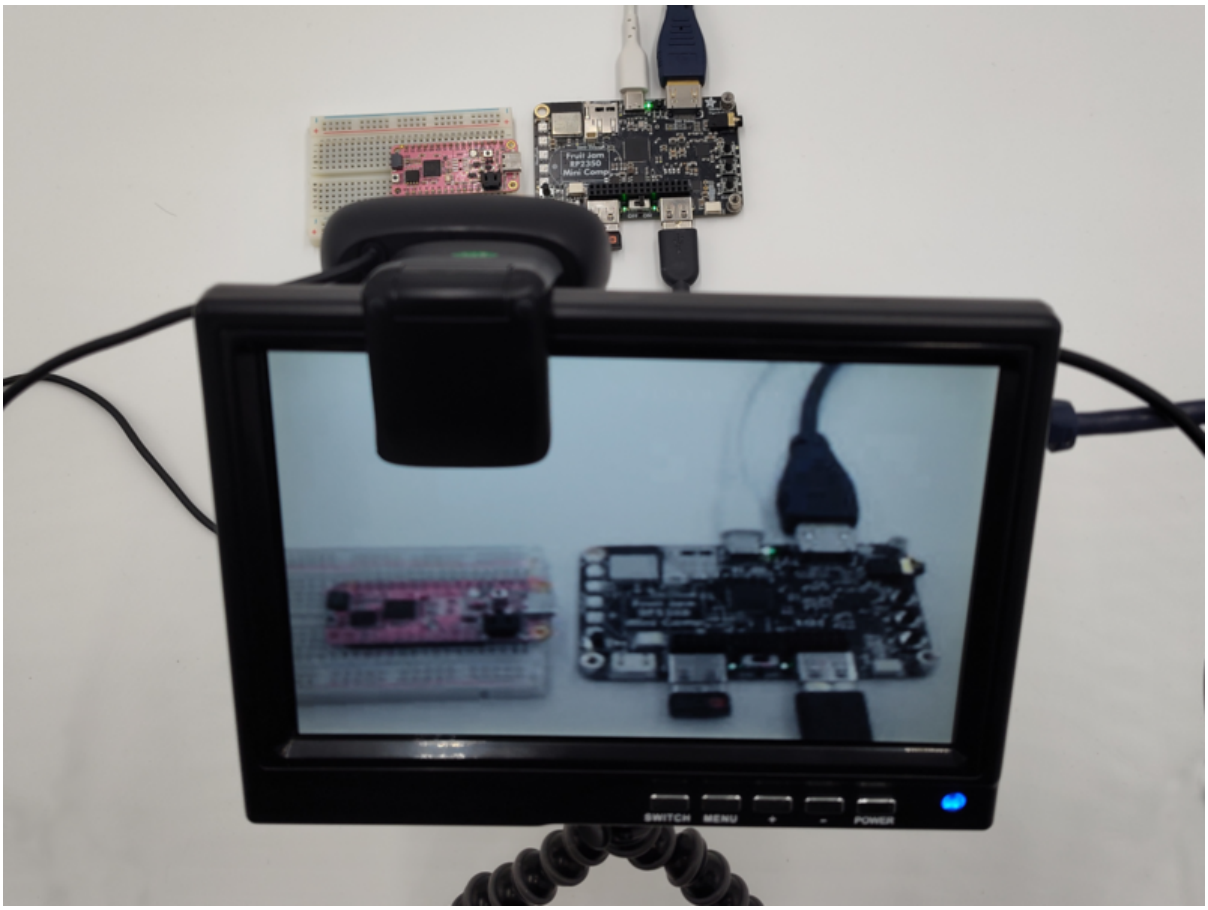
USB Host Camera on Fruit Jam

Created by Tim C



https://learn.adafruit.com/usb-host-camera-on-fruit-jam

Last updated on 2026-09-29 02:22:14 PM UTC

Table of Contents

Overview 3

- Parts

Install CircuitPython 5

- CircuitPython Quickstart
- Safe Mode
- Flash Resetting UF2

USB Host Camera Basics 9

- Isochronous Buffer Size
- Detect the Camera
- Configure Mode
- Start the Camera
- Save a Photo
- Live Preview via DVI

Camera with Effects 12

- Effects Gallery

Code 17

- Getting the Program Files
- Drive Structure

Use 29

- Choose Effect
- Take Photo

Camera with BLE Thermal Printer 31

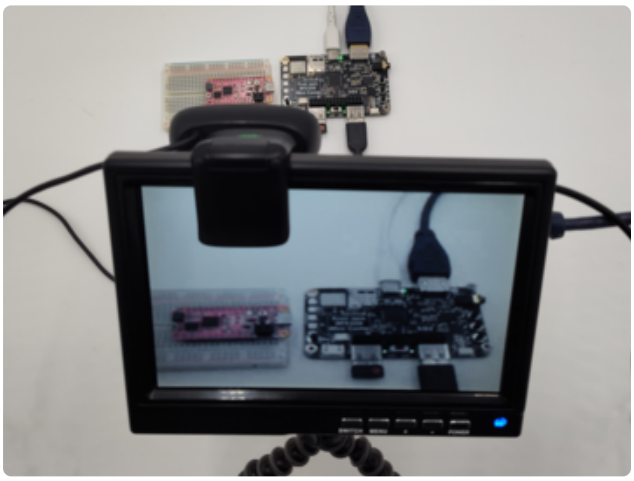
Code 33

- Getting the Program's Files
- Drive Structure

Use 40

- Change Brightness
- Full Color Preview
- Print Photo

Overview

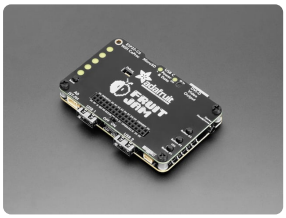


CircuitPython has recently gained support () for isochronous communication over USB host on RP2040 and RP2350. This unlocks a few new types of USB devices that can work on the Adafruit Fruit Jam. USB cameras are the group that I have added support for.

This guide demonstrates two projects that use standard USB webcams:

1. Saving photos to the CPSAVES drive or SD card with several filters that can be applied.
2. Converting photos to black and white with Floyd Stenberg dithering and print them over BLE to a thermal printer

Parts



Featured

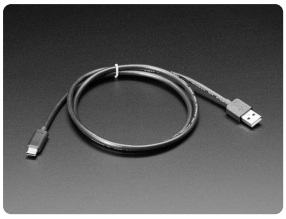
Adafruit Fruit Jam - Mini RP2350 Computer

We were catching up on a recent hackaday hackchat with eben upton and learned some fun facts:...

https://www.adafruit.com/product/6200

USB Camera

Any standard USB webcam that supports MJPEG at sizes of 320x240 or less. Tested successfully on Logitech models C270 () and C920 ().



Featured

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

Featured

HDMI Cable - 1 meter

Connect two HDMI devices together with this basic HDMI cable. It has nice molded grips for easy installation, and is 1 meter long (about 3 feet). This is a HDMI 1.3...

https://www.adafruit.com/product/608

Cat BLE Thermal Printer

This small feline-styled BLE thermal printer is available on Amazon ()

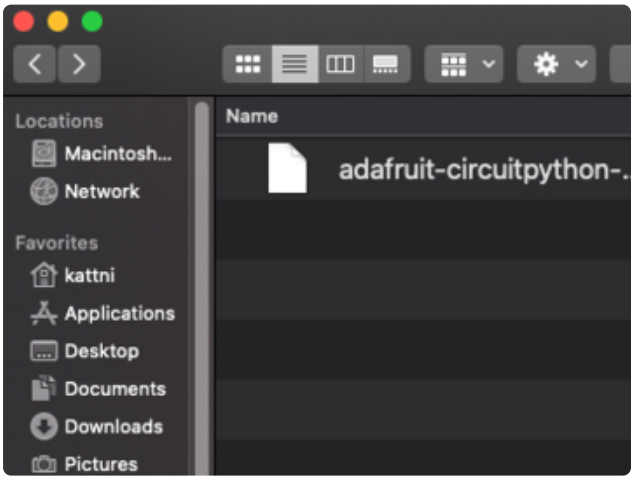
Install CircuitPython

CircuitPython () is a derivative of MicroPython () 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

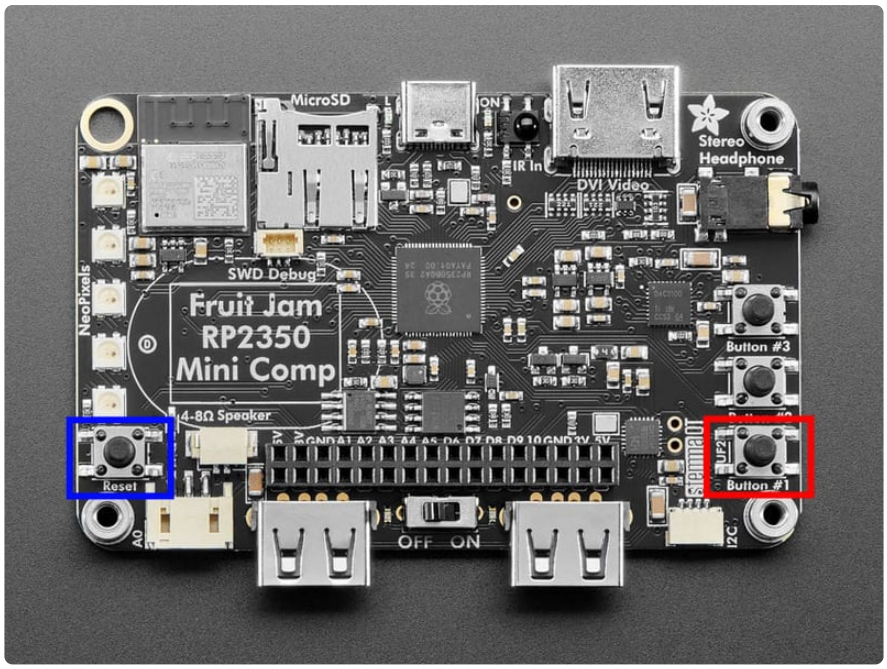
Follow this step-by-step to quickly get CircuitPython running on your board.

Text emphasized with a yellow exclamation:	Support for USB host cameras was added in CircuitPython 11.0.0.alpha-1. As of 9/24/26 it is the latest development release of CircuitPython. Scroll down to the development release section of the downloads page!



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

Save it wherever is convenient for you.

