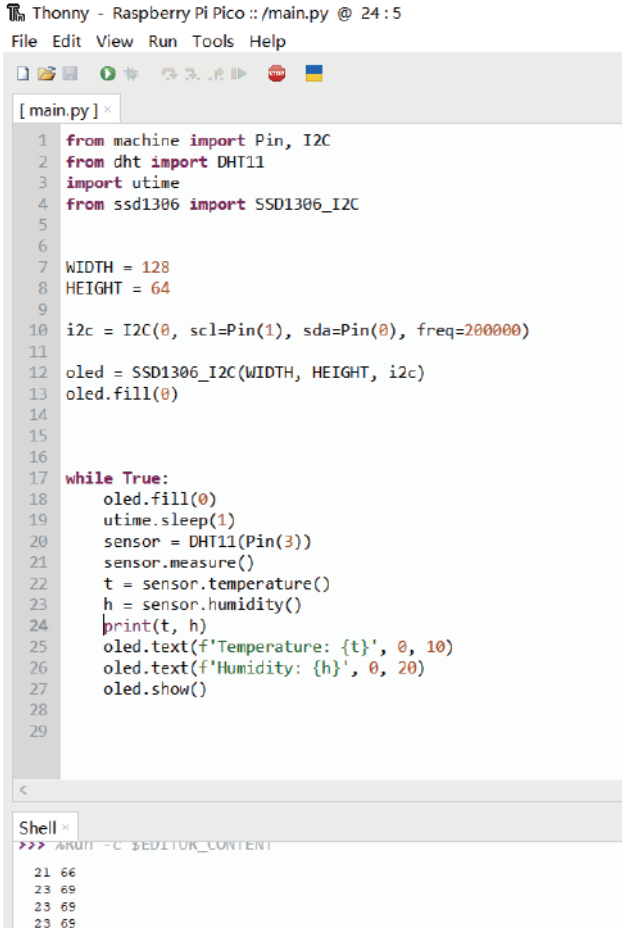
Connect the SCL Pin from OLED display module to GPIO Pin 1 (GP1) on Raspberry Pi Pico W.
Connect the SDA Pin from OLED display module to GPIO Pin 0 (GP0) on Raspberry Pi Pico W.
Connect the VCC Pin from OLED display module to 3V3(OUT) on Raspberry Pi Pico W.
Connect the GND Pin from OLED display module to GND on Raspberry Pi Pico W.



2. CODING

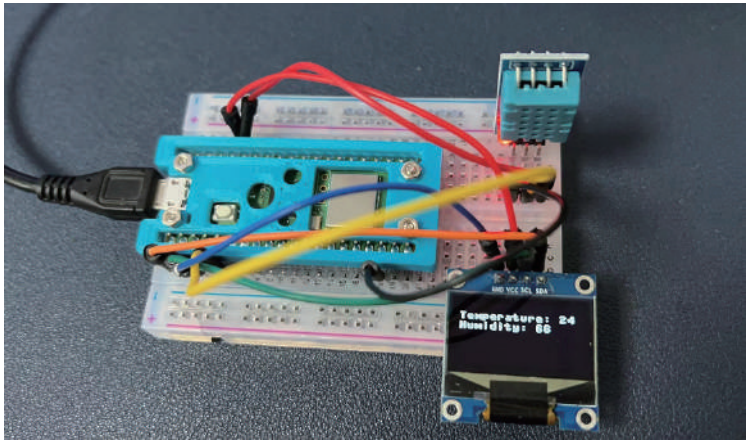
Make sure you have already create lib folder on Raspberry Pi Pico and uploaded the ssd1306.py file into the folder, please refer to Demo 4.

Open Thonny IDE and the Raspberry Pi Pico W has been plugged into the USB port. Create a new file and typing following code:



3. TESTING

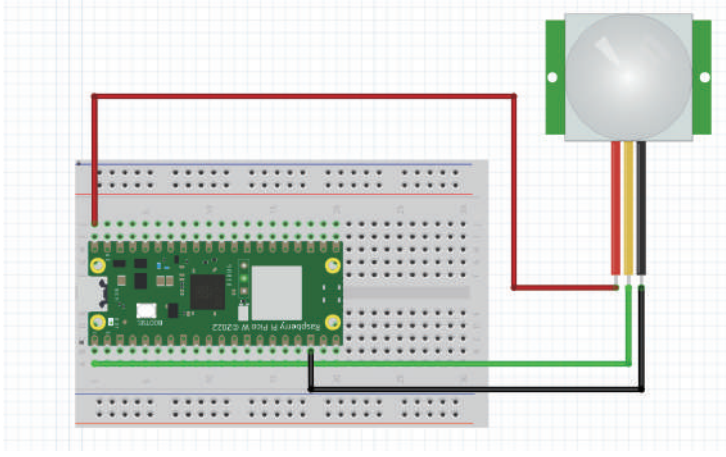
You can see the temperature and humidity information displayed on OLED screen.



DEMO 7 MOTION DETECTION AND SHOW THE RESULT ON WEB

1. WIRING

Connect VCC Pin from PIR sensor to on 5V Pin on Raspberry Pi Pico W.
Connect GND Pin from PIR sensor to GND on Raspberry Pi Pico W.
Connect OUT Pin from PIR sensor to GPIO 0 (GP0) on Raspberry Pi Pico W.



2. CODING

Open Thonny IDE, and connect the Raspberry Pi Pico W to USB Port on PC by using MicroUSB cable.
Input following code to detect the motion and send to web page.

Thonny - Raspberry Pi Pico :: /main.py @ 30 : 41

File Edit View Run Tools Help

[main.py]

```
from machine import Pin
import network
import socket
import utime
import machine


PIR = Pin(15, Pin.IN, Pin.PULL_UP)

SSID = 'H[illegible]'
PASS = '[illegible]'

def connect():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(SSID, PASS)
    while wlan.isconnected() == False:
        print("waiting for connection...")
        utime.sleep(1)
    ip = wlan.ifconfig()[0]
    print(f'Connect to IP: {ip}')
    return ip

def open_socket(ip):
    addr = (ip, 80)
    connection = socket.socket()
    connection.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    connection.bind(addr)
    connection.listen(1)
    print(f'Binding to 80 port on {ip}')
    return connection
```

Thonny - Raspberry Pi Pico :: /main.py @ 30 : 41

File Edit View Run Tools Help

[main.py]

```

def open_socket(ip):
    addr = (ip, 80)
    connection = socket.socket()
    connection.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    connection.bind(addr)
    connection.listen(1)
    print(f'Binding to 80 port on {ip}')
    return connection

def webpage(reading):
    html = f"""
            <!DOCTYPE html>
            <html>
            <head>
            <title> Pico W web motion detect system</title>
            <meta http-equiv="refresh" content="10">
            </head>
            <body>
            <p>{reading}</p>
            </body>
            </html>
            """
    return str(html)

def serve(connection):
    while True:
        if PIR.value() == 0:
            reading = "There is no motion detected...."
        else:
            reading = 'Alarm! Motion detected, please check the fence and call 911...'
```

Thonny - Raspberry Pi Pico :: /main.py @ 42 : 29

File Edit View Run Tools Help

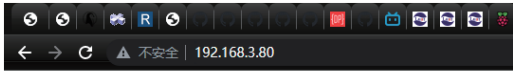
[main.py]

```
            </body>
            </html>
            """
    return str(html)

def serve(connection):
    while True:
        if PIR.value() == 0:
            reading = "There is no motion detected...."
        else:
            reading = 'Alarm! Motion detected, please check the fence and call 911...'

        client = connection.accept()[0]
        request = client.recv(1024)
        request = str(request)
        html = webpage(reading)
        client.send(html)
        client.close()


try:
    ip = connect()
    connection = open_socket(ip)
    serve(connection)
except KeyboardInterrupt:
    utime.sleep(3)
    machine.reset()
```

there are almost the same with previous demonstration. you can refer to it and modify the reading and sending the properly information to webpage.

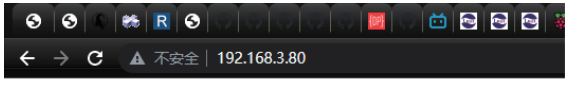
3. TESTING

Open a browser and typing the IP address:



There is no motion detected....

wave your hand before the PIR sensor and you will get this:



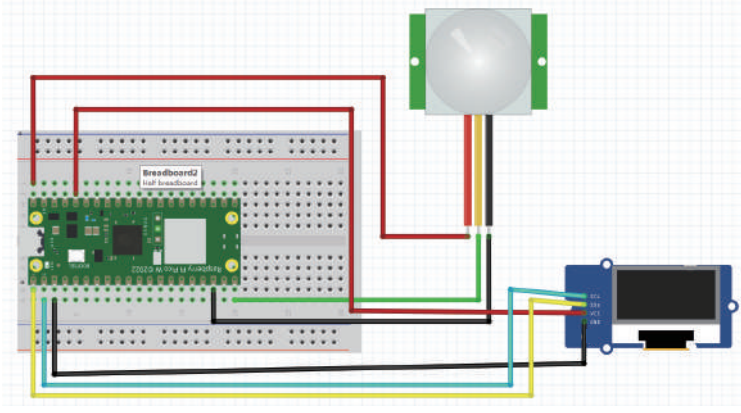
Alarm! Motion detected, please check the fence and call 911...

DEMO 8 MOTION DETECTION AND SHOW THE RESULT ON 0.96-INCH OLED

1. WIRING

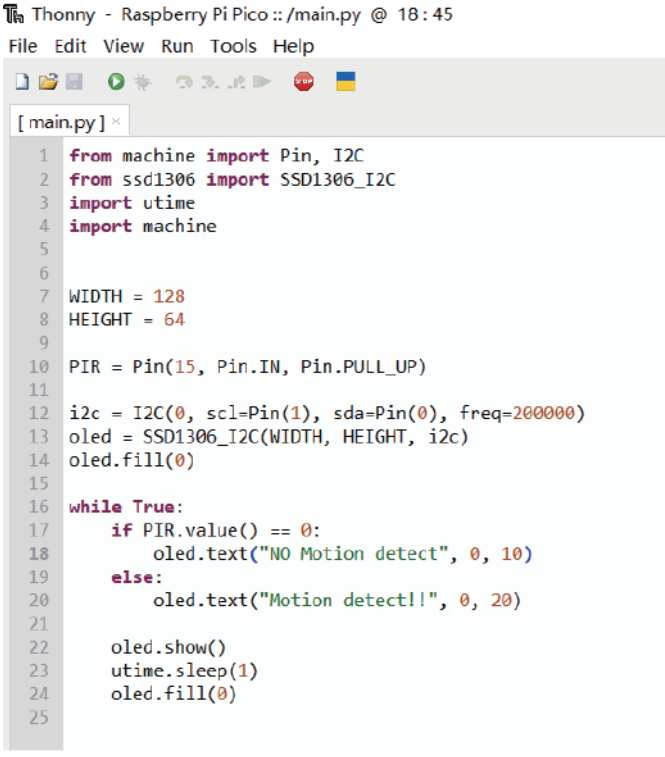
Keep the PIR sensor's connection.

Connect the SDA Pin from OLED screen to GPIO Pin 0 (GP0) to Raspberry Pi Pico W.
Connect the SCL Pin from OLED screen to GPIO Pin 1 (GP1) to Raspberry Pi Pico W.
Connect the VCC Pin from OLED screen to 3V3(OUT) to Raspberry Pi Pico W.
Connect the GND Pin from OLED screen to GND to Raspberry Pi Pico W.



2. CODING

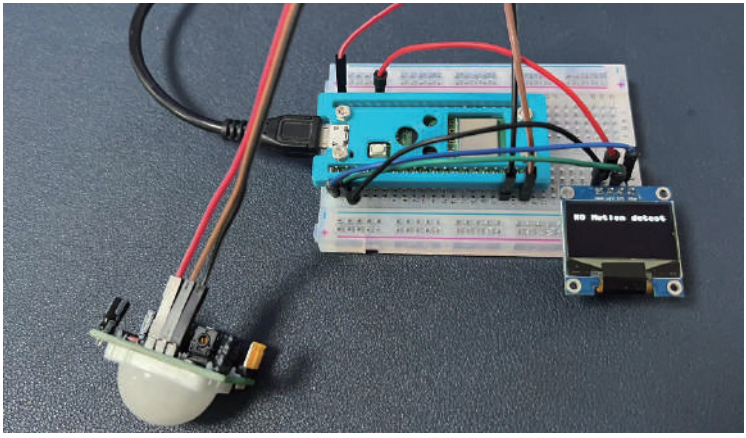
Open Thonny IDE and input following code and save it to main.py on Raspberry Pi Pico W.

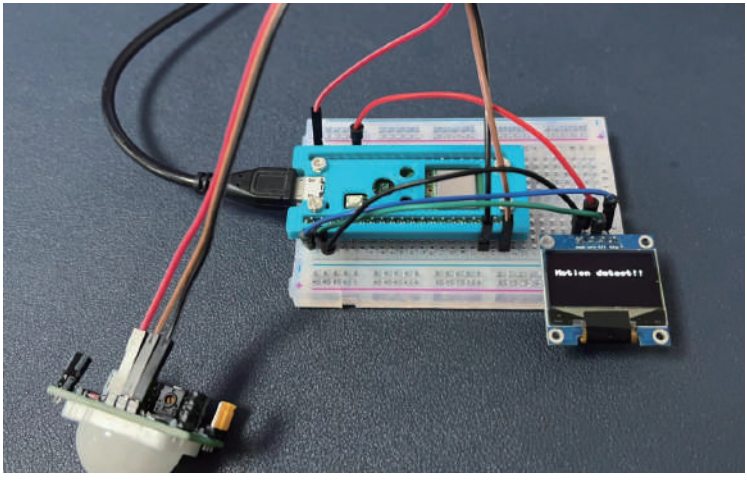



```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import utime
import machine


WIDTH = 128
HEIGHT = 64

PIR = Pin(15, Pin.IN, Pin.PULL_UP)

i2c = I2C(0, scl=Pin(1), sda=Pin(0), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
oled.fill(0)

while True:
    if PIR.value() == 0:
        oled.text("NO Motion detect", 0, 10)
    else:
        oled.text("Motion detect!!", 0, 20)

    oled.show()
    utime.sleep(1)
    oled.fill(0)

```

3. TESTING

Wave your hand before PIR sensor:



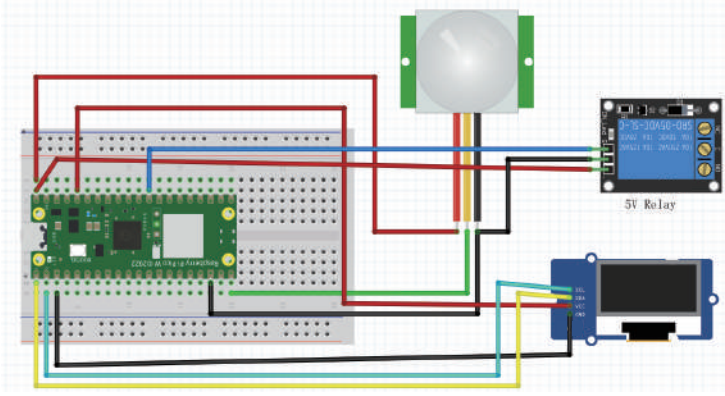


and you can modify the code to build more funny project.

DEMO 9 MOTION DETECTION AND TRIGGERING SINGLE CHANNEL RELAY

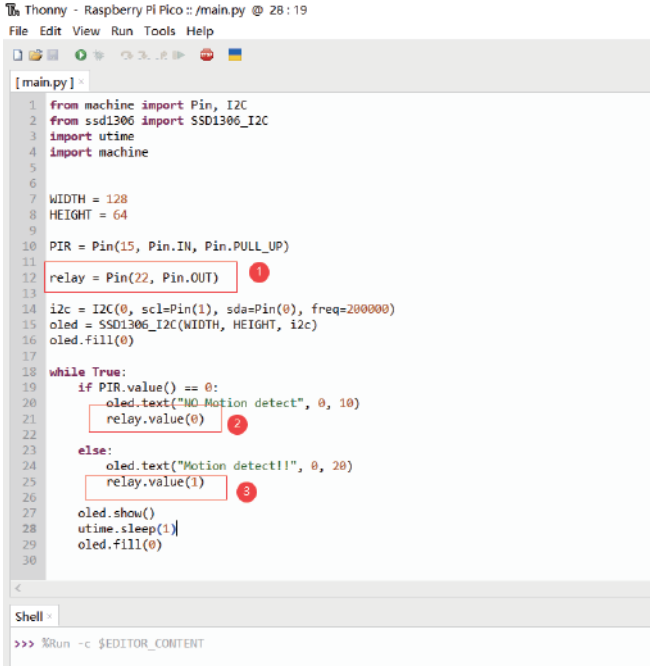
1. WIRING

Keep the previous demo's circuit, and adding 1 channel relay module to the circuit.
Connect the 5V Pin from Relay module to 5V Pin on Raspberry Pi Pico W.
Connect the GND Pin from Relay module to GND Pin on Raspberry Pi Pico W.
Connect the S Pin from Relay module to GPIO Pin 22 (GP22) on Raspberry Pi Pico W.



