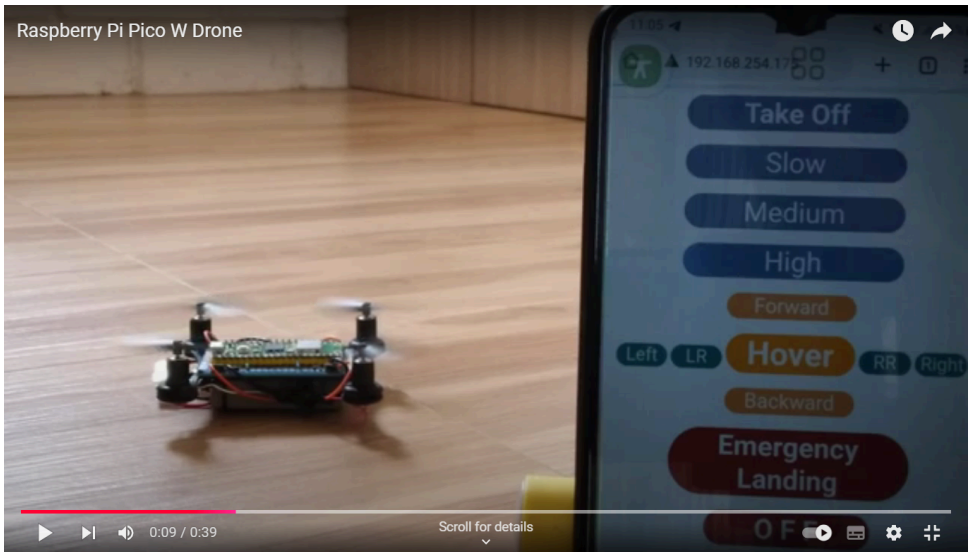
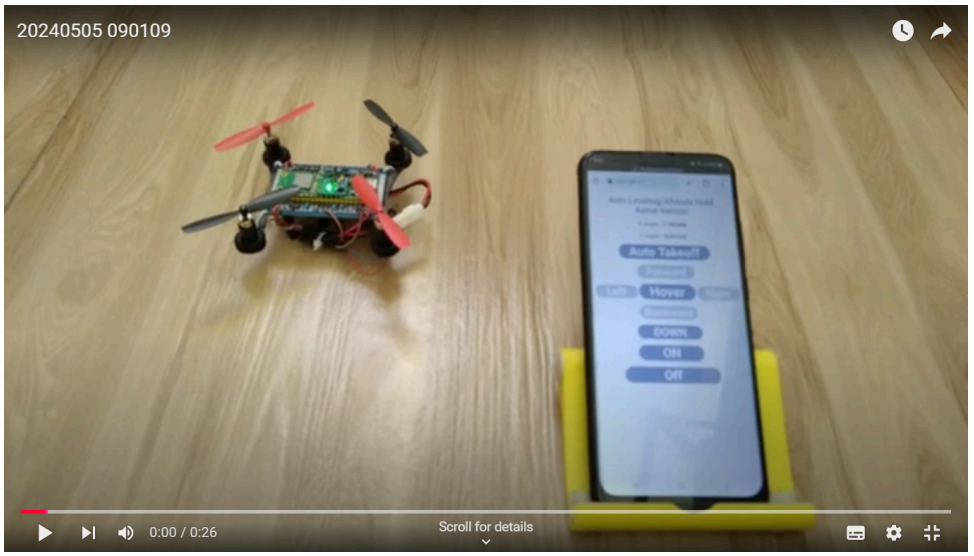
Raspberry Pi Pico W Drone
(Woobu Autonomous Drone)
Tutorial
By Ellen Red
(ellenrapps)



Prototype 1



Prototype 2

Introduction:

This tutorial is all about the Raspberry Pi Pico W Drone based on my YouTube video and my GitHub repository. I named this Raspberry Pi Pico W Drone the Woobu Autonomous Drone.

Woobu Autonomous Drone uses Raspberry Pi Pico W and MPU6050 as the main hardware, and 95.6% of the code is Micropython, 2.4% HTML, and 2% JavaScript.

As shown in my videos (https://www.youtube.com/watch?v=ECWFBZKwFYE and https://www.youtube.com/watch?v=JJ19gocz3pc), the current setups have hardware issues. Despite this limitation, the completed code published on my GitHub page is functional and can be integrated into your own drone hardware setup. The completed code's license is "The Unlicense", which means that it is a free and unencumbered software released into the public domain. This Raspberry Pi Pico W Drone (Woobu Autonomous Drone) tutorial is, however, copyrighted but free.

The code allows a drone to autonomously fly itself, autonomously hover, autonomously balance itself, and autonomously land itself. The code can read X and Y angles faster than a blink of an eye and display these X, Y angles on your mobile browser. On your mobile browser, these buttons are provided: Auto Takeoff, Forward, Left, Right, Backward, Hover, Down, On, and Off. The code purposely disabled the Forward, Left, Right, and Backward buttons.

The following are the future milestones for Woobu Autonomous Drone:

1) MPU6050's gyroscope and accelerometer shall be used to measure altitude and distance, indoor and outdoor
2) MPU6050 shall not only be used in autonomous drone setups, but also in other flying machine setups

Hardware Setup (Warning! For this tutorial, don't connect the drone battery. When you figure out this tutorial, be free to experiment.)

