Motion-Reactive LED Devil Horns

Created by Erin St Blaine

https://learn.adafruit.com/motion-reactive-led-devil-horns

Last updated on 2026-08-13 12:27:06 PM UTC

Table of Contents

Overview 3

- Parts
- You'll Also Need

Wiring Diagram 6

CircuitPython 6

- CircuitPython Quickstart
- Safe Mode
- Flash Resetting UF2

Code the Horns 10

- Upload the Code and Libraries to the RP2040 Prop-Maker Feather
- Configuration
- Accelerometer Setup
- Resting Ember Pulse
- Shake Gesture
- Head-Tilt Gesture
- Progressive Horn Charge
- Layering the Effects

Electronics Assembly 23

- Troubleshooting

Create the Horns 25

Final Assembly 29

Overview

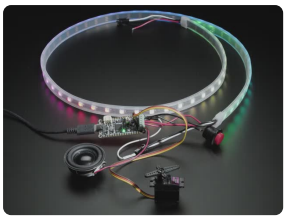
Bring out your naughty side with these DIY devil horns. The horns are made from paper mache and filled with high-density LED lights that flicker and glow like burning embers.

The lights are controlled by a PropMaker Feather RP2040. This board has got an onboard accelerometer that can read your head gestures and trigger different animations based on your movements. There is CircuitPython sample code that animates the horns with a slow lava-flow animation when you tilt your head, and a flare-up spark animation when you give your head a shake.

This is an intermediate level project that requires a little careful soldering and a lot of hot glue.

The power behind this project is the Adafruit RP2040 PropMaker Feather. It has an onboard accelerometer, onboard battery charging, and a screw terminal for easy wire connection. This board is a workhorse and can make your project interactive in a lot of different ways.

Parts



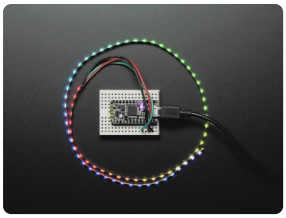
Featured

Adafruit RP2040 Prop-Maker Feather with I2S Audio Amplifier

The Adafruit Feather series gives you lots of options for a small, portable, rechargeable microcontroller board. By picking a feather and stacking on a FeatherWing you can create...

https://www.adafruit.com/product/5768

These are the highest density pixels we carry. They're tiny and a bit delicate, and can be tricky to solder, but they're just right for illuminating a small space and getting buttery smooth LED animations.



Featured

Ultra Skinny NeoPixel 1515 LED Strip 4mm wide

We have all sorts of LED strips for a wide range of needs. Chonky strips? We got those!

https://www.adafruit.com/product/4368

Snap-on Enclosure for Adafruit FeatherWing RP2040 Prop-Maker

Here is a cute and minimal enclosure for your Adafruit RP2040 Prop-Maker Feather with I2S Audio Amplifier to keep it safe...

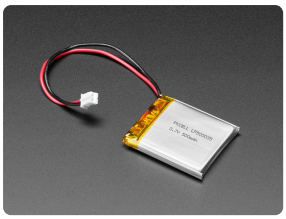
https://www.adafruit.com/product/6504

JST 2-pin Extension Cable with On/Off Switch - JST PH2

By popular request - we now have a way you can turn on-and-off Lithium Polymer batteries without unplugging them.This PH2 Female/Male JST 2-pin Extension...

https://www.adafruit.com/product/3064

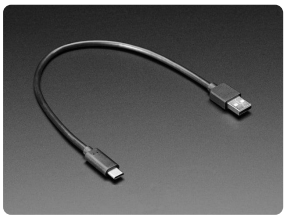
This little battery will run the horns for several hours, depending on how bright you set them. It hides easily on top of your 2" headband and can be recharged via USB, without having to remove it and put it on a charger.



Lithium Ion Polymer Battery - 3.7v 500mAh

Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...

https://www.adafruit.com/product/1578



USB Type A to Type C Cable - 1ft - 0.3 meter

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/4473

Get some hook up wire in 3 different colors: for this project I'm using black, green, and red.

Silicone Cover Stranded-Core Wire - 2m 26AWG Black (×3)

Silicone-sheathing wire is super-flexible and soft, and its also strong! Able to handle up to 200°C and up to 600V, it will do when PVC covered wire wimps out. We like this wire...

https://www.adafruit.com/product/1881

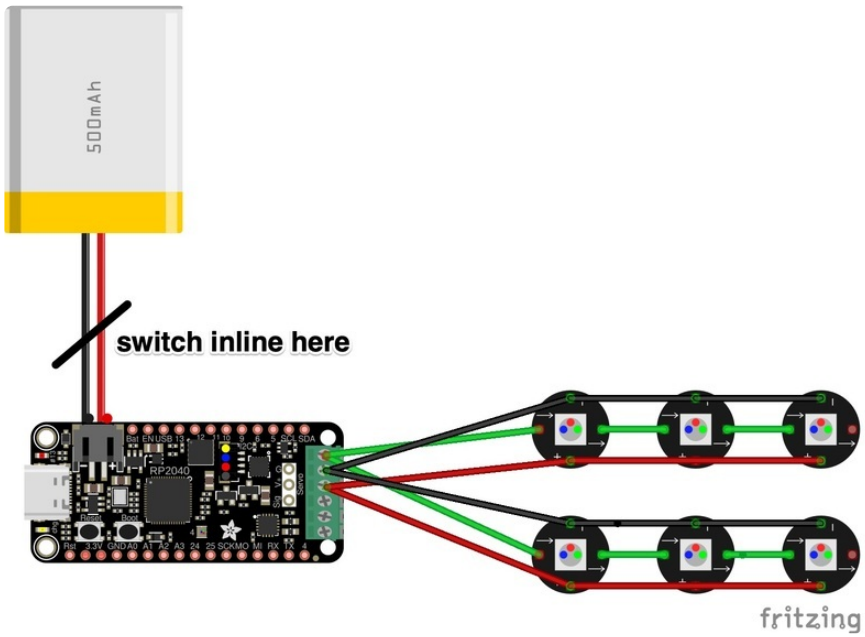
You'll Also Need

- A 2" wide headband - I got mine at Dollar Tree, or here's one on Amazon (https://adafru.it/1aHo)
- Titebond III Wood Glue (https://adafru.it/1aHp)
- Paper towels, saran wrap & aluminum foil
- Wig clips or hair combs to secure the headband to your hair
- Faux Flowers or other decor items, and craft wire to hold it all together