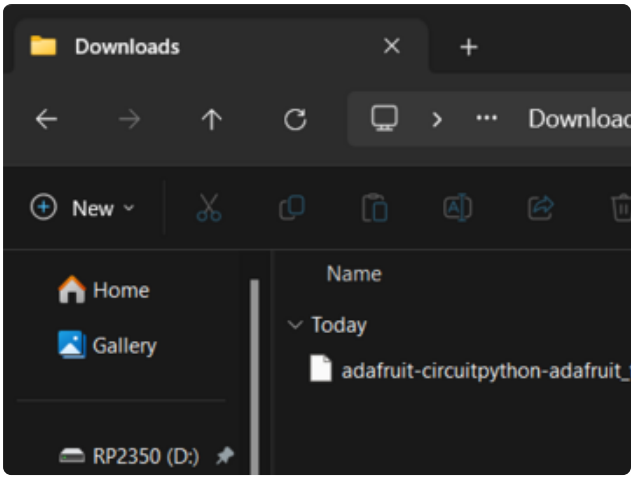


To enter the bootloader, hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset button** (highlighted in red or blue above). **Continue to hold the BOOT/BOOTSEL button until the RP2350 drive appears!**

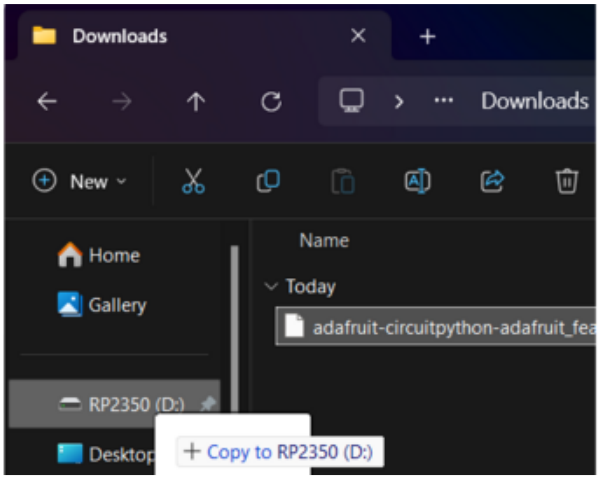
If the drive does not appear, release all the buttons, and then repeat the process above.

You can also start with your board unplugged from USB, press and hold the BOOTSEL button (highlighted in red above), continue to hold it while plugging it into USB, and wait for the drive to appear before releasing the button.

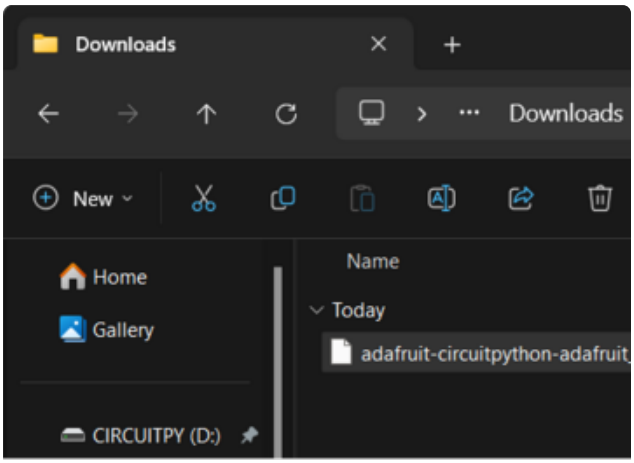
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **RP2350**.



Drag the **adafruit-circuitpython-boardname-language-version.uf2** file to **RP2350.**



The **RP2350** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it, you're done! :)

Safe Mode

You want to edit your **code.py** or modify the files on your **CIRCUITPY** drive, but find that you can't. Perhaps your board has gotten into a state where **CIRCUITPY** is read-only. You may have turned off the **CIRCUITPY** drive altogether. Whatever the reason, safe mode can help.

Safe mode in CircuitPython does not run any user code on startup, and disables auto-reload. This means a few things. First, safe mode bypasses any code in **boot.py** (where you can set **CIRCUITPY** read-only or turn it off completely). Second, it does not run the code in **code.py**. And finally, it does not automatically soft-reload when data is written to the **CIRCUITPY** drive.

Therefore, whatever you may have done to put your board in a non-interactive state, safe mode gives you the opportunity to correct it without losing all of the data on the **CIRCUITPY** drive.

Entering Safe Mode

To enter safe mode when using CircuitPython, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED (highlighted in green above) will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

In Safe Mode

If you successfully enter safe mode on CircuitPython, the LED will intermittently blink yellow three times.

If you connect to the serial console, you'll find the following message.

```
Auto-reload is off.
Running in safe mode! Not running saved code.

CircuitPython is in safe mode because you pressed the reset button during boot. Press again to

Press any key to enter the REPL. Use CTRL-D to reload.
```

You can now edit the contents of the **CIRCUITPY** drive. Remember, your code will not run until you press the reset button, or unplug and plug in your board, to get out of safe mode.

Flash Resetting UF2

If your board ever gets into a really weird state and CIRCUITPY doesn't show up as a disk drive after installing CircuitPython, try loading this 'nuke' UF2 to RP2350. which

will do a 'deep clean' on your Flash Memory. **You will lose all the files on the board**, but at least you'll be able to revive it! After loading this UF2, follow the steps above to re-install CircuitPython.

USB Host Camera Basics

Support for USB host cameras is provided by the Adafruit CircuitPython USB Host Camera () library. This page covers the basic setup and usage of the library to view and capture photos from a USB camera. See the following guide pages for complete projects that are ready to run.

Isochronous Buffer Size

USB devices with isochronous communication need a ring buffer () to store incoming packets while CircuitPython is doing other things. The `CIRCUITPY_USB_HOST_ISO_BUFFER_SIZE` configuration in **settings.toml** is the way to declare the size allocated for that buffer. A size of `16384` has been found to work nicely for a USB web camera sending 320x240 pixel frames.

Here is an example **settings.toml** file with the buffer configured.

```
CIRCUITPY_USB_HOST_ISO_BUFFER_SIZE = 16384
```

Detect the Camera

The `UVCCamera()` initializer will automatically search for a supported camera connected to a USB host port on the Fruit Jam. It is good practice to make your project code retry until a camera is found in case you happen to power up the board without the camera connected. This loop from the simpletest script demonstrates how to check for the camera every 2 seconds until it one is found.

```
import time

import adafruit_usb_host_camera

camera = None
while camera is None:
    try:
        # Finds the camera among the attached devices (a keyboard, a mouse...)
        camera = adafruit_usb_host_camera.UVCCamera()
```

```
    except ValueError:
        print("Plug a USB camera into the USB host port")
        time.sleep(2)
```

Configure Mode

Most USB cameras support multiple different modes, each with different capture formats and sizes. The list of supported modes is communicated by the camera inside of its USB descriptor. To make things easy, the `UVCCamera` class has the `find_mode(width, height)` function. Pass in the width and height that you want to capture photos at, and the function will return a `Mode` object corresponding to the closest match that the camera supports.

On the Fruit Jam, 320x240 works best, if you want a live preview from the camera displayed over DVI. 640x480 works as well, but saving a photo takes longer, and the display cannot show the preview at high frames per second (FPS).

```
print("Camera modes:", camera.modes)
mode = camera.find_mode(320, 240)
print("Using", mode)
```

Start the Camera

To start the camera so it will begin sending back image data, use the `start_camera()` function. It accepts a `Mode` object representing the capture mode that you want to use.

```
camera.start(mode)
```

Save a Photo

To capture a photo and save it as a jpeg file, use the `save_jpeg()` function and pass a path for where to save the photo. The path must be mounted as writable to CircuitPython. The easiest options are inside of the **/saves/** directory, or on an SD card.

```
camera.save_jpeg("/saves/photo.jpg")
```

Live Preview via DVI

To show a live preview from the camera on a display connected to the Fruit Jam's DVI output, the `camera.bitmap` property and `update_bitmap()` function are used. The `picogame` module is the fastest way to render the preview. You can use it instead of `displayio` for maximizing the preview FPS.

First initialize the display configured for 320x240, you can skip this the display is already set to 320x240 with 16 bit color depth from your **settings.toml** file. In that case just use `framebuffer = supervisor.runtime.display.framebuffer`.

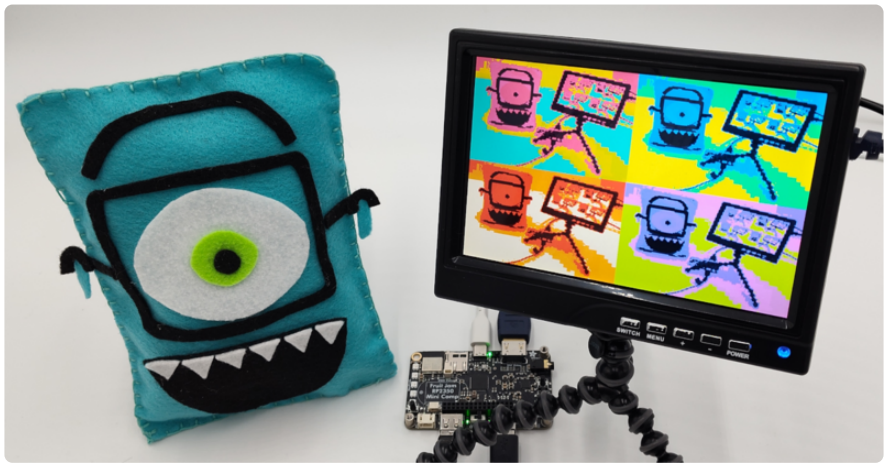
Next set up the camera in the same way shown above. Then get the bitmap object from the camera and set up the Picogame `Sprite` and `layers`.

