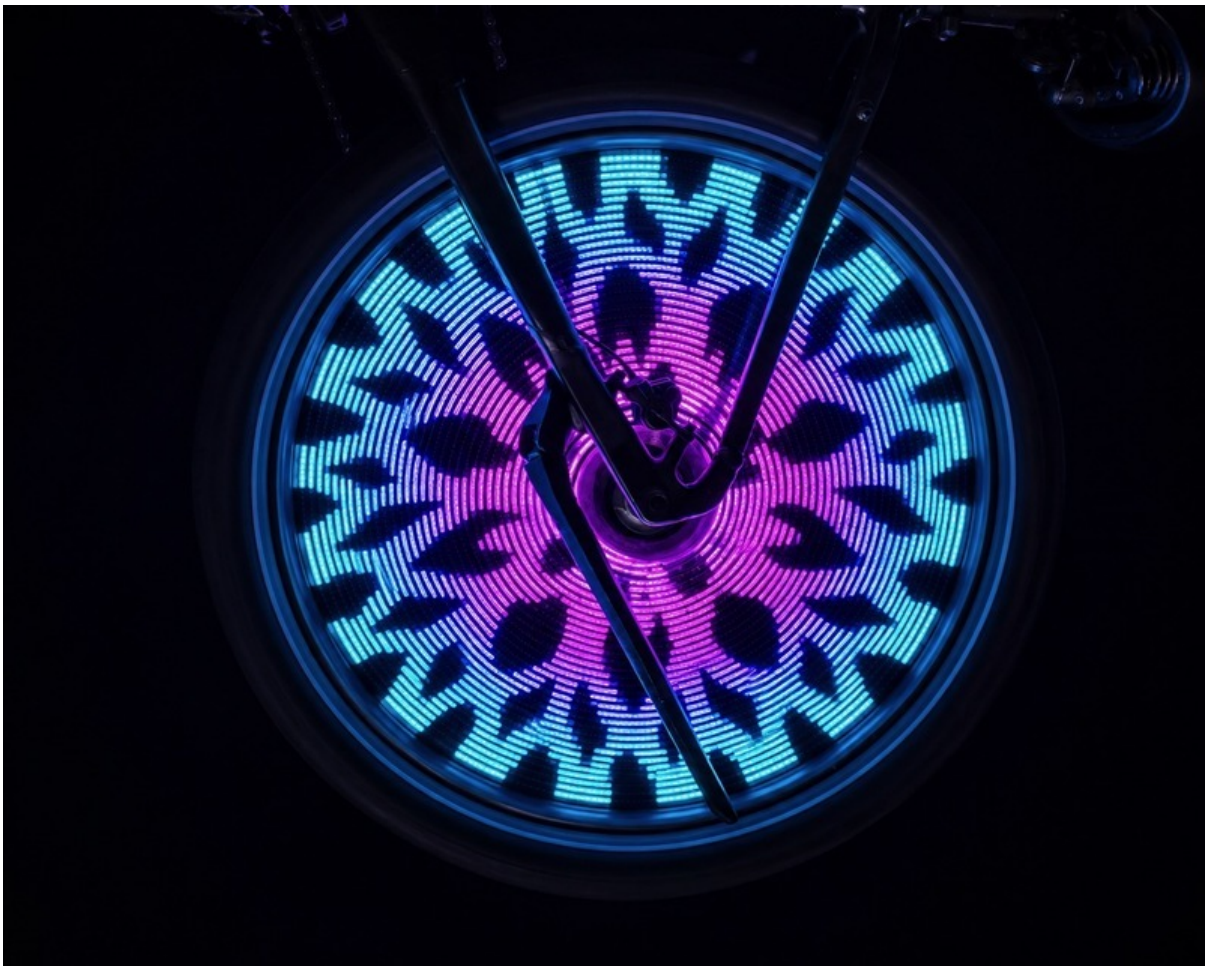
CircuitPython POV Bike Wheel

Created by Erin St Blaine



https://learn.adafruit.com/circuitpython-pov-bike-wheel

Last updated on 2026-09-29 08:12:06 PM UTC

Table of Contents

Overview 3

- Parts

Wiring Diagram 6

Install CircuitPython 6

- CircuitPython Quickstart
- Safe Mode
- Flash Resetting UF2

Test Code 10

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

Electronics Assembly 12

- Troubleshooting
- Seal Your Connections. And Then Seal Them Some More.