Tools

- Soldering iron & accessories
- A tiny flat head screwdriver - here's a great set from our shop (http://adafru.it/424)
- Hot glue gun & sticks

Wiring Diagram



This project uses two NeoPixel strips wired in parallel, so they're always perfectly in sync with each other. The NeoPixel strands attach to the screw terminals as shown:

- +5v to +5v
- G to G
- DI to NEO

Plug your inline switch into the JST port on the board, and plug the battery into the other end of the switch.

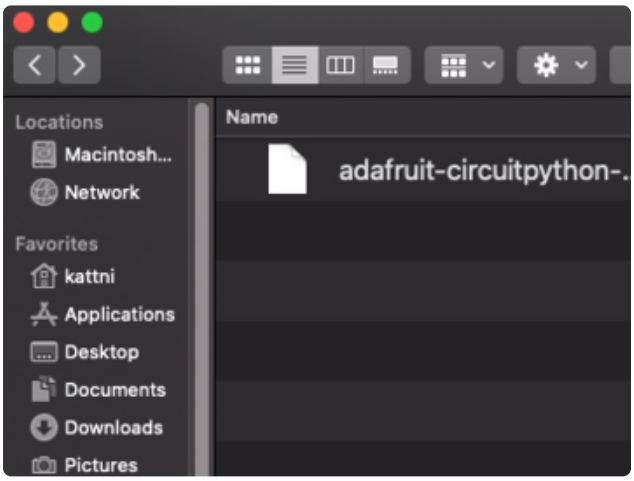
CircuitPython

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

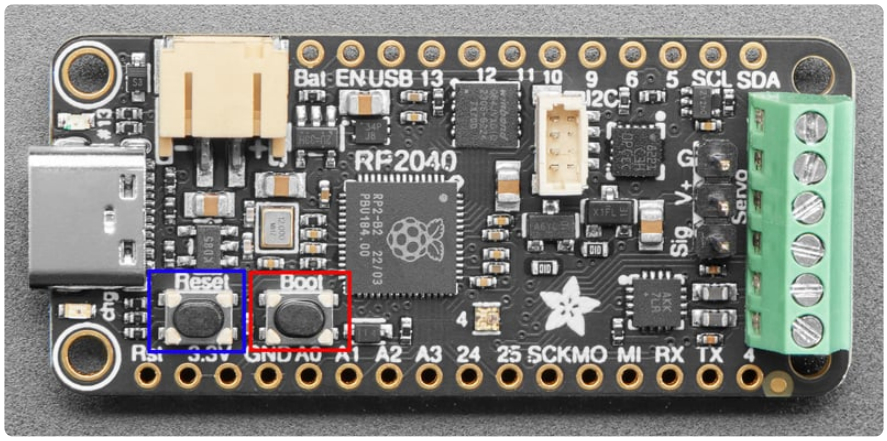
Follow this step-by-step to quickly get CircuitPython running on your board.

https://adafru.it/18EH



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

Save it wherever is convenient for you.

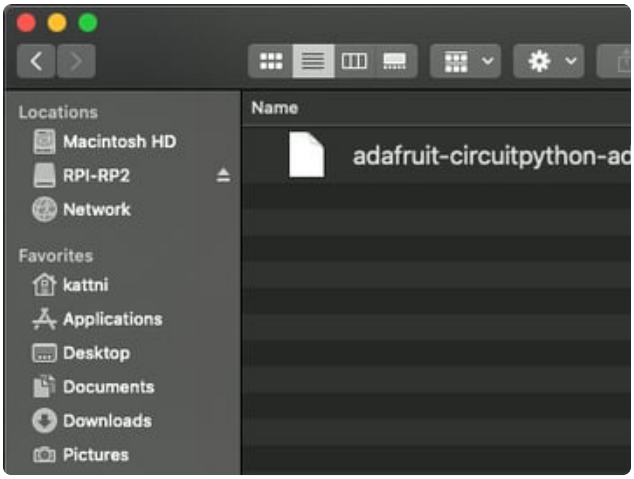


To enter the bootloader, hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset button** (highlighted in red or blue above). **Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!**

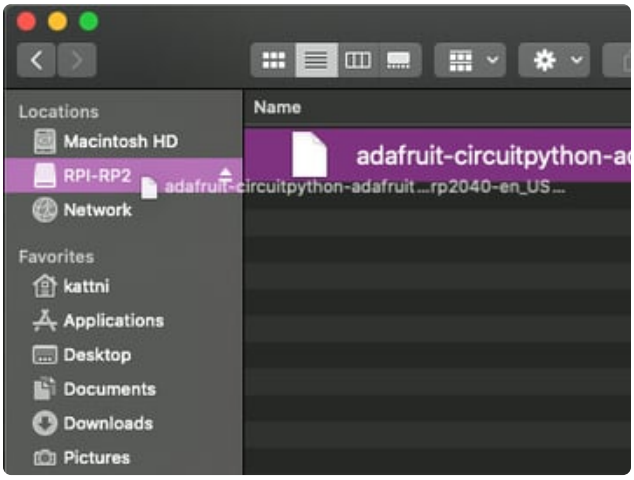
If the drive does not appear, release all the buttons, and then repeat the process above.

