Adding a Bubble Level to the MEMENTO

Created by Melissa LeBlanc-Williams

https://learn.adafruit.com/adding-a-bubble-level-to-the-memento

Last updated on 2026-08-23 07:31:20 PM UTC

Table of Contents

Overview 3
• Parts
Install CircuitPython 4
• CircuitPython Quickstart
Code the Bubble Level 7
• How the Code Works
Use the Bubble Level 15

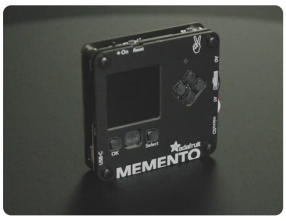
Overview

Have you ever tried taking a top-down photo only to find that you were off by just a little bit and the subject suddenly looks a bit warped? By using the MEMENTO Python-programmable camera with its built-in accelerometer, it's easy to add a bubble level that will help you keep the photos straight and level and get that perfect photo.

The bubble level not only works with horizontal photos so that the camera isn't rotated slightly while taking photos, but also top-down photos so that you can get a perfectly level shot.

This guide uses the MEMENTO Basic Camera example (https://adafru.it/18ef) as a base and builds on top of that. It makes use of a little bit of space on the bottom bar between the mode and image type.

Parts

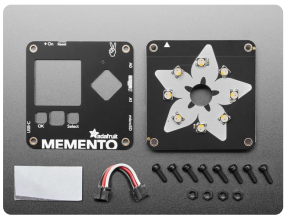


Featured

MEMENTO - Python Programmable DIY Camera - Bare Board

Make memories, or just a cool camera-based project, with Adafruit's MEMENTO Camera Board. It's a development board with everything you need to create...

https://www.adafruit.com/product/5420

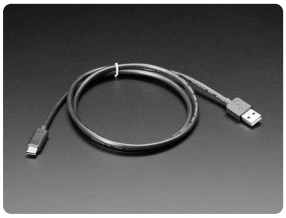


Featured

Adafruit MEMENTO Camera Enclosure & Hardware Kit

Once you've picked up your MEMENTO Camera and you're ready to take it out into the world, here is a chic and minimalist enclosure that will look great on the...

https://www.adafruit.com/product/5843



USB Type A to Type C Cable - approx 1 meter / 3 ft long

As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...

https://www.adafruit.com/product/4474

Install CircuitPython

CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your board.

Text emphasized with a yellow exclamation:	Make sure you use CircuitPython 9.0.0 final or later. CircuitPython 9.0.0-beta.1 and earlier have a bug that can corrupt the fileystem.

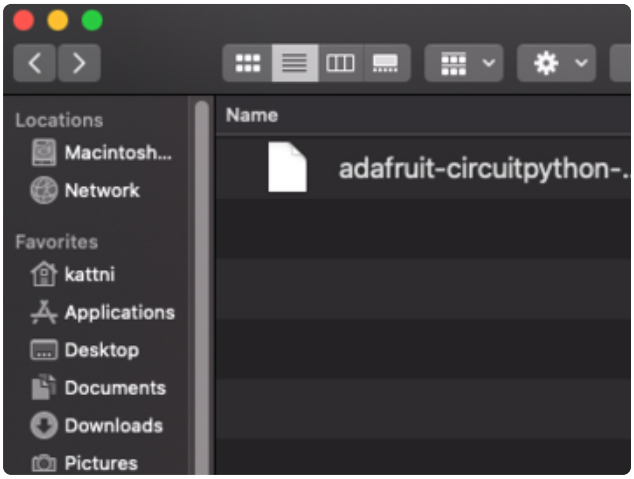
```
import storage
storage.erase_filesystem()
```

Your board will reboot after running this.