2. CODING

Modify previous demo's code as following:



3. TESTING

Wave your hand before the PIR sensor, the relay will be turn on and turn off. once you connect the device such as sound alarm device or just connect to the electrical fence, your own motion detect system is ready to go.

DEMO 10 USAGE OF UPS FOR PICO WH

WHAT IS UPS PICO LITE?

This is a portable power supply for the Raspberry Pi Pico and Pico W.

Which supports the use of a single 18650 lithium battery for power supply, which can free your Pico from the shackles of cables and make it more convenient for you to DIY Pico projects.

The remain power of battery can be read from the 4 LED indicators. You can also read battery voltage and current information by connecting the USB-to-TTL cable to the serial port pins of the UPS.

The charging prompt during charging is very intuitive, and when the battery is fully charged, it will stop charging, and the 'Standby indicator' on the panel will light up.

All GPIOs is leading out and providing clear silkscreens which are very convenient for Pico experiments. It is plug-and-play module.

NOTE:

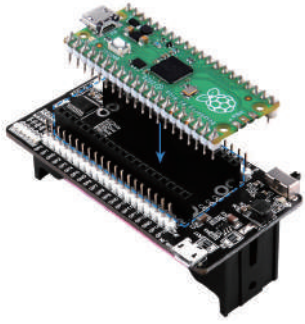
This device is a power supply device, please pay attention to the positive and negative poles of the battery. Reverse battery connection may damage your UPS device or Pico development board. Please check the circuit carefully before turning on the power switch to avoid equipment damage caused by short circuit.
Supports 18650 lithium battery Only!

It is normal that the current consumption can still be read when the PICO is not connected and the power switch is turned on.

After all, the board also consumes a small amount of electricity, which can be ignored. MicroUSB port on UPS which marked 'USB_IN' is only for charging, DO NOT Connect it to USB Port on laptop or PC, or it may damage the USB Port!

HOW TO CONNECT THE PICO W WITH UPS PICO LITE?

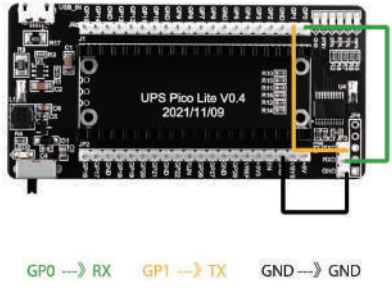
Just connect as following figure.



NOTE: Do not mistake the positive and negative poles, otherwise it will cause equipment damage or fire.

HOW TO READ VOLTAGE AND CURRENT FROM UPS VIA PICO W?

Connect **GP0** to **RX pin**, **GP1** to **TX pin** on UPS's JP3 Pin header.



Open a new file and paste following code and press run button on the Thonny IDE.

```
from machine import UART, Pin
import time

# create an instance. Connect GP0 to UPS's RX pin, GP1 to UPS's TX pin

uart1 = UART(0, baudrate=115200, tx=Pin(0), rx=Pin(1))

# loop
while True:
    try:
        data = uart1.readline()   # read data from UPS.
        if data != None:
            data = data.decode('utf-8')
            data = data.rstrip('\n')
            #print(data)

            print("Battery Voltage:{}mV".format(data.split('|')[0]))
            print("Current of Charging In:{}mA
".format(data.split('|')[1]))
            batt_cur = float(data.split('|')[2]) / 10.0
            print("Consuming Current:{}mA".format(batt_cur))

            sign = data.split('|')[2]
            print("sign: {}".format(sign))
            if float(sign) < 0.0:
                print("Battery is charging....")
            else:
                print("Power is consuming...")

        time.sleep(2)
        print("-"*38)
    except ValueError:
        pass
    except IndexError:
        pass
```

Thonny - Raspberry Pi Pico :: /main.py @ 19 : 43

File Edit View Run Tools Help

[main.py]

```
from machine import UART, Pin
import time

# create an instance. Connect GP0 to UPS's RX pin, GP1 to UPS's TX pin
uart1 = UART(0, baudrate=115200, tx=Pin(0), rx=Pin(1))

# loop
while True:
    try:
        data = uart1.readline()   # read data from UPS.
        if data != None:
            data = data.decode('utf-8')
            data = data.rstrip('\n')
#             print(data)

            print("Battery Voltage:{} mV".format(data.split('|')[0]))
            print("Current of Charging In:{} mA".format(data.split('|')[1]))
            batt_cur = float(data.split('|')[2]) / 10.0
            print("Consuming Current:{} mA".format(batt_cur))

            sign = data.split('|')[2]
            print("sign: {}".format(sign))
            if float(sign) < 0.0:
                print("Battery is charging....")
            else:
                print("Power is consuming...")

        time.sleep(2)
        print("-"*38)
    except ValueError:
        pass
    except IndexError:
        pass
```

RESULT

Thonny - Raspberry Pi Pico :: /main.py @ 19 : 43

File Edit View Run Tools Help

[main.py]

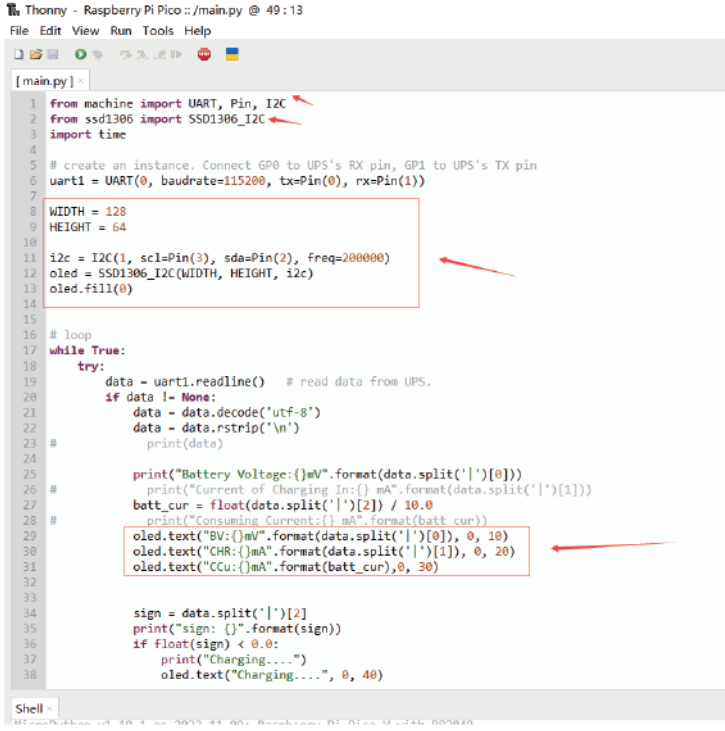
```
1 from machine import UART, Pin
2 import time
```

Shell

```
Battery Voltage:3609.936000 mV
Current of Charging In:1.861818 mA
Consuming Current:506.0326 mA
sign: 5060.326000
Power is consuming...
---------------------------------------
Battery Voltage:3607.268000 mV
Current of Charging In:1.861822 mA
Consuming Current:505.3024 mA
sign: 5053.024000
Power is consuming...
---------------------------------------
Battery Voltage:3607.268000 mV
Current of Charging In:1.861822 mA
Consuming Current:505.6585 mA
sign: 5056.585000
Power is consuming...
---------------------------------------
Battery Voltage:3607.268000 mV
Current of Charging In:1.861822 mA
Consuming Current:505.6585 mA
sign: 5056.585000
Power is consuming...
---------------------------------------
Battery Voltage:3609.936000 mV
Current of Charging In:1.861818 mA
Consuming Current:505.6762 mA
sign: 5056.762000
Power is consuming...
---------------------------------------
Battery Voltage:3607.268000 mV
Current of Charging In:1.861822 mA
Consuming Current:506.0146 mA
sign: 5060.146000
Power is consuming...
---------------------------------------
Battery Voltage:3609.936000 mV
Current of Charging In:1.861818 mA
Consuming Current:506.3889 mA
sign: 5063.889000
Power is consuming...
---------------------------------------
Battery Voltage:3609.936000 mV
Current of Charging In:1.861818 mA
Consuming Current:506.3889 mA
sign: 5063.889000
Power is consuming...
```

HOW TO SHOW THE BATTERY'S INFORMATION ON OLED SCREEN?

Just modify the code:




```
from machine import UART, Pin, I2C
from ssd1306 import SSD1306_I2C
import time

# create an instance. Connect GP0 to UPS's RX pin, GP1 to UPS's TX pin
uart1 = UART(0, baudrate=115200, tx=Pin(0), rx=Pin(1))

WIDTH = 128
HEIGHT = 64

i2c = I2C(1, scl=Pin(3), sda=Pin(2), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
oled.fill(0)


# loop
while True:
    try:
        data = uart1.readline()   # read data from UPS.
        if data != None:
            data = data.decode('utf-8')
            data = data.rstrip('\n')
#             print(data)

            print("Battery Voltage:{}mV".format(data.split('|')[0]))
#             print("Current of Charging In:{} mA".format(data.split('|')[1]))
            batt_cur = float(data.split('|')[2]) / 10.0
#             print("Consuming Current:{} mA".format(batt_cur))
            oled.text("BV:{}mV".format(data.split('|')[0]), 0, 10)
            oled.text("CHR:{}mA".format(data.split('|')[1]), 0, 20)
            oled.text("CCu:{}mA".format(batt_cur),0, 30)


            sign = data.split('|')[2]
            print("sign: {}".format(sign))
            if float(sign) < 0.0:
                print("Charging....")
                oled.text("Charging....", 0, 40)
```

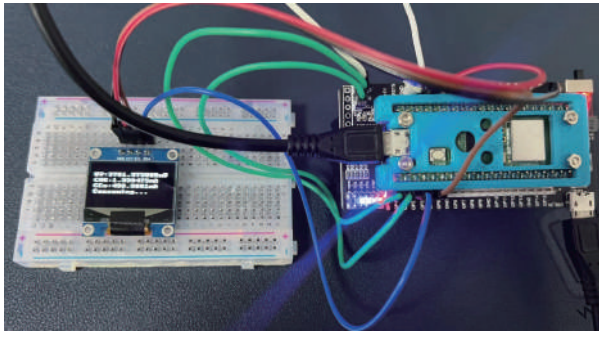
Thonny - Raspberry Pi Pico :: /main.py @ 49 : 13

File Edit View Run Tools Help

[main.py]

```
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
oled.fill(0)


# loop
while True:
    try:
        data = uart1.readline()   # read data from UPS.
        if data != None:
            data = data.decode('utf-8')
            data = data.rstrip('\n')
#             print(data)

            print("Battery Voltage:{}mV".format(data.split('|')[0]))
#             print("Current of Charging In:{} mA".format(data.split('|')[1]))
            batt_cur = float(data.split('|')[2]) / 10.0
#             print("Consuming Current:{} mA".format(batt_cur))
            oled.text("BV:{}mV".format(data.split('|')[0]), 0, 10)
            oled.text("CHR:{}mA".format(data.split('|')[1]), 0, 20)
            oled.text("CCu:{}mA".format(batt_cur),0, 30)


            sign = data.split('|')[2]
            print("sign: {}".format(sign))
            if float(sign) < 0.0:
                print("Charging....")
                oled.text("Charging....", 0, 40)
            else:
                print("Consuming...")
                oled.text("Consuming...", 0, 40)
        oled.show()
        time.sleep(2)
        oled.fill(0)
        print("-"*30)
    except ValueError:
        pass
    except IndexError:
        pass
```

and save it to main.py. click run button.

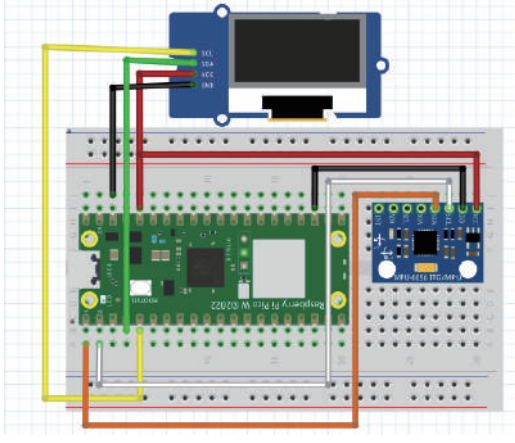


DEMO 11 BUILD A ROBOT FACE BY USING MPU6050 AND OLED DISPLAY

1. WIRING

Wiring of OLED display:
Connect SDA Pin from OLED display module to GPIO Pin 2 (GP2) on Raspberry Pi Pico W.
Connect SCL Pin from OLED display module to GPIO Pin 3 (GP3) on Raspberry Pi Pico W.
Connect VCC Pin from OLED display module to 3V3 (OUT) on Raspberry Pi Pico W.
Connect GND Pin from OLED display module to GND on Raspberry Pi Pico W.

Wiring of MPU6050 IMU:
Connect SDA Pin from MPU6050 module to GPIO Pin 0 (GP0) on Raspberry Pi Pico W.
Connect SCL Pin from MPU6050 module to GPIO Pin 1 (GP1) on Raspberry Pi Pico W.
Connect VCC Pin from MPU6050 module to 3V3 (OUT) on Raspberry Pi Pico W.
Connect GND Pin from MPU6050 module to GND on Raspberry Pi Pico W.