To follow this tutorial, you need these 6 things:

1) Raspberry Pi Pico W
2) MPU6050
3) Laptop or desktop
4) Female jumper wires (4 pcs)
5) USB cable wire
6) Smartphone

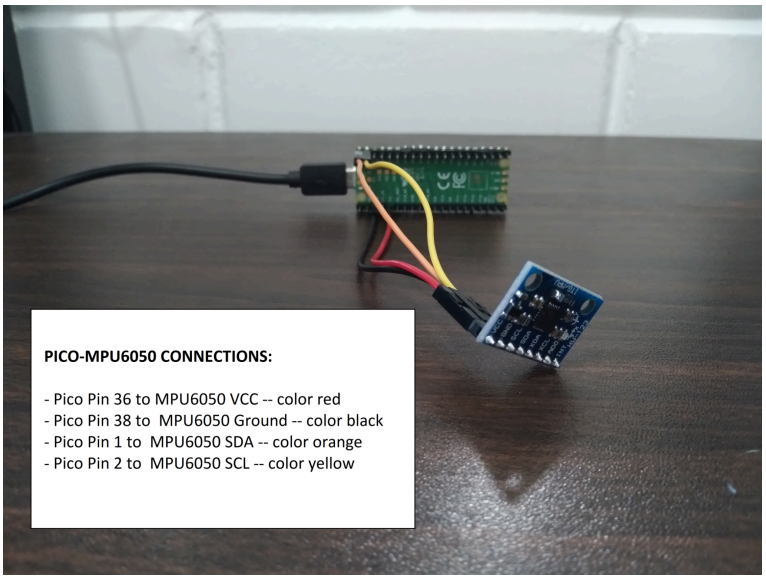


Figure 1

Figure 1 shows how the Pico and MPU should be connected.

- Connect Pico Pin 36 to MPU VCC
- Connect Pico Pin 38 to MPU Ground
- Connect Pin 1 to MPU SDA
- Connect Pin 2 to MPU SCL

It is important to note that for this tutorial, the distinction between Pin and GPIO should be made. Figure 2 below shows the difference between GPIO and Pin. See the official [Raspberry Pi Pico Pinout].

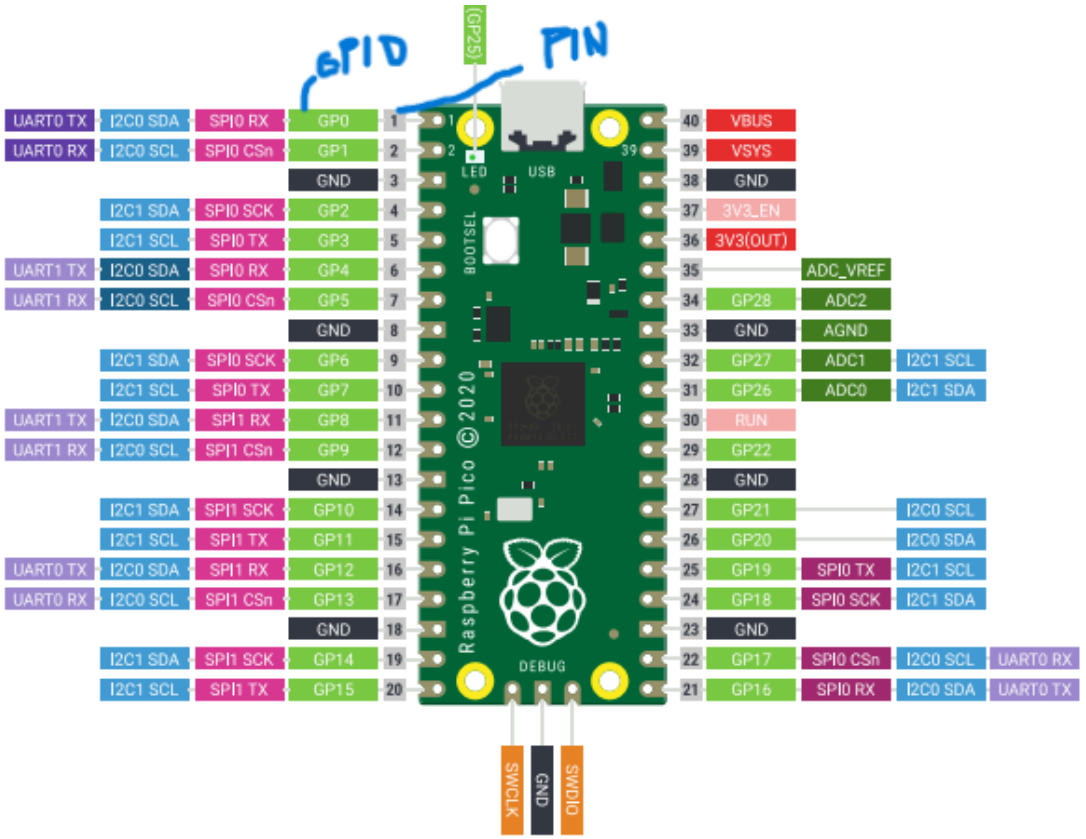


Figure 2

Connect the Pico to your desktop or laptop via a USB cable wire. To connect Pico to your smartphone, complete the software setup first. At the end of the software setup and code explanation, I will explain how to connect the Pico to your smartphone.

Software Set-up

For the software setup, we will be using the Thonny IDE to write our Micropython, JavaScript, and HTML code.

Figure 3

Configure Thonny to connect to Pico. Raspberrypi.org has a tutorial for this.

Next, copy all code from my GitHub repository to Thonny.