https://adafru.it/18e6

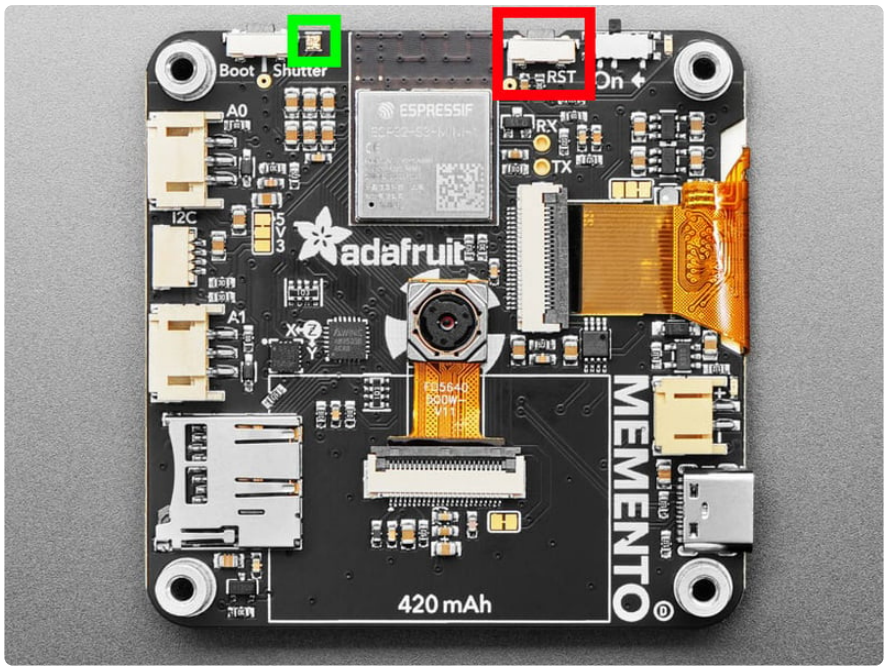
Text emphasized with a yellow exclamation:	As of CircuitPython 9, you'll need to create a folder called "sd" on your CIRCUITPY drive to mount the microSD card, if it's not already there.

https://adafru.it/19ei



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.



Plug your board into your computer, using a known-good data-sync cable, directly, or via an adapter if needed.

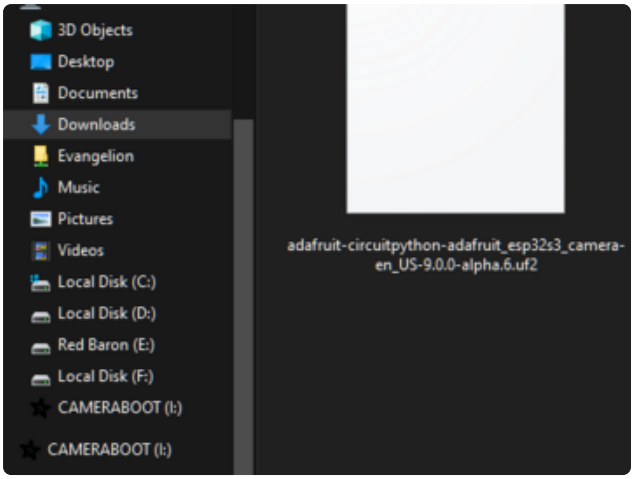
Double-click the **reset** button (highlighted in red above), and you will see the **RGB status LED(s)** turn green (highlighted in green above). If you see red, try another port, or if you're using an adapter or hub, try without the hub, or different adapter or hub.

For this board, tap reset and wait for the LED to turn purple, and as soon as it turns purple, tap reset again. The second tap needs to happen while the LED is still purple.