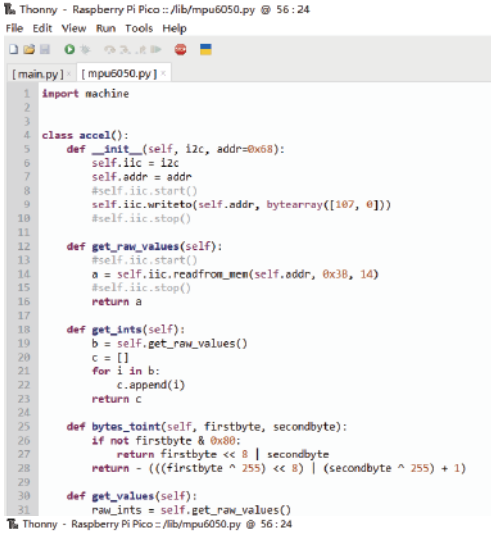


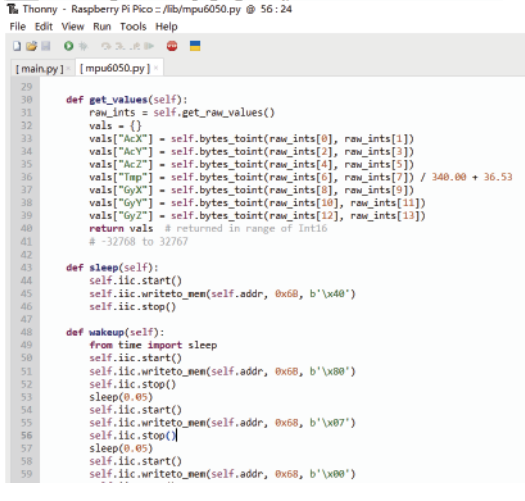
2. CODING

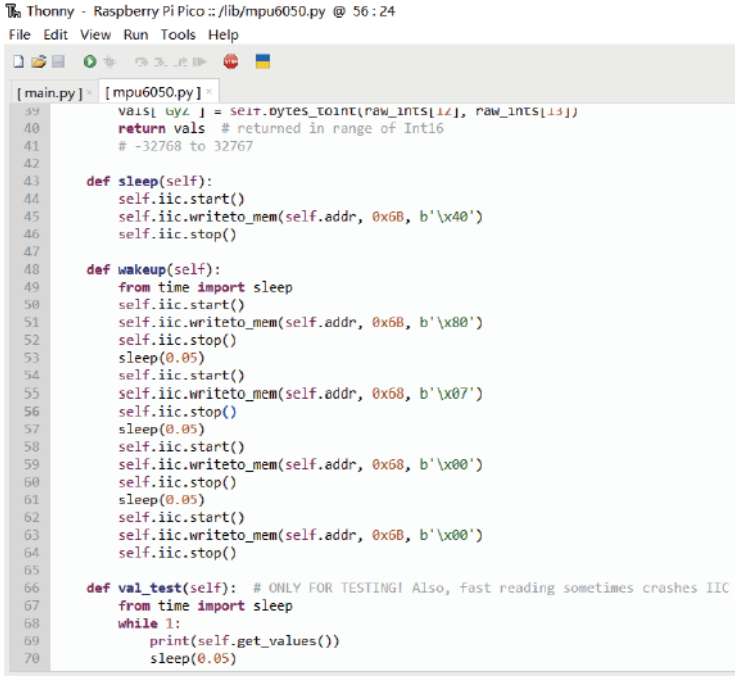
Open Thonny IDE and create a new file.
Input following code and save it to Raspberry Pi Pico W's lib folder and named it "mpu6050.py"

```
import machine


class accel():
    def __init__(self, i2c, addr=0x68):
        self.iic = i2c
        self.addr = addr
        #self.iic.start()
        self.iic.writeto(self.addr, bytearray([107, 0]))
        #self.iic.stop()

    def get_raw_values(self):
        #self.iic.start()
        a = self.iic.readfrom_mem(self.addr, 0x3B, 14)
        #self.iic.stop()
        return a

    def get_ints(self):
        b = self.get_raw_values()
        c = []
        for i in b:
            c.append(i)
        return c

    def bytes_toint(self, firstbyte, secondbyte):
        if not firstbyte & 0x80:
            return firstbyte << 8 | secondbyte
        return - (((firstbyte ^ 255) << 8) | (secondbyte ^ 255) + 1)

    def get_values(self):
        raw_ints = self.get_raw_values()
        vals = {}
        vals["AcX"] = self.bytes_toint(raw_ints[0], raw_ints[1])
```

```
    vals["AcY"] = self.bytes_toint(raw_ints[2], raw_ints[3])
    vals["AcZ"] = self.bytes_toint(raw_ints[4], raw_ints[5])
    vals["Tmp"] = self.bytes_toint(raw_ints[6], raw_ints[7]) / 340.00 + 36.53
    vals["GyX"] = self.bytes_toint(raw_ints[8], raw_ints[9])
    vals["GyY"] = self.bytes_toint(raw_ints[10], raw_ints[11])
    vals["GyZ"] = self.bytes_toint(raw_ints[12], raw_ints[13])
    return vals  # returned in range of Int16
    # -32768 to 32767

def sleep(self):
    self.iic.start()
    self.iic.writeto_mem(self.addr, 0x6B, b'\x40')
    self.iic.stop()

def wakeup(self):
    from time import sleep
    self.iic.start()
    self.iic.writeto_mem(self.addr, 0x6B, b'\x80')
    self.iic.stop()
    sleep(0.05)
    self.iic.start()
    self.iic.writeto_mem(self.addr, 0x68, b'\x07')
    self.iic.stop()
    sleep(0.05)
    self.iic.start()
    self.iic.writeto_mem(self.addr, 0x68, b'\x00')
    self.iic.stop()
    sleep(0.05)
    self.iic.start()
    self.iic.writeto_mem(self.addr, 0x6B, b'\x00')
    self.iic.stop()

def val_test(self):  # ONLY FOR TESTING! Also, fast reading sometimes crashes IIC
    from time import sleep
    while 1:
        print(self.get_values())
        sleep(0.05)
```


Thonny - Raspberry Pi Pico :: /lib/mpu6050.py @ 56 : 24
File Edit View Run Tools Help
[main.py]
[mpu6050.py]
import machine
class accel():
def __init__(self, i2c, addr=0x68):
self.iic = i2c
self.addr = addr
#self.iic.start()
self.iic.writeto(self.addr, bytearray([107, 0]))
#self.iic.stop()
def get_raw_values(self):
#self.iic.start()
a = self.iic.readfrom_mem(self.addr, 0x3B, 14)
#self.iic.stop()
return a
def get_ints(self):
b = self.get_raw_values()
c = []
for i in b:
c.append(i)
return c
def bytes_toint(self, firstbyte, secondbyte):
if not firstbyte & 0x80:
return firstbyte << 8 | secondbyte
return - (((firstbyte ^ 255) << 8) | (secondbyte ^ 255) + 1)
def get_values(self):
raw_ints = self.get_raw_values()


Thonny - Raspberry Pi Pico :: /lib/mpu6050.py @ 56 : 24
File Edit View Run Tools Help
[main.py]
[mpu6050.py]
def get_values(self):
raw_ints = self.get_raw_values()
vals = {}
vals["AcX"] = self.bytes_toint(raw_ints[0], raw_ints[1])
vals["AcY"] = self.bytes_toint(raw_ints[2], raw_ints[3])
vals["AcZ"] = self.bytes_toint(raw_ints[4], raw_ints[5])
vals["Tmp"] = self.bytes_toint(raw_ints[6], raw_ints[7]) / 340.00 + 36.53
vals["GyX"] = self.bytes_toint(raw_ints[8], raw_ints[9])
vals["GyY"] = self.bytes_toint(raw_ints[10], raw_ints[11])
vals["GyZ"] = self.bytes_toint(raw_ints[12], raw_ints[13])
return vals # returned in range of Int16
# -32768 to 32767
def sleep(self):
self.iic.start()
self.iic.writeto_mem(self.addr, 0x68, b'\x40')
self.iic.stop()
def wakeup(self):
from time import sleep
self.iic.start()
self.iic.writeto_mem(self.addr, 0x68, b'\x00')
self.iic.stop()
sleep(0.05)
self.iic.start()
self.iic.writeto_mem(self.addr, 0x68, b'\x07')
self.iic.stop()
sleep(0.05)
self.iic.start()
self.iic.writeto_mem(self.addr, 0x68, b'\x00')



once it done. create a new file and input following code.

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import mpu6050
import utime
from random import randint


WIDTH = 128
HEIGHT = 64
```

```
i2c_oled = I2C(1, sda=Pin(2), scl=Pin(3), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c_oled)
oled.fill(0)


i2c_mpu6050 = I2C(0, sda=Pin(0), scl=Pin(1), freq=200000)

imu = mpu6050.accel(i2c_mpu6050)

while True:
    data = imu.get_values()

    cx = WIDTH / 2
    cy = HEIGHT /2

    rec_x = int(cy - cy * data['AcY']/16384.0)
    rec_y = int(cx - cx * data['AcX']/16384.0  - 20)

    oled.fill_rect( rec_x, rec_y, 15, 15, 1)
    oled.fill_rect( rec_x+30, rec_y, 15, 15, 1)
    oled.show()
    utime.sleep(randint(1, 5) * 0.1)
    oled.fill(0)
    oled.fill_rect( rec_x, rec_y, 13, 13, 1)
    oled.fill_rect( rec_x+30, rec_y, 13, 13, 1)
    oled.show()
    utime.sleep(0.2)
    oled.fill(0)
```

Thonny - Raspberry Pi Pico :: /main.py @ 1 : 2

File Edit View Run Tools Help

[main.py] * [mpu6050.py]

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import mpu6050
import utime
from random import randint


WIDTH = 128
HEIGHT = 64

i2c_oled = I2C(1, sda=Pin(2), scl=Pin(3), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c_oled)
oled.fill(0)


i2c_mpu6050 = I2C(0, sda=Pin(0), scl=Pin(1), freq=200000)

imu = mpu6050.accel(i2c_mpu6050)

while True:
    data = imu.get_values()

    cx = WIDTH / 2
    cy = HEIGHT /2

    rec_x = int(cy - cy * data['AcY']/16384.0)
    rec_y = int(cx - cx * data['AcX']/16384.0  - 20)
```

Thonny - Raspberry Pi Pico :: /main.py @ 1 : 2

File Edit View Run Tools Help

[main.py] * [mpu6050.py]

```
HEIGHT = 64

i2c_oled = I2C(1, sda=Pin(2), scl=Pin(3), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c_oled)
oled.fill(0)


i2c_mpu6050 = I2C(0, sda=Pin(0), scl=Pin(1), freq=200000)

imu = mpu6050.accel(i2c_mpu6050)

while True:
    data = imu.get_values()

    cx = WIDTH / 2
    cy = HEIGHT /2

    rec_x = int(cy - cy * data['AcY']/16384.0)
    rec_y = int(cx - cx * data['AcX']/16384.0  - 20)

    oled.fill_rect( rec_x, rec_y, 15, 15, 1)
    oled.fill_rect( rec_x+30, rec_y, 15, 15, 1)
    oled.show()
    utime.sleep(randint(1, 5) * 0.1)
    oled.fill(0)
    oled.fill_rect( rec_x, rec_y, 13, 13, 1)
    oled.fill_rect( rec_x+30, rec_y, 13, 13, 1)
    oled.show()
    utime.sleep(0.2)
    oled.fill(0)
```

HOW IT WORKS?

1. Import the Pin class and I2C class from the machine library and then import the SSD1306_I2C class from the library, import framebuf library, and setting the variables of the OLED display.

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import mpu6050
import utime
from random import randint


WIDTH = 128
HEIGHT = 64
```

2. Create instance of the bus of I2C, named it i2c, and create an instance, named oled, after that, clear the screen with fill method.

```
i2c_oled = I2C(1, sda=Pin(2), scl=Pin(3), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c_oled)

# clear screen
oled.fill(0)
```

3. Create IMU instance by using mpu6050's accel class.

```
i2c_mpu6050 = I2C(0, sda=Pin(0), scl=Pin(1), freq=200000)

imu = mpu6050.accel(i2c_mpu6050)
```

4. Create a loop to read the data by using get_values() method, and it returns a dictionary type result. here we using AcX and AcY for this demo.

```
while True:
    data = imu.get_values()

    cx = WIDTH/2  # center of screen, x position
    cy = HEIGHT/2 # center of screen, y position
```

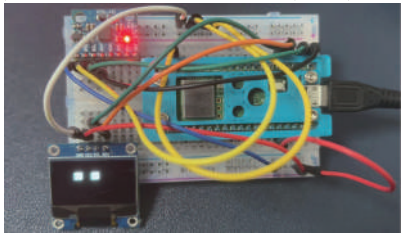
```
# x pos of robot left eye
rec_x = int(cy - cy * data['AcY']/16384.0)

# y pos of robot left eye
rec_y = int(cx - cx * data['AcX']/16384.0)
```

5. Draw a rectangle and fill it by using fill_rect() method, the arguments are : (x position, y position, length, width, colour=1)
for example: oled.fill_rect(10,10, 22, 43, 1) means draw a rectangle outline 10,10 to 22,43, colour=1.

```
oled.fill_rect(rec_x, rec_y, 15, 15, 1)    # draw left eye
oled.fill_rect(rec_x+30, rec_y, 15, 15, 1) # draw right eye

oled.show()  # put the rectangle on screen

utime.sleep(randint(1, 5) * 0.1)  # delay random time

oled.fill(0)  # clear screen

oled.fill_rect(rec_x, rec_y, 13, 13, 1)
oled.fill_rect(rec_x+30, rec_y, 13, 13, 1)

oled.show()
utime.sleep(randint(1,5) * 0.1)
oled.fill(0)
```

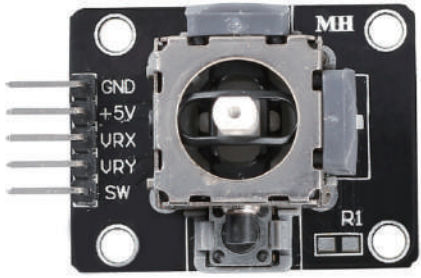
3. TESTING

when you tilt the breadboard, the IMU's data will change the position of the eye, create your own robot project will be very funny.



DEMO 12 PS2 JOYSTICK CONTROL THE EYE OF ROBOT

JOYSTICK MODULE

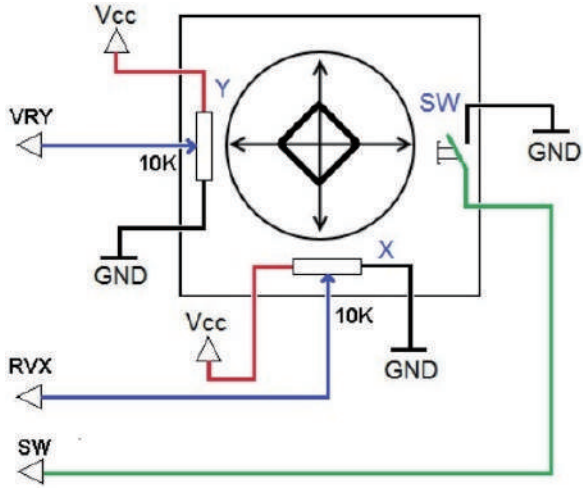


The basic idea of a joystick is to translate the movement of a stick into electronic information that a computer can process.

In order to communicate a full range of motion to the computer, a joystick needs to measure the stick' s position on two axes – the X-axis (left to right) and the Y-axis (up and down). Just as in basic geometry, the X-Y coordinates pinpoint the stick' s position exactly.

To determine the location of the stick, the joystick control system simply monitors the position of each shaft. The conventional analog joystick design does this with two potentiometers, or variable resistors.

The joystick also has a digital input that is actuated when the joystick is pressed down.



- The x-axis and y-axis values are analog values that vary from 0 to 65535.
- The Z-axis is a digital value with a status of 1 or 0.

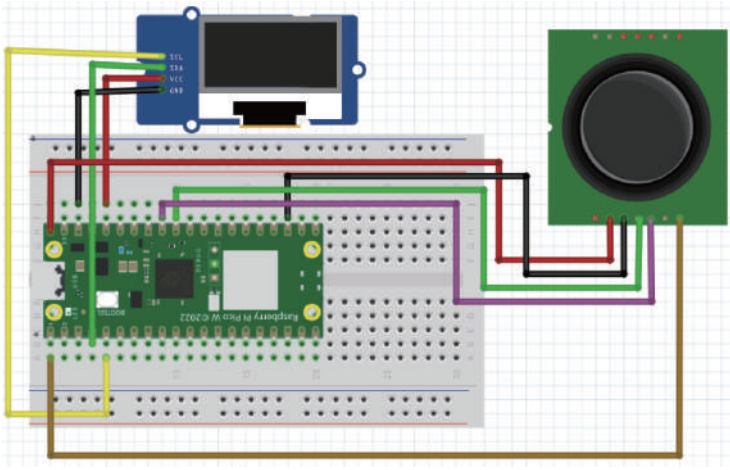
1. WIRING

Wiring of OLED display:

- Connect SDA Pin from OLED display module to GPIO Pin 2 (GP2) on Raspberry Pi Pico W.
- Connect SCL Pin from OLED display module to GPIO Pin 3 (GP3) on Raspberry Pi Pico W.
- Connect VCC Pin from OLED display module to 3V3 (OUT) on Raspberry Pi Pico W.
- Connect GND Pin from OLED display module to GND on Raspberry Pi Pico W.

Wiring PS2 Joystick:

- Connect 5V+ Pin from PS2 Joystick module to 5V Pin on Raspberry Pi Pico W.
- Connect GND Pin from PS2 Joystick module to GND Pin on Raspberry Pi Pico W.
- Connect VRX Pin from PS2 Joystick module to GP26 (ADC0) on Raspberry Pi Pico W.
- Connect VRY Pin from PS2 Joystick module to GP27 (ADC1) on Raspberry Pi Pico W.
- Connect SW Pin from PS2 Joystick module to GP0 on Raspberry Pi Pico W.



2. CODING

Open Thonny IDE and create a new file, input following code.