Bike Assembly 17

Code the POV Wheel 20

- Configuration
- Image Size and Repeats
- Wheel Speed
- Image Position
- Hall Sensor Settings

Create Images 38

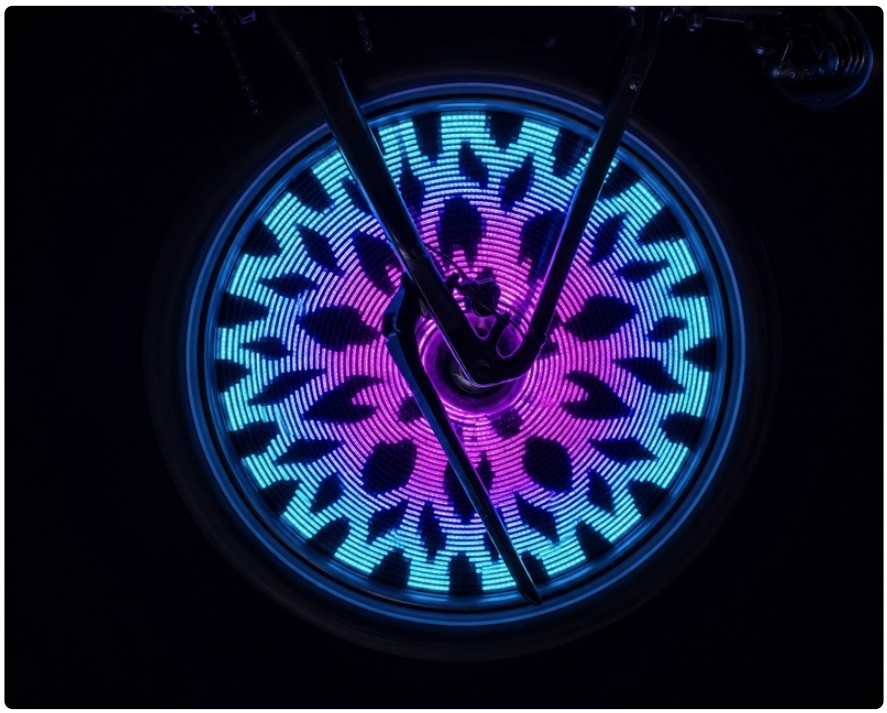
Overview

Light up your bike for festival season or nighttime Critical Mass rides. Stand out from the crowd with a custom persistence-of-vision bike wheel display, and upload dozens of your own images to create the ride you've always dreamed of.

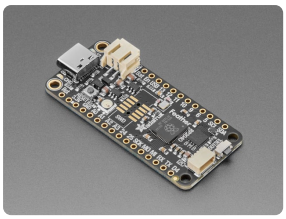
Persistence of vision is the effect that happens when your eyes briefly retain an image after it disappears. By flashing LEDs in carefully timed patterns as the bike wheel spins, those individual points of light appear to blend together into a complete floating image.

This guide shows you how to build a three-spoke, double-sided bike wheel display. There's some tricky soldering required to get everything wired neatly and securely in place, so this is not a beginner build.

Sample CircuitPython code and images are included, and it's easy to swap in your own artwork or adapt the layout for your particular wheel and LED setup.



Parts

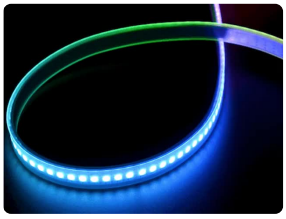


Featured

Adafruit Feather RP2040

A new chip means a new Feather, and the Raspberry Pi RP2040 is no exception. When we saw this chip we thought "this chip is going to be awesome when we give it the Feather...