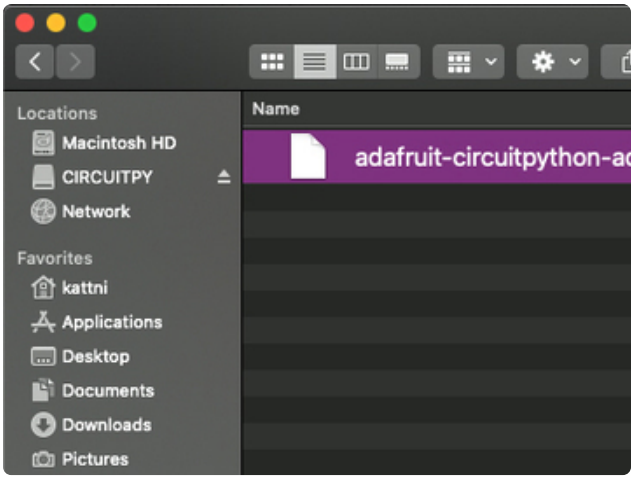
If you do not see the LED turning purple, you will need to reinstall the UF2 bootloader. See the **Factory Reset** page in this guide for details.

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **CAMERABOOT**. Drag the **adafruit-circuitpython-adafruit_esp32s3_camera-etc.uf2** file to **CAMERABOOT**.



The **BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

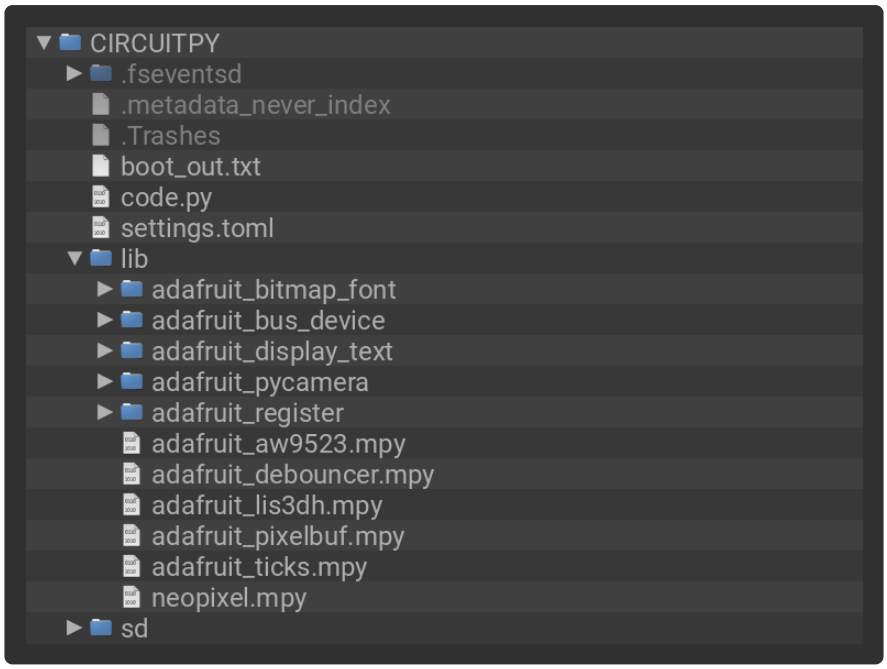
That's it!

Code the Bubble Level

The project uses a specific set of CircuitPython libraries, and the main **code.py** file. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