Inside the main loop, update the preview by calling `update_bitmap()` and `picogame.render()`

```
# --------- Display Setup ---------

WIDTH, HEIGHT = 320, 240

# Manual display init
displayio.release_displays()
framebuffer = picodvi.Framebuffer(
    WIDTH,
    HEIGHT,
    clk_dp=board.CKP,
    clk_dn=board.CKN,
    red_dp=board.D0P,
    red_dn=board.D0N,
    green_dp=board.D1P,
    green_dn=board.D1N,
    blue_dp=board.D2P,
    blue_dn=board.D2N,
    color_depth=16,
)
# Or automatic display init

# Inside settings.toml:
# CIRCUITPY_DISPLAY_WIDTH=320
# CIRCUITPY_DISPLAY_COLOR_DEPTH=16

# Here in code.py:
# framebuffer = supervisor.runtime.display.framebuffer

# Either way, wrap the framebuffer with picogame
render_buffer = picogame.Framebuffer(framebuffer, WIDTH, HEIGHT, native_rgb565=True)


# --------- Camera Setup ---------
# ... same as shown above ...


# --------- Picogame Sprite and Layers setup ---------

# The first access captures a frame and creates the bitmap.
bitmap = None
while bitmap is None:
    try:
        bitmap = camera.bitmap
    except RuntimeError:
```

```
        pass  # no complete frame in time; keep trying

# update_bitmap() decodes each frame into this same bitmap, and picogame draws
# that memory (no copy) straight into the DVI framebuffer.
sprite = picogame.Sprite(picogame.Bitmap(bitmap, bitmap.width, bitmap.height), x=0, y=0)
layers = [sprite]


# --------- Main Loop ---------
while True:
    try:
        camera.update_bitmap()
    except RuntimeError:
        continue  # no complete frame in time; keep trying
    picogame.render(render_buffer, layers, None, 0, 0, WIDTH, HEIGHT)
```

Camera with Effects

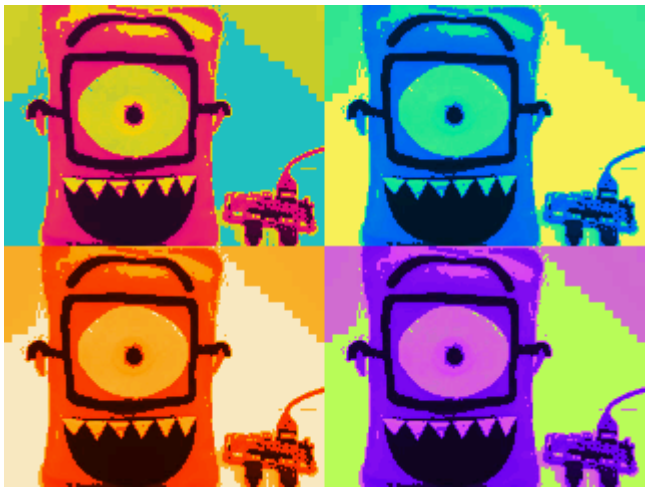


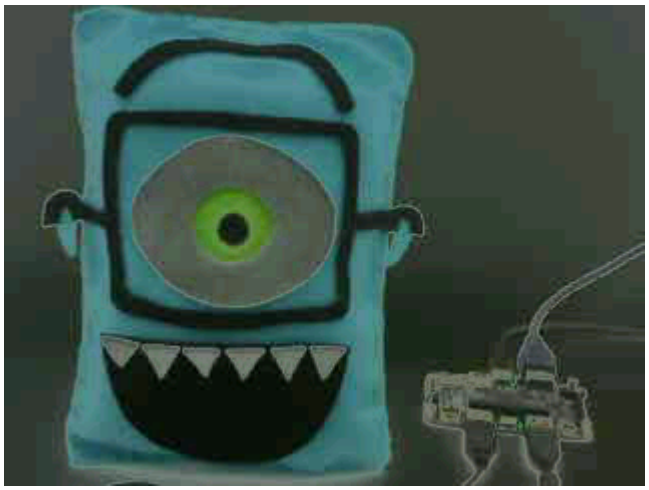
This section covers an effects camera project. Live preview from the USB camera is shown on the display connected to the Fruit Jam's DVI output. Several different effects are available which modify colors and other aspects of the images. A number of the effects are 2x2 color grids, inspired by Andy Worhol's artwork.

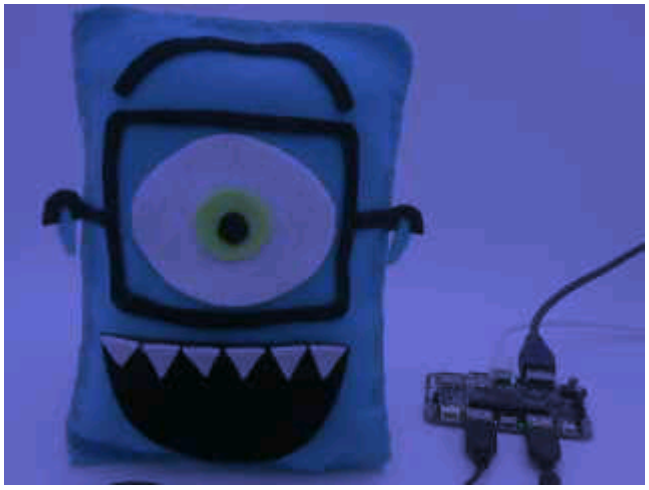
Text emphasized with a blue camera: If you point the camera at the display showing the live preview it functions as a basic video feedback rig.

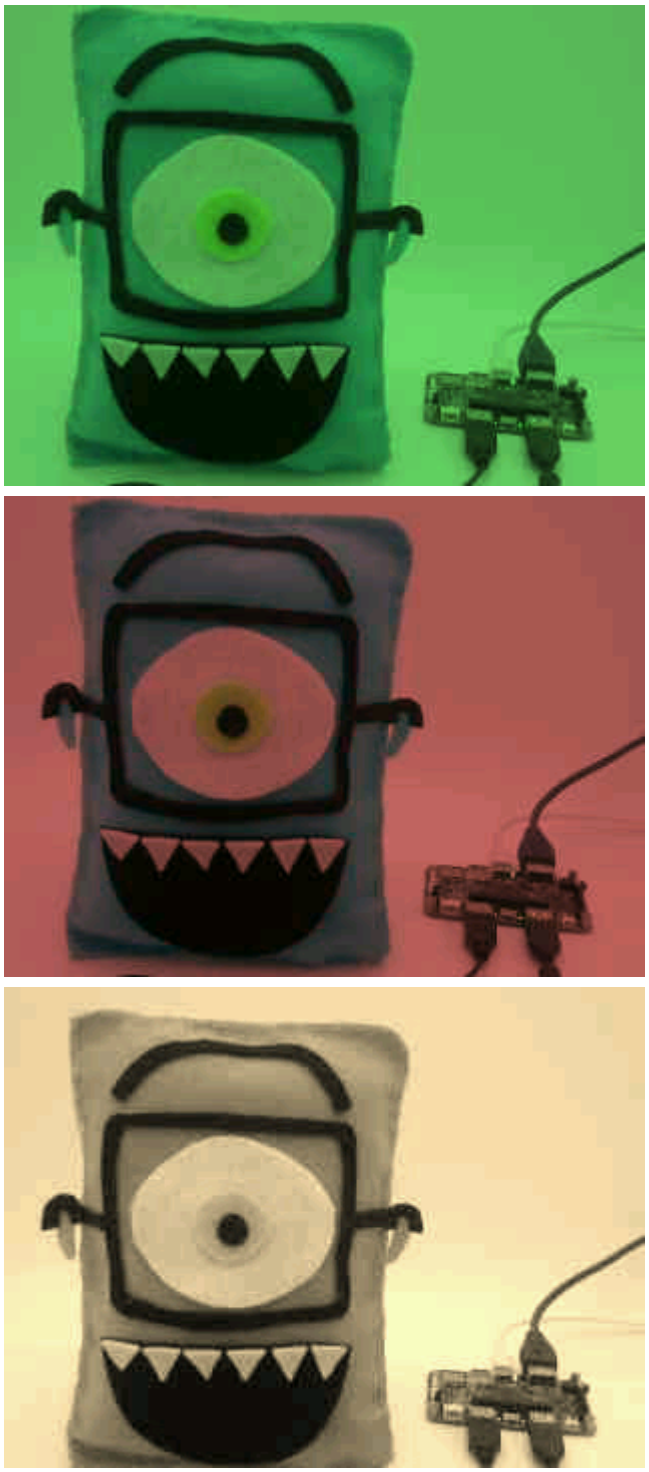
Effects Gallery

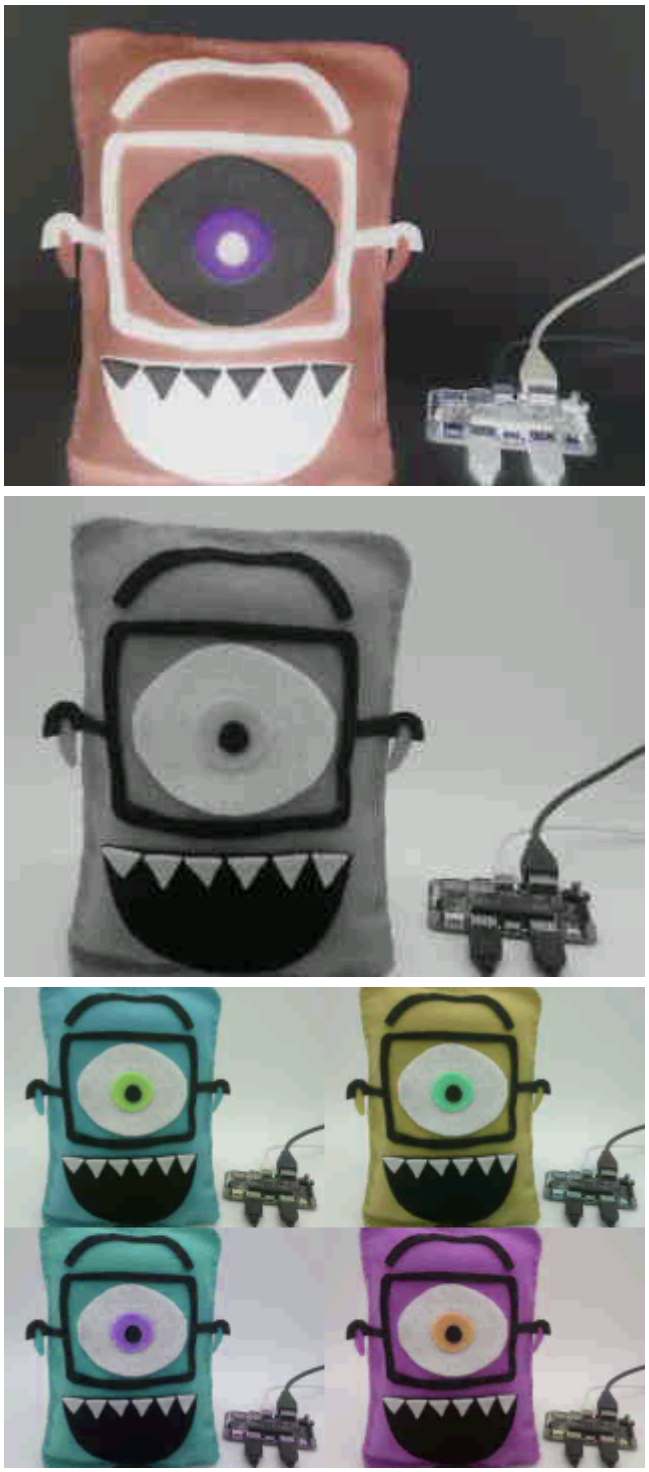
This gallery shows several of the available photo effects from the project.