```
from machine import Pin, I2C, ADC
from ssd1306 import SSD1306_I2C
import utime
from random import randint


WIDTH = 128
HEIGHT = 64

i2c_oled = I2C(1, sda=Pin(2), scl=Pin(3), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c_oled)
oled.fill(0)

vrx = ADC(0)
vry = ADC(1)
sw = Pin(0, Pin.IN, Pin.PULL_UP)
```

```
# mapping the reading from 240-65535 to  0-128, screen width
def x_map(reading):
    return int((reading - 240) * (128 - 0) / (65535 - 240) + 0)

# mapping the reading from 240-65535 to  0-64, screen height
def y_map(reading):
    return int((reading - 240) * (64 - 0) / (65535 - 240) + 0)


while True:

    cx = WIDTH / 2
    cy = HEIGHT /2

    read_vrx = vrx.read_u16()
    read_vry = vry.read_u16()

    x_rate = x_map(read_vrx) - 88.0
    y_rate = y_map(read_vry) - 44.0
    print(x_rate, y_rate)
    if (cx + x_rate) <= 0:
        rec_x = 0
    else:
        rec_x = int(cx +  x_rate)
        if rec_x >=90:
            rec_x = 128 - 45

    if (cy + y_rate) <= 0:
        rec_y = 0
    else:
        rec_y = int(cy +  y_rate)
        if rec_y >=64:
            rec_y = 64 - 15 - 5
#
    if sw.value() == 0:
        utime.sleep(0.02)
        if sw.value() == 0:
            oled.fill(0)
            oled.text("What you want? ", 0 ,10)
```

```
        utime.sleep(0.5)

    oled.fill_rect( rec_x, rec_y, 15, 15, 1)
    oled.fill_rect( rec_x+30, rec_y, 15, 15, 1)
    oled.show()
    utime.sleep(randint(1, 3) * 0.1)
    oled.fill(0)
    oled.fill_rect( rec_x, rec_y, 13, 13, 1)
    oled.fill_rect( rec_x+30, rec_y, 13, 13, 1)
    oled.show()
    utime.sleep(randint(1, 3) * 0.1)
    oled.fill(0)
```

Thonny - Raspberry Pi Pico :: /main.py @ 63 : 17

File Edit View Run Tools Help

[main.py]

```
1  from machine import Pin, I2C, ADC
2  from ssd1306 import SSD1306_I2C
3  import utime
4  from random import randint
5
6
7  WIDTH = 128
8  HEIGHT = 64
9
10 i2c_oled = I2C(1, sda=Pin(2), scl=Pin(3), freq=200000)
11 oled = SSD1306_I2C(WIDTH, HEIGHT, i2c_oled)
12 oled.fill(0)
13
14 vrx = ADC(0)
15 vry = ADC(1)
16 sw = Pin(0, Pin.IN, Pin.PULL_UP)
17
18 # mapping the reading from 240-65535 to  0-128, screen width
19 def x_map(reading):
20     return int((reading - 240) * (128 - 0) / (65535 - 240) + 0)
21
22 # mapping the reading from 240-65535 to  0-64, screen height
23 def y_map(reading):
24     return int((reading - 240) * (64 - 0) / (65535 - 240) + 0)
25
26
27 while True:
28
29     cx = WIDTH / 2
30     cy = HEIGHT /2
```

Thonny - Raspberry Pi Pico :: /main.py @ 63 : 17

File Edit View Run Tools Help

[main.py]

```
    cx = WIDTH / 2
    cy = HEIGHT /2

    read_vrx = vrx.read_u16()
    read_vry = vry.read_u16()

    x_rate = x_map(read_vrx) - 88.0
    y_rate = y_map(read_vry) - 44.0
    print(x_rate, y_rate)
    if (cx + x_rate) <= 0:
        rec_x = 0
    else:
        rec_x = int(cx +  x_rate)
        if rec_x >=90:
            rec_x = 128 - 45

    if (cy + y_rate) <= 0:
        rec_y = 0
    else:
        rec_y = int(cy +  y_rate)
        if rec_y >=64:
            rec_y = 64 - 15 - 5
#
    if sw.value() == 0:
        utime.sleep(0.02)
        if sw.value() == 0:
            oled.fill(0)
            oled.text("What you want? ", 0 ,10)
            utime.sleep(0.5)

```

Thonny - Raspberry Pi Pico :: /main.py @ 63 : 17

File Edit View Run Tools Help

[main.py]

```
        rec_x = 0
    else:
        rec_x = int(cx +  x_rate)
        if rec_x >=90:
            rec_x = 128 - 45

    if (cy + y_rate) <= 0:
        rec_y = 0
    else:
        rec_y = int(cy +  y_rate)
        if rec_y >=64:
            rec_y = 64 - 15 - 5
#
    if sw.value() == 0:
        utime.sleep(0.02)
        if sw.value() == 0:
            oled.fill(0)
            oled.text("What you want? ", 0 ,10)
            utime.sleep(0.5)

    oled.fill_rect( rec_x, rec_y, 15, 15, 1)
    oled.fill_rect( rec_x+30, rec_y, 15, 15, 1)
    oled.show()
    utime.sleep(randint(1, 3) * 0.1)
    oled.fill(0)
    oled.fill_rect( rec_x, rec_y, 13, 13, 1)
    oled.fill_rect( rec_x+30, rec_y, 13, 13, 1)
    oled.show()
    utime.sleep(randint(1, 3) * 0.1)
    oled.fill(0)

```

HOW IT WORKS?

1. Import the Pin class and I2C class from the machine library and then import the SSD1306_I2C class from the library, and setting the variables of the OLED display.

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import utime
from random import randint


WIDTH = 128
HEIGHT = 64
```

2. Create instance of the bus of I2C, named it i2c, and create an instance, named oled, after that, clear the screen with fill method.

```
i2c_oled = I2C(1, sda=Pin(2), scl=Pin(3), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c_oled)

# clear screen
oled.fill(0)
```

3. Create instance of the bus of I2C, named it i2c, and create an instance, named oled, after that, clear the screen with fill method.

```
i2c_oled = I2C(1, sda=Pin(2), scl=Pin(3), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c_oled)

# clear screen
oled.fill(0)
```

4. Create variable to PS2 joystick.

```
vrx = ADC(0)   # x-axis
vry = ADC(1)   # y-axis
sw = Pin(0, Pin.IN, Pin.PULL_UP)  # button
```

5. Create two new functions to mapping the raw value to match the screen width and height.
mapping function's formula is:

```
def _map(x, in_min, in_max, out_min, out_max):
```

```
    return int((x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min)

# mapping the reading from 240-65535 to  0-128, screen width
def x_map(reading):
    return int((reading - 240) * (128 - 0) / (65535 - 240) + 0)

# mapping the reading from 240-65535 to  0-64, screen height
def y_map(reading):
    return int((reading - 240) * (64 - 0) / (65535 - 240) + 0)
```

6. Create the main loop and reading from the joystick, and mapping the raw value from 0-128 for x-axis and 0-64 for y-axis, and in order to setting the range that robot eyes can move to.

```
while True:

    cx = WIDTH / 2
    cy = HEIGHT /2

    read_vrx = vrx.read_u16()
    read_vry = vry.read_u16()

x_rate = x_map(read_vrx) - 88.0
# 88.0 can be modified when your raw reading is not point to 0.0 when PS2 joystick is untouched.

y_rate = y_map(read_vry) - 44.0
# same as the 88.0, 44.0 is an alternative number, as an calibration number, change it by compare it with x_rate, and y_rate, make it 0.0, 0.0

print(x_rate, y_rate)

    if (cx + x_rate) <= 0:
        rec_x = 0
    else:
        rec_x = int(cx +  x_rate)
        if rec_x >=90:
            rec_x = 128 - 45

    if (cy + y_rate) <= 0:
```

```
        rec_y = 0
    else:
        rec_y = int(cy +  y_rate)
        if rec_y >=64:
            rec_y = 64 - 15 - 5
```

7. Check the button's status, once it has been pushed down, will print some string on OLED display, and we adding utime.sleep(0.02) to prevent the button from generating level jitter when you press the button.

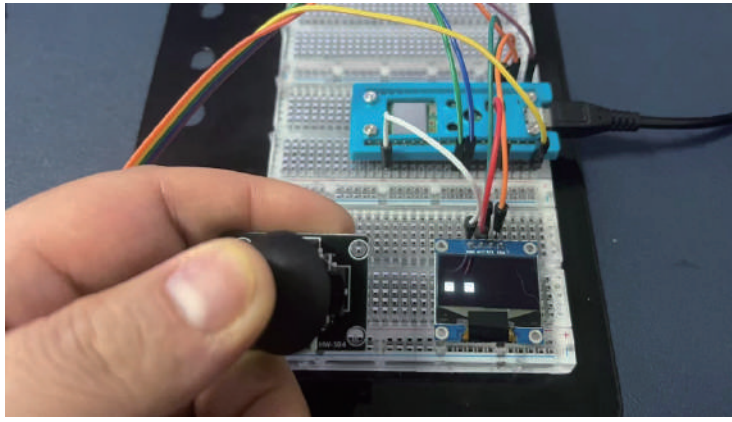
```
    if sw.value() == 0:
        utime.sleep(0.02)
        if sw.value() == 0:
            oled.fill(0)
            oled.text("What you want? ", 0 ,10)
            utime.sleep(0.5)
```

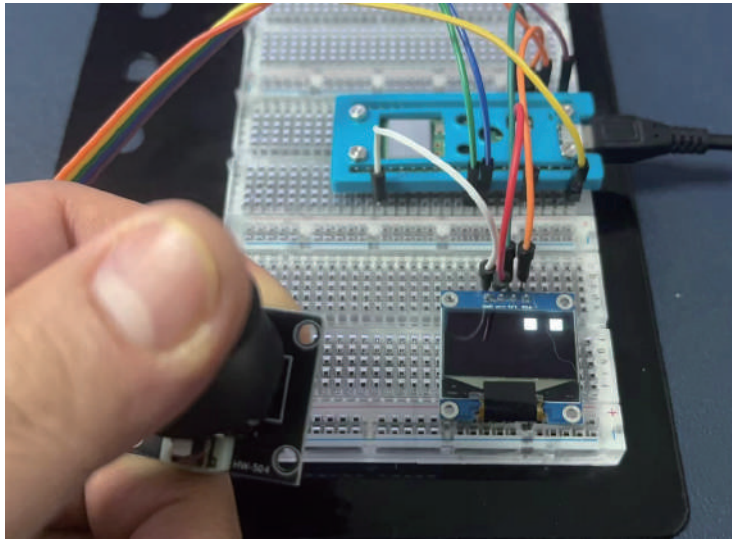
8. Move the robot eyes which generate by two rectangles on screen.

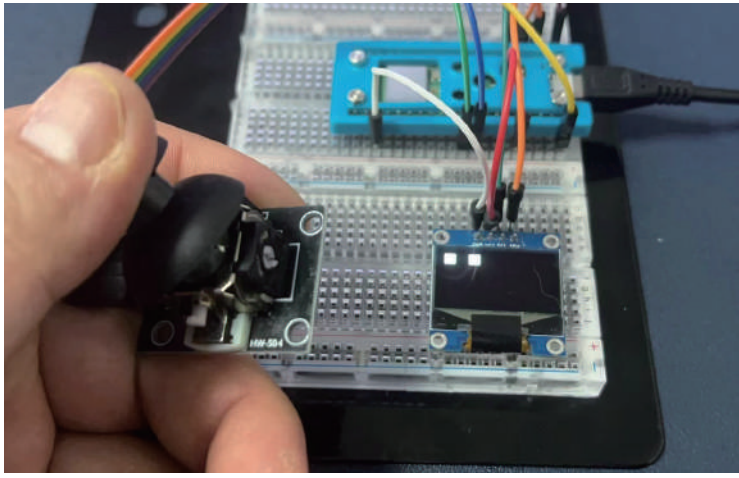
```
    oled.fill_rect( rec_x, rec_y, 15, 15, 1)
    oled.fill_rect( rec_x+30, rec_y, 15, 15, 1)
    oled.show()
    utime.sleep(randint(1, 3) * 0.1)
    oled.fill(0)
    oled.fill_rect( rec_x, rec_y, 13, 13, 1)
    oled.fill_rect( rec_x+30, rec_y, 13, 13, 1)
    oled.show()
    utime.sleep(randint(1, 3) * 0.1)
    oled.fill(0)
```

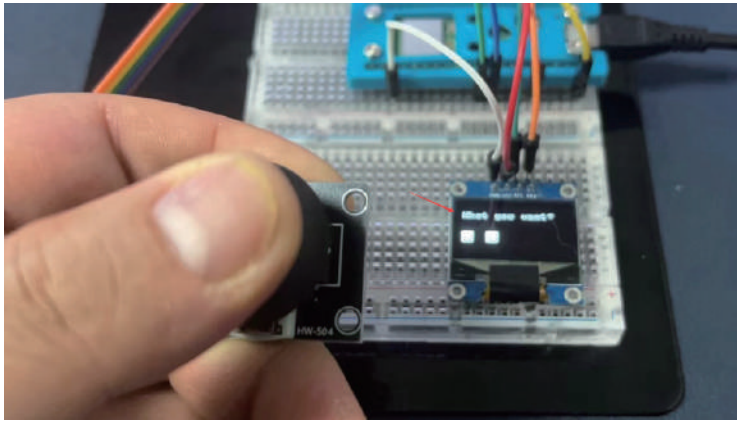
3. TESTING

Save the code to Raspberry Pi Pico W and named it "main.py" , click the run button on menu bar, and move the PS2 Joystick nob, you will see the eyes will follow your joystick move around on the OLED screen.






HW-504


HW-504

DEMO 13 STEPPER MOTOR

WHAT IS STEPPER MOTOR?

Stepper motors are DC brushless and synchronous motors. They rotate in discrete steps of predefined values and are able to rotate both clockwise and anticlockwise. Unlike other DC motors, they provide a precise position control according to the number of steps per revolution for which the motor is designed. That means a complete one revolution of a stepper motor is divided into a discrete number of steps. They are commonly used in CNC machines, Robotics, 2D and 3D printers.

28BYJ-48 is a uni-polar 5V stepper motor that takes electrical signals as input and rotates by converting these input signals into mechanical rotation. It consists of 4 stationary coils rated at +5V. These coils are known as a stator and make a ring around the rotor. Because of 5 volts operating voltage, we can easily drive this motor from any microcontroller such as Raspberry Pi Pico, ESP32, ESP8266, Arduino, etc. It has a 1/64 reduction gear set and therefore moves in precise 512 steps per revolution. These motors are silent in comparison to other DC motors and servo motors. You can achieve positional control easily without needing extra circuitry and components.

STRIDE ANGLE

This stepper motor has a stride angle of 5.625 degrees. That means 28BYJ-48 will complete one revolution in (360/5.625) 64 steps by taking one step at a time and in one step it covers a 5.625-degree distance. However, the stepper motor can also be used in full-step mode. In full-step mode, the angle of each step is 11.25 degrees. That means the motor completes its one revolution in 32 steps instead (360/11.25).

Therefore, in order to move one step forward or backward, the coils of the motor energize with a particular sequence.

STEPS PER REVOLUTION & STEP ANGLE

The output shaft of this particular stepper motor is driven through a gear ratio of 64:1 that is also known as the speed variation ratio. This suggests that after the inside motor

rotates 64 times then the shaft will complete one rotation.

Therefore, we can conclude that:

In order to complete one full rotation of the shaft a total of 2048 steps will be required. This is known as the steps per revolution calculated by multiplying 32 and 64 (32×64=2048).

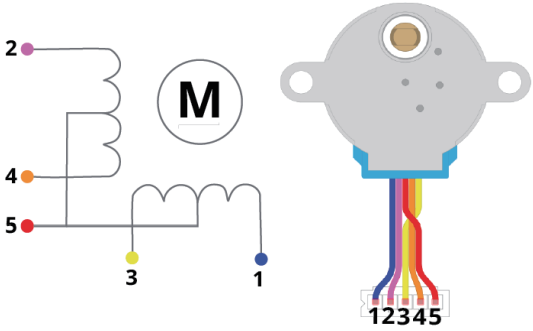
Moreover, this will enable the motor to have a step angle of 360°/2048 steps = 0.18°/step.

SPECIFICATIONS

- It is a unipolar 5 pin coil with a rated DC voltage of 5V.
- Has 4 phases with a stride angle of 5.625°/64.
- Speed variation ratio is 1/64
- The frequency of this stepper motor is 100Hz and insulated power is 600VAC/1mA/1s.
- The half-step method is recommended for driving this stepper motor.
- The value of pull in torque for a stepper motor is 300 gf-cm.

PINOUT

The following figure shows the pinout diagram of 28BYJ-48 stepper motor. It consists of 5 pins. Out of these 5 pins, four pins are used to provide sequence logic to the coils and one pin is a +5 volts supply pin.



PIN CONFIGURATION DETAILS

Pin Number	Coil Number	Colour
1	4	Blue
2	2	Pink
3	3	Yellow
4	1	Orange
5	VCC	Red

Coil 1-Coil 4: These are coils used to control the step sequence of the stepper motor. One end of each coil is connected with +5V and the other end will be connected with ULN2003 driver output.

VCC: Used to apply +5 volt supply to the stepper motor. This voltage appears across the coils when a specific coil is ground through a control sequence.

ULN2003 STEPPER MOTOR DRIVER MODULE

To use a 28BYJ-28 stepper motor with Raspberry Pi Pico W, we will be required to attach it with the ULN2003 motor driver. This is necessary because the current consumption of 28BYJ-48 is around 240mA.

That means the current required to drive coils by applying a sequence of control signals is also almost 200mA.

The pins of Raspberry Pi Pico W cannot provide current of this magnitude.