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.

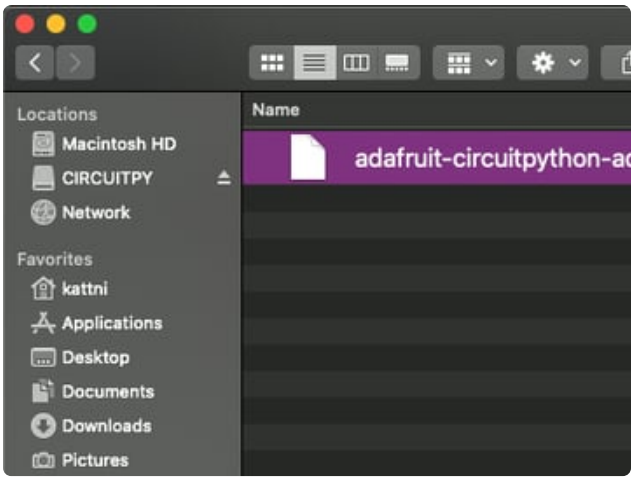
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **RPI-RP2**.



Drag the **adafruit_circuitpython_etc.uf2** file to **RPI-RP2.**



The **RPI-RP2** 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 RPI-RP2. 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.

https://adafru.it/1aHm

Code the Horns

You can access the code and necessary libraries by downloading the Project Bundle.

To do this, click on the **Download Project Bundle** button in the window below. It will download to your computer as a zipped folder.

```
# SPDX-FileCopyrightText: 2026 Erin St Blaine for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Motion-Reactive Devil Horns
# Quiet ember pulse at rest
# Fast traveling flare on sharp movement
# Progressive demon charge on head tilt
#
# Adafruit RP2040 Prop-Maker Feather
#
# Pixel 0 = horn base
# Last pixel = horn tip

import math
import random
import time

import adafruit_lis3dh
import board
import digitalio
import neopixel


# ------------------------------------------------------------------------
# CONFIGURATION
# ------------------------------------------------------------------------

PIXEL_COUNT = 21
BRIGHTNESS = 0.9
FRAME_DELAY = 0.02

# Ember pulse timing.
EMBER_PULSE_SPEED = 1.4

# Small random flickers near the base.
```

```
EMBER_FLICKER_CHANCE = 8
EMBER_FLICKER_DECAY = 0.88

# Sharp movement required to trigger the traveling flare.
FLARE_TRIGGER = 6.0
FLARE_COOLDOWN = 0.9

# Traveling flare behavior.
FLARE_STEP_TIME = 0.02
FLARE_WIDTH = 5

# Head tilt behavior.
#
# X axis is near 0 when your head is level.
# Tilt begins affecting the horns around this value.
TILT_TRIGGER = 3.0

# Around this X reading, the horn is fully charged.
TILT_FULL = 5.0

# Lower = slower/smoother response.
# Higher = faster response.
TILT_SMOOTHING = 0.10


# ---------------------------------------------------------------------------
# POWER AND NEOPIXEL SETUP
# ---------------------------------------------------------------------------

external_power = digitalio.DigitalInOut(board.EXTERNAL_POWER)
external_power.direction = digitalio.Direction.OUTPUT
external_power.value = True

pixels = neopixel.NeoPixel(
    board.EXTERNAL_NEOPIXELS,
    PIXEL_COUNT,
    brightness=BRIGHTNESS,
    auto_write=False,
    pixel_order=neopixel.GRB,
)


# ---------------------------------------------------------------------------
# ACCELEROMETER SETUP
# ---------------------------------------------------------------------------

i2c = board.I2C()
accelerometer = adafruit_lis3dh.LIS3DH_I2C(i2c)
accelerometer.range = adafruit_lis3dh.RANGE_4_G

previous_x, previous_y, previous_z = accelerometer.acceleration

tilt_level = 0.0
last_flare_time = -FLARE_COOLDOWN