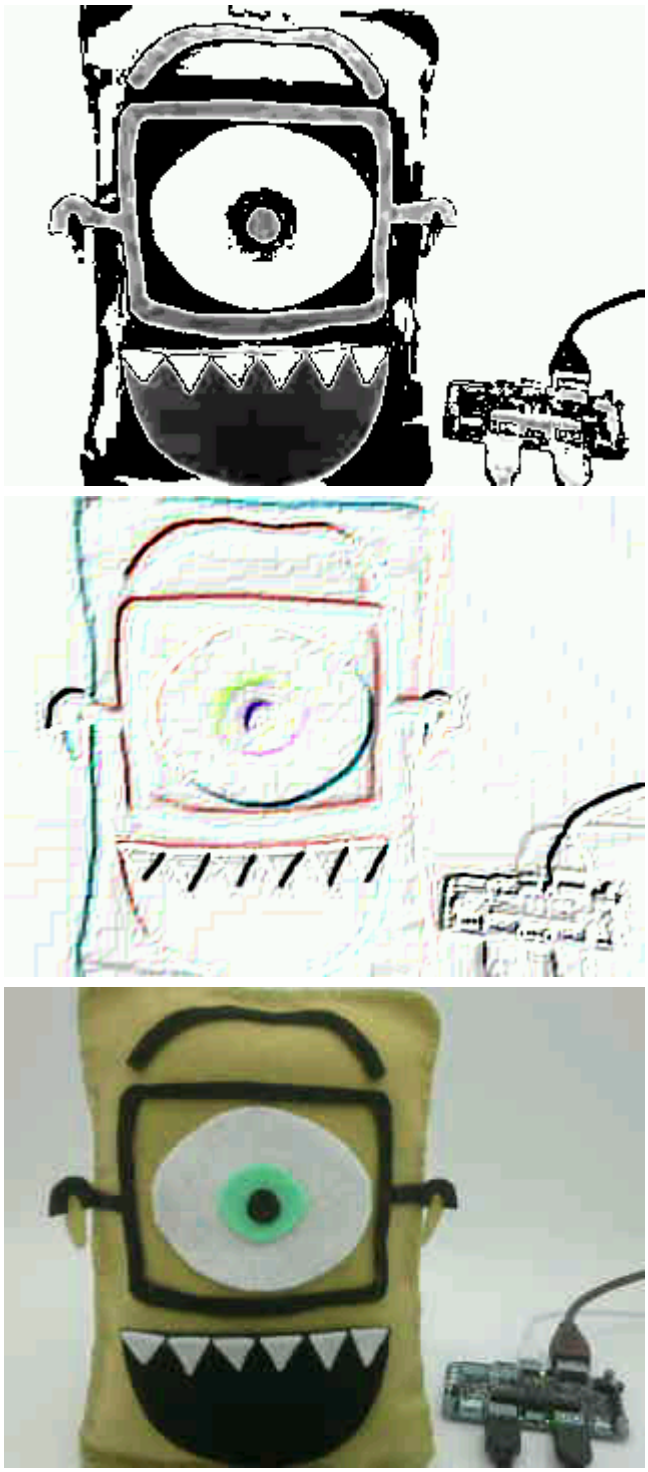












Code

Getting the Program Files

To use the application, you need to obtain **code.py** with the program, and the other project files to place on the Fruit Jam **CIRCUITPY** drive.

Thankfully, this can be done in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries, the **code.py** file, and other project files in a zip file.

Connect your board to your computer via a known good data+power USB cable. The board should show up in your File Explorer/Finder (depending on your operating system) as a flash drive named **CIRCUITPY**.