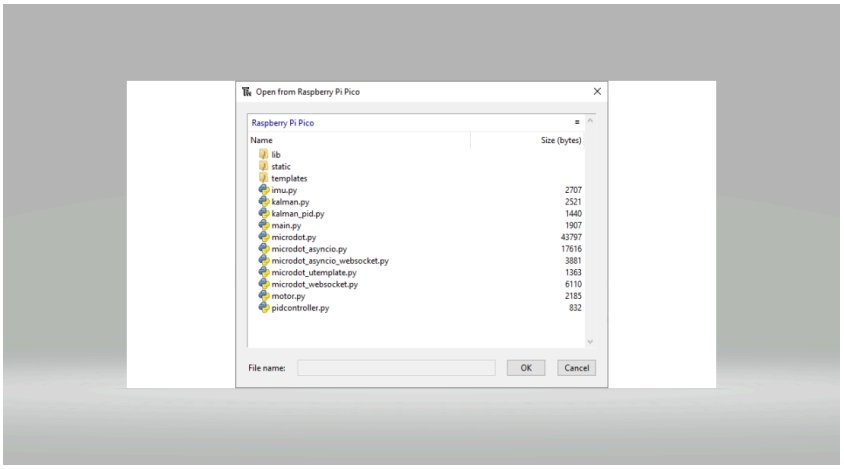


Figure 4

Figure 4 shows you how the code should be organized inside Thonny. The lib folder is automatically created by Thonny when you install packages. For this project, install these 2 packages:

1) ujson: an ultra-fast JSON encoder and decoder for Micropython
2) utemplate: very lightweight, memory-efficient template engine which compiles to Micropython

Install ujson and utemplate via Thonny->Tools->Manage packages. Then create 2 folders, one named static, and another named as templates. Inside the static folder, create a JavaScript file named control.js and copy the code from GitHub. Inside the templates folder, create an HTML file named index.html and copy the code from GitHub.

For the Woobu Autonomous Drone project, I used the following external libraries:

1) Microdot library by Miguel Grinberg: https://blog.miguelgrinberg.com/post/microdot-yet-another-python-web-framework
2) Kalman Filter library by Roche Christopher: https://github.com/rocheparadox/Kalman-Filter-Python-for-mpu6050/blob/master/Kalman.py
3) PID Controller library by Mark Tomczak: https://github.com/shivaay1/Equsant-Self-Balancing-Robot-Python/blob/master/pidcontroller.py

Microdot allows you to create a web application with a simple and lightweight syntax. Microdot's late 2023/early 2024 version is used here.

Kalman is an algorithm that combines several sources of uncertainty to provide a more accurate estimate of the state of the system, here the X and Y angles of the drone.

PID, which stands for Proportional-Integral-Derivative, algorithm ensures that the drone will stabilize itself to the target. When Kalman and PID are combined, Kalman estimates the drone's state variables, and PID uses these estimates to generate the control signal, resulting in increased accuracy.

The Kalman library by Roche Christopher is the Python (also applicable for Micropython) implementation of Kalman. The PID library by Mark Tomczak is the Python (also applicable to Micropython) implementation of PID.

Copy the Microdot, Kalman, and PID libraries from my GitHub repository, as I tested these versions numerous times. New versions of the libraries may not be compatible with my own code. After you become familiar with the code, you can use and test the latest versions of these libraries.

The Code

Here, we will dig into the code that I personally developed.

Given that you have set up everything, from hardware to software, the Raspberry Pi Pico W will first check the main.py file. For testing, that is, with the current hardware setup for this tutorial, it is best to name this file main1.py. For production, name this file main.py.

main.py

```
# Network enables the use of the wireless network features on the
Raspberry Pi Pico W.
import network
from microdot_asyncio import Microdot, Response, send_file
from microdot_asyncio_websocket import with_websocket
from microdot_utemplate import render_template
import ujson
# File to control drone motors
import motor
# File to get MPU's raw accelerometer and gyroscope values
import imu
# Implement both Kalman and PID
import kalman_pid

#----Start network code----
# Insert your own network ssid
ssid = " "
# Insert your own network password
password = " "

# There are 3 network.WLAN options. Access Point(AP) option is
chosen here. AP creates wifi as an access point for other devices,
```

```
like your smartphone, to connect to. With the AP option, you do
not need your traditional wifi connection from your home internet.
ap = network.WLAN(network.AP_IF)
# Connect to Pico's wifi using ssid and password
ap.config(essid=ssid, password=password)
# Turn on Pico's wifi
ap.active(True)


while ap.active == False:
    pass


print('Connection successful')
print(ap.ifconfig())
#----End network code----


#----Start web application (We are basically building a web app
here using Microdot.)----
# Initialize MicroDot class and assign it as an app variable
app = Microdot()
# Set the default content type to 'text/html'
Response.default_content_type = 'text/html'

# This defines a route for a specific URL. In this case, the URL
is the root URL of the application, in particular, the single
forward slash (/).
@app.route('/')

# Async means it can run concurrently with other tasks using the
Micropython asyncio library.
# The def index(request) means that the function named index will
be called when a client requests the root URL ("/") of the web
application.
# The request parameter represents the incoming HTTP request.
# The index.html is the name of the HTML template file to be
rendered.
async def index(request):
    return render_template('index.html')

# This is the part of the code that allows the display of the X
```

```
and Y angles on your mobile browser.
@app.route('/update')
def update(request):
    # Call filter_pid function from kalman_pid file
    values_kalman_pid = kalman_pid.filter_pid()
# The filter_pid function returns 4 values. The 4 values are
k_angle_x, k_angle_y, PIDx, and PIDy. The first value is Kalman's
x angle, and the second value is Kalman's y angle. The third
value, representing x angle and fourth value, representing the y
angle, are the results after combining both the Kalman and PID
calculations. Only the first and second values are called here,
namely, Kalman X angle and Kalman Y angle.
    kalman_x_angle = values_kalman_pid[0]
    kalman_y_angle = values_kalman_pid[1]

# Initialize a variable named data and assign it a dictionary with
the Kalman X and Kalman Y values
    data = {
    "kalman_x_angle" : kalman_pid_x_angle
    "kalman_y_angle" : kalman_pid_y_angle
    }

# This converts data into a JSON-formatted string. This data is
fetched using the constant variable named getSensorReadings from
the control.js file. In particular, search for these under
getSensorReadings:
document.getElementById('x_angle').innerHTML
=response.kalman_x_angle;
document.getElementById('y_angle').innerHTML
=response.kalman_y_angle;


    response = ujson.dumps(data)
    return response


@app.route("/ws")
# WebSocket is a real-time communication protocol that enables a
persistent connection between a client (your mobile browser) and a
server (Raspberry Pi Pico W server).
@with_websocket
async def drone_commands(request, ws):
```

```
    while True:
        websocket_message = await ws.receive()
# If "Auto Takeoff" is clicked on your mobile browser, it calls
the auto_takeoff_button function from the motor file, enabling the
drone to automatically take off.
        if "auto_takeoff" in websocket_message:
            motor.auto_takeoff_button()

# If "Hover" is clicked on your mobile browser, it calls the
hover_button() function from the motor file, enabling the drone's
motors to hover.
        if "hover" in websocket_message:
            motor.hover_button()

# If "DOWN" is clicked on your mobile browser, it calls the
down_button() function from the motor file, enabling the drone's
motors to automatically go down.
         if "down" in websocket_message:
            motor.down_button()

# If "ON" is clicked on your mobile browser, it calls the
on_manual_button() function from the motor file, turning on the
drone's motors.
        if "on_manual" in websocket_message:
            motor.on_manual_button()

# If "Off" is clicked on your mobile web browser, it calls the
off_manual_button() function from the motor file, turning off the
drone's motors.
        if "off_manual" in websocket_message:
            motor.off_manual_button()
# The await keyword means that the execution of the coroutine is
suspended until the sending operation is complete. Once a message
is sent, the coroutine resumes execution. A coroutine is a special
type of function that can suspend and resume its execution,
allowing other tasks to run concurrently.
        await ws.send('')

# Static route is for serving static files such as CSS and
JavaScript.
@app.route("/static/<path:path>")
def static(request, path):
```

```
    if ".." in path:
        # Directory traversal is not allowed.
        return "Not found", 404
    return send_file("static/" + path)


def start_server():
    print('Starting microdot app')
    try:
        app.run(port=80)
    except:
        app.shutdown()

# Start Microdot app
start_server()
#----End web application----
```