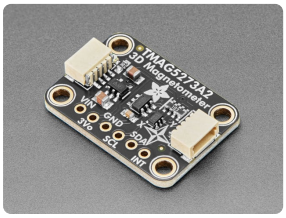
https://www.adafruit.com/product/4884



Adafruit DotStar Digital LED Strip - White 144 LED/m - One Meter (×2)

Move over NeoPixels, there's a new LED strip in town! These fancy new DotStar LED strips are a great upgrade for people who have loved and used NeoPixel strips for a few years but...

https://www.adafruit.com/product/2242



Adafruit TMAG5273 (A2) 3D Hall Effect Magnetometer

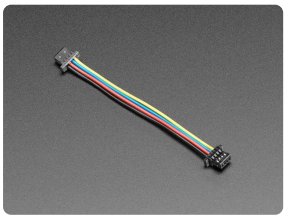
The Adafruit TMAG5273 (A2) 3D Hall Effect Magnetometer Breakout is a low-power linear 3D Hall-effect sensor designed for a wide range of industrial and personal...

https://www.adafruit.com/product/6490

High-strength 'rare earth' magnet

Yow! These things are super powerful. .47" diameter and .18" thick discs. Great for use with your SpokePOV Kit. If you have an aluminium frame, use tape/foam sticky tape to...

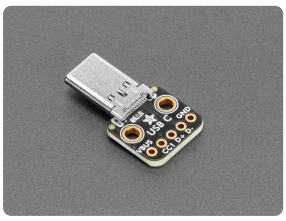
https://www.adafruit.com/product/9



STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long

This 4-wire cable is 50mm / 1.9" long and fitted with JST SH female 4-pin connectors on both ends. Compared with the chunkier JST PH these are 1mm pitch instead of 2mm, but...

https://www.adafruit.com/product/4399



Adafruit USB Type C Plug Breakout

Throw out all those Mini and Micro B USB cables you have in a plastic bin - the next generation of USB connectors is here with USB C! You've seen these...

https://www.adafruit.com/product/5978

USB C to USB C Cable - USB 3.1 Gen 4 with E-Mark - 6" long

As technology changes and adapts, so does Adafruit! Rather than the regular USB A, this cable has USB C to USB C plugs!USB C is the latest...

https://www.adafruit.com/product/4198

This is my favorite wire for projects that will flex and move a lot. Get 4 different colors: I used black, red, green, and yellow.

Silicone Cover Stranded-Core Wire - 2m 26AWG Black (×4)

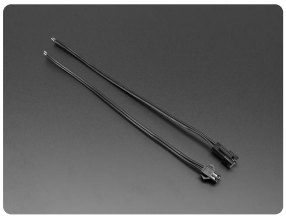
Silicone-sheathing wire is super-flexible and soft, and its also strong! Able to handle up to 200°C and up to 600V, it will do when PVC covered wire wimps out. We like this wire...

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

3-pin JST SM Plug + Receptacle Cable Set

These 3-wire cables are 163mm (6.4") long and come as a set. One cable has a JST SM type connector plug on the end. The other cable has a matching JST SM type receptacle...

https://www.adafruit.com/product/1663



2-pin JST SM Plug + Receptacle Cable Set

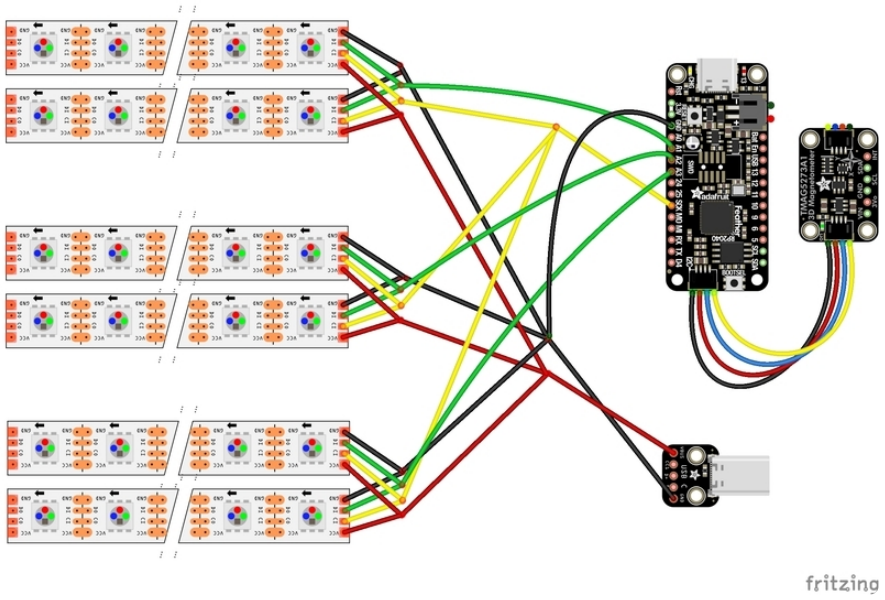
These 2-wire cables are 16cm long and come as a set. One side has a 2-pin JST SM type connector plug on the end. The other side has a matching 2-pin JST SM type...