We can do this in one go. In the example code block below, click the **Download Project Bundle** button to download the necessary libraries and the **code.py** file as a zip file. Extract the contents of the zip file, open the directory **MEMENTO/Memento_Bubble_Level/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:




```
# SPDX-FileCopyrightText: Copyright (c) 2023 john park for Adafruit Industries
# SPDX-FileCopyrightText: 2026 Melissa LeBlanc-Williams for Adafruit Industries
#
# SPDX-License-Identifier: MIT
"""Camera preview with an accelerometer-driven bubble level for Adafruit MEMENTO."""

import math
import time

import adafruit_pycamera
import displayio
import vectorio

pycam = adafruit_pycamera.PyCamera()

# Settings
PHOTO_RESOLUTION = 8  # 0-12 preset resolutions:
#                     0: 240x240, 1: 320x240, 2: 640x480, 3: 800x600, 4: 1024x768,
#                     5: 1280x720, 6: 1280x1024, 7: 1600x1200, 8: 1920x1080, 9: 2048x1536,
#                     10: 2560x1440, 11: 2560x1600, 12: 2560x1920

LED_LEVEL = 0 # 0-4 preset brightness levels

FULL_SCALE_DEGREES = 8.0  # Bubble reaches the end at this much error
GREEN_DEGREES = 1.5
AMBER_DEGREES = 4.0
SMOOTHING = 0.18  # 0.0 = frozen, 1.0 = no filtering

# The gap between these angles prevents AUTO mode from flickering.
DOWN_ENTER_DEGREES = 35
DOWN_EXIT_DEGREES = 45

# Level geometry within the 240 x 32 band below the live preview.
BAR_WIDTH = 64
BAR_HEIGHT = 25
BAR_BOTTOM_MARGIN = 2
```

```
BUBBLE_RADIUS = 6

# Constants and Calculated Values
AUTO = 0
DOWN = 1
HORIZON = 2

GREEN = 0x00FF88
AMBER = 0xFFC43D
RED = 0xFF4D5A

BAR_LEFT = pycam.display.width // 2 - BAR_WIDTH // 2
BAR_TOP = pycam.display.height - BAR_HEIGHT - BAR_BOTTOM_MARGIN
BAR_CENTER_X = BAR_LEFT + BAR_WIDTH // 2
BAR_CENTER_Y = BAR_TOP + BAR_HEIGHT // 2
HORIZONTAL_TRAVEL = BAR_WIDTH // 2 - BUBBLE_RADIUS - 2
VERTICAL_TRAVEL = BAR_HEIGHT // 2 - BUBBLE_RADIUS - 2

def clamp(value, low, high):
    return max(low, min(high, value))

def level_color(error_degrees):
    if error_degrees <= GREEN_DEGREES:
        return GREEN
    if error_degrees <= AMBER_DEGREES:
        return AMBER
    return RED

def update_level(mode, _x_angle, _y_angle, _roll_angle):
    """Move and recolor the retained vector bubble."""
    down_references.hidden = mode != DOWN
    horizon_references.hidden = mode == DOWN

    if mode == DOWN:
        bubble_x = BAR_CENTER_X + int(
            clamp(_x_angle / FULL_SCALE_DEGREES, -1.0, 1.0) * HORIZONTAL_TRAVEL
        )
        bubble_y = BAR_CENTER_Y + int(
            clamp(_y_angle / FULL_SCALE_DEGREES, -1.0, 1.0) * VERTICAL_TRAVEL
        )
        error_amount = max(abs(_x_angle), abs(_y_angle))
    else:
        bubble_x = BAR_CENTER_X + int(
            clamp(_roll_angle / FULL_SCALE_DEGREES, -1.0, 1.0)
            * HORIZONTAL_TRAVEL
        )
        bubble_y = BAR_CENTER_Y
        error_amount = abs(_roll_angle)

    bubble_outline.x = bubble_x
    bubble_outline.y = bubble_y
    bubble_fill.x = bubble_x
    bubble_fill.y = bubble_y
    bubble_palette[0] = level_color(error_amount)

def take_photo(camera):
```

```
    camera.tone(1200, 0.04)
    camera.tone(1600, 0.04)
    try:
        camera.display_message("snap", color=0x00DD00)
        camera.capture_jpeg()
    except TypeError:
        camera.display_message("Capture failed", color=0xFF0000, scale=2)
        time.sleep(0.5)
    except RuntimeError:
        camera.display_message("No SD card", color=0xFF0000, scale=2)
        time.sleep(0.5)
    finally:
        camera.live_preview_mode()

pycam.mode = 0  # JPEG
pycam.resolution = PHOTO_RESOLUTION
pycam.led_level = LED_LEVEL
pycam.effect = 0

# Replace PyCamera's resolution/SD top bar with retained vector shapes. The
# camera preview begins at y=32, so this group never covers it.
black_palette = displayio.Palette(1)
black_palette[0] = 0x000000
white_palette = displayio.Palette(1)
white_palette[0] = 0xFFFFFF
bubble_palette = displayio.Palette(1)
bubble_palette[0] = RED

def white_rect(width, height, x, y):
    return vectorio.Rectangle(
        pixel_shader=white_palette, width=width, height=height, x=x, y=y
    )