control.js

```
// This automatically makes the opacity of the button that you
click to 100% and turns all other buttons to 0.5 or 50% opacity.
var buttonSelector = document.querySelectorAll('button');
buttonSelector.forEach(function (el){
    el.addEventListener("click", function(ev){
        buttonSelector.forEach(function(button) {
button.style.opacity = '0.5' })
        this.style.opacity = 1
    })
});


// Use the fetch JavaScript API to show on your mobile browser the
Kalman readings for X and Y angles every 10 milliseconds.
document.addEventListener("DOMContentLoaded", function(event) {
    const getSensorReadings = function() {
        fetch('/update')
            .then((resp) => resp.json())
            .then(function(response) {
```

```
                document.getElementById('x_angle').innerHTML
=response.kalman_x_angle;
                document.getElementById('y_angle').innerHTML
=response.kalman_y_angle;
        });
    }

    getSensorReadings();
    setInterval(getSensorReadings, 10);  // Update every 10
milliseconds
});

// ${location.host} refers to Raspberry Pi Pico W's IP address.
// 192.168.4.1 is the IP address of the Raspberry Pi Pico server.
var targetUrl = `ws://${location.host}/ws`;
var websocket;
window.addEventListener("load", onLoad);

function onLoad() {
  initializeSocket();
}

function initializeSocket() {
  // TargetUrl is the URL of the WebSocket server to connect to.
  // It uses 4 schemes, one of them is ws, which is unencrypted.
  websocket = new WebSocket(targetUrl);
  websocket.onopen = onOpen;
  websocket.onclose = onClose;
  websocket.onmessage = onMessage;
}

// This is triggered when a WebSocket connection is established,
indicating that the connection is ready to send and receive data.
function onOpen(event) {
  console.log("Websocket Open...");
}

// This is triggered when a WebSocket connection is closed.
function onClose(event) {
  console.log("Websocket Close...");
  setTimeout(initializeSocket, 2000);
}
```

```
// This is triggered when a message is received from the server
over the WebSocket connection.
function onMessage(event) {
  console.log("WebSocket message received:", event);
}

// Send data to a server over a WebSocket connection
function sendMessage(message) {
  websocket.send(message);
}

// Send WebSocket message "auto_takeoff" to server
function auto_takeoff() {
    sendMessage(auto_takeoff);
}

// Send WebSocket message "hover" to server
function hover() {
    sendMessage(hover);
}

// Send WebSocket message "down" to server
function down() {
    sendMessage(down);
}

// Send WebSocket message "on_manual" to server
function on_manual() {
    sendMessage(on_manual);
}

// Send WebSocket message "off_manual" to server
function off_manual() {
     sendMessage(off_manual);
}
```

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width,
initial-scale=1">
<link rel="icon" href="data:,">

<!--Start style code-->
<style>
html{font-family: Helvetica; display:inline-block; margin: 0px
auto; text-align: center;}
h1{color: #0F3376; padding: 2vh;}p{font-size: 1rem;}
.button_main{display: inline-block; border: none; border-radius:
75px; color: white; text-decoration: none}
.button_hover{background-color: #4682B4 ; font-size: 36px;
width:39%}
.button_forward_backward{background-color: #4682B4; font-size:
30px; width:40%; opacity:0.5}
.button_left_right{background-color: #4682B4; font-size: 30px;
width:29%; opacity:0.5}
.button_on{background-color: #4682B4; font-size: 34px; width:44%}
.button_off{background-color: #4682B4; font-size: 34px; width:64%}
.button_down{background-color: #4682B4; font-size: 30px;
width:44%}
.button_up{background-color: #4682B4; font-size: 34px; width:64%}
</style>
<!--End style code-->
</head>