https://www.adafruit.com/product/2880

You'll Also Need

- Clear self-fusing silicone tape ()
- Clear 1/2" heat shrink tubing ()
- Hot glue gun & sticks
- Heat gun
- Zip ties
- Soldering iron & accessories

Wiring Diagram



The wiring for this project is fairly complex.

For a double-sided wheel, use six LED strips arranged in pairs. The two strips in each pair are wired identically, so the left and right sides of your bike will show the same image. Treat these like one double-sided strip once they're connected.

Working from the **IN** end of all strips (arrows pointing away from your solder joints:)

- **VCC** (red wires) connect all together and to **VBUS** on the USB breakout
- **GND** (black wires) connect all together and to **GND** on the USB breakout, and also to **G** on the Feather
- **CI** (yellow wires) connect all together and to **SCK** on the Feather
- **DI** (green wires) connect in pairs, then the pairs are connected to **A1**, **A2**, and **A3** on the Feather

Use either Stemma port on the hall effect sensor to plug into the Feather's Stemma port.

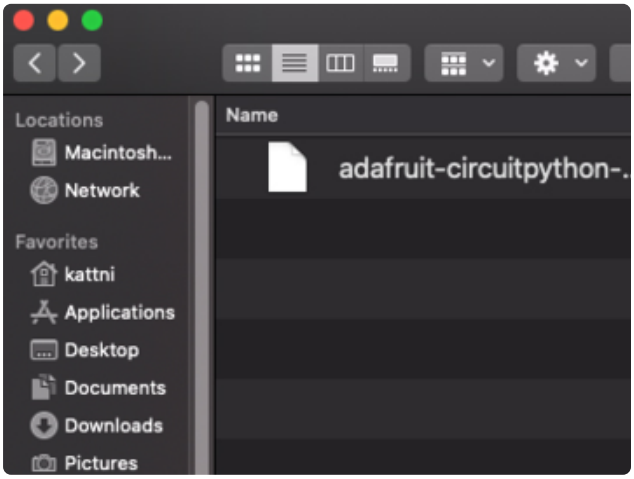
Install CircuitPython

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.

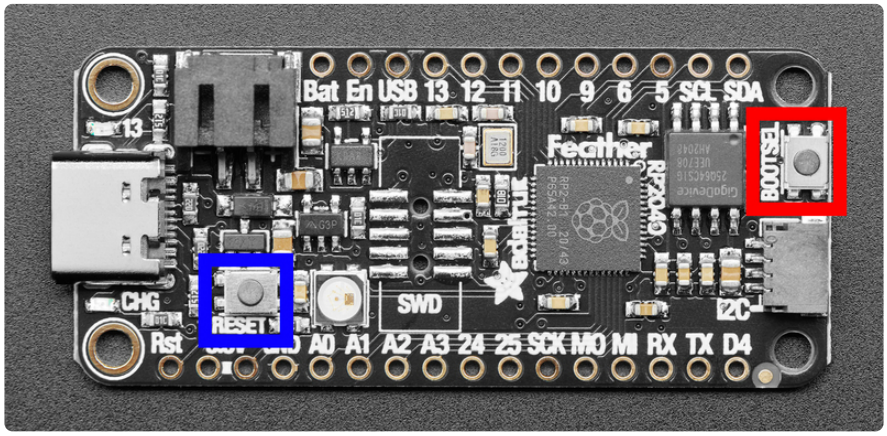
CircuitPython Quickstart

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



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

Save it wherever is convenient for you.