level_group = displayio.Group()

# Four filled rectangles make an outlined vial.
level_group.append(white_rect(BAR_WIDTH, 1, BAR_LEFT, BAR_TOP))
level_group.append(
    white_rect(BAR_WIDTH, 1, BAR_LEFT, BAR_TOP + BAR_HEIGHT - 1)
)
level_group.append(white_rect(1, BAR_HEIGHT, BAR_LEFT, BAR_TOP))
level_group.append(
    white_rect(1, BAR_HEIGHT, BAR_LEFT + BAR_WIDTH - 1, BAR_TOP)
)

# DOWN gets a small two-axis target.
down_references = displayio.Group()
down_references.append(white_rect(9, 1, BAR_CENTER_X - 4, BAR_CENTER_Y))
down_references.append(white_rect(1, 9, BAR_CENTER_X, BAR_CENTER_Y - 4))
level_group.append(down_references)

# HORIZON gets three vertical vial marks.
horizon_references = displayio.Group()
for offset in (-HORIZONTAL_TRAVEL // 2, 0, HORIZONTAL_TRAVEL // 2):
    tick_height = 15 if offset == 0 else 9
    horizon_references.append(
```

```
        white_rect(
            1, tick_height, BAR_CENTER_X + offset, BAR_CENTER_Y - tick_height // 2
        )
    )
level_group.append(horizon_references)

bubble_outline = vectorio.Circle(
    pixel_shader=white_palette,
    radius=BUBBLE_RADIUS + 1,
    x=BAR_CENTER_X,
    y=BAR_CENTER_Y,
)
bubble_fill = vectorio.Circle(
    pixel_shader=bubble_palette,
    radius=BUBBLE_RADIUS - 1,
    x=BAR_CENTER_X,
    y=BAR_CENTER_Y,
)
level_group.append(bubble_outline)
level_group.append(bubble_fill)

# Add it on top of the toolbar
pycam.splash.append(level_group)
pycam.display.refresh()

mode_setting = AUTO
active_mode = HORIZON
screen_x, screen_y, forward = pycam.accel.acceleration
zero_down_x = 0.0
zero_down_y = 0.0
zero_horizon = 0.0
last_print = time.monotonic()

while True:
    frame = pycam.continuous_capture()
    pycam.keys_debounce()

    raw = pycam.accel.acceleration
    screen_x += (raw[0] - screen_x) * SMOOTHING
    screen_y += (raw[1] - screen_y) * SMOOTHING
    forward += (raw[2] - forward) * SMOOTHING

    angle_from_vertical = math.degrees(
        math.atan2(
            math.sqrt(screen_x * screen_x + screen_y * screen_y),
            abs(forward),
        )
    )

    if mode_setting == AUTO:
        if active_mode == HORIZON and angle_from_vertical <= DOWN_ENTER_DEGREES:
            active_mode = DOWN
        elif active_mode == DOWN and angle_from_vertical >= DOWN_EXIT_DEGREES:
            active_mode = HORIZON
    else:
        active_mode = mode_setting
```

```
205
206     # atan2 keeps the reading stable even if total measured gravity varies.
207     x_angle = math.degrees(math.atan2(screen_x, abs(forward))) - zero_down_x
208     y_angle = math.degrees(math.atan2(screen_y, abs(forward))) - zero_down_y
209     roll_angle = math.degrees(math.atan2(screen_x, abs(screen_y))) - zero_horizon
210
211     if frame is not None and hasattr(frame, "width"):
212         pycam.blit(frame)
213
214     update_level(active_mode, x_angle, y_angle, roll_angle)
215     pycam.display.refresh()
216
217     if pycam.select.fell:
218         mode_setting = (mode_setting + 1) % 3
219         names = ("AUTO", "DOWN", "HORIZON")
220         pycam.display_message(names[mode_setting], color=0xFFFFFF, scale=2)
221         time.sleep(0.2)
222
223     if pycam.ok.fell:
224         if active_mode == DOWN:
225             zero_down_x += x_angle
226             zero_down_y += y_angle
227             message = "DOWN zeroed"
228         else:
229             zero_horizon += roll_angle
230             message = "HORIZON zeroed"
231         pycam.display_message(message, color=0x00DD00, scale=2)
232         time.sleep(0.2)
233
234     if pycam.shutter.short_count:
235         take_photo(pycam)
236
237     if pycam.card_detect.fell:
238         print("SD card removed")
239         pycam.unmount_sd_card()
240         pycam.display.refresh()
241
242     if pycam.card_detect.rose:
243         print("SD card inserted")
244         pycam.display_message("Mounting SD", color=0xFFFFFF, scale=2)
245         try:
246             pycam.mount_sd_card()
247         except OSError as error:
248             print("SD mount failed:", error)
249             pycam.display_message("SD failed", color=0xFF0000, scale=2)
250             time.sleep(0.5)
251         pycam.display.refresh()
```