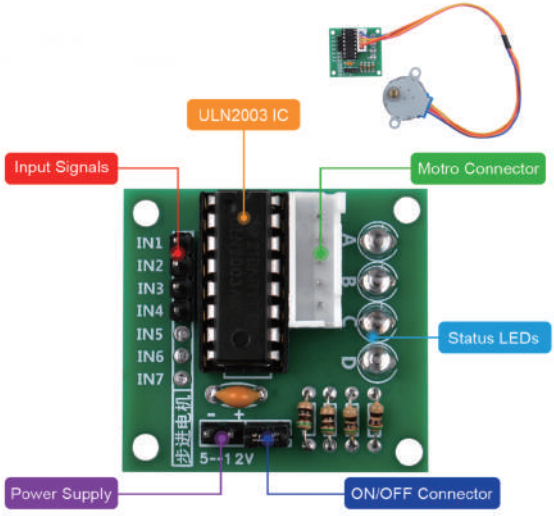
Therefore, we need a ULN2003 driver which translates low current output of Raspberry Pi Pico W pins into higher current that meets the requirement of stepper motor control signals.

The ULN2003 breakout board has high current and voltage than a single transistor and therefore it can drive a stepper motor easily by enabling our Raspberry Pi Pico W.

STEPPER MOTOR DRIVER MODULE PINOUT

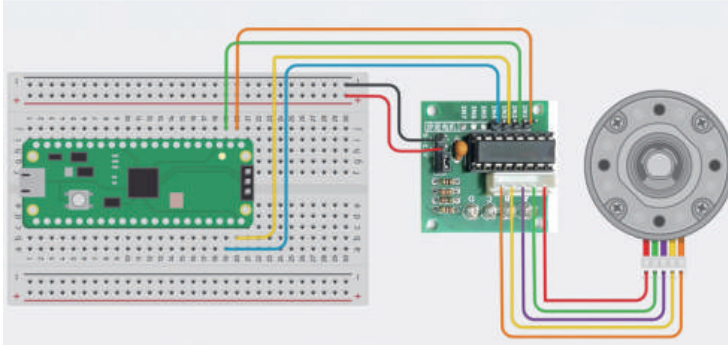
ULN2003 driver IC consists of 7 Darlington pair transistor outputs. Each output can drive 500mA and 50V load.

The input to each 7 Darlington pair transistor will be a signal from our microcontroller. To drive a stepper motor, this driver board uses only four input pins (IN1, IN2, IN3, and IN4).



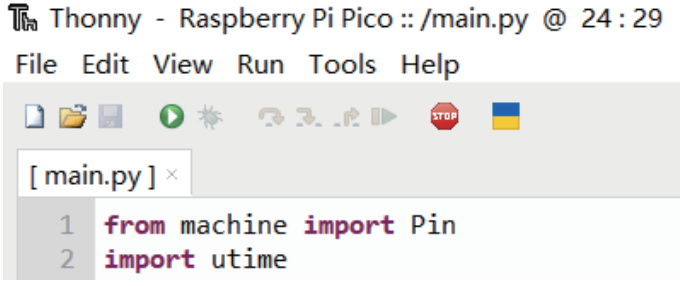
1. WIRING

Connect the IN1 Pin from Stepper motor driver board to GP15 on Raspberry Pi Pico W.
Connect the IN2 Pin from Stepper motor driver board to GP14 on Raspberry Pi Pico W.
Connect the IN3 Pin from Stepper motor driver board to GP16 on Raspberry Pi Pico W.
Connect the IN4 Pin from Stepper motor driver board to GP17 on Raspberry Pi Pico W.
Connect the 5V Pin from Stepper motor driver board to 5V on Raspberry Pi Pico W.
Connect the GND Pin from Stepper motor driver board to GND on Raspberry Pi Pico W.



2. CODING

Now that we are ready with the connections, it is time to look at the code. Start by importing the pin module from the machine library, We also import utime to add some delays.



Next we create a list of pins configured as outputs. A stepper motor essentially works by providing a sequence of values to these four pins.

[main.py]

```
from machine import Pin
import utime


pins = [
    Pin(15, Pin.OUT), #IN1
    Pin(14, Pin.OUT), #IN2
    Pin(16, Pin.OUT), #IN3
    Pin(17, Pin.OUT), #IN4
    ]
```

Then we create a list of sequences to be provided to the stepper motor. As you can see, we create sequences where the 1 value shifts through the different positions in the list.

[main.py]

```
from machine import Pin
import utime


pins = [
    Pin(15, Pin.OUT), #IN1
    Pin(14, Pin.OUT), #IN2
    Pin(16, Pin.OUT), #IN3
    Pin(17, Pin.OUT), #IN4
    ]


full_step_sequence = [
    [1, 0, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 1, 0],
    [0, 0, 0, 1]
    ]
```

Now in a forever loop, we step through the sequence and apply each value of the sequence to each one of the pins. We also add a little bit of delay so we can notice the turn.

[main.py]

```
from machine import Pin
import utime


pins = [
    Pin(15, Pin.OUT), #IN1
    Pin(14, Pin.OUT), #IN2
    Pin(16, Pin.OUT), #IN3
    Pin(17, Pin.OUT), #IN4
    ]


full_step_sequence = [
    [1, 0, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 1, 0],
    [0, 0, 0, 1]
    ]

while True:
    for step in full_step_sequence:
        for i in range(len(pins)):
            pins[i].value(step[i])
            utime.sleep(0.001)
```

3. TESTING

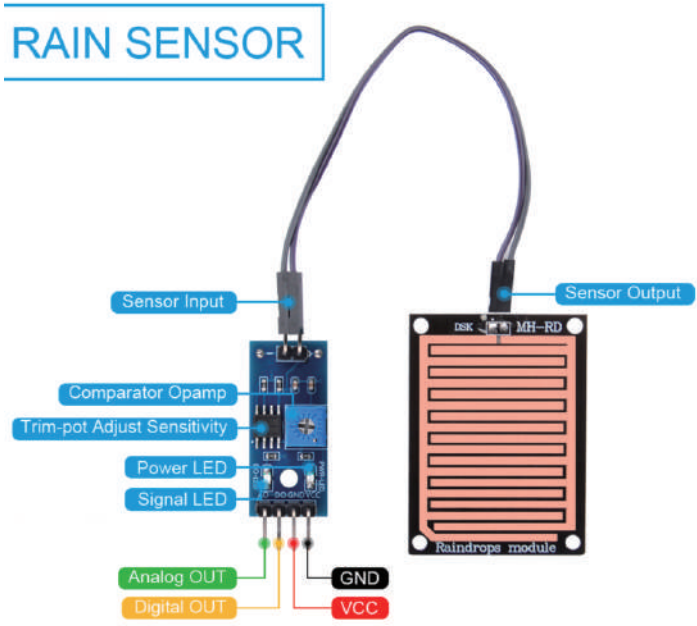
To see the demo of above code, copy this code to Thonny IDE and upload it to Raspberry Pi Pico W. Also, attach the stepper motor and ULN2003 with Raspberry Pi Pico W according to the connection diagram above.

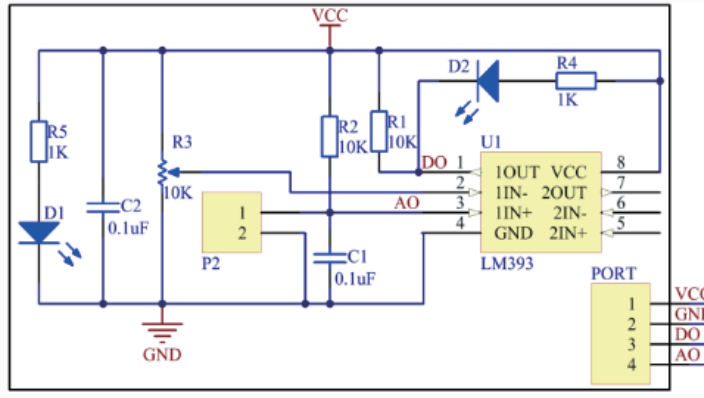
The stepper motor will move in a clockwise direction.

DEMO 14 RAIN DROP SENSOR BUILD IOT DEVICE

The rain detection module detects rain on the board. Place the rain detection board in the open air. When it is raining, the rain detection module will sense the raindrops and send signals to the Raspberry Pi Pico W

When rain falls on top of the sensor the resistivity of the conductive plates changes, and by measuring the changes in the resistance, we can determine the intensity of the rainfall. The more intense the rainfall the lower the resistance.





VCC is the power supply pin of the Rain Detection Sensor that can be connected to 3.3V or 5V of the supply. But do note that the analog output will vary depending upon the provided supply voltage.

GND is the ground pin of the board and it should be connected to the ground pin of the Raspberry Pi Pico W.

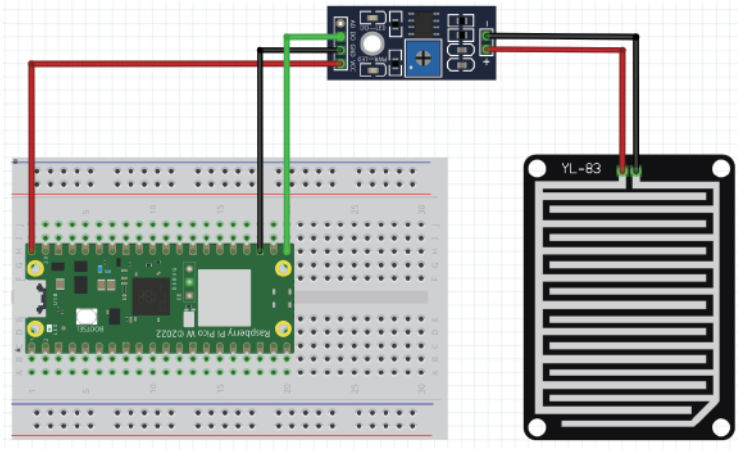
DOUT is the Digital output pin of the board, output low indicates rain is detected, and high indicates no rain condition.

AOUT is the Analog output pin of the board that will give us an analog signal in between VCC and ground.

1. WIRING

Connect the + Pin from rain drops driver board to Detection Board Pin.
Connect the - Pin from rain drops driver board to Detection Board Pin.

Connect the VCC Pin from rain drops driver board to VBUS Pin on Raspberry Pi Pico W.
Connect the GND Pin from rain drops driver board to GND Pin on Raspberry Pi Pico W.
Connect the DO Pin from rain drops driver board to GP16 Pin on Raspberry Pi Pico W.



2. CODING

Now we create a new file in thonny IDE and import machine library and utime library.

File Edit View Run Tools Help

[main.py]

```
1 from machine import Pin
2 import utime
3
```

Create an instance of sensor by using Pin class.

File Edit View Run Tools Help

[main.py]

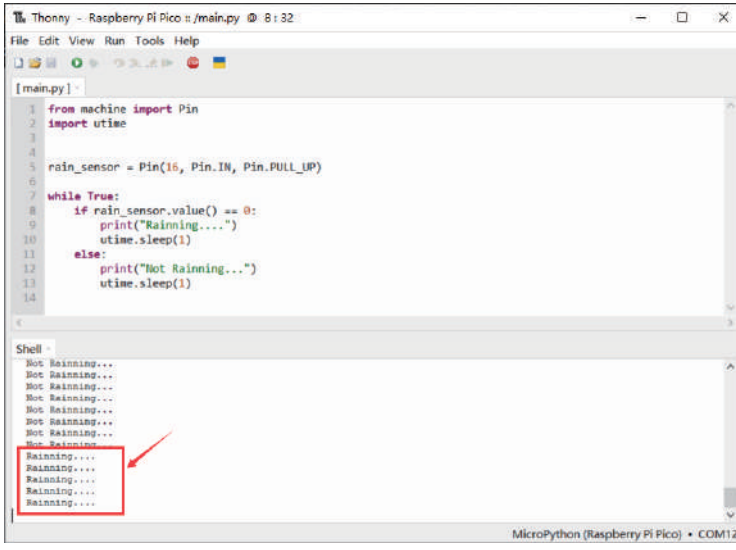
```
1 from machine import Pin
2 import utime
3
4
5 rain_sensor = Pin(16, Pin.IN, Pin.PULL_UP)
6
```

And reading sensor' s value and if the value is equal to 0 means the pin has been pulling down, it means:

When rain falls on top of the sensor the resistivity of the conductive plates changes, and by measuring the changes in the resistance, we can determine the intensity of the rainfall. The more intense the rainfall the lower the resistance.

3. TESTING

Save the file to Raspberry Pi Pico W and named it main.py.
once we put some water on the detection board, you will see following figure:



DEMO 15 SERVO PROJECT

A servo is generally composed of the following parts: case, shaft, gear system, potentiometer, DC motor, and embedded board.

It works like this:
The microcontroller sends out PWM signals to the servo, and then the embedded board in the servo receives the signals through the signal pin and controls the motor inside to turn.

As a result, the motor drives the gear system and then motivates the shaft after deceleration.

The shaft and potentiometer of the servo are connected together. When the shaft rotates, it drives the potentiometer, so the potentiometer outputs a voltage signal to the embedded board. Then the board determines the direction and speed of rotation based on the current position, so it can stop exactly at the right position as defined and hold there.

The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse Width Modulation.

The servo expects to see a pulse every 20 ms.

The length of the pulse will determine how far the motor turns.

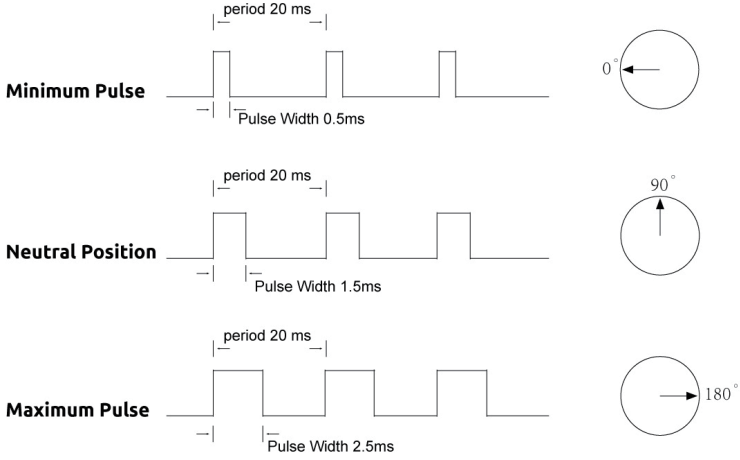
For example, a 1.5ms pulse will make the motor turn to the 90-degree position (neutral position).

When a pulse is sent to a servo that is less than 1.5 ms, the servo rotates to a position and holds its output shaft some number of degrees counterclockwise from the neutral point.

When the pulse is wider than 1.5 ms the opposite occurs.

The minimal width and the maximum width of pulse that will command the servo to turn to a valid position are functions of each servo.

Generally, the minimum pulse will be about 0.5 ms wide and the maximum pulse will be 2.5 ms wide.



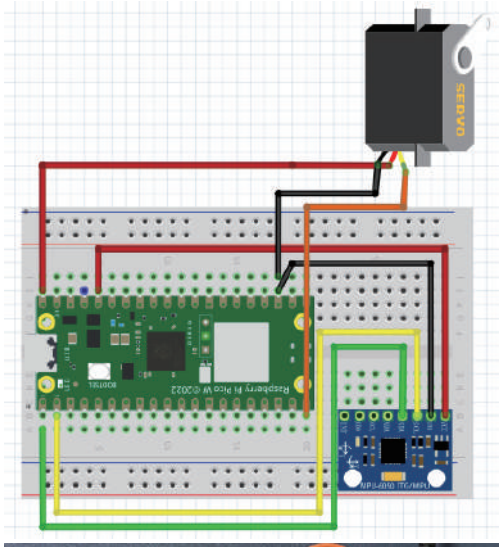
1. WIRING

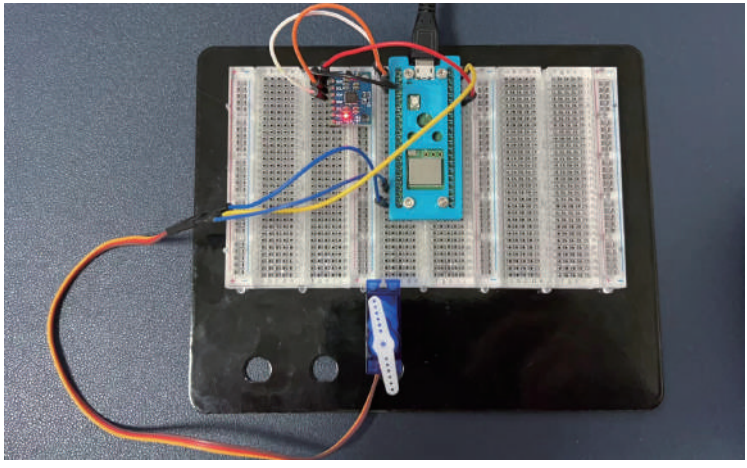
Servo

Connect the red wire from servo to VBUS(+5V) on Raspberry Pi Pico W.
Connect the brown wire from servo to GND on Raspberry Pi Pico W.
Connect the yellow wire from servo to GP15 on Raspberry Pi Pico W.