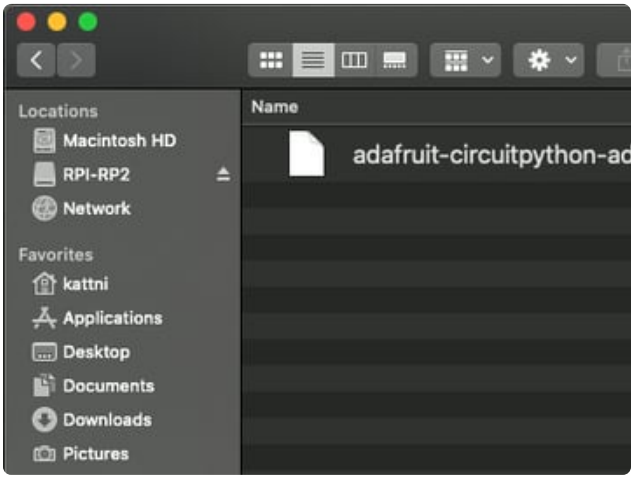


To enter the bootloader, hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset button** (highlighted in red or blue above). **Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!**

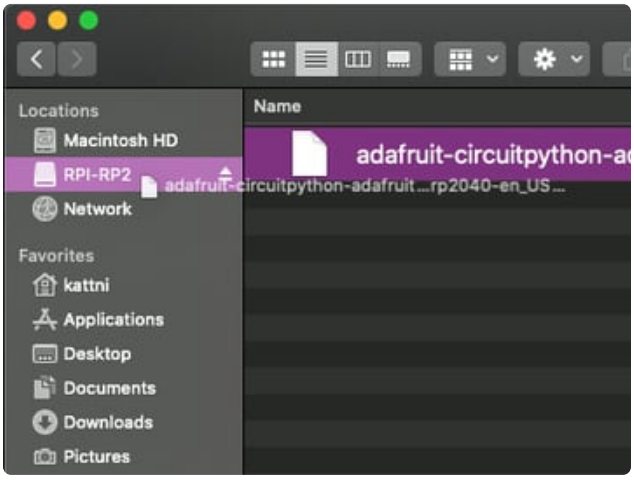
If the drive does not appear, release all the buttons, and then repeat the process above.

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.

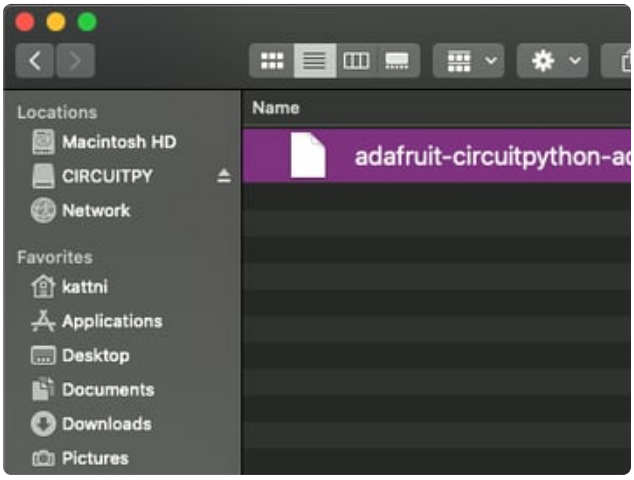
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



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.

Test Code

This code will light up the three pairs of LED strips in red, green, and blue. Use this code during the build to make sure all the strips are working and to help keep the strips together in the correct order during assembly.

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

"""Simple three-spoke DotStar color test using the adafruit_dotstar library."""

import time

import adafruit_dotstar
import board

NUM_PIXELS = 36

CLOCK_PIN = board.SCK
DATA_PINS = (board.A1, board.A2, board.A3)

COLORS = (
    (255, 0, 0),  # Red
    (0, 255, 0),  # Green
    (0, 0, 255),  # Blue
)
```