```

```
# ---------------------------------------------------------------------------
# EMBER STATE
# ---------------------------------------------------------------------------

ember_flicker = [0.0] * PIXEL_COUNT


# ---------------------------------------------------------------------------
# FLARE STATE
# ---------------------------------------------------------------------------

flare_active = False
flare_position = 0.0
last_flare_step = time.monotonic()


# ---------------------------------------------------------------------------
# COLOR HELPERS
# ---------------------------------------------------------------------------

def scale_color(color, amount):
    """Scale an RGB color by a brightness amount."""

    amount = max(0.0, min(1.0, amount))

    return tuple(
        min(255, int(channel * amount))
        for channel in color
    )


def blend_colors(base_color, overlay_color, amount):
    """Blend two RGB colors."""

    amount = max(0.0, min(1.0, amount))

    return tuple(
        int(base + (overlay - base) * amount)
        for base, overlay in zip(base_color, overlay_color)
    )


def ember_color(position, pulse_amount):
    """Return the resting ember color for one pixel."""

    # Fade strongly from the horn base toward the tip.
    position_fade = max(
        0.0,
        1.0 - position / (PIXEL_COUNT - 1)
    )

    # Concentrate most of the glow in the lower half.
    position_fade = position_fade ** 1.8

    # Gentle breathing glow.
    brightness = (
```

```
        0.10
        + pulse_amount * 0.18
    ) * position_fade

    # Dark red with a trace of orange.
    color = (255, 24, 3)

    return scale_color(color, brightness)


def flare_color(strength):
    """Create the bright yellow-orange motion flare."""

    strength = max(0.0, min(1.0, strength))

    red = 255
    green = int(100 + 155 * strength)
    blue = int(8 + 120 * strength)

    return (
        red,
        min(255, green),
        min(135, blue),
    )


def charge_color(strength):
    """Red-orange glow for the sustained head-tilt charge."""

    strength = max(0.0, min(1.0, strength))

    red = 255
    green = int(30 + 90 * strength)
    blue = int(2 + 12 * strength)

    return (
        red,
        green,
        blue,
    )


# ---------------------------------------------------------------------------
# MOTION + TILT DETECTION
# ---------------------------------------------------------------------------

def read_motion_and_tilt():
    """Measure sharp motion and update the head-tilt level."""
    # pylint: disable=global-statement

    global previous_x, previous_y, previous_z
    global tilt_level

    x, y, z = accelerometer.acceleration

    delta_x = x - previous_x
```

```
    delta_y = y - previous_y
    delta_z = z - previous_z

    previous_x = x
    previous_y = y
    previous_z = z

    acceleration_change = math.sqrt(
        delta_x * delta_x
        + delta_y * delta_y
        + delta_z * delta_z
    )

    # Head tilt uses the X axis.
    # abs() means left and right tilts behave identically.
    tilt_amount = abs(x)

    raw_tilt = (
        tilt_amount - TILT_TRIGGER
    ) / (TILT_FULL - TILT_TRIGGER)

    raw_tilt = max(0.0, min(1.0, raw_tilt))

    # Smooth the charge so it grows and recedes rather than jittering.
    tilt_level += (
        raw_tilt - tilt_level
    ) * TILT_SMOOTHING

    return acceleration_change


# ---------------------------------------------------------------------------
# EMBER ANIMATION
# ---------------------------------------------------------------------------

def update_embers():
    """Update small, irregular ember flickers."""

    for index in range(PIXEL_COUNT):
        ember_flicker[index] *= EMBER_FLICKER_DECAY

    # Occasional subtle flare-ups near the horn base.
    if random.random() < EMBER_FLICKER_CHANCE / 100.0:
        flicker_pixel = random.randint(0, min(4, PIXEL_COUNT - 1))
        ember_flicker[flicker_pixel] = random.uniform(0.15, 0.45)

        # Let some flicker spill into the next pixel.
        if flicker_pixel + 1 < PIXEL_COUNT:
            ember_flicker[flicker_pixel + 1] = max(
                ember_flicker[flicker_pixel + 1],
                ember_flicker[flicker_pixel] * 0.45,
            )


# ---------------------------------------------------------------------------
# TRAVELING FLARE
```

```
# ------------------------------------------------------------------------

def start_traveling_flare():
    """Start a bright flare at the horn base."""
    # pylint: disable=global-statement

    global flare_active
    global flare_position
    global last_flare_step

    flare_active = True
    flare_position = -2.0
    last_flare_step = time.monotonic()


def update_traveling_flare():
    """Move the flare toward the horn tip."""
    # pylint: disable=global-statement

    global flare_active
    global flare_position
    global last_flare_step

    if not flare_active:
        return

    current_time = time.monotonic()

    elapsed_steps = int(
        (current_time - last_flare_step) / FLARE_STEP_TIME
    )

    if elapsed_steps > 0:
        flare_position += elapsed_steps
        last_flare_step += elapsed_steps * FLARE_STEP_TIME

    # End after the flare tail has fully moved past the tip.
    if flare_position - FLARE_WIDTH > PIXEL_COUNT - 1:
        flare_active = False


def flare_amount_for_pixel(index):
    """Return flare brightness for a particular pixel."""

    if not flare_active:
        return 0.0

    distance = flare_position - index

    if distance < 0 or distance > FLARE_WIDTH:
        return 0.0

    normalized = 1.0 - distance / FLARE_WIDTH

    # Keeps the flare front broad and bright.
    return normalized ** 0.45
```

```
317
318
319  # ---------------------------------------------------------------------------
320  # HEAD-TILT CHARGE
321  # ---------------------------------------------------------------------------
322
323  def charge_amount_for_pixel(index):
324      """Return how strongly the tilt effect affects this pixel."""
325
326      if tilt_level <= 0.0:
327          return 0.0
328
329      # As tilt increases, the glow climbs farther toward the horn tip.
330      fill_position = tilt_level * (PIXEL_COUNT + 1)
331
332      distance = fill_position - index
333
334      if distance <= 0:
335          return 0.0
336
337      # Pixels comfortably below the fill line get the full tilt strength.
338      if distance >= 2.0:
339          return tilt_level
340
341      # Soft leading edge.
342      return tilt_level * (distance / 2.0)
343
344
345  # ---------------------------------------------------------------------------
346  # DISPLAY
347  # ---------------------------------------------------------------------------
348
349  def draw_pixels():
350      """Draw ember pulse, tilt charge, and motion flare."""
351
352      current_time = time.monotonic()
353
354      # Slow breathing pulse.
355      pulse_wave = (
356          math.sin(current_time * EMBER_PULSE_SPEED) + 1.0
357      ) / 2.0
358
359      # Keep the pulse gentle rather than fading completely out.
360      pulse_amount = 0.25 + pulse_wave * 0.75
361
362      for index in range(PIXEL_COUNT):
363          base_color = ember_color(
364              index,
365              pulse_amount,
366          )
367
368          # Add small independent flickers.
369          if ember_flicker[index] > 0:
370              flicker_color = scale_color(
371                  (255, 48, 4),
372                  ember_flicker[index],
```

```
            )

            base_color = tuple(
                min(
                    255,
                    base_color[channel] + flicker_color[channel],
                )
                for channel in range(3)
            )

        # ---------------------------------------------------------
        # HEAD-TILT CHARGE
        # ---------------------------------------------------------

        charge_amount = charge_amount_for_pixel(index)

        if charge_amount > 0:
            charge_overlay = charge_color(charge_amount)

            base_color = blend_colors(
                base_color,
                charge_overlay,
                0.20 + charge_amount * 0.70,
            )

        # ---------------------------------------------------------
        # TRAVELING FLARE
        # ---------------------------------------------------------

        flare_amount = flare_amount_for_pixel(index)

        if flare_amount > 0:
            overlay = flare_color(flare_amount)

            overlay_amount = (
                0.72 + flare_amount * 0.28
            )

            final_color = blend_colors(
                base_color,
                overlay,
                overlay_amount,
            )
        else:
            final_color = base_color

        pixels[index] = final_color

    pixels.show()


# ---------------------------------------------------------------------
# MAIN LOOP
# ---------------------------------------------------------------------

try:
```

```
    while True:
        motion_change = read_motion_and_tilt()
        now = time.monotonic()

        if (
            motion_change >= FLARE_TRIGGER
            and now - last_flare_time >= FLARE_COOLDOWN
        ):
            start_traveling_flare()
            last_flare_time = now

        update_embers()
        update_traveling_flare()
        draw_pixels()

        time.sleep(FRAME_DELAY)

finally:
    pixels.fill((0, 0, 0))
    pixels.show()
    external_power.value = False
```