MPU6050

Connect VCC Pin from MPU6050 module to 3V3(OUT) Pin on Raspberry Pi Pico W.
Connect GND Pin from MPU6050 module to GND Pin Raspberry Pi Pico W.
Connect SDA Pin from MPU6050 module to GP0 Pin Raspberry Pi Pico W.
Connect SCL Pin from MPU6050 module to GP1 Pin Raspberry Pi Pico W.


SERVO



2. CODING

Now first we create a new file and import the libraries.
here we import PWM and I2C for servo and MPU6050 module.
math module provides access to the mathematical functions defined by the C standard.
for example:

math.sqrt(*x*)

Return the square root of *x*.

math.pow(*x*, *y*)

Return `x` raised to the power `y`. Exceptional cases follow the IEEE 754 standard as far as possible. In particular, `pow(1.0, x)` and `pow(x, 0.0)` always return `1.0`, even when `x` is a zero or a NaN. If both `x` and `y` are finite, `x` is negative, and `y` is not an integer then `pow(x, y)` is undefined, and raises `ValueError`.

Unlike the built-in `**` operator, `math.pow()` converts both its arguments to type `float`. Use `**` or the built-in `pow()` function for computing exact integer powers.

Changed in version 3.11: The special cases `pow(0.0, -inf)` and `pow(-0.0, -inf)` were changed to return `inf` instead of raising `ValueError`, for consistency with IEEE 754.

math.atan2(*y*, *x*)

Return `atan(y / x)`, in radians. The result is between `-pi` and `pi`. The vector in the plane from the origin to point `(x, y)` makes this angle with the positive X axis. The point of `atan2()` is that the signs of both inputs are known to it, so it can compute the correct quadrant for the angle. For example, `atan(1)` and `atan2(1, 1)` are both `pi/4`, but `atan2(-1, -1)` is `-3*pi/4`.

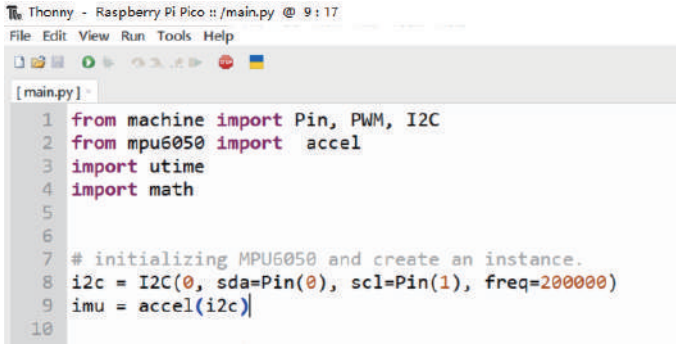
Thonny - Raspberry Pi Pico :: /main.py @ 9 : 17

File Edit View Run Tools Help

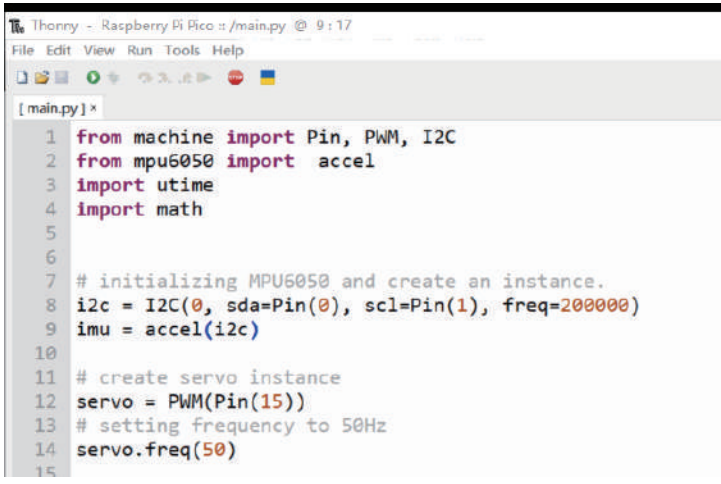
[main.py] ×

```
1  from machine import Pin, PWM, I2C
2  from mpu6050 import  accel
3  import utime
4  import math
5
```

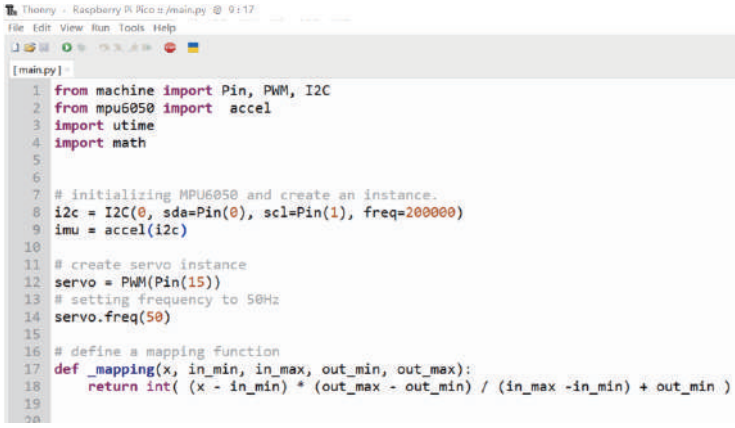
and then initializing MPU6050 module via using I2C class to create an instance.



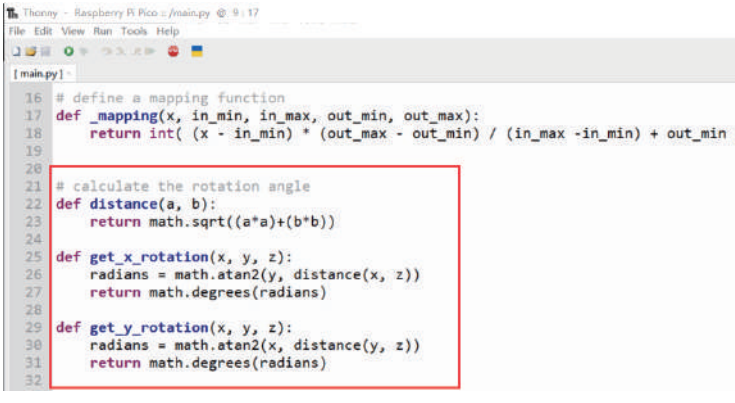
after that, we create another instance of servo.



In order to control the servo smoothly, we need to create a function to mapping the data from IMU.



and then we create 3 functions to calculate the rotation angle.



and create a function to control servo:

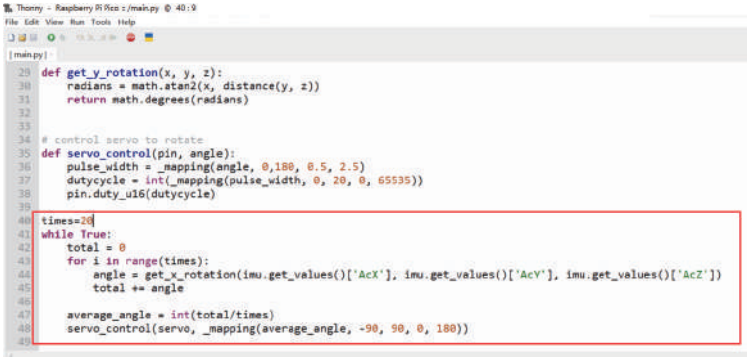
Thonny - Raspberry Pi Pico :: /main.py @ 9 : 17

File Edit View Run Tools Help

[main.py]

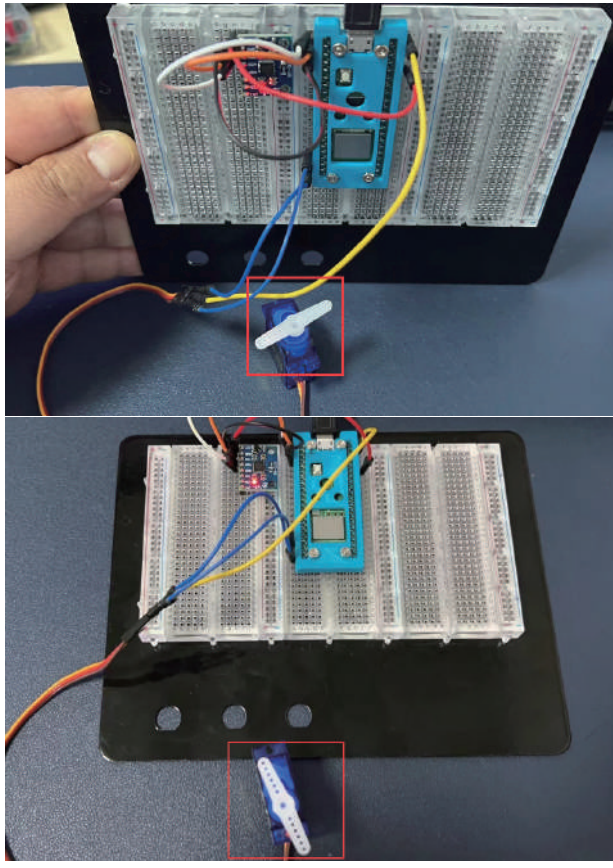
```
        return math.degrees(radians)

def get_y_rotation(x, y, z):
    radians = math.atan2(x, distance(y, z))
    return math.degrees(radians)


# control servo to rotate
def servo_control(pin, angle):
    pulse_width = _mapping(angle, 0,180, 0.5, 2.5)
    dutycycle = int(_mapping(pulse_width, 0, 20, 0, 65535))
    pin.duty_u16(dutycycle)
```

we create a variable to store the times we want to smooth the angle rotations. and in a loop, we check the IMU' s value and get the "AcX" , "AcY" , "AcZ" values as the arguments of **get_x_rotation()** function. we will get the angle of each time, and we average it by divide the sum of angle by times, it will smooth the rotation, and then send the final result to **servo_control()** function, and mapping it from -90 degree to 90 degree when get the average angle of it.

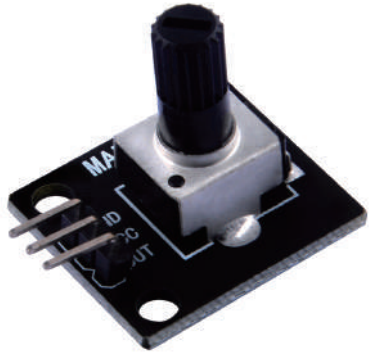


3. TESTING

When you flip the IMU up and down, you will find that the servo arm will follow the flip to change the angle.



DEMO 16 POTENTIOMETER PROJECT



A potentiometer is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider.

If only two terminals are used, one end and the wiper, it acts as a variable resistor or rheostat.

The measuring instrument called a potentiometer is essentially a voltage divider used for measuring electric potential (voltage); the component is an implementation of the same principle, hence its name.

Potentiometers are commonly used to control electrical devices such as volume controls on audio equipment. Potentiometers operated by a mechanism can be used as position transducers, for example, in a joystick. Potentiometers are rarely used to directly control significant power (more than a watt), since the power dissipated in the potentiometer would be comparable to the power in the controlled load.

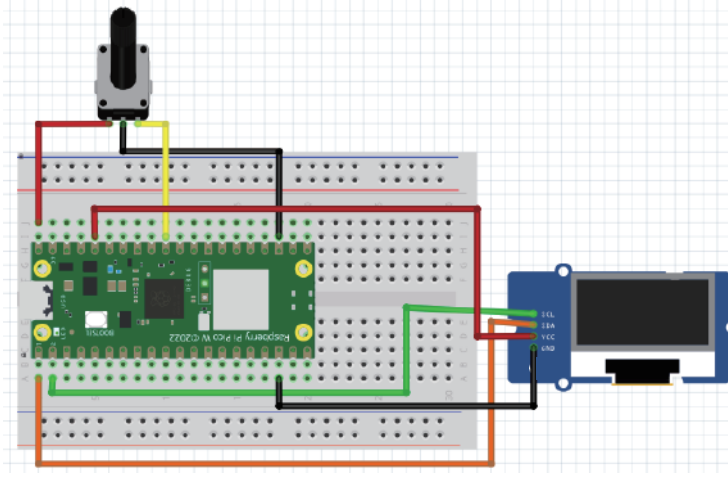
1. WIRING

OLED

Connect VCC Pin from OLED display to 3V3(OUT) Pin on Raspberry Pi Pico W.
Connect GND Pin from OLED display to GND Pin on Raspberry Pi Pico W.
Connect SDA Pin from OLED display to GP0 Pin on Raspberry Pi Pico W.
Connect SCL Pin from OLED display to GP1 Pin on Raspberry Pi Pico W.

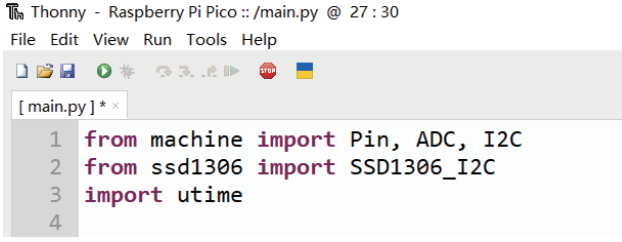
Potentiometer

Connect VCC Pin from Potentiometer module to VBUS(+5v) Pin on Raspberry Pi Pico W.
Connect GND Pin from Potentiometer module to GND Pin on Raspberry Pi Pico W.
Connect OUT Pin from Potentiometer module to GP26 Pin on Raspberry Pi Pico W.

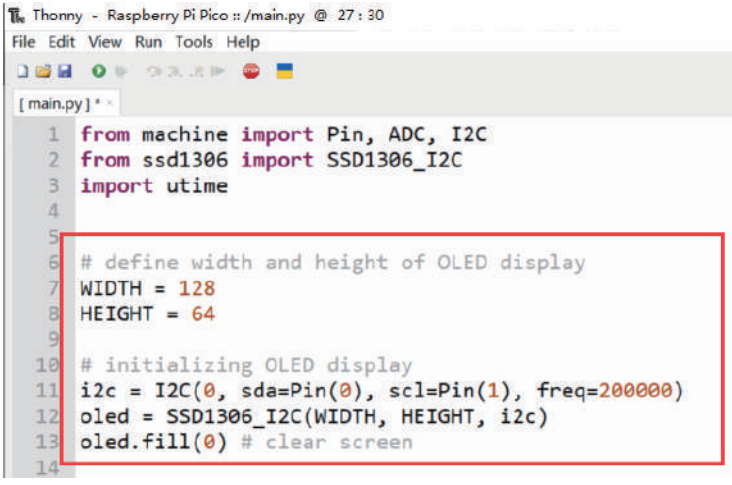


2. CODING

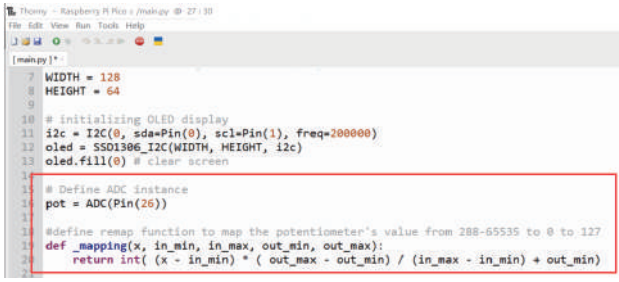
Open Thonny IDE and create a new file, import the machine library and utime library. Here we need using ADC class to generate a new instance about analogue device reading.



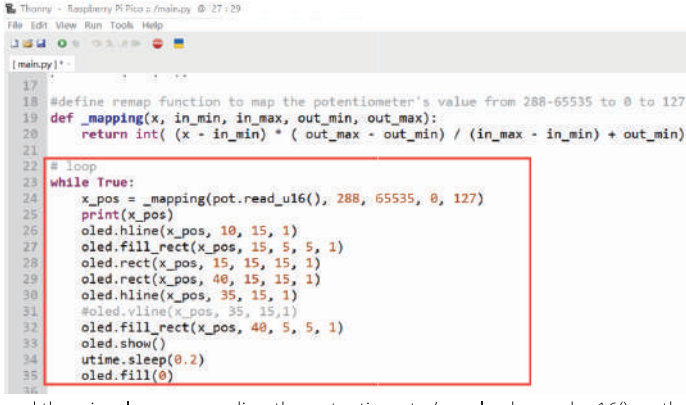
create instance of OLED display.



create ADC instance by using ADC class, here we connect the GP26, so put Pin(26) object to ADC class will create new instance of ADC, we called it "pot" .



also we need to create a function to map the raw value of potentiometer from 288 to 65535 to 0-127 which is the width of OLED display.



and then, in a loop, we reading the potentiometer' s value by read_u16() method. and mapped it from 288-65535 to 0-127.
we draw a horizontal line from mapped data as x position, and y=10, length=15, colour=1.
and then fill_rect() means drawing filling rectangle on x=x_pos, y=15, width=5, height=5, colour=1.
and we use oled.show() method to draw the buffer to OLED display. and delay for 0.2 second and then clean screen by using fill(0) method.