```
22
23
24  class SpokeDotStar(adafruit_dotstar.DotStar):
25      """DotStar strip that keeps its pixels lit when its pins are released.
26
27      All three spokes share one clock pin, so only one strip can own the pins
28      at a time. The stock ``deinit`` blanks the strip before releasing the
29      pins; this version releases them and leaves the last colors showing.
30      """
31
32      def deinit(self) -> None:
33          if self._spi:
34              self._spi.deinit()
35          else:
36              self.dpin.deinit()
37              self.cpin.deinit()
38
39
40  def write_dotstar(data_pin, color):
41      """Fill one DotStar strip with a solid color."""
42      with SpokeDotStar(CLOCK_PIN, data_pin, NUM_PIXELS) as strip:
43          strip.fill(color)
44
45
46  while True:
47      for strip_pin, strip_color in zip(DATA_PINS, COLORS):
48          write_dotstar(strip_pin, strip_color)
49
50      time.sleep(1)
```

https://github.com/adafruit/Adafruit_Learning_System_Guides/blob/main/CircuitPython_Bike_Wheel_POV/test/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 Prop-Maker Feather's **CIRCUITPY** drive.

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

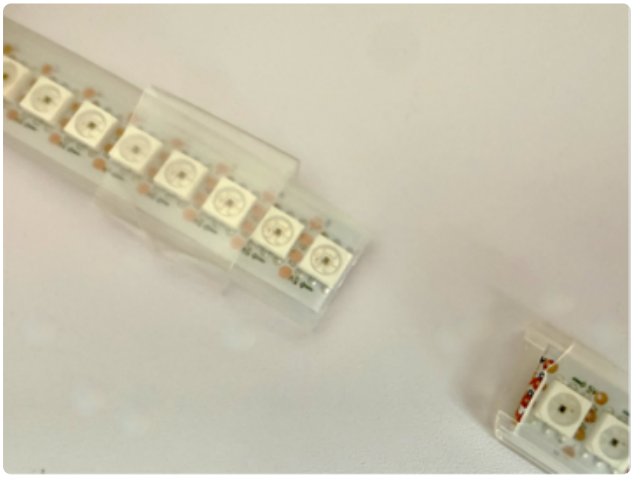
Your RP2040 Prop-Maker Feather **CIRCUITPY** drive should look like this after copying the **lib** folder, and the **code.py** file.

If your strips have a different number of pixels than 36, or if you soldered to different pins, change the code here.