Extract the contents of the zip file, copy the **lib** directory files to **CIRCUITPY/lib**. Copy the **code.py** file to your **CIRCUITPY** drive. The program should self start.

```
# SPDX-FileCopyrightText: Copyright (c) 2026 Tim Cocks for Adafruit Industries
#
# SPDX-License-Identifier: MIT
"""USB webcam with live filters on the Fruit Jam.

Frames come from a UVC webcam on a USB host port (adafruit_usb_host_camera),
are decoded with jpegio, run through the currently selected bitmapfilter effect,
and shown on the DVI output with picogame.

Taking a photo saves the camera's JPEG as-is; when an effect is selected the
filtered picture is written alongside it as a .bmp. Photos go to the SD card
when one is mounted, otherwise to /saves (the CPSAVES drive).

Controls, from a USB keyboard on the other host port (or the serial console):
  right / left arrow -> next / previous effect
  up arrow           -> show the plain picture (no effect), keeping the
                        place in the list
  down arrow         -> back to the effect at that place in the list
  1-9, 0             -> shortcuts to every other effect: 1 is the first
                        effect, 2 the third, ... 0 the nineteenth
  space or enter     -> take a photo
  R                  -> remount CPSAVES so the computer sees the new photos

The Fruit Jam's own buttons work without a keyboard: button 1 and button 2
step down and up the list of effects, and button 3 takes a photo.
"""

import binascii
import gc
import os
import struct
import sys
import time

import bitmapfilter
import bitmaptools
import board
import displayio
import jpegio
import keypad
import picodvi
import picogame
import storage
```

```
import supervisor
import terminalio
from ulab import numpy as np

import adafruit_usb_host_camera

# pylint: disable=too-many-locals, global-statement

WIDTH, HEIGHT = 320, 240
LABEL_HEIGHT = 14
LABEL_SECONDS = 2

KEY_UP = "\x1b[A"
KEY_DOWN = "\x1b[B"
KEY_RIGHT = "\x1b[C"
KEY_LEFT = "\x1b[D"
KEYS_SHUTTER = (" ", "\r", "\n")
# Number keys jump to every other effect: 1 -> effects[1], 2 -> effects[3], ...
# 0 comes after 9, like on the keyboard.
KEYS_NUMBER = "1234567890"

displayio.release_displays()
framebuffer = picodvi.Framebuffer(
    WIDTH,
    HEIGHT,
    clk_dp=board.CKP,
    clk_dn=board.CKN,
    red_dp=board.D0P,
    red_dn=board.D0N,
    green_dp=board.D1P,
    green_dn=board.D1N,
    blue_dp=board.D2P,
    blue_dn=board.D2N,
    color_depth=16,
)
# jpegio decodes into `bitmap`, the effects change it in place, and picogame
# draws that same memory straight into the DVI scanout buffer.
bitmap = displayio.Bitmap(WIDTH, HEIGHT, 65535)
sprite = picogame.Sprite(picogame.Bitmap(bitmap, WIDTH, HEIGHT), x=0, y=0)
target = picogame.Framebuffer(framebuffer, WIDTH, HEIGHT, native_rgb565=True)

WHITE = picogame.rgb565(255, 255, 255)
GREEN = picogame.rgb565(0, 221, 0)
RED = picogame.rgb565(255, 0, 0)
BLACK = picogame.rgb565(0, 0, 0)
CLEAR = picogame.rgb565(255, 0, 255)  # stands for "transparent" in the label

# A strip along the bottom of the screen for the effect name and messages
label = picogame.Canvas(WIDTH, LABEL_HEIGHT, transparent=CLEAR)
label.move(0, HEIGHT - LABEL_HEIGHT)
label.clear(CLEAR)
label_until = 0
picture_layers = [sprite]  # the color grids swap in their four panels


def refresh():
```

```
    layers = picture_layers
    if time.monotonic() < label_until:
        layers = layers + [label]
    picogame.render(target, layers, None, 0, 0, WIDTH, HEIGHT)


def display_message(text, color=WHITE, seconds=LABEL_SECONDS):
    """Show ``text`` over the bottom of the picture for a while."""
    global label_until  # noqa: PLW0603
    label.clear(CLEAR)
    label.text(4, 1, " %s " % text, color, terminalio.FONT, BLACK)
    label_until = time.monotonic() + seconds
    refresh()


# --- Effects -----------------------------------------------------------
# These are the adafruit_pycamera.imageprocessing effects. That package needs
# the PyCamera's hardware to import, so the ones used here are repeated.

greyscale_weights = bitmapfilter.ChannelMixer(
    0.299, 0.587, 0.114,
    0.299, 0.587, 0.114,
    0.299, 0.587, 0.114,
)  # fmt: skip
inverse_greyscale_weights = bitmapfilter.ChannelMixer(
    1 - 0.299, 1 - 0.587, 1 - 0.114,
    1 - 0.299, 1 - 0.587, 1 - 0.114,
    1 - 0.299, 1 - 0.587, 1 - 0.114,
)  # fmt: skip
sepia_weights = bitmapfilter.ChannelMixer(
    0.393, 0.769, 0.189,
    0.349, 0.686, 0.168,
    0.272, 0.534, 0.131,
)  # fmt: skip
negative_weights = bitmapfilter.ChannelScaleOffset(-1, 1, -1, 1, -1, 1)
red_weights = bitmapfilter.ChannelScale(1, 0.5, 0.5)
green_weights = bitmapfilter.ChannelScale(0.5, 1, 0.5)
blue_weights = bitmapfilter.ChannelScale(0.5, 0.5, 1)
bright_weights = bitmapfilter.ChannelScale(2.0, 2.0, 2.0)
low_contrast_weights = bitmapfilter.ChannelScaleOffset(0.5, 0.5, 0.5, 0.5, 0.5, 0.5)
swap_rb_weights = bitmapfilter.ChannelMixer(0, 0, 1, 0, 1, 0, 1, 0, 0)
swap_bg_weights = bitmapfilter.ChannelMixer(1, 0, 0, 0, 0, 1, 0, 1, 0)
swap_rg_weights = bitmapfilter.ChannelMixer(0, 1, 0, 1, 0, 0, 0, 0, 1)

blur_weights = (1, 2, 1, 2, 4, 2, 1, 2, 1)
sharpen_weights = (-1, -2, -1, -2, 13, -2, -1, -2, -1)
emboss_weights = (-2, -1, 0, -1, 0, 1, 0, 1, 2)
blur_more = [
    4, 15, 24, 15, 4,
    15, 61, 97, 61, 15,
    24, 97, 154, 97, 24,
    15, 61, 97, 61, 15,
    4, 15, 24, 15, 4,
]  # fmt: skip

# The 'ironbow' false color palette, three bytes (R, G, B) per entry
```

```
_IRONBOW = binascii.unhexlify(
    "ffffffffffffffefff7fef7f7fdf7f7fdf7f7fcf7effcefefffefefffefeffaefe7fae7"
    "e7fde7e7fde7e7f8e7def8dedeffdedeffdedefeded6fed6d6f5d6d6f5d6d6f4d6cef4ce"
    "ceffceceffcecefacec6fac6c6f5c6c6f5c6c6f0c6bdf0bdbdbfbdbdbfbdbdbebdb5beb5"
    "b5bdb5b5bdb5b5bcb5b5bcb5adafadadafadadaaadadaaada5ada5a5ada5a5a8a5a5a8a5"
    "9cbf9c9cbf9c9cbe9c9cbe9c94b59494b59494b49494b4948caf8c8caf8c8caa8c8caa8c"
    "84a58484a58484a08484a0847b7f7b7b7f7b7b7e7b7b7e7b737d73737d73737c73737c73"
    "6b7f6b6b7f6b6b7a6b6b7a6b637d63637d636378636378635a5f5a5a5f5a5a5e5a5a5e5a"
    "525552525552525452525452 4a5f4a4a5f4a4a5a4a4a5a4a4a554a425542425042425042"
    "423f42393f39393e39393e39393d39313d31313c31313c31312f31292f29292a29292a29"
    "292d29212d21212821212821211f21181f18181e18181e1818151810151010141010141 0"
    "100f10080f08080a08080a0808050800050000000000000000000800001000001808002 1"
    "0800290800290800311000391000421000 4a18005218005a18006318006b21006b210073"
    "21007b29007b31007b31007b39007b39007b42007b4a007b4a00845200845200845a0084"
    "6300846300846b00846b008473008c7b008c7b008c84008c84058c8c058c94058c94058c"
    "9c058c9c058ca5058ca5058cad058cb5058cb50a8cbd0a8cbd0a8cbd0f84c6147bc6157b"
    "c61573c61e6bce1f6bce2863ce2863ce2d5ad62a52d62f52d62f4ade3c42de3d42de3e39"
    "de3e31de3f31de5029e75529e75a29e75a21e75f21e75421e75521e75e18e75f18e75f18"
    "ef7810ef7d10ef7a10ef7f08ef7c08ef7d08ef7d08ef7e08ef7f08efa008efa508f7aa08"
    "f7af10f7b410f7b510f7be10f7bf10f7a810f7a810f7ad10f7aa10f7af10f7bc10f7bd10"
    "f7be10f7bf10f7bf10fff018fff518fffa18ffff18fff418fff518fffe18fffe21ffff21"
    "fff829fffd31fffd42fffa52fffa63fffa6bffff7bffff8cfffc94fffca5fffdb5fffdbd"
    "fffecefffedeffffefffff18"
)
ironbow_palette = displayio.Palette(256)
for _i in range(256):
    ironbow_palette[_i] = tuple(_IRONBOW[3 * _i : 3 * _i + 3])
del _IRONBOW


def color_dodge_func(a, b):
    return a / (1 - b) if b != 1 else 1


color_dodge = bitmapfilter.blend_precompute(color_dodge_func)

# Scratch copy of the frame for "sketch"
aux = displayio.Bitmap(WIDTH, HEIGHT, 65535)


def sketch(b):
    bitmapfilter.mix(b, inverse_greyscale_weights)
    memoryview(aux)[:] = memoryview(b)
    bitmapfilter.morph(aux, blur_more)
    bitmapfilter.blend(b, aux, b, color_dodge)
    bitmapfilter.mix(b, inverse_greyscale_weights)


# --- Color grids ------------------------------------------------------------
# The frame is decoded at half size and shown four times, each panel gets
# painted with its own colors.

GRID_WIDTH, GRID_HEIGHT = WIDTH // 2, HEIGHT // 2
GRID_QUANTILES = (18, 46, 76)  # where the bands split, in percent of the pixels
GRID_SOFT = 64  # 0 paints flat bands .. 255 blends each fully into the next
GRID_EDGE_SLACK = 4  # keep the palettes until a band edge moves this far
```

```
212
213 GRID_SETS = (
214     (
215         (0x1B0A1E, 0xE6007E, 0xFFD400, 0x22C1C3),
216         (0x06131F, 0x0057FF, 0x00E5A0, 0xFFF35C),
217         (0x1F0606, 0xFF2E00, 0xFF9E00, 0xFFE9C4),
218         (0x0D0518, 0x6A00F4, 0xE040FB, 0xB8FF5C),
219     ),
220     (
221         (0x14100C, 0xC8102E, 0xF5F0E1, 0xE8C547),
222         (0x0C1014, 0x1D3F8F, 0xF5F0E1, 0xE8C547),
223         (0x0C1410, 0x2E8B57, 0xF5F0E1, 0xE8C547),
224         (0x140C14, 0x6B2D5C, 0xF5F0E1, 0xE8C547),
225     ),
226     (
227         (0x0A1A0A, 0xFF2D95, 0xFFE14D, 0x1F6B3A),
228         (0x0A1A0A, 0xFF6A00, 0xFFF2B0, 0x1F6B3A),
229         (0x0A1A0A, 0x00B3FF, 0xD6FAFF, 0x1F6B3A),
230         (0x0A1A0A, 0xB026FF, 0xFFD6FF, 0x1F6B3A),
231     ),
232     (
233         (0x06131F, 0x0057FF, 0x00E5A0, 0xFFF35C),
234         (0x1A0016, 0xFF0080, 0x00FFF0, 0xFFFFFF),
235         (0x001A0E, 0x00FF6A, 0x003CFF, 0xFFFB00),
236         (0x120014, 0xFF4D00, 0xFF00C8, 0x8CFFFF),
237     ),
238     (
239         (0x0A0A0A, 0x3D3D3D, 0x8A8A8A, 0xF0F0F0),
240         (0x0A1018, 0x2A4A6A, 0x6A9AC0, 0xE8F4FF),
241         (0x180A0A, 0x6A2A2A, 0xC08A6A, 0xFFF0E0),
242         (0x0A180F, 0x2A6A3D, 0x8AC0A0, 0xE8FFF0),
243     ),
244 )
245
246 grid_source = displayio.Bitmap(GRID_WIDTH, GRID_HEIGHT, 65535)
247 grid_pixels = np.frombuffer(grid_source, dtype=np.uint8)
248 grid_panels = []
249 grid_palettes = []
250 grid_layers = []
251 for _i in range(4):
252     _panel = displayio.Bitmap(GRID_WIDTH, GRID_HEIGHT, 65535)
253     grid_panels.append(_panel)
254     grid_palettes.append(displayio.Palette(256))
255     grid_layers.append(
256         picogame.Sprite(
257             picogame.Bitmap(_panel, GRID_WIDTH, GRID_HEIGHT),
258             x=(_i % 2) * GRID_WIDTH,
259             y=(_i // 2) * GRID_HEIGHT,
260         )
261     )
262 grid_edges = None  # the band edges the palettes were built for
263 grid_loaded = None  # and the palette set
264
265
266 def grid_band_edges():
267     """Brightness (0-255) at each of GRID_QUANTILES, from a sample of the
```

```
    pixels of grid_source."""
    step = 2 * 13  # every 13th pixel; each is two bytes, RRRRRGGG GGGBBBBB
    high = grid_pixels[0::step]
    low = grid_pixels[1::step]
    red = high // 8
    green_low = low // 32
    green = (high - red * 8) * 8 + green_low
    blue = low - green_low * 32
    luma = np.sort(red * (8 * 0.299) + green * (4 * 0.587) + blue * (8 * 0.114))
    return [0] + [int(luma[len(luma) * q // 100]) for q in GRID_QUANTILES] + [256]


def grid_build_palette(palette, colors, edges):
    """Spread a panel's four colors over the brightness bands."""
    for level in range(4):
        a = colors[level]
        b = colors[min(level + 1, 3)]
        ar, ag, ab = a >> 16, (a >> 8) & 0xFF, a & 0xFF
        dr, dg, db = (b >> 16) - ar, ((b >> 8) & 0xFF) - ag, (b & 0xFF) - ab
        low, high = edges[level], edges[level + 1]
        for y in range(low, high):
            # how far through the band, softened toward the flat color
            t = (y - low) * GRID_SOFT // (high - low)
            palette[y] = (
                (ar + (dr * t >> 8)) << 16
                | (ag + (dg * t >> 8)) << 8
                | (ab + (db * t >> 8))
            )


def color_grid(set_index):
    """Make the effect that paints grid_source into the four panels."""

    def effect(_):
        global grid_edges, grid_loaded  # noqa: PLW0603
        edges = grid_band_edges()
        # Building the palettes is the slow part, so keep them while the
        # picture's brightness holds steady.
        if grid_loaded != set_index or any(
            abs(new - old) >= GRID_EDGE_SLACK for new, old in zip(edges, grid_edges)
        ):
            for palette, colors in zip(grid_palettes, GRID_SETS[set_index]):
                grid_build_palette(palette, colors, edges)
            grid_edges = edges
            grid_loaded = set_index
        source = memoryview(grid_source)
        for panel, palette in zip(grid_panels, grid_palettes):
            memoryview(panel)[:] = source
            bitmapfilter.false_color(panel, palette)

    return effect


# The swap grid's panels: plain, then each pair of color channels traded
SWAP_GRID_WEIGHTS = (None, swap_rb_weights, swap_bg_weights, swap_rg_weights)
```

```
324
325 def swap_grid(_):
326     """Paint grid_source into the four panels, the top left one as it is and
327     the others with two of their color channels swapped."""
328     source = memoryview(grid_source)
329     for panel, weights in zip(grid_panels, SWAP_GRID_WEIGHTS):
330         memoryview(panel)[:] = source
331         if weights is not None:
332             bitmapfilter.mix(panel, weights)
333
334
335 def grid_to_bitmap():
336     """Copy the four panels into the full size bitmap, to save as a photo."""
337     for i, panel in enumerate(grid_panels):
338         bitmaptools.blit(bitmap, panel, (i % 2) * GRID_WIDTH, (i // 2) * GRID_HEIGHT)
339
340
341 effects = [
342     ("none", lambda b: None),
343     ("greyscale", lambda b: bitmapfilter.mix(b, greyscale_weights)),
344     ("negative", lambda b: bitmapfilter.mix(b, negative_weights)),
345     ("sepia", lambda b: bitmapfilter.mix(b, sepia_weights)),
346     ("red cast", lambda b: bitmapfilter.mix(b, red_weights)),
347     ("green cast", lambda b: bitmapfilter.mix(b, green_weights)),
348     ("blue cast", lambda b: bitmapfilter.mix(b, blue_weights)),
349     ("solarize", bitmapfilter.solarize),
350     ("ironbow", lambda b: bitmapfilter.false_color(b, ironbow_palette)),
351     ("blur", lambda b: bitmapfilter.morph(b, blur_weights)),
352     ("sharpen", lambda b: bitmapfilter.morph(b, sharpen_weights)),
353     ("emboss", lambda b: bitmapfilter.morph(b, emboss_weights, add=0.5)),
354     ("bright", lambda b: bitmapfilter.mix(b, bright_weights)),
355     ("low contrast", lambda b: bitmapfilter.mix(b, low_contrast_weights)),
356     ("swap r/b", lambda b: bitmapfilter.mix(b, swap_rb_weights)),
357     ("sketch", sketch),
358 ]
359 FIRST_GRID = len(effects)
360 for _i in range(len(GRID_SETS)):
361     effects.append(("color grid %d" % (_i + 1), color_grid(_i)))
362 effects.append(("swap grid", swap_grid))
363 effect_index = 0
364 # Up arrow sets this to show the plain picture without losing effect_index
365 bypass = False
366
367
368 def active_effect():
369     """The index into ``effects`` of what is on screen."""
370     return 0 if bypass else effect_index
371
372
373 # --- Saving photos ------------------------------------------------------
374
375
376 def photo_directory():
377     try:
378         storage.getmount("/sd")
379         return "/sd"
```

```
    except OSError:
        return "/saves"


def next_photo_number(directory):
    number = 0
    for name in os.listdir(directory):
        if name.startswith("img") and name[3:7].isdigit():
            number = max(number, int(name[3:7]))
    return number + 1


def save_bmp(path, source):
    """Write an RGB565_SWAPPED bitmap as a 24 bit .bmp file."""
    _width, _height = source.width, source.height
    row_size = (_width * 3 + 3) & ~3
    # Build the file in memory and write it at once: many small writes to the
    # flash drive are several times slower.
    data = bytearray(54 + row_size * _height)
    data[0:2] = b"BM"
    struct.pack_into("<IHHI", data, 2, len(data), 0, 0, 54)
    struct.pack_into(
        "<IiiHHIIiiII", data, 14, 40, _width, _height, 1, 24, 0, 0, 0, 0, 0, 0
    )
    out = np.frombuffer(data, dtype=np.uint8)
    pixels = np.frombuffer(source, dtype=np.uint8)
    for y in range(_height):
        row = pixels[y * _width * 2 : (y + 1) * _width * 2]
        high = row[0::2]  # RRRRRGGG
        low = row[1::2]  # GGGBBBBB
        red = high // 8
        green_low = low // 32
        start = 54 + (_height - 1 - y) * row_size  # .bmp rows run bottom to top
        stop = start + _width * 3
        out[start:stop:3] = (low - green_low * 32) * 8
        out[start + 1 : stop : 3] = (high - red * 8) * 32 + green_low * 4
        out[start + 2 : stop : 3] = red * 8
    with open(path, "wb") as f:
        f.write(data)


def take_photo(_camera, _mode, _frame):
    """Save ``frame`` (the JPEG on screen) and, with an effect selected, the
    filtered picture that was made from it."""
    _active = active_effect()
    effect_name = effects[_active][0]
    try:
        directory = photo_directory()
        base = "%s/img%04d" % (directory, next_photo_number(directory))
        display_message("snap", GREEN)
        _camera.save_jpeg(base + ".jpg", _frame)
        print("Saved", base + ".jpg")
        if effect_name != "none":
            display_message("saving " + effect_name, GREEN)
            path = "%s_%s.bmp" % (base, effect_name.replace(" ", "_").replace("/", ""))
            if _active >= FIRST_GRID:
```

```
436                 grid_to_bitmap()
437             # Writing the .bmp takes a few seconds, far longer than the camera's
438             # stream can be buffered for, so pause it meanwhile.
439             _camera.stop()
440             try:
441                 t0 = time.monotonic()
442                 save_bmp(path, bitmap)
443                 print("Saved %s in %.1fs" % (path, time.monotonic() - t0))
444             finally:
445                 _camera.start(_mode)
446         display_message("saved " + base, GREEN)
447     except OSError as exception:
448         # Read-only or full drive. /saves only holds about 2 MB.
449         print("Save failed:", exception)
450         display_message("Save failed: %s" % exception, RED)
451     gc.collect()
452
453
454 def remount_saves():
455     """Remount /saves. The computer only reads the CPSAVES drive when it mounts
456     it, so photos saved since then stay hidden until the drive comes back."""
457     display_message("Remounting CPSAVES", GREEN)
458     try:
459         storage.remount("/saves", readonly=False)
460         print("Remounted CPSAVES")
461         display_message("Remounted CPSAVES", GREEN)
462     except (OSError, RuntimeError) as exception:
463         print("Remount failed:", exception)
464         display_message("Remount failed: %s" % exception, RED)
465
466
467 # --- Keyboard and buttons ---------------------------------------------------
468
469
470 def read_keys():
471     """Return the keys typed since the last call. A USB keyboard plugged into
472     the Fruit Jam types into sys.stdin, the same as the serial console does.
473     Arrow keys arrive as three character escape sequences."""
474     available = supervisor.runtime.serial_bytes_available
475     if not available:
476         return []
477     data = sys.stdin.read(available)
478     keys = []
479     i = 0
480     while i < len(data):
481         if data[i] == "\x1b" and data[i + 1 : i + 2] == "[" and i + 2 < len(data):
482             keys.append(data[i : i + 3])
483             i += 3
484         else:
485             keys.append(data[i])
486             i += 1
487     return keys
488
489
490 # The board's own buttons stand in for three of the keys
491 try:
```

```
    buttons = keypad.Keys(
        (board.BUTTON1, board.BUTTON2, board.BUTTON3),
        value_when_pressed=False,
        pull=True,
    )
except ValueError:
    # Something else holds the button pins: imported over the top of another
    # program (its objects survive Ctrl-C). Run with the keyboard only.
    buttons = None
    print("Board buttons are in use by another program; keyboard only.")
BUTTON_KEYS = (KEY_LEFT, KEY_RIGHT, KEYS_SHUTTER[0])


def read_buttons():
    """Return the keys the board's buttons stand for, for each button pressed
    since the last call."""
    keys = []
    while buttons is not None:
        event = buttons.events.get()
        if event is None:
            break
        if event.pressed:
            keys.append(BUTTON_KEYS[event.key_number])
    return keys


# --- Main ----------------------------------------------------------------

camera = None
while camera is None:
    try:
        # Finds the camera among the attached devices; the keyboard is one too
        camera = adafruit_usb_host_camera.UVCCamera()
    except ValueError:
        print("Plug a USB camera into a USB host port")
        display_message("Plug in a USB camera", RED)
        time.sleep(1)

mode = camera.find_mode(WIDTH, HEIGHT)
decoder = jpegio.JpegDecoder()
print("Filtered camera ready:", mode)
print("Arrow keys choose the effect, space takes a photo, R remounts CPSAVES.")
print("Board buttons: 1/2 choose the effect, 3 takes a photo.")
camera.start(mode)
display_message("arrows: effect  space: photo  R: remount", seconds=4)

try:
    while True:
        try:
            frame = camera.capture()
        except RuntimeError:
            continue  # no complete frame in time; keep trying
        width, height = decoder.open(camera.add_huffman_tables(frame))
        # The color grids show the picture four times at half size
        active = active_effect()
        grid = active >= FIRST_GRID
```

```
        destination = grid_source if grid else bitmap
        # Shrink larger pictures to fit: each scale step halves the size.
        scale = 0
        while (width >> scale) > destination.width or (
            height >> scale
        ) > destination.height:
            scale += 1
        try:
            decoder.decode(destination, scale=scale)
        except (RuntimeError, ValueError):
            continue  # damaged frame
        effects[active][1](destination)
        picture_layers = grid_layers if grid else [sprite]
        refresh()

        for key in read_keys() + read_buttons():
            if key == KEY_RIGHT:
                effect_index = (effect_index + 1) % len(effects)
                bypass = False
            elif key == KEY_LEFT:
                effect_index = (effect_index - 1) % len(effects)
                bypass = False
            elif key == KEY_UP:
                bypass = True
            elif key == KEY_DOWN:
                bypass = False
            elif key in KEYS_NUMBER:
                effect_index = min(2 * KEYS_NUMBER.index(key) + 1, len(effects) - 1)
                bypass = False
            elif key in KEYS_SHUTTER:
                take_photo(camera, mode, frame)
                break
            elif key in {"r", "R"}:
                remount_saves()
        if active_effect() != active:
            print("Filter:", effects[active_effect()][0])
            display_message(effects[active_effect()][0])
finally:
    camera.stop()
```