<body>
<p><font size="5.7" color=#4682B4><b>Auto Leveling/Altitude
Hold</br>Aerial Vehicle</b></font></p>

<!--Display Kalman's X angle-->
<p>X Angle: <strong><span id='x_angle'>--.-</span></strong></p>
<!--Display Kalman's Y angle-->
<p>Y Angle: <strong><span id='y_angle'>--.-</span></strong></p>

<!--Start button labels-->
<!--Code purposely disabled the Forward, Left, Right, and Backward
buttons-->
<p><button class="button_main button_up"
```

```
onclick="auto_takeoff()"><b>Auto Takeoff</b></button></p>
<p><button class="button_main button_forward_backward"
type="button" disabled><b>Forward</b></button></p>
<p>
<button class="button_main button_left_right" type="button"
disabled><b>Left</b></button>
<button class="button_main button_hover"
onclick="hover()"><b>Hover</b></button>
<button class="button_main button_left_right" type="button"
disabled><b>Right</b></button>
</p>
<p><button class="button_main button_forward_backward"
type="button" disabled><b>Backward</b></button></p>
<p><button class="button_main button_down"
onclick="down()"><b>DOWN</b></button></p>
<p><button class="button_main button_on"
onclick="on_manual()"><b>ON</b></button></p>
<p><button class="button_main button_off"
onclick="off_manual()"><b>Off</b></button></p>
<!--End button labels-->

<!--Link to an external JavaScript file, in this case
static/control.js-->
<script src="static/control.js"></script>

</body>
</html>
```

imu.py

```
# Machine module provides access to Pico's hardware peripherals.
import machine
# Import Pin creates Pin objects to control Pico GPIO pins.
# Import I2C creates an I2C object and interact with I2C devices
connected to Pico's I2C pins.
from machine import Pin, I2C

# Define I2C bus
# Note that sda=machine.Pin(0) means "Connect Pin 1 to MPU SDA" in
```

```
the hardware setup, while scl=machine.Pin(1) means "Connect Pin 2
to MPU SCL" in the hardware setup.
i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1))

# Device address on the I2C bus
MPU6050_ADDR = 0x68

# PWR_MGMT_1 memory address
MPU6050_PWR_MGMT_1 = 0x6B

# Accelerometer and Gyroscope's high and low registers for each
axis
MPU6050_ACCEL_XOUT_H = 0x3B
MPU6050_ACCEL_XOUT_L = 0x3C
MPU6050_ACCEL_YOUT_H = 0x3D
MPU6050_ACCEL_YOUT_L = 0x3E
MPU6050_ACCEL_ZOUT_H = 0x3F
MPU6050_ACCEL_ZOUT_L = 0x40
MPU6050_GYRO_XOUT_H = 0x43
MPU6050_GYRO_XOUT_L = 0x44
MPU6050_GYRO_YOUT_H = 0x45
MPU6050_GYRO_YOUT_L = 0x46
MPU6050_GYRO_ZOUT_H = 0x47
MPU6050_GYRO_ZOUT_L = 0x48

# This refers to the accelerometer's LSB/g (least significant bits
per gravitational force) sensitivity. The value can be changed to
suit your hardware setup. Refer to the MPU6050 manual.
MPU6050_LSBG = 16384.0

# This refers to the gyroscope's LSB/g sensitivity. The value can
be changed to suit your hardware setup. Refer to the MPU6050
manual.
MPU6050_LSBDS = 131.0


# Set all bits in the PWR_MGMT_1 register to 0
def mpu6050_init(i2c):
    i2c.writeto_mem(MPU6050_ADDR, MPU6050_PWR_MGMT_1, bytes([0]))


# Combine the accelerometer and gyroscope's high and low registers
```

```
def combine_register_values(h, l):
    if not h[0] & 0x80:
        return h[0] << 8 | l[0]
    return -((h[0] ^ 255) << 8) |  (l[0] ^ 255) + 1


# Get accelerometer values
def mpu6050_get_accel(i2c):
    mpu6050_init(i2c)
    accel_x_h = i2c.readfrom_mem(MPU6050_ADDR,
MPU6050_ACCEL_XOUT_H, 1)
    accel_x_l = i2c.readfrom_mem(MPU6050_ADDR,
MPU6050_ACCEL_XOUT_L, 1)
    accel_y_h = i2c.readfrom_mem(MPU6050_ADDR,
MPU6050_ACCEL_YOUT_H, 1)
    accel_y_l = i2c.readfrom_mem(MPU6050_ADDR,
MPU6050_ACCEL_YOUT_L, 1)
    accel_z_h = i2c.readfrom_mem(MPU6050_ADDR,
MPU6050_ACCEL_ZOUT_H, 1)
    accel_z_l = i2c.readfrom_mem(MPU6050_ADDR,
MPU6050_ACCEL_ZOUT_L, 1)

    accel_x = combine_register_values(accel_x_h, accel_x_l) /
MPU6050_LSBG
    accel_y = combine_register_values(accel_y_h, accel_y_l) /
MPU6050_LSBG
    accel_z = combine_register_values(accel_z_h, accel_z_l) /
MPU6050_LSBG
    return accel_x, accel_y, accel_z