https://github.com/adafruit/Adafruit_Learning_System_Guides/blob/main/MEMENTO/Memento_Bubble_Level/code.py

How the Code Works

The code starts off with a few imports and a number of settings. We'll start by going over the settings. The first couple of settings are from the basic camera code and set the Resolution and LED modes:

```
# Settings
PHOTO_RESOLUTION = 8  # 0-12 preset resolutions:
#                     0: 240x240, 1: 320x240, 2: 640x480, 3: 800x600, 4: 1024x768,
#                     5: 1280x720, 6: 1280x1024, 7: 1600x1200, 8: 1920x1080, 9: 2048x1536,
#                     10: 2560x1440, 11: 2560x1600, 12: 2560x1920

LED_LEVEL = 0 # 0-4 preset brightness levels
```

The next series of settings are for the bubble level itself:

```
FULL_SCALE_DEGREES = 8.0  # Bubble reaches the end at this much error
GREEN_DEGREES = 1.5
AMBER_DEGREES = 4.0
SMOOTHING = 0.18  # 0.0 = frozen, 1.0 = no filtering

# The gap between these angles prevents AUTO mode from flickering.
DOWN_ENTER_DEGREES = 35
DOWN_EXIT_DEGREES = 45

# Level geometry within the 240 x 32 band below the live preview.
BAR_WIDTH = 64
BAR_HEIGHT = 25
BAR_BOTTOM_MARGIN = 2
BUBBLE_RADIUS = 6
```

Here are the settings specific to the bubble level:

- **FULL_SCALE_DEGREES:** The amount of tilt between one end of the level and the and the other on each axis
- **GREEN_DEGREES:** The tolerance in degrees for the bubble to show green
- **AMBER_DEGREES:** The tolerance in degrees for the bubble to show amber. This should always be larger than GREEN_DEGREES.
- **SMOOTHING:** This value is between 0.0 and 1.0 and controls how fast the bubble moves side to side. If it's too low, it will move very slowly and too fast will make it appear jittery. This is similar to controlling the viscosity of the liquid that the bubble moves through with low numbers being more viscous.
- **DOWN_ENTER_DEGREES:** This is the point in degrees that the bubble level changes modes and enters into a downward type level.
- **DOWN_EXIT_DEGREES:** This is the point in degrees that the bubble level changes modes and enters into a horizontal type level.

- **BAR_WIDTH:** The width of the bubble level bar
- **BAR_HEIGHT:** The height of the bubble level bar
- **BAR_BOTTOM_MARGIN:** The amount of gap from the bottom of the display
- **BUBBLE_RADIUS:** The size of the bubble itself

Helper Functions

There are a few functions that help with either simplifying the code by grouping sections that need to run multiple times or by just making the code easier to read:

`clamp()`: This function will keep a given value within a certain range. Any number outside this range will be set to the end of the range that it exceeds.

`level_color()`: This will return the color based on how close to the amount of levelness the camera is off by.