https://github.com/adafruit/Adafruit_Learning_System_Guides/blob/main/Fruit_Jam/Fruit_Jam_USB_Host_Camera/camera_effects/code.py

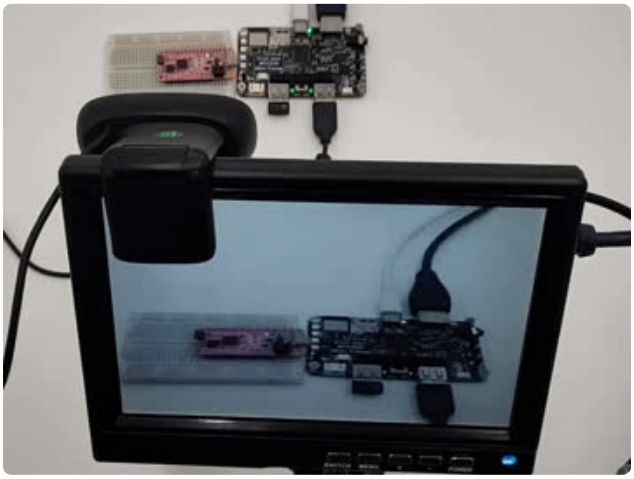
Drive Structure

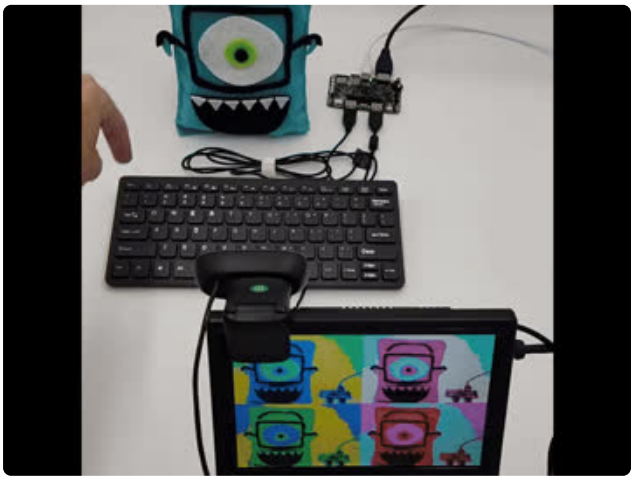
After copying the files, your drive should look like the listing below. It can contain other files as well, but must contain these at a minimum.