Basic functons

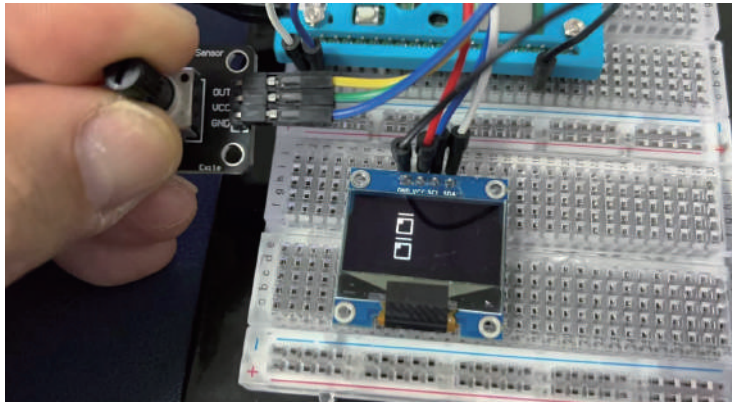
```
oled.poweroff()     # power off the display, pixels persist in memory
oled.poweron()      # power on the display, pixels redrawn
oled.contrast(0)    # dim
oled.contrast(255)  # bright
oled.invert(1)      # display inverted
oled.invert(0)      # display normal
oled.rotate(True)   # rotate 180 degrees
oled.rotate(False)  # rotate 0 degrees
oled.show()         # write the contents of the FrameBuffer to display memory
```

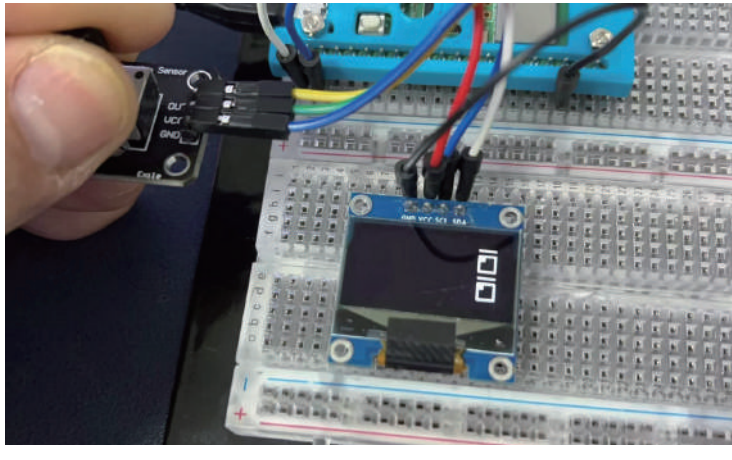
Subclassing FrameBuffer provides support for graphics primitives

```
oled.fill(0)                    # fill entire screen with colour=0
oled.pixel(0, 10)               # get pixel at x=0, y=10
oled.pixel(0, 10, 1)            # set pixel at x=0, y=10 to colour=1
oled.hline(0, 8, 4, 1)          # draw horizontal line x=0, y=8, width=4, colour=1
oled.vline(0, 8, 4, 1)          # draw vertical line x=0, y=8, height=4, colour=1
oled.line(0, 0, 127, 63, 1)     # draw a line from 0,0 to 127,63
oled.rect(10, 10, 107, 43, 1)   # draw a rectangle outline 10,10 to 117,53, colour=1
oled.fill_rect(10, 10, 107, 43, 1)  # draw a solid rectangle 10,10 to 117,53, colour=1
oled.text('Hello World', 0, 0, 1)   # draw some text at x=0, y=0, colour=1
oled.scroll(20, 0)              # scroll 20 pixels to the right

# draw another FrameBuffer on top of the current one at the given coordinates
import framebuf
fbuf = framebuf.FrameBuffer(bytearray(8 * 8 * 1), 8, 8, framebuf.MONO_VLSB)
fbuf.line(0, 0, 7, 7, 1)
oled.blit(fbuf, 10, 10, 0)      # draw on top at x=10, y=10, key=0
oled.show()
```

3. TESTING

When you rotating the knob of potentiometer, you will find that the drawing on screen will move towards right. If you have two of them, you can build your own "pot-game" by rotating of the knob.


Sensor
OUT
VCC
GND
Enle

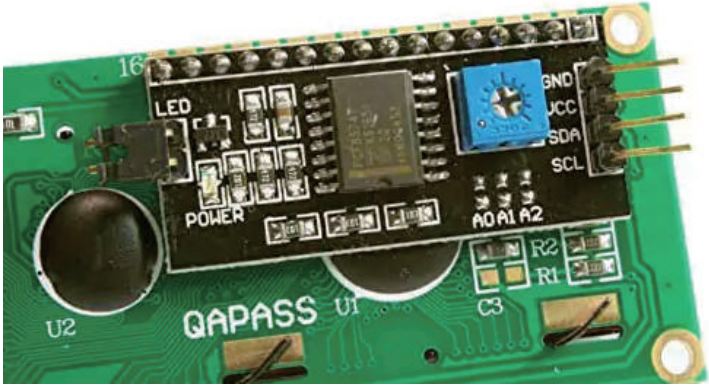

Sensor
OUT
VCC
GND
Enle

DEMO 17 LCD1602 PROJECT

As the name indicates 16X2, means it can display characters in 16 columns and 2 rows and LCD is the Liquid Crystal Display.

It can display total of 16X2=32 characters. Each character is made with 5width x 8height pixel dots =total 40 pixel dots for one character.

To make a character, pixel dots should be lit black according to the character shape and other pixels should be in off state. So for every character all the 40 pixel dots are to be instructed weither to be lit on or off.



It uses I2C interface protocol to transfer the data using only 2 pins SDA and SCL. In this method we are using 16X2 LCD display module which has I2C adapter connected to 16 pins of normal LCD display and outputs only 4 pins.

The I2c adapter has PCF8574 8-Bit I/O Expander IC which converts the I2C data from the microcontrollers to parallel data required to the LCD display module. This adapter has a jumper to turn ON/OFF the backlight which can save the power consumption, and also has an inbuilt potentiometer to adjust the contrast of the LCD display module.

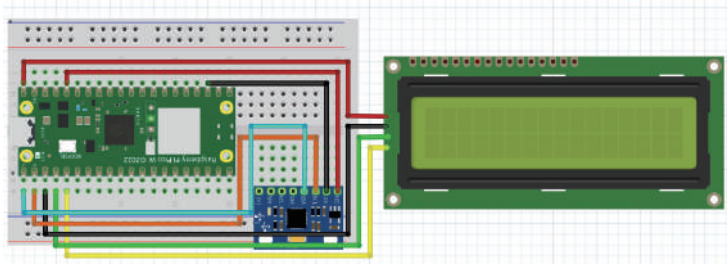
1. WIRING

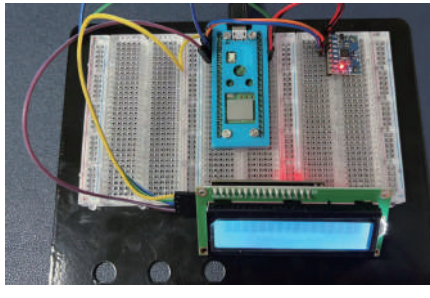
MPU6050

Connect the VCC Pin from MPU6050 module to 3V3(OUT) Pin on Raspberry Pi Pico W.
Connect the GND Pin from MPU6050 module to GND Pin on Raspberry Pi Pico W.
Connect the SDA Pin from MPU6050 module to GP0 Pin on Raspberry Pi Pico W.
Connect the SCL Pin from MPU6050 module to GP1 Pin on Raspberry Pi Pico W.

LCD1602 Display Module

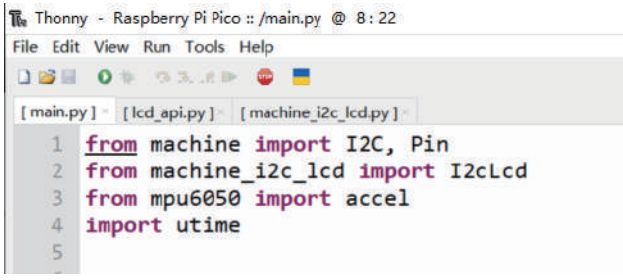
Connect the VCC Pin from LCD1602 Display to VBUS(5V) Pin on Raspberry Pi Pico W.
Connect the GND Pin from LCD1602 Display to GND Pin on Raspberry Pi Pico W.
Connect the SDA Pin from LCD1602 Display to GP2 Pin on Raspberry Pi Pico W.
Connect the SCL Pin from LCD1602 Display to GP3 Pin on Raspberry Pi Pico W.





2. CODING

Note: LCD display I2C requires few pre coded libraries
* **lcd_api.py**
* **machine_i2c_lcd.py**
here is the fully code of lcd_api.py, or you can download from internet.
https://github.com/geeekpi/picokitadv/blob/main/libs/lcd_api.py
and
https://github.com/geeekpi/picokitadv/blob/main/libs/machine_i2c_lcd.py
Copy and paste the libraries to your Raspberry Pi Pico W' s lib folder.
and then create a new file and import the libraries.



and then initializing the MPU6050 instance.

after that, initializing the LCD1602 display instance.

In a loop, reading the values from MPU6050.

After getting sensor raw data we can calculate acceleration and angular velocity by dividing sensor raw data with their sensitivity scale factor as follows:

Accelerometer values in g (g force)

Acceleration along the X axis = (Accelerometer X axis raw data/16384) g.

Acceleration along the Y axis = (Accelerometer Y axis raw data/16384) g.

Acceleration along the Z axis = (Accelerometer Z axis raw data/16384) g.

Gyroscope values in °/s (degree per second)

Angular velocity along the X axis = (Gyroscope X axis raw data/131) °/s.

Angular velocity along the Y axis = (Gyroscope Y axis raw data/131) °/s.

Angular velocity along the Z axis = (Gyroscope Z axis raw data/131) °/s.

Temperature value in °/c (degree per Celsius)

Temperature in degrees C = ((temperature sensor data)/340 + 36.53) °/c.

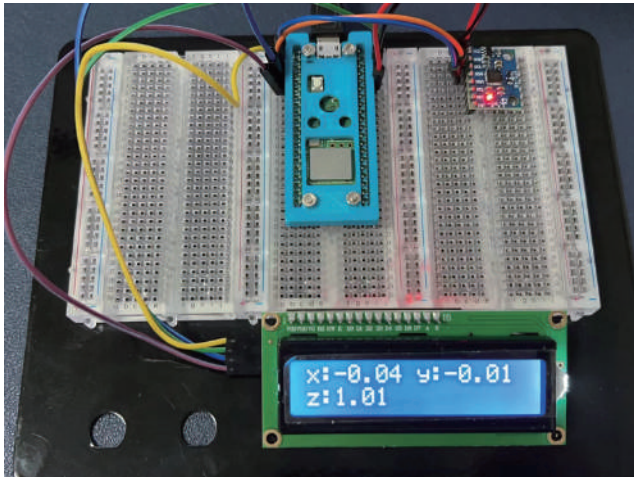
Thonny - Raspberry Pi Pico :: /main.py @ 15 : 37

File Edit View Run Tools Help

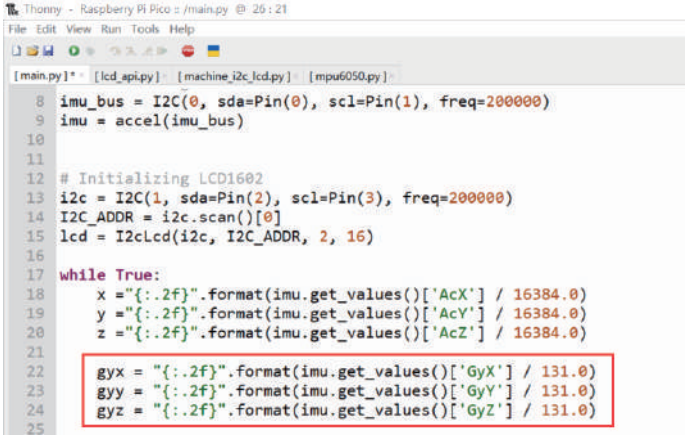
[main.py] [lcd_api.py] [machine_i2c_lcd.py] [mpu6050.py]

```
imu_bus = I2C(0, sda=Pin(0), scl=Pin(1), freq=200000)
imu = accel(imu_bus)


# Initializing LCD1602
i2c = I2C(1, sda=Pin(2), scl=Pin(3), freq=200000)
I2C_ADDR = i2c.scan()[0]
lcd = I2cLcd(i2c, I2C_ADDR, 2, 16)

while True:
    x ="{:.2f}".format(imu.get_values()['AcX'] / 16384.0)
    y ="{:.2f}".format(imu.get_values()['AcY'] / 16384.0)
    z ="{:.2f}".format(imu.get_values()['AcZ'] / 16384.0)

    lcd.move_to(0,0)
    lcd.putstr("x:"+ x)
    lcd.move_to(8, 0)
    lcd.putstr("y:"+y)
    lcd.move_to(0, 1)
    lcd.putstr("z:"+z)
    utime.sleep(0.01)
```

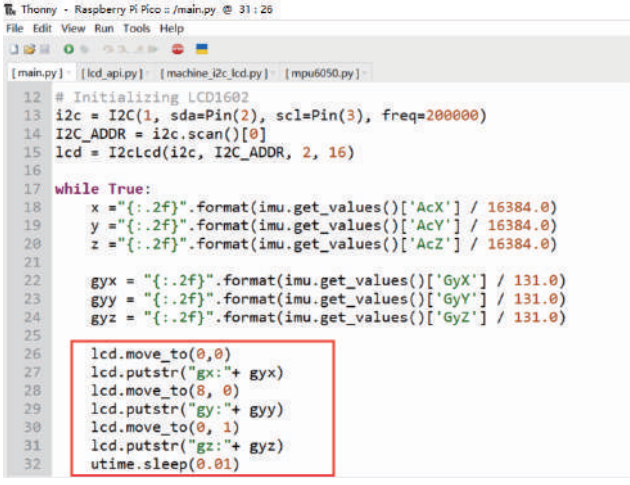
We' ve got the Accelerometer values and shows on LCD1602 display.