# Get gyroscope values
def mpu6050_get_gyro(i2c):
    mpu6050_init(i2c)
    gyro_x_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_GYRO_XOUT_H,
1)
    gyro_x_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_GYRO_XOUT_L,
1)
    gyro_y_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_GYRO_YOUT_H,
1)
    gyro_y_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_GYRO_YOUT_L,
1)
```

```
    gyro_z_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_GYRO_ZOUT_H,
1)
    gyro_z_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_GYRO_ZOUT_L,
1)

    gyro_x = combine_register_values(gyro_x_h, gyro_x_l) /
MPU6050_LSBDS
    gyro_y = combine_register_values(gyro_y_h, gyro_y_l) /
MPU6050_LSBDS
    gyro_z = combine_register_values(gyro_z_h, gyro_z_l) /
MPU6050_LSBDS
    return gyro_x, gyro_y, gyro_z
```

kalman_pid.py

```
import machine, time, math
from machine import Pin, I2C
from time import sleep
import imu
from kalman import KalmanAngle
from pidcontroller import PIDController

# Connect Pico Pin 1 to MPU6050 SDA
# Connect Pico Pin 2 to MPU6050 SCL
i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1))
# Instantiate a Kalman object
kalman_x = KalmanAngle()
kalman_y = KalmanAngle()


def acceleration_radius_to_degrees():
    while True:
# Mathematical formula for converting radius to degrees
     radius_to_degrees = 180/3.14159
# Read the accelerometer values
     accel_x, accel_y, accel_z = imu.mpu6050_get_accel(i2c)
# Convert accelerometer values from radius to degrees
     ax_angle = math.atan(accel_y/math.sqrt(math.pow(accel_x,2) +
math.pow(accel_z, 2)))*radius_to_degrees
        ay_angle =
math.atan((-1*accel_x)/math.sqrt(math.pow(accel_y,2) +
math.pow(accel_z, 2)))*radius_to_degrees
        return ax_angle, ay_angle


def filter_pid():
    while True:
# Calculate time element change, here known as dt
        t_now = time.ticks_ms()
        last_read_time = 0.0
        dt = (t_now - last_read_time)/1000.0
# Call acceleration_radius_to_degrees function which returns 2
values: ax_angle, ay_angle
        ax_angle, ay_angle = acceleration_radius_to_degrees()
# Call the gyroscope values from mpu6050_get_gyro(i2c) function
```

```
from imu file
        gyro_x, gyro_y, gyro_z = imu.mpu6050_get_gyro(i2c)

# GetAngle function from kalman file has 3 parameters: 1)
accelerometer angle, 2) gyroscope angle, and 3) time element
change.
        k_angle_x = kalman_x.getAngle(ax_angle, gyro_x, dt)
        k_angle_y = kalman_y.getAngle(ay_angle, gyro_y, dt)
        x, y = k_angle_x, k_angle_y
        newx = x
        newy = y
# PIDController class from pidcontroller file has 3 parameters: P
for Proportional, I for Integral, D for Derivative. Change the
values of the PID to suit your hardware setup.
        PID = PIDController(P=0.2, I=0.02, D=1)

# The step function from pidcontroller file takes one parameter,
that is, the current value of newx, which is the current value of
kalman x for PIDx and current value of newy, which is the value of
kalman y angle for PIDy. This part is the combined calculation of
Kalman and PID.
        PIDx = PID.step(newx)
       PIDy = PID.step(newy)
        result = (k_angle_x, k_angle_y, PIDx, PIDy)
        return(result)
```

motor.py

```
from machine import Pin, PWM
import time
import imu
import kalman_pid

# Connect motor 1 to Pico GPIO 22
motor1 = PWM(Pin(22))
# Connect motor 2 to Pico GPIO 18
motor2 = PWM(Pin(18))
# Connect motor 3 to Pico GPIO 12
motor3 = PWM(Pin(12))
```

```
# Connect motor 4 to Pico GPIO 4
motor4 = PWM(Pin(4))
# Frequency is set to 30.
freq = 30
# Set the initial duty cycle of the PWM signal to 0, so that when
you connect a battery to your completed hardware setup the default
speed is 0.
duty_u16 = 0


# Speed is set to 5%, 0 being off and 100 is full speed.
def slow():
    motor1.duty_u16(3251) # 3251 is equals to 5% speed
    motor2.duty_u16(3251)
    motor3.duty_u16(3251)
    motor4.duty_u16(3251)


# Speed is set to 90%
def high():
    motor1.duty_u16(58522) # 58522 is equals to 90% speed
    motor2.duty_u16(58522)
    motor3.duty_u16(58522)
    motor4.duty_u16(58522)


# Speed is set to 50%
def hover():
    motor1.duty_u16(32512) # 32512 is equals to 50% speed
    motor2.duty_u16(32512)
    motor3.duty_u16(32512)
    motor4.duty_u16(32512)


# Speed is set to 0%.
def off():
    motor1.duty_u16(0) # 0 is equals to 0% speed
    motor2.duty_u16(0)
    motor3.duty_u16(0)
    motor4.duty_u16(0)

# Automatic takeoff button function
```

```
def auto_takeoff_button():
    start = time.ticks_ms()
    while True:
# Call the filter_pid function from kalman_pid file. This function
returns 4 values: 1) k_angle_x, 2) k_angle_y, 3) PIDx, 4) PIDy.
Only the third and fourth values are called here.
        values_kalman_pid = kalman_pid.filter_pid()
        pid_x = values_kalman_pid[2]
        pid_y = values_kalman_pid[3]