`update_level()`: This function will take the accelerometer values along with the current mode and update the UI elements to reflect the current level.

`take_photo()`: This function is the photo capture section from the basic camera example.

`white_rectangle():` This function just creates and returns a white vectorio rectangle.

UI Elements

The UI elements are made from vectorio shapes. Since there are no line elements, they are mostly created using white rectangles. First the bar is added and then the downward facing and horizontal facing level elements are added, which display depending on the angle of the level.

The bubble itself is drawn next with an outline of white and a fill of the color determined by the amount of error. Afterwards, the rest of the UI is drawn as normal.

Main Loop

The main loop involves reading the accelerometer, applying a smoothing factor so that the bubble doesn't move too fast, and monitoring which mode the bubble level is in. After that, the `update_level()` function is called to update the UI elements.

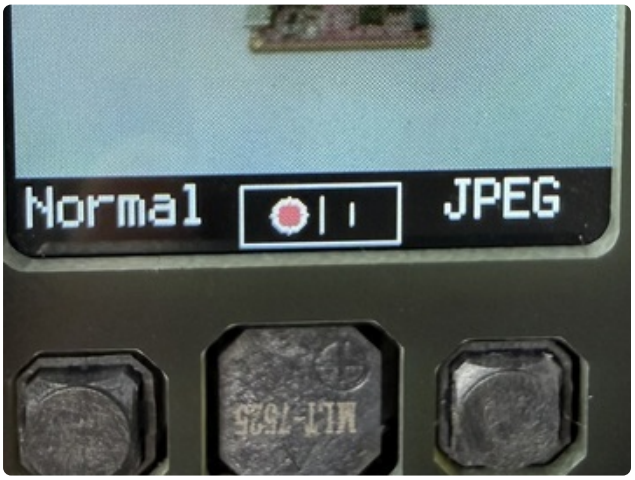
The rest of the main loop involves checking for the SD card and updating the pycam library by calling its refresh method.

Use the Bubble Level

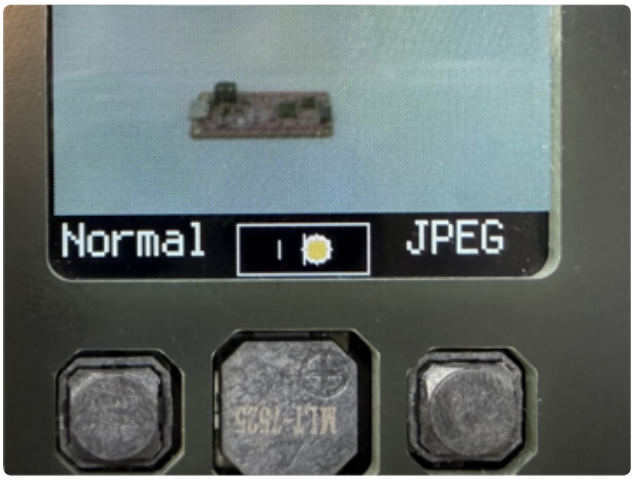
The bubble level is pretty simple to use. You would use the camera like normal, except you would use the bubble level on the bottom of the camera to adjust the angle of your camera in order to keep it straight. The level has 2 modes that can be switched between just by tilting the camera down or forward.

Horizontal Mode

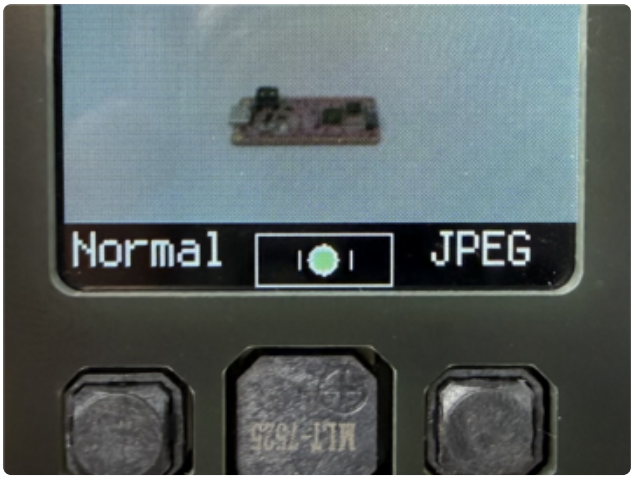
For horizontal mode, the bubble shows the amount of tilt to either side. The level is drawn in a more traditional level style with lines and the bubble changes color with the amount of tilt.