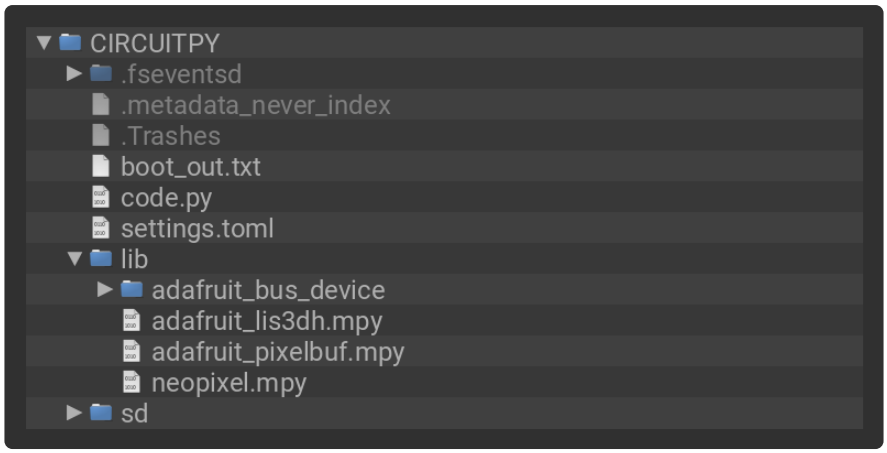
https://github.com/adafruit/Adafruit_Learning_System_Guides/blob/main/Devil_Horns/code.py

Upload the Code and Libraries to the RP2040 Prop-Maker Feather

After downloading the Project Bundle, plug your RP2040 Prop-Maker Feather into the computer's USB port with a known good USB data+power cable. You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called **CIRCUITPY**. Unzip the folder and copy the following items to the RP2040 PropMaker Feather's **CIRCUITPY** drive.

- **lib** folder
- **code.py**

Your RP2040 PropMaker Feather **CIRCUITPY** drive should look like this after copying the **lib** folder, and the **code.py** file.



The code gives the horns three different behaviors. At rest, they glow with a slow, smoldering ember pulse. A sharp head shake sends a bright flare racing from the base of the horns toward the tips. Tilt your head to either side and the horns gradually "charge up," glowing brighter and farther toward the tips the farther you lean. The two gesture effects can overlap with the ember animation, which keeps the horns feeling alive all the time.

Configuration

The main settings are gathered together near the top of the code so you can easily customize the look and responsiveness of the horns.

```
PIXEL_COUNT = 21
BRIGHTNESS = 0.9
FRAME_DELAY = 0.02
```

`PIXEL_COUNT` should match the number of NeoPixels in your horn strand. In this build, there are 21 pixels, with pixel `0` at the base of the horn and the last pixel at the tip.

`BRIGHTNESS` sets the maximum overall brightness. Use a number between `0.0` and `1.0`.

`FRAME_DELAY` controls how often the animation updates. Lower numbers update more quickly.

Accelerometer Setup

The PropMaker Feather's onboard LIS3DH accelerometer is what makes the horns gesture-reactive.

```
i2c = board.I2C()
accelerometer = adafruit_lis3dh.LIS3DH_I2C(i2c)
accelerometer.range = adafruit_lis3dh.RANGE_4_G
```

The code uses the sensor in two different ways.

For the **shake gesture**, it looks at how much the X, Y, and Z acceleration readings have changed since the previous frame.

For the **head tilt**, it watches the X-axis directly. Since the Feather is mounted with the USB port pointing forward on top of the head, the X-axis changes strongly when you tilt your head from side to side.

If you've oriented your board differently (i.e. anything other than right on top of the head with the USB port facing forward) you will need to adjust the code to read from a different axis in order to make that animation work.

Resting Ember Pulse

When you're holding still, the horns glow like banked coals instead of simply sitting at one solid color.

```
EMBER_PULSE_SPEED = 1.4

EMBER_FLICKER_CHANCE = 8
EMBER_FLICKER_DECAY = 0.88
```

`EMBER_PULSE_SPEED` controls the speed of the slow breathing effect.

`EMBER_FLICKER_CHANCE` controls how often a small random flicker appears near the base of the horn. Raise this number for more activity.

`EMBER_FLICKER_DECAY` controls how quickly those individual flickers disappear. A value closer to `1.0` makes them linger longer.

The base ember color is created in `ember_color()`. If you'd like to change the color, the numbers are red, green, blue with values from 0 to 255.

```
color = (255, 24, 3)
```

Shake Gesture

The shake effect is based on **change in acceleration**, rather than the absolute orientation of the board.