# If pid_x is less than or equal to -4, and less than or equal to
1 and if pid_y is less than or equal to -6, and less than or equal
to 1, call high function which has a 90% speed, and print "Up High
Plain" on Thonny Shell. After 200 milliseconds, call hover
function which has a 50% speed and print "Hover High Time", and
break from the loop. Else, call function hover and print "Up High
Hover Else" and break from the loop. Adjust the numbers to suit
your own hardware setup.
        if -4 <= pid_x <= 1 and -6 <= pid_y <= 1:
            high()
            print('Up High Plain')

            if time.ticks_diff(time.ticks_ms(), start) > 200:
                hover()
                print('Hover High Time')
                break

        else:
            hover()
            print('Up High Hover Else')
            break



# Call the hover function which has a speed of 50% and print
"Hover"
def hover_button():
    hover()
    print('Hover')
```

```
def down_button():
    start = time.ticks_ms()
    while True:
        values_kalman_pid = kalman_pid.filter_pid()
        pid_x = values_kalman_pid[2]
        pid_y = values_kalman_pid[3]

# If pid_x is less than or equal to -4, and less than or equal to
1 and if pid_y is less than or equal to -6, and less than or equal
to 1, call slow function which has a 5% speed, and print "Down
Plain". After 200 milliseconds, call off function which has a 0%
speed and print "Down Off Time" and break from the loop. Else,
call function hover and print "Down Hover Else" and break from the
loop. Adjust the numbers to suit your own hardware setup.

     if -4 <= pid_x <= 1 and -6 <= pid_y <= 1:
            slow()
            print('Down Plain')

            if time.ticks_diff(time.ticks_ms(), start) > 200:
                off()
                print('Down Off Time')
                break

        else:
            hover()
            print('Down Hover Else')
            break


# Call the slow function, which has a speed of 5% and the word
"on" is printed on Thonny Shell.
def on_manual_button():
    slow()
    print('On')