In **horizontal mode**, if your camera is off by a larger degree, then the bubble will show up as red. This would mean a tilt angle of **more than 4 degrees** if using the default values.



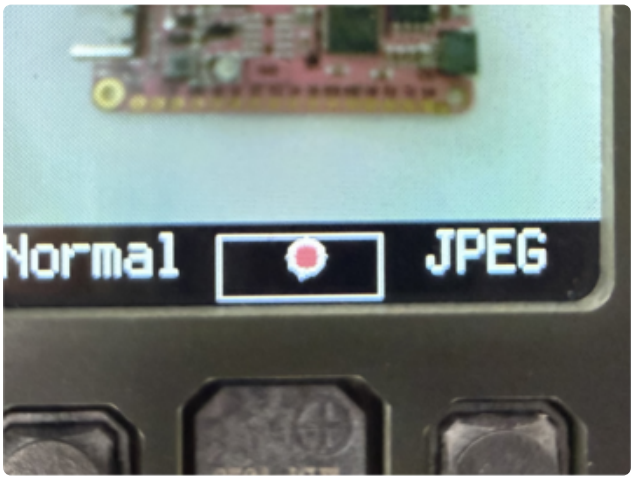
In **horizontal mode**, if your camera is off by a smaller degree, then the bubble will show up as yellow. This would mean a tilt angle **between 1.5 and 4 degrees** if using the default values.



In **horizontal mode**, if your camera is relatively level, then the bubble will show up as green. This would mean a tilt angle of **1.5 degrees or less** if using the default values.

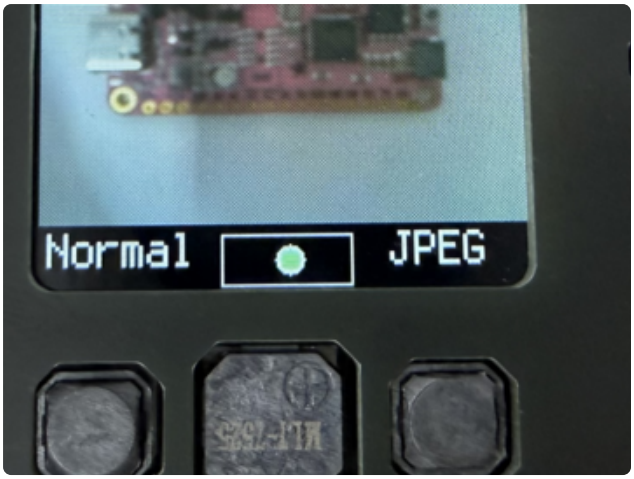
Vertical Mode

For vertical mode, the bubble shows the amount of tilt in 2 different axes. The level is drawn with a plus to indicate the optimal center and the bubble changes color with the amount of tilt.



In **vertical mode**, if your camera is off by a larger degree in either axis, then the bubble will show up as red. This would mean a tilt angle of **more than 4 degrees** if using the default values.

In **vertical mode**, if your camera is off by a smaller degree in either axis, then the bubble will show up as yellow. This would mean a tilt angle **between 1.5 and 4 degrees** if using the default values.



In **vertical mode**, if your camera is relatively level in both axes, then the bubble will show up as green. This would mean a tilt angle of **1.5 degrees or less** if using the default values.

This bubble level could probably be pretty easily adapted to work with other cameras besides the basic camera. Feel free to play with the values in the script and see how it affects the bubble level to get it to your liking and most of all, have fun!