Every frame, the code compares the current accelerometer readings with the previous readings.

A sharp head shake causes this value to jump.

The most useful adjustment for this gesture is:

`FLARE_TRIGGER 6.0`

A higher number makes it less sensitive, a lower number more sensitive.

There is also a cooldown:

`FLARE_COOLDOWN = 0.9`

This prevents one vigorous movement from repeatedly restarting the animation. Make this number higher if you're getting too many re-triggers.

Head-Tilt Gesture

The second gesture works differently. Instead of looking for sudden motion, it measures how far the head is tilted. You can tilt in either direction.

`TILT_TRIGGER` determines how far you need to tilt before the charge begins.

`TILT_FULL` determines the tilt reading at which the horn reaches its full charge.

Progressive Horn Charge

The amount of tilt also determines **how far up the horn the glow travels**.

`fill_position = tilt_level * (PIXEL_COUNT + 1)`

At a small tilt, only the pixels near the base are affected. As the tilt increases, the fill position moves closer to the tip.

Layering the Effects

The real trick in this code is that these aren't three completely separate animations. They are drawn in layers.

Inside `draw_pixels()`, each pixel begins with the resting ember color.

```
base_color = ember_color(
    index,
    pulse_amount,
)
```

The random ember flicker is added next.

Then the head-tilt charge is blended over the ember:

```
base_color = blend_colors(
    base_color,
    charge_overlay,
    0.20 + charge_amount * 0.70,
)
```

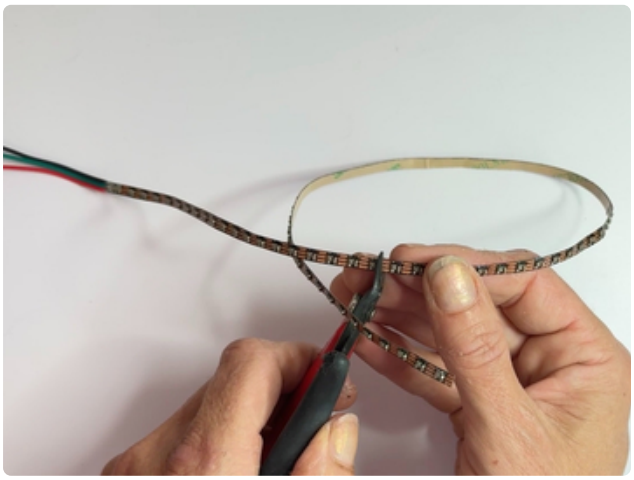
Finally, if a traveling flare is passing through that pixel, the bright flare is blended on top.

```
final_color = blend_colors(
    base_color,
    overlay,
    overlay_amount,
)
```

This layering means the underlying ember animation never really stops. The tilt and shake effects simply build on top of it.

You can even tilt your head while triggering a flare, and both gestures will contribute to the final color.

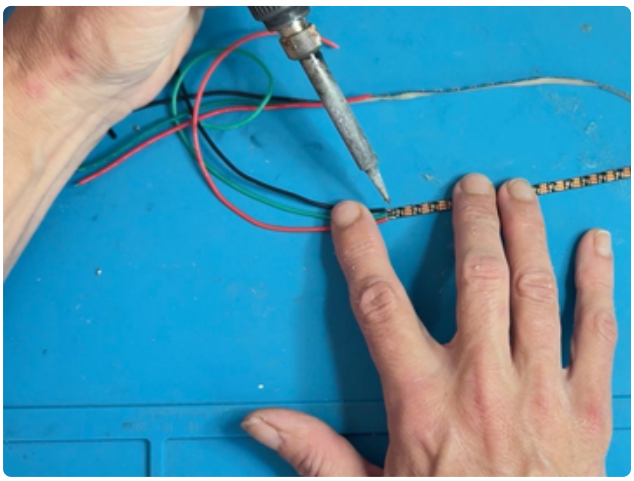
Electronics Assembly



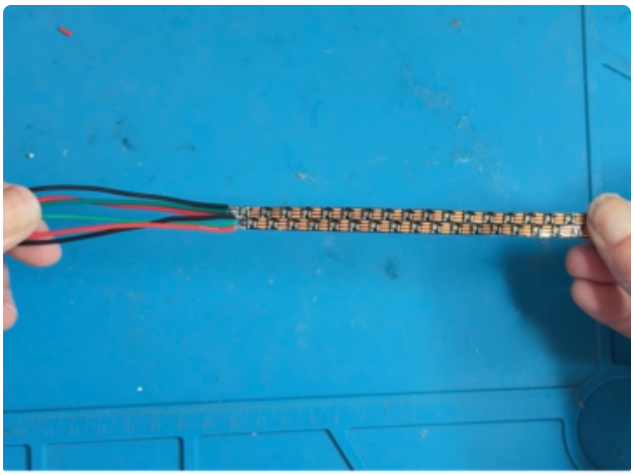
The NeoPixel strip comes with three wires attached to the IN end for easy connection. These will be used for the first horn.

Cut the strip to the length you want for your first horn. It's not a bad idea to make it a little longer than you think - lengthening this strip is tricky, but shortening it later is easy.

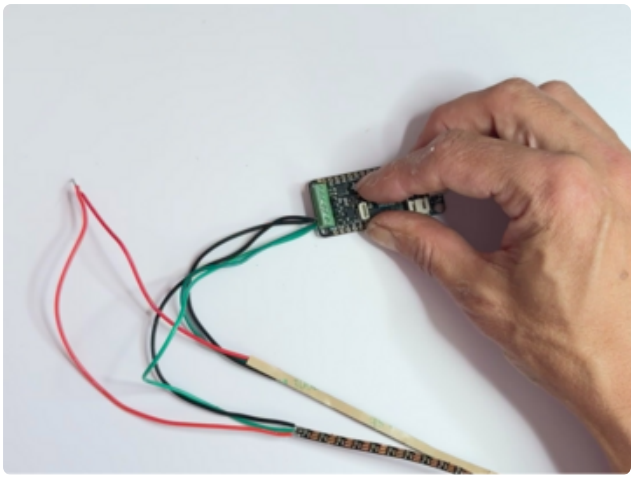
It's helpful to cut on the very front edge of the copper pads, leaving a full pad on the cut-off side to make it easier to solder. These pads are tiny, and soldering to a half-size pad can be extra tricky.