Use

You can use the built-in buttons on the Fruit Jam or a USB keyboard to control the camera app.

Choose Effect





On-board Buttons

Cycle through the available effects with the onboard **button 1** and **button 2** on the Fruit Jam.

USB Keyboard

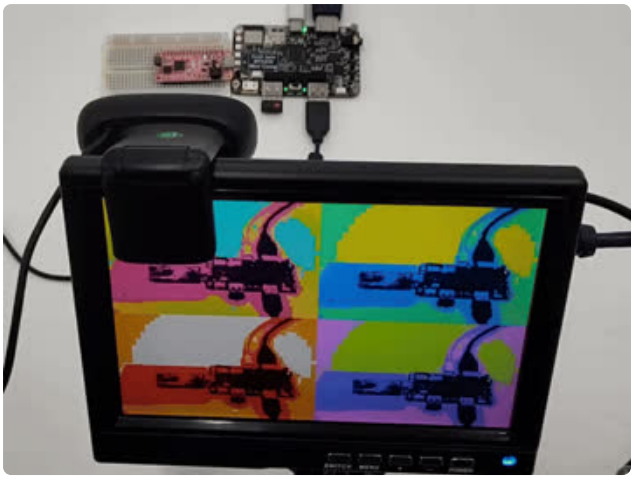
The **left arrow** and **right arrow** keys on a USB keyboard can be used to go through the effects forward and backwards.

The **up arrow** key will pause the current effect switching to a clean pass-through of the video signal.

The **down arrow** key will resume the current effect after it has been paused. Switching back from the clean pass-through to the current effect.

The **number keys 0-9** will jump to different effects in the list. There are just over 20 effects total, each number on the keyboard acts as a hotkey to every other effect in the list. That means around half of the effects can be reached with a single button press, and the other half can be reached in two button presses, the correct hotkey then a left or right arrow press.

Take Photo



On-board Buttons

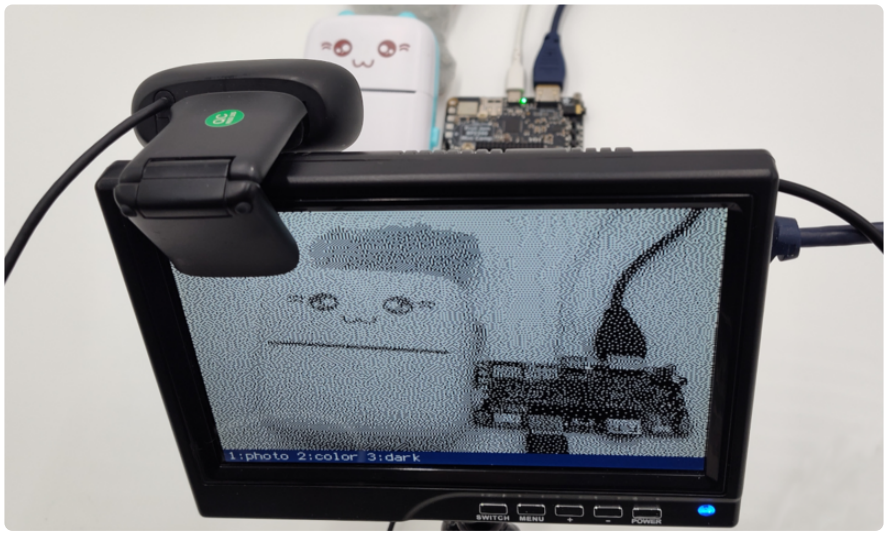
Button 3 on the Fruit Jam will save a photo of the current frame.

USB Keyboard

The **space**, or **enter** buttons will save the current photo.

Both methods will save the photo to an SD card if one is mounted, or the CPSAVES drive if not. Plain photos are saved as **.jpg** files, the version with effects applied is saved as a **.bmp** file.

Camera with BLE Thermal Printer



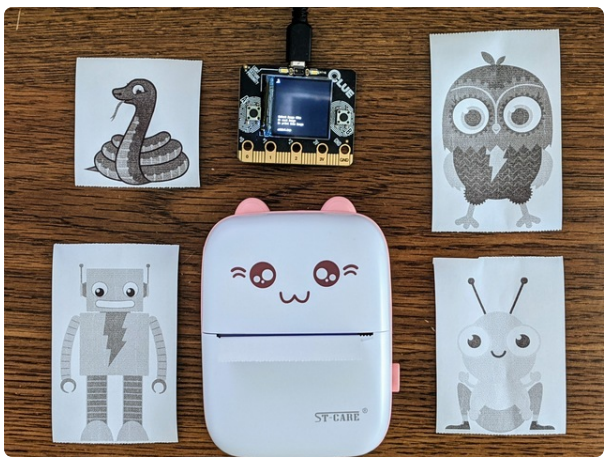
This section of the guide covers a photo printer project that uses cute feline thermal printer that communicates over BLE.

Text emphasized with a blue info:

The CircuitPython code only supports the "Cat" printer models GD02, and MXW01. Other models may not work without changes to the printer library code.

Cat BLE Thermal Printer
This small feline-styled BLE thermal printer is available on Amazon ()

Thanks to bitbank2 () who originally published the Arduino code for interfacing with these printers. And to Jeff for initially porting it to CircuitPython and writing the original cat printer guide for the CLUE.



BLE Thermal "Cat" Printer with CircuitPython
By Jeff Epler

https://learn.adafruit.com/ble-thermal-cat-printer-with-circuitpython

Code

Getting the Program's Files

To use the application, you need to obtain **code.py** with the program, and the other project files to place on the Fruit Jam **CIRCUITPY** drive.

Thankfully, this can be done in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries, the **code.py** file, and other project files in a zip file.

Connect your board to your computer via a known good data+power USB cable. The board should show up in your File Explorer/Finder (depending on your operating system) as a flash drive named **CIRCUITPY**.

Extract the contents of the zip file, copy the **lib** directory files to **CIRCUITPY/lib**. Copy the **code.py** file, as well as **thermalprinter.py** and **fruitjam_ble/** to your **CIRCUITPY** drive. The program should self start.

```
# SPDX-FileCopyrightText: Copyright (c) 2026 Tim Cocks for Adafruit Industries
#
# SPDX-License-Identifier: MIT
"""Photo booth: take a picture with a USB webcam on the Fruit Jam and print it
on a Bluetooth "cat printer" through the Fruit Jam's ESP32-C6.

Live view:  button 1 takes a photo, button 2 switches between the dithered
        and the color preview, button 3 changes the print lightness.
Photo view: button 1 prints, button 2 goes back to the live view,
        button 3 changes the print lightness.

The keys 1, 2 and 3 on the serial console work like the buttons.
"""

import sys
import time

import bitmapfilter
import bitmaptools
import board
import digitalio
import displayio
import jpegio
import keypad
import picodvi
import picogame
import supervisor
import terminalio
import ulab.numpy as np

# fruitjam_ble has to come before adafruit_ble
```

```
import fruitjam_ble  # pylint: disable=wrong-import-order
from adafruit_ble import BLERadio
from adafruit_ble.advertising import Advertisement
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement

import adafruit_usb_host_camera
from thermalprinter import CatPrinter, MXW01Printer

# pylint: disable=global-statement

WIDTH, HEIGHT = 320, 240
# The camera runs in one mode for both the preview and the photo, because
# switching modes can stall some cameras.
CAMERA_WIDTH, CAMERA_HEIGHT = 640, 480
PRINT_WIDTH, PRINT_HEIGHT = 384, 288  # the printer is 384 dots wide
PRINTER_NAMES = ("GB0", "GT0", "MX0", "MX1", "YT0")
# Newer printers with the MXW01 protocol
MXW01_NAMES = ("MXW01",)
FEED_ROWS = 80  # blank rows after a photo, so it clears the tear-off edge

# Thermal prints come out dark, so the photo is lightened with a gamma curve
# before dithering. Button 3 cycles through these.
LIGHTNESS = (("normal", 1.0), ("light", 0.75), ("lighter", 0.55), ("dark", 1.4))

BLACK = picogame.rgb565(0, 0, 0)
WHITE = picogame.rgb565(255, 255, 255)
STATUS_BG = picogame.rgb565(0, 0, 80)

displayio.release_displays()
framebuffer = picodvi.Framebuffer(
    WIDTH,
    HEIGHT,
    clk_dp=board.CKP,
    clk_dn=board.CKN,
    red_dp=board.D0P,
    red_dn=board.D0N,
    green_dp=board.D1P,
    green_dn=board.D1N,
    blue_dp=board.D2P,
    blue_dn=board.D2N,
    color_depth=16,
)
target = picogame.Framebuffer(framebuffer, WIDTH, HEIGHT, native_rgb565=True)

buttons = keypad.Keys(
    (board.BUTTON1, board.BUTTON2, board.BUTTON3), value_when_pressed=False, pull=True
)

# Bitmaps are RGB565_SWAPPED, which is picogame's transfer byte order, so
# picogame draws their memory directly.
preview = displayio.Bitmap(WIDTH, HEIGHT, 65536)  # camera image
shown = displayio.Bitmap(WIDTH, HEIGHT, 65536)  # dithered preview or photo
photo = displayio.Bitmap(CAMERA_WIDTH, CAMERA_HEIGHT, 65536)  # full size photo
print_source = displayio.Bitmap(PRINT_WIDTH, PRINT_HEIGHT, 65536)
# Dithering into a 2-color bitmap crashes (see dither_bug.md), so dither into a
# 65536-color bitmap holding 0 and 65535, and pack the rows with ulab.
```

```
print_dithered = displayio.Bitmap(PRINT_WIDTH, PRINT_HEIGHT, 65536)
BIT_WEIGHTS = np.array([1, 2, 4, 8, 16, 32, 64, 128], dtype=np.uint8)
ROW_BYTES = PRINT_WIDTH // 8


def picogame_bitmap(bitmap):
    return picogame.Bitmap(bitmap, bitmap.width, bitmap.height)


preview_sprite_bitmap = picogame_bitmap(preview)
shown_sprite_bitmap = picogame_bitmap(shown)
image_sprite = picogame.Sprite(shown_sprite_bitmap, x=0, y=0)
status_bar = picogame.Canvas(WIDTH, 14)
status_bar.move(0, HEIGHT - 14)
layers = [image_sprite, status_bar]


def refresh():
    picogame.render(target, layers, None, 0, 0, WIDTH, HEIGHT)


def status(text):
    print(text)
    status_bar.clear(STATUS_BG)
    status_bar.text(3, 1, text, WHITE, terminalio.FONT)
    refresh()


def read_input():
    """Return the button that was pressed (0, 1 or 2) or None."""
    event = buttons.events.get()
    if event and event.pressed:
        return event.key_number
    if supervisor.runtime.serial_bytes_available:
        _key = sys.stdin.read(1)
        if _key in "123":
            return ord(_key) - ord("1")
    return None


# ---- camera ----

camera = None
decoder = jpegio.JpegDecoder()


def open_camera():
    global camera  # noqa: PLW0603
    camera = None
    while camera is None:
        try:
            # Finds the camera among the attached devices (a keyboard, a mouse...)
            camera = adafruit_usb_host_camera.UVCCamera()
        except ValueError:
            status("Plug in a USB camera")
            time.sleep(1)
```

```
    camera.start(camera.find_mode(CAMERA_WIDTH, CAMERA_HEIGHT))


def power_cycle_camera():
    """Some cameras stop sending image data after a while; switching the USB
    host port off and on again brings them back."""
    status("Camera stalled, restarting it")
    power = digitalio.DigitalInOut(board.USB_HOST_5V_POWER)
    power.switch_to_output(True)
    power.value = False
    time.sleep(1.5)
    power.value = True
    power.deinit()
    time.sleep(3)
    open_camera()


timeouts = 0


def capture_jpeg():
    """Return the newest camera frame as a JPEG, or None if none came in time."""
    global timeouts  # noqa: PLW0603
    try:
        frame = camera.capture()
    except RuntimeError:
        timeouts += 1
        if timeouts >= 5:
            timeouts = 0
            power_cycle_camera()
        return None
    timeouts = 0
    return camera.add_huffman_tables(frame)


# ---- image processing ----

lightness = 0


def lighten(bitmap):
    gamma = LIGHTNESS[lightness][1]
    if gamma != 1.0:
        bitmapfilter.lookup(bitmap, lambda v: v**gamma)


def dither(dest, source):
    bitmaptools.dither(
        dest,
        source,
        displayio.Colorspace.RGB565_SWAPPED,
        bitmaptools.DitherAlgorithm.FloydStenberg,
    )


def show_dithered_preview():
```

```
    lighten(preview)
    dither(shown, preview)


def render_photo():
    """Scale, lighten and dither `photo` for the printer and for the screen."""
    bitmaptools.rotozoom(print_source, photo, scale=PRINT_WIDTH / CAMERA_WIDTH)
    lighten(print_source)
    dither(print_dithered, print_source)
    bitmaptools.rotozoom(preview, photo, scale=WIDTH / CAMERA_WIDTH)
    lighten(preview)
    dither(shown, preview)


def packed_rows():
    """The dithered print image as 1 bit per pixel, 1 = black, with the
    leftmost pixel of each byte in the lowest bit."""
    pixels = np.frombuffer(print_dithered, dtype=np.uint16)
    black = np.array(pixels.reshape((PRINT_HEIGHT * ROW_BYTES, 8)) == 0, dtype=np.uint8)
    return np.array(np.dot(black, BIT_WEIGHTS), dtype=np.uint8).tobytes()


# ---- printer ----

ble = None
connection = None
printer = None


def start_ble():
    global ble  # noqa: PLW0603
    if ble is None:
        status("Starting Bluetooth")
        ble = BLERadio(fruitjam_ble.start_bluetooth())


def is_printer(adv):
    name = adv.complete_name or adv.short_name
    if name:
        print("saw", name, adv.address, adv.rssi)
        if name.startswith(PRINTER_NAMES + MXW01_NAMES):
            return True
    return is_mxw01(adv) or (
        isinstance(adv, ProvideServicesAdvertisement) and CatPrinter in adv.services
    )


def is_mxw01(adv):
    name = adv.complete_name or adv.short_name
    if name and name.startswith(MXW01_NAMES):
        return True
    return (
        isinstance(adv, ProvideServicesAdvertisement)
        and MXW01Printer.advertised_uuid in adv.services
    )