# Off function, which has a speed of 0% and the word "off" is
printed on Thonny Shell
def off_manual_button():
    off()
    print('Off')
```

That’s it for the code that I originally developed.

Testing, Testing: Connect Pico to Smartphone

This is the last part of the tutorial. Make sure to follow the hardware and software setups.

To connect the Pico to your smartphone, make sure you set your own ssid and password in the main.py file.

For my demonstration, I set up my ssid to tw82>29&$

On Thonny, open the main.py file and then click the green button called Run Current Script. On your smartphone, go to connection, then click Wi-fi, and toggle to On. On the available networks, tw82>29&$ should appear.

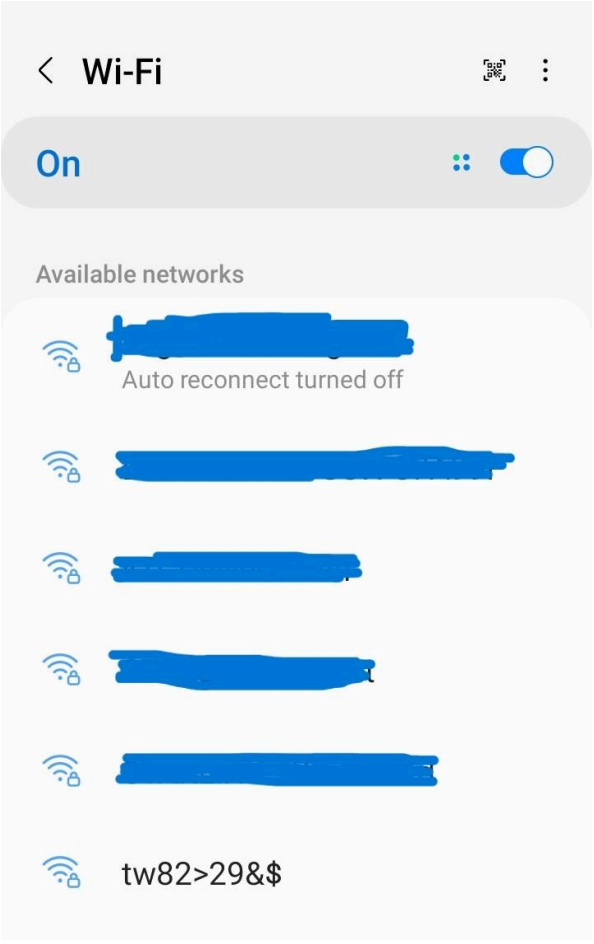


Figure 5

Click on the tw82>29&$ network and you will be asked to enter your chosen password.

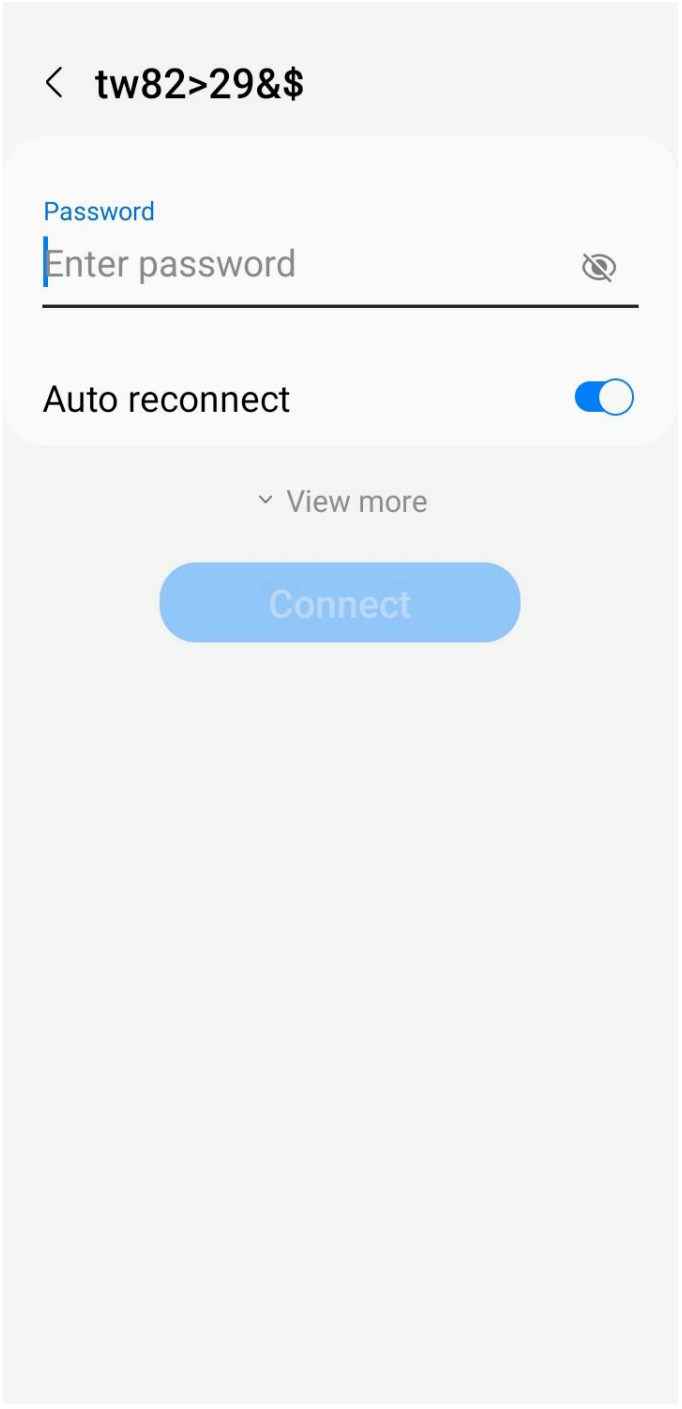


Figure 6

Once you are connected to the tw82>29&$ network, on Thonny Shell, this should appear:

Figure 7

Notice the 192.168.4.1 on Thonny Shell. Enter 192.168.4.1 on your smartphone's web browser and click the "On" button. This should appear on your mobile browser:

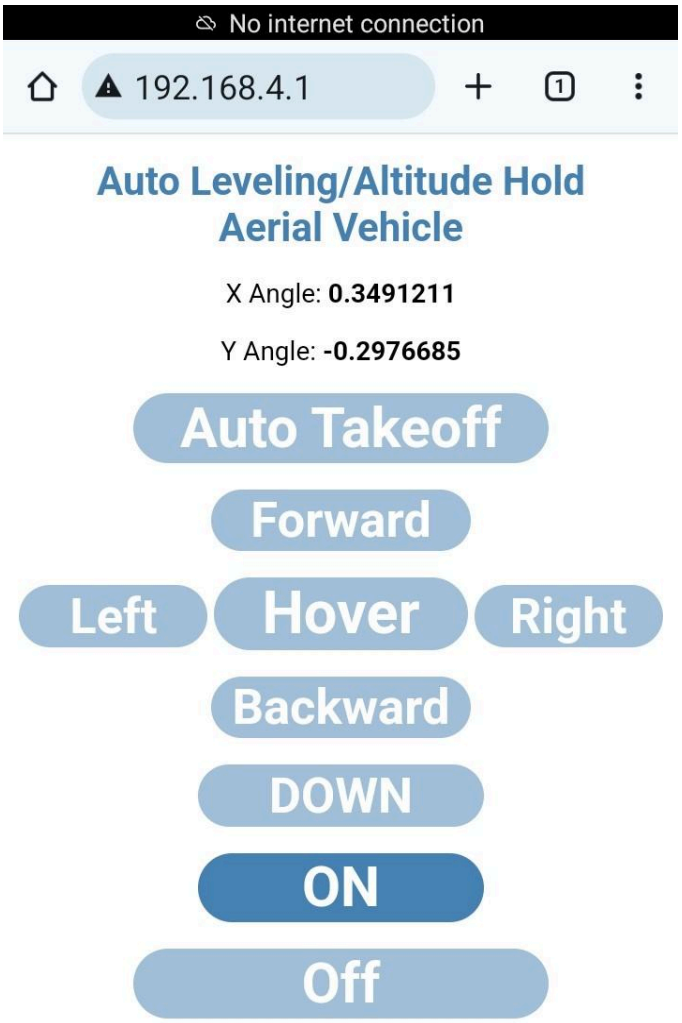


Figure 8

That’s it.

To know more about my hardware and software development projects, visit my:

Website: https://ellenrapps.com/
GitHub: https://github.com/ellenrapps
YouTube: https://www.youtube.com/@ellenrapps
X.com: https://x.com/ellenrapps

Sources:

1. Raspberry Pi Pico W drone:
https://www.youtube.com/watch?v=ECWFBZKwFYE&themeRefresh=1
https://www.youtube.com/watch?v=JJ19gocz3pc
https://github.com/ellenrapps/Woobu-Autonomous-Drone

2. Raspberry Pi Pico Pinout:
https://datasheets.raspberrypi.com/pico/Pico-R3-A4-Pinout.pdf

3. Connect Pico to Thonny:
https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/2

4. MPU6050:
https://www.youtube.com/watch?v=2ufkfd-oFrY
https://www.youtube.com/watch?v=NkBL_VxJ77g

5. Microdot:
https://blog.miguelgrinberg.com/post/microdot-yet-another-python-web-framework

6. Kalman Filter:
https://github.com/rocheparadox/Kalman-Filter-Python-for-mpu6050/blob/master/Kalman.py
https://github.com/nihalpasham/micropython_sensorfusion

7. Proportional-Integral-Derivative (PID):
https://github.com/shivaay1/Equsant-Self-Balancing-Robot-Python/blob/master/pidcontroller.py