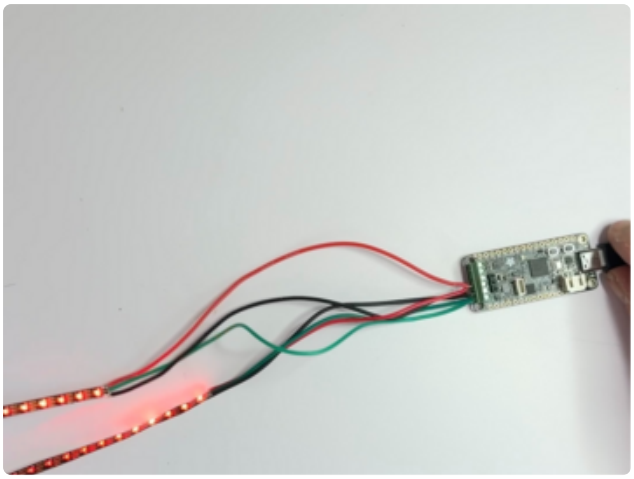
Solder a red, black, and green wire to the IN end of the cut strip to make a second strip. Be sure to get the wires in the correct configuration - you can use the first strip as a reference, just make it match.

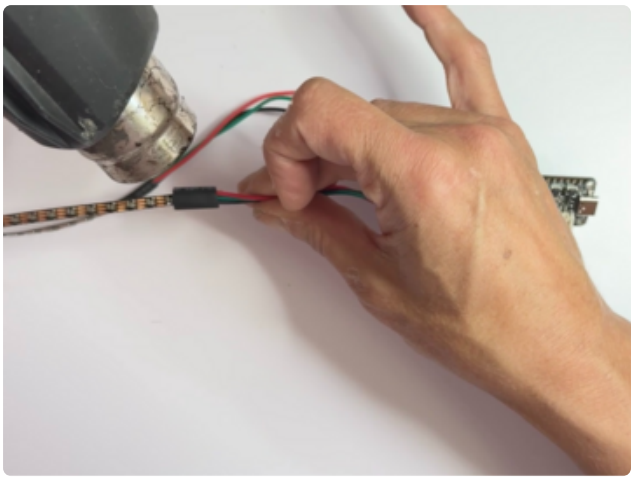


Trim the second strip to match the length of the first. Count the number of LEDs in your strip and write this number down somewhere. We'll use it later in the code to refine the animations.



Strip a little shielding from the other end of the wires. Twist the two color coded pairs together. Insert both black wires into G, both green into NEO and both red into +5v on the PropMaker. Tug on the wires to make sure the connection is secure.





Plug in your battery or USB cable to power up the PropMaker. If all goes well, your lights will come on. Shake the controller to test the animation modes.

If everything's working, add some heat shrink to the wire connections on the LED strips to hold the solder joints firmly in place.

Troubleshooting

If your lights don't come on or behave the way you expect, here are a few things to try:

- Try re-uploading the software. Check the serial console for any error messages being printed there.

- Make sure your Data IN is connected to the NEO pin and double check all your wiring.
- Try disconnecting the strips and connecting just one at a time. If your solder joints are bridging or touching at any point, or if the wires are crossed at all, the whole project will short out. If one works and the other doesn't, check your solder joints. This is a tiny, tricky soldering job so take your time and double check all your connections.

Create the Horns

Create the basic shape for your horns using aluminum foil. You can make them long and curvy if you want, but too many tight twists will make the foil difficult to remove later on. I went with a basic goat-horn shape.

Wrap the horn forms in saran wrap. This will keep the outer layer from sticking to the foil.

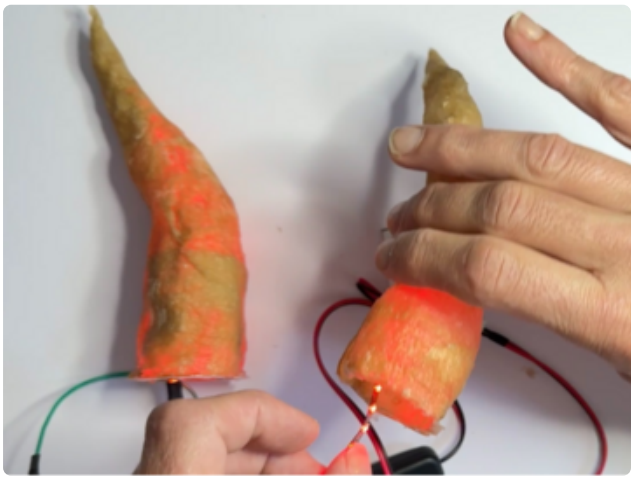
Cut up one or two paper towels into small strips. Squeeze some wood glue onto a disposable surface. I'm using Titebond III - this is one of the strongest and easiest to work-with glues out there. It takes a while to dry but is really strong, and the resulting material feels a bit like leather, with a slight flexibility to it.

Mix a little water in with the glue using a paint brush or stirrer you don't care about -- this stuff is hard to get out once it starts drying. Just add enough water so the glue is a bit runny and more like a paint consistency than a glue consistency.