```

```
256
257 def connect_printer():
258     """Find and connect to the printer, unless still connected."""
259     global connection, printer  # noqa: PLW0603
260     if connection is not None and connection.connected:
261         return
262     connection = printer = None
263     start_ble()
264     status("Looking for the printer...")
265     found = None
266     for adv in ble.start_scan(ProvideServicesAdvertisement, Advertisement, timeout=15):
267         if is_printer(adv):
268             found = adv
269             break
270     ble.stop_scan()
271     if found is None:
272         raise RuntimeError("No printer found. Is it on?")
273     status("Connecting to " + (found.complete_name or "printer"))
274     connection = ble.connect(found, timeout=10)
275     if CatPrinter not in connection:
276         connection.disconnect()
277         connection = None
278         raise RuntimeError("That is not a cat printer")
279     # Both kinds of printer have the same service UUID but different protocols
280     printer = connection[MXW01Printer if is_mxw01(found) else CatPrinter]
281
282
283 def print_photo():
284     rows = packed_rows()
285     connect_printer()
286     blank = bytes(ROW_BYTES)
287     if isinstance(printer, MXW01Printer):
288         printer.print_bitmap(
289             rows + blank * FEED_ROWS,
290             progress=lambda done: status("Printing %d%%" % (100 * done)),
291         )
292         return
293     for y in range(PRINT_HEIGHT):
294         printer.print_bitmap_row(
295             rows[y * ROW_BYTES : (y + 1) * ROW_BYTES], reverse_bits=False
296         )
297         if y % 32 == 0:
298             status("Printing %d%%" % (100 * y // PRINT_HEIGHT))
299     status("Feeding paper")
300     for _ in range(FEED_ROWS):
301         printer.print_bitmap_row(blank, reverse_bits=False)
302
303
304 # ---- main loop ----
305
306
307 def live_help():
308     status(
309         "1:photo 2:%s 3:%s"
310         % ("color" if dithered else "dither", LIGHTNESS[lightness][0])
311     )
```

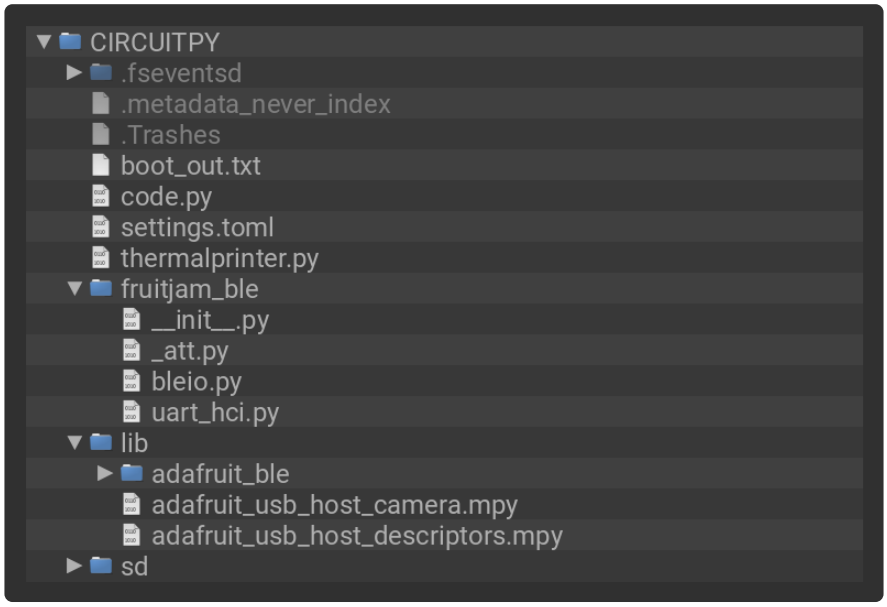
```
312
313
314 def photo_help():
315     status("1:print 2:retake 3:%s" % LIGHTNESS[lightness][0])
316
317
318 open_camera()
319 dithered = True
320 frozen = False
321 live_help()
322 last_poll = time.monotonic()
323 while True:
324     key = read_input()
325     if key == 2:
326         lightness = (lightness + 1) % len(LIGHTNESS)
327         if frozen:
328             render_photo()
329             photo_help()
330         else:
331             live_help()
332     elif frozen:
333         if key == 0:
334             try:
335                 print_photo()
336                 photo_help()
337             except Exception as e:  # pylint: disable=broad-except
338                 # keep the booth running
339                 print("Print failed:", repr(e))
340                 status("Print failed. 1:retry 2:retake")
341         elif key == 1:
342             frozen = False
343             live_help()
344     elif key == 0:
345         status("Taking photo...")
346         jpeg = None
347         while jpeg is None:
348             jpeg = capture_jpeg()
349         decoder.open(jpeg)
350         decoder.decode(photo)
351         render_photo()
352         image_sprite.bitmap = shown_sprite_bitmap
353         frozen = True
354         photo_help()
355     elif key == 1:
356         dithered = not dithered
357         live_help()
358
359     if not frozen:
360         jpeg = capture_jpeg()
361         if jpeg is not None:
362             decoder.open(jpeg)
363             decoder.decode(preview, scale=1)
364             if dithered:
365                 show_dithered_preview()
366                 image_sprite.bitmap = shown_sprite_bitmap
367             else:
```

```
368                 image_sprite.bitmap = preview_sprite_bitmap
369         refresh()
370
371         # The Bluetooth code only handles events when asked, so check on the
372         # printer now and then to notice when it goes away.
373         if connection is not None and time.monotonic() - last_poll > 1:
374             last_poll = time.monotonic()
375             if not connection.connected:
376                 connection = printer = None
```

https://github.com/adafruit/Adafruit_Learning_System_Guides/blob/main/Fruit_Jam/Fruit_Jam_USB_Host_Camera/camera_printer/code.py

Drive Structure

After copying the files, your drive should look like the listing below. It can contain other files as well, but must contain these at a minimum.



Use

You can use the built-in buttons on the Fruit Jam or a USB keyboard to control the printer app.

Change Brightness

Button 3 on the Fruit Jam or the **number 3 key** on a USB keyboard will cycle through the different available brightness levels: normal, light, lighter, and dark. This will lighten or darken the photo before applying the Floyd Stenberg dither which will influence the density of regions with white and black pixels in the resulting printed photo.

Full Color Preview

To see a full color preview on the display, press **button 2** on the Fruit Jam or the **number 2 key** on a USB keyboard.

Print Photo

Printing the photo is a two step process, first freeze the current frame and confirm it looks how you want. If it looks good, then send it to the printer. Use **button 3** on the Fruit Jam or the **number 3 key** on a USB keyboard. First press to capture the current frame and freeze it on the display, then press again to send it to the thermal printer.

If the captured frame isn't right, press **button 2** on the Fruit Jam or the **number 2 key** on a USB keyboard to re-take the photo instead of sending it to the printer.