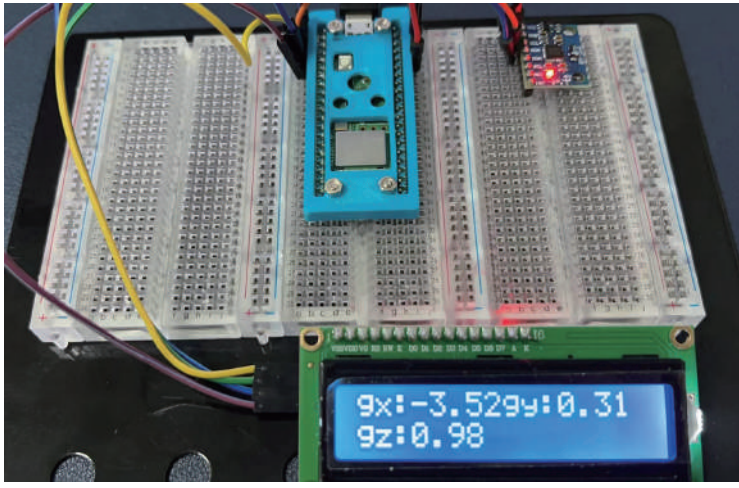
If we change the data to Angular velocity, just adding following code:



and then we show the result:



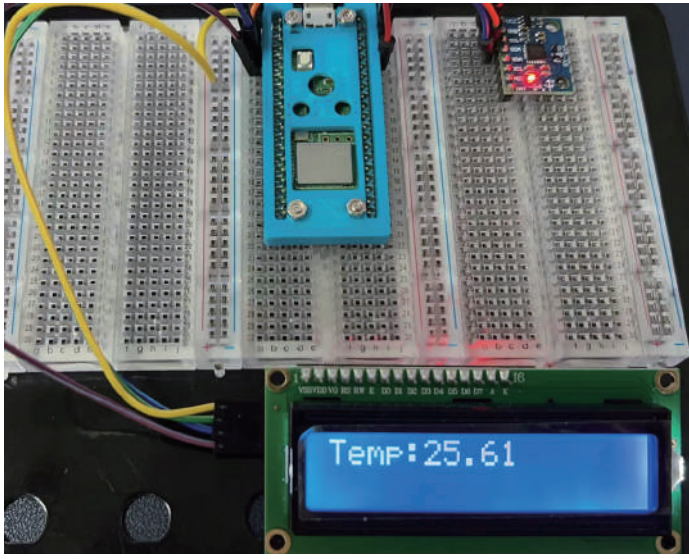
Result:



If you want to get the temperature information:

```
while True:
#     x ="{:.2f}".format(imu.get_values()['AcX'] / 16384.0)
#     y ="{:.2f}".format(imu.get_values()['AcY'] / 16384.0)
#     z ="{:.2f}".format(imu.get_values()['AcZ'] / 16384.0)
#
#     gyx = "{:.2f}".format(imu.get_values()['GyX'] / 131.0)
#     gyy = "{:.2f}".format(imu.get_values()['GyY'] / 131.0)
#     gyz = "{:.2f}".format(imu.get_values()['GyZ'] / 131.0)
    temp = "{:.2f}".format(imu.get_values()['Tmp'])

    lcd.move_to(0,0)
    lcd.putstr("Temp:"+ temp)
    utime.sleep(3)
```

Result:



You can also calculate the angle of pitch and angle of roll by adding following code. first, you need to import math library and then define some functions:

Thonny - Raspberry Pi Pico :: /main.py @ 50 : 33

File Edit View Run Tools Help

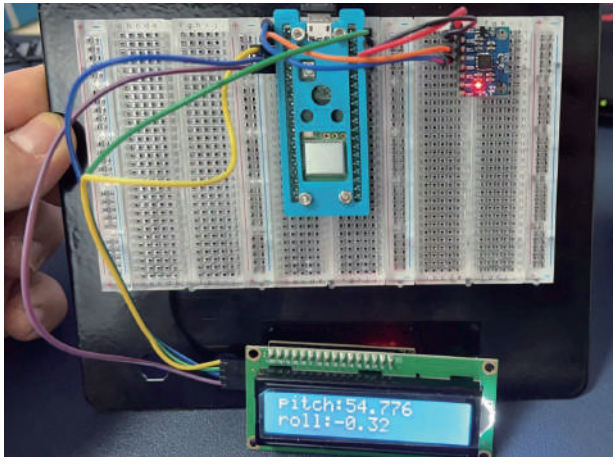
[main.py] [lcd_api.py] [machine_i2c_lcd.py] [mpu6050.py]

```
18 # mapping function
19 def _mapping(x, in_min, in_max, out_min, out_max):
20     return (x - in_min) * ( out_max - out_min) / (in_max - in_min) + out_min
21
22 def dist(a,b):
23     return math.sqrt((a*a)+(b*b))
24
25 def get_y_rotation(x,y,z):
26     radians = math.atan2(x,dist(y,z))
27     return -math.degrees(radians)
28
29 def get_x_rotation(x,y,z):
30     radians = math.atan2(y, dist(x, z))
31     return math.degrees(radians)
32
```

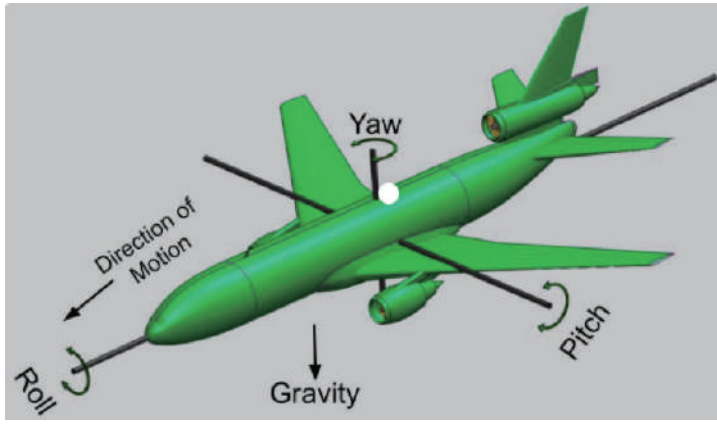
and then reading the raw data and calculate the angle.

```
while True:
    x = imu.get_values()['AcX'] / 16384.0
    y = imu.get_values()['AcY'] / 16384.0
    z = imu.get_values()['AcZ'] / 16384.0

    gyx = "{:.2f}".format(imu.get_values()['GyX'] / 131.0)
    gyy = "{:.2f}".format(imu.get_values()['GyY'] / 131.0)
    gyz = "{:.2f}".format(imu.get_values()['GyZ'] / 131.0)

    temp = "{:.2f}".format(imu.get_values()['Tmp'])

    roll = "{:.2f}".format(get_y_rotation(x, y, z))
    pitch = "{:.2f}".format(get_x_rotation(x, y, z))

    lcd.move_to(0,0)
    lcd.putstr("pitch:"+ str(pitch))
    lcd.move_to(0,1)
    lcd.putstr("roll:"+ str(roll))
    utime.sleep(0.5)
```

create ADC instance by using ADC class, here we connect the GP26, so put Pin(26) object to ADC class will create new instance of ADC, we called it "pot" .



It can be your RC model' s controller by using this demo code.

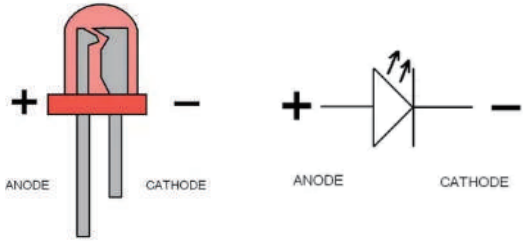


DEMO 18 LED PROJECT

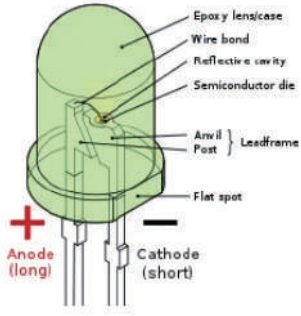
Let’ s make a level state detector by using MPU6050 and several LEDs.

LED

Semiconductor light-emitting diode is a type of component which can turn electric energy into light energy via PN junctions. By wavelength, it can be categorized into laser diode, infrared light-emitting diode and visible light-emitting diode which is usually known as light-emitting diode (LED).



Diode has unidirectional conductivity, so the current flow will be as the arrow indicates in figure circuit symbol. You can only provide the anode with a positive power and the cathode with a negative. Thus the LED will light up. Each LED has two legs, the longer one is called anode, the shorter one called cathode, you cannot connect them inversely due to it only has one way to let the current flow.



There is fixed forward voltage drop in the LED, so it cannot be connected with the circuit directly because the supply voltage can outweigh this drop and cause the LED to be burnt.

The forward voltage of the red, yellow, and green LED is 1.8 V and that of the white one is 2.6 V.

Most LEDs can withstand a maximum current of 16 to 25 mA, so we need to connect a current limiting resistor in series. here we are using 220Ω Resister to limit the current.

We try to make a level state tester with MPU6050 and LEDs, when we keep level, only green and blue LED lights are on, when our pitch angle is greater than 15 degrees or less than -15 degrees, we keep the yellow led on all the time, and the red one blinks every 0.5s.

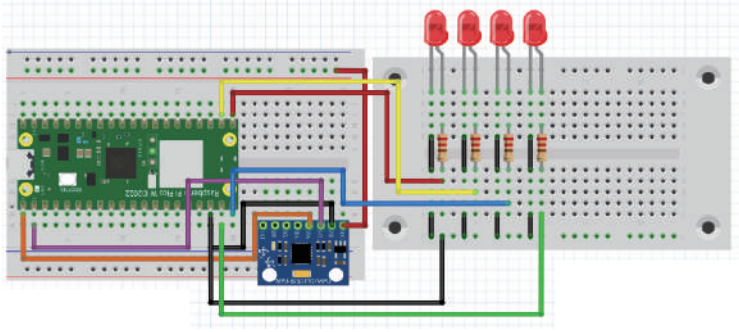
1. WIRING

MPU6050

Connect the VCC Pin from MPU6050 module to 3V3(OUT) Pin on Raspberry Pi Pico W.
Connect the GND Pin from MPU6050 module to GND Pin on Raspberry Pi Pico W.
Connect the SDA Pin from MPU6050 module to GP0 Pin on Raspberry Pi Pico W.
Connect the SCL Pin from MPU6050 module to GP1 Pin on Raspberry Pi Pico W.

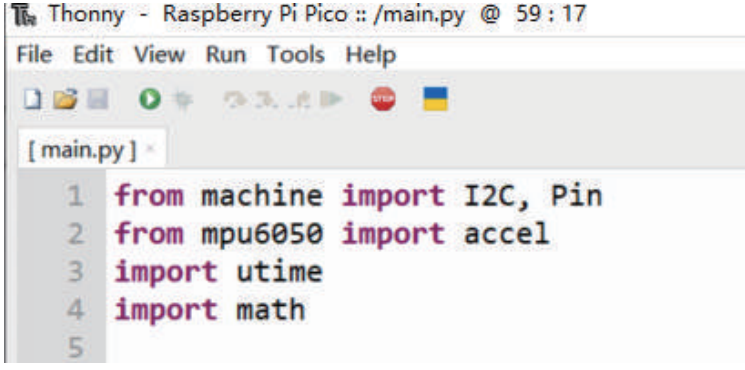
LED

Red: Anode(Long leg) connect to 220Ω Resister and connect to GP16 on Raspberry Pi Pico W.
Red: Cathode (shorter leg) connect to GND Pin on Raspberry Pi Pico W.
Yellow: Connects to GP17 on Raspberry Pi Pico W at the same way.
Blue: Connects to GP15 on Raspberry Pi Pico W at the same way.
Green: Connects to GP14 on Raspberry Pi Pico W at the same way.

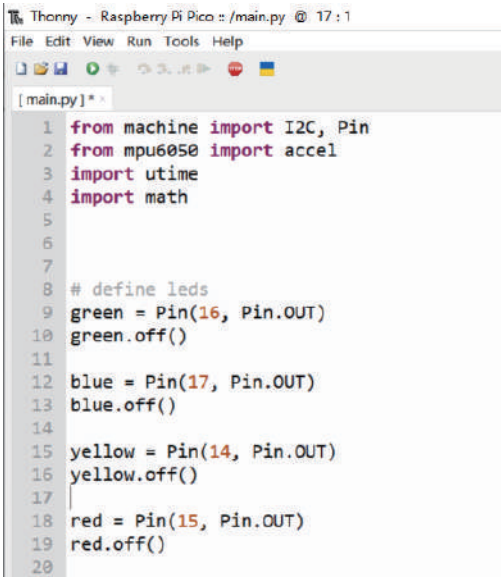


2. CODING

Create a new file in Thonny IDE, and import libraries.



Create LED objects and initializing.



Create an instance of MPU6050 module.

create several functions for calculate the raw data comes from MPU6050 module.

Thonny - Raspberry Pi Pico :: /main.py @ 17 : 1

File Edit View Run Tools Help

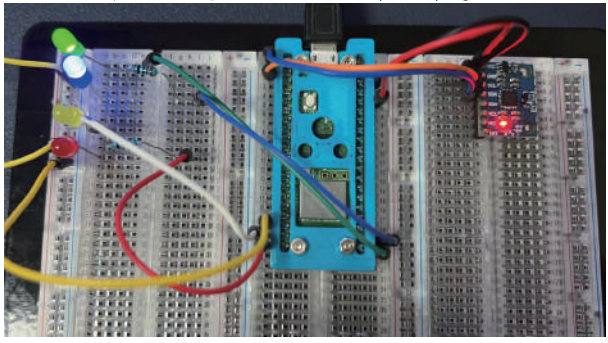
[main.py] *

```
18 red = Pin(15, Pin.OUT)
19 red.off()
20
21 # Initializing MPU6050
22 imu_bus = I2C(0, sda=Pin(0), scl=Pin(1), freq=200000)
23 imu = accel(imu_bus)
24
25 # mapping function
26 def _mapping(x, in_min, in_max, out_min, out_max):
27     return (x - in_min) * ( out_max - out_min) / (in_max - in_min) + out_min
28
29 def dist(a,b):
30     return math.sqrt((a*a)+(b*b))
31
32 def get_y_rotation(x,y,z):
33     radians = math.atan2(x,dist(y,z))
34     return -math.degrees(radians)
35
36 def get_x_rotation(x,y,z):
37     radians = math.atan2(y, dist(x, z))
38     return math.degrees(radians)
39
```

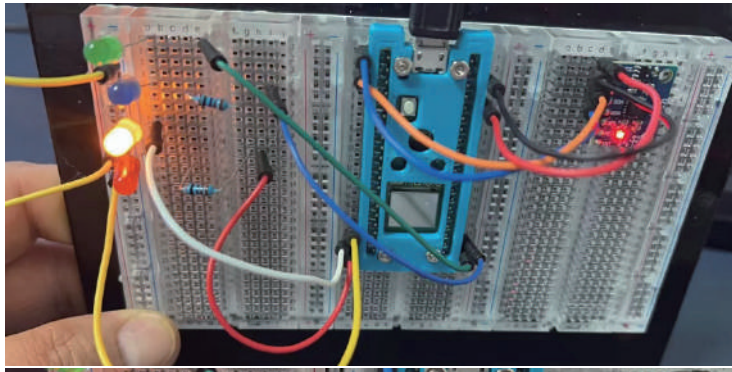
In a loop, We obtain the original data and obtain the current radian through atan2 in the math library, then calculate the angle, and then judge the attitude change by the size of the angle, so as to light up the LED light.

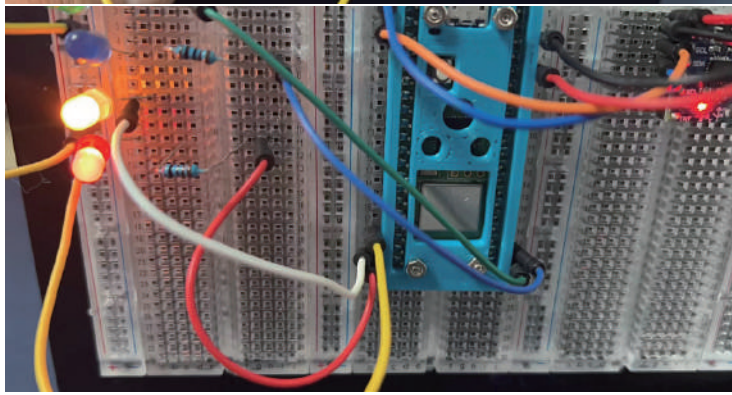
3. TESTING

When the experimental platform is stationary and lying flat on the table:



When the inclination of the experimental platform is greater than +/-15 degrees:





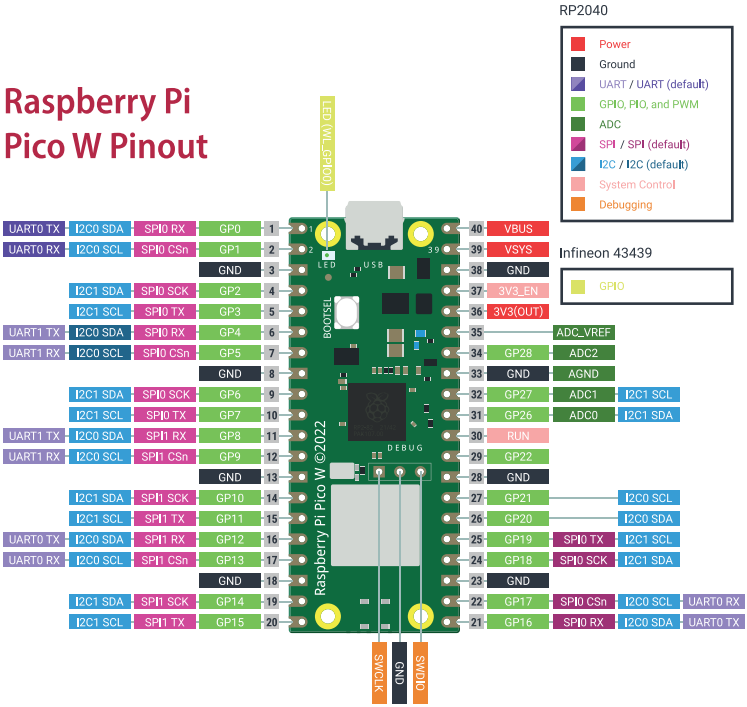
The yellow led will lights up and the red one will blink.

Ok, That' s all, thanks for watching and if you have any issue with those demonstration please kindly contact us and all of the demo code can be found in our GitHub website: https://github.com/geeekpi/picowkit

More information please visit:
https://wiki.52pi.com/index.php?title=EP-0159

INNOVATING THE FUTURE OF SCIENCE AND TECHNOLOGY

Raspberry Pi Pico W Pinout



Email: tech@52pi.net