Dip the paper towel strips in the glue mixture and smooth them down over the form, rubbing with your fingers to obscure any obvious edges. The paper towels soak up a lot of glue so you can keep the paper mache fairly thin. This glue will dry very strong, so don't use more than 1-2 layers. The thicker it is, the less light will get through.

Let them dry for at least 24 hours.

The paper will dry a darker brown color, which to me looks just like goat horns. They also have an interesting texture, thanks to the paper towels. They are sandable and paintable, so you may add details and shading if you like, but remember that an opaque paint will also block the light transmission, so be sparing. I'm really happy with the way they look with no paint whatsoever.

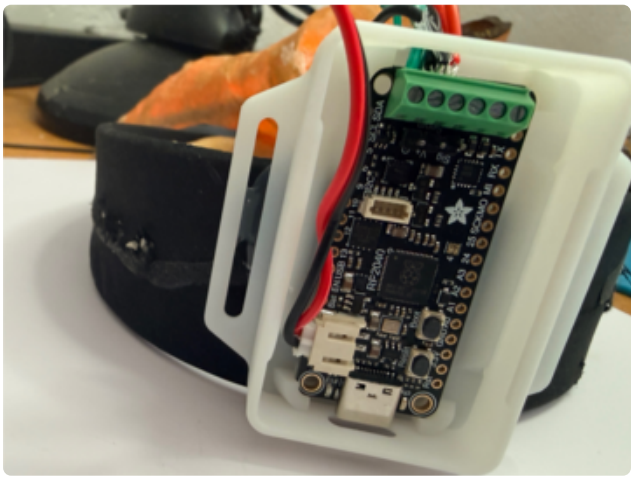


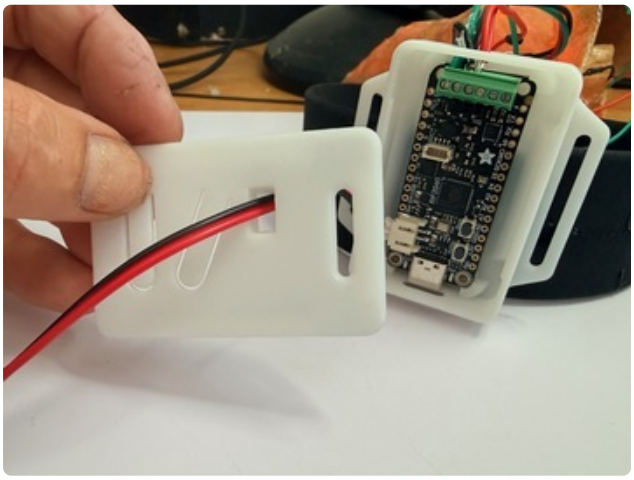
Peel the adhesive off the back of the LED strips and insert them into the horns with the lights facing inwards, so the whole horn illuminates. You'll want to stick the lights firmly to the back wall of the horns - so spend a little time deciding which way they'll face on the headband before you commit.

The adhesive on the strips is helpful for initial placement but won't hold up to longer term use. These strips really look best when they're stuck to one wall rather than "floating" in the middle - if the front side of the lights get too close to the edge you can see the individual pixels rather than the diffused glow.

Test them out with the animations by shaking and tilting the board. You can change the brightness of the LEDs in the code, and tweak them to get exactly the look you're going for.

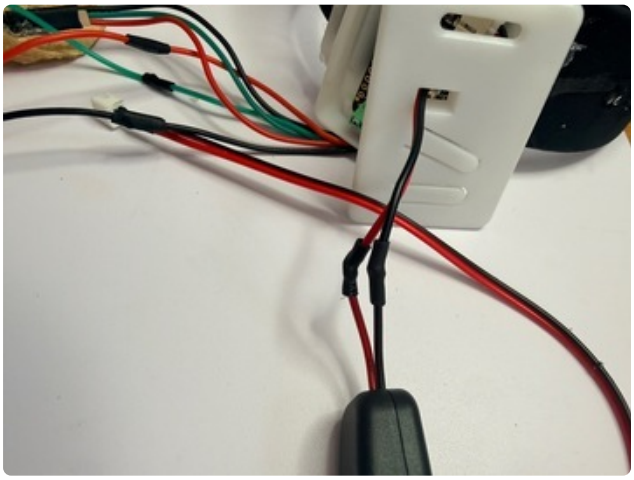
Final Assembly





Place the PropMaker inside the plastic case. You will need to remove the wires from the screw terminal to get it in there, but this is a great opportunity to adjust the length of the wires so they just reach the board without a lot of extra loops.

Thread the female connector from the switch through the hole in the top of the case. Plug it into into the onboard JST connector.



Close up the case and use some hot glue to secure it to the very top of the headband. I used my heat gun to bend the side slots down just a bit so it forms to the top of the headband a little better, and added a little glue to the top of the case to keep it from popping off.

The head-tilt code works best if the board is placed directly on the top of the head, with the USB port facing forwards.

Secure the on/off switch on one side of the case, and carefully glue the battery down to the other side. Your placement may vary, depending on how you're planning to mount the horns, but put the switch someplace accessible. Don't worry about the battery being accessible - the PropMaker has onboard charging capabilities, so you won't need to remove the battery for charging.

While you're charging, make sure the switch is open / turned on. The switch makes a physical break in the line from the battery, so the controller can't charge the battery if the switch is in the closed / off position.

Mount the horns on the headband with hot glue. I used a whole lot of glue, some on the inside edge and some on the outside, so they feel super secure. This paper mache mixture is flexible like leather and gives a really good surface hold with the hot glue.

Add flowers, grapes, skulls, or lace to hide the electronics. I used craft wire and hot glue to secure everything. I also sewed a couple of wig clips to the underside of the headband so they grab my hair, allowing me to toss my head with abandon on the dance floor without fear of losing my horns.