```
NUM_PIXELS = 36

CLOCK_PIN = board.SCK
DATA_PINS = (board.A1, board.A2, board.A3)
```

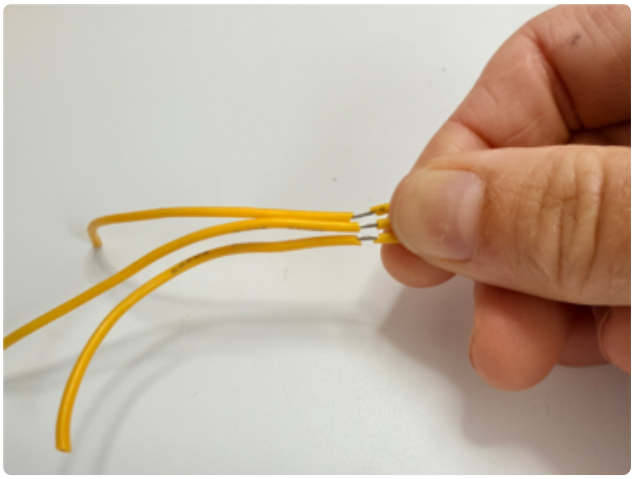
Electronics Assembly

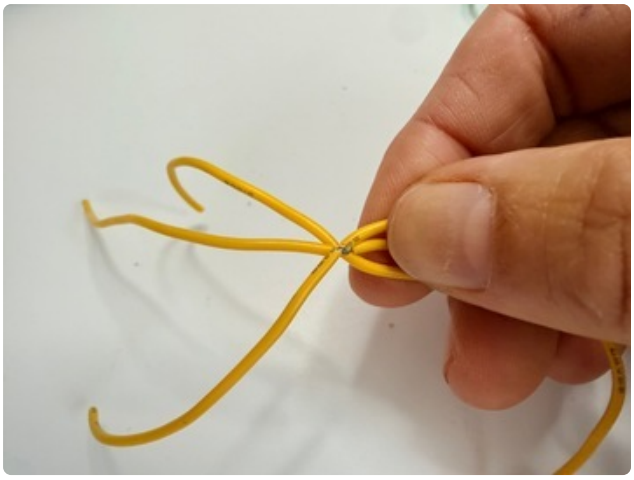


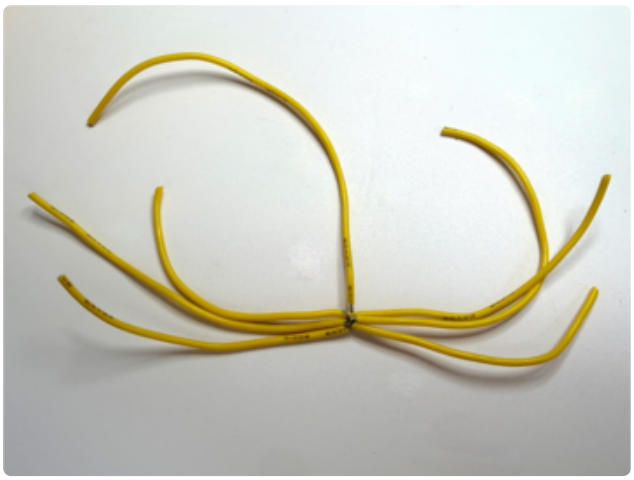
My bike has 27.5 inch wheels, and each spoke fits 36 pixels.

Figure out the length you want for your bike and cut 6 strips to that length. When cutting, leave the full width of the copper pads on the IN end of the cut-off strip. This will make soldering the strips a whole lot easier - these pads are tiny so do yourself a favor here and keep them intact.

Slip some clear heat shrink on to both cut ends of the strips. Don't shrink it down until you're done testing, but it's good to put it on now so you don't forget later.







The 6 clock pins (CI) on the LED strips will all connect to the same pin on the Feather: SCK.

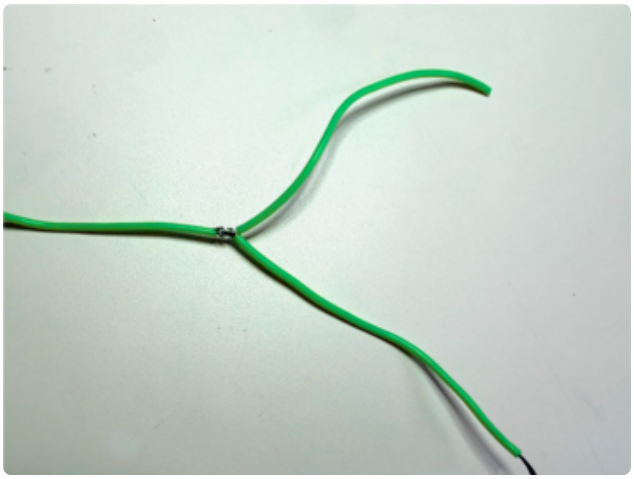
Cut three yellow wires to about 5-6" long. Find the center, and carefully cut through the shielding on the silicone wires with your fingernail or a pair of wire strippers to expose a little bit of the wire in the middle.

Twist the three wires together so the exposed wires all snuggle together. You want a good solid connection of the metal here. Now you have six wires all neatly joined together. These six wires will connect to the CI pins on your six strips.

Cut one more 3-4" piece of yellow wire -- this one will connect all these wires to the Feather. Wrap it around the metal connection point so you have a seven wire bundle. Solder the connection thoroughly so all the wires have good connectivity. Cover the joint with heat shrink.

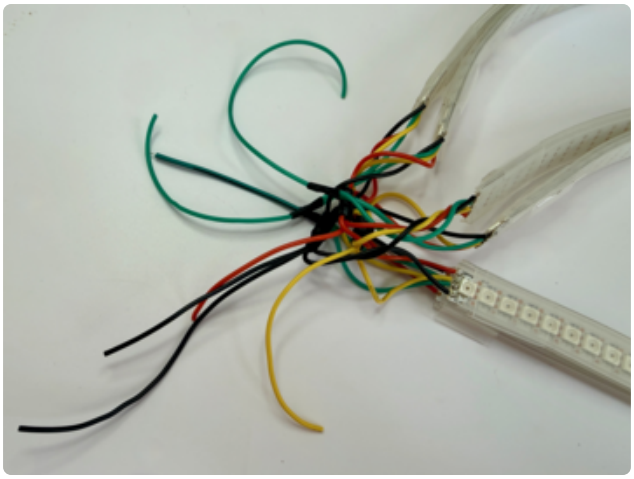
Repeat this process with the red wires - you want a 7-wire bundle there too. The 7th wire will connect to VCC on the USB breakout.

For the black wires, use the same technique but end up with 8 wires (so, 4 twisted pairs). The ground wires need to go to both the Feather G and the USB breakout G so we need one more in the black bundle.



For the green (data) wires: each LED pair will connect to a different data pin, so we need to keep them separate and not bundle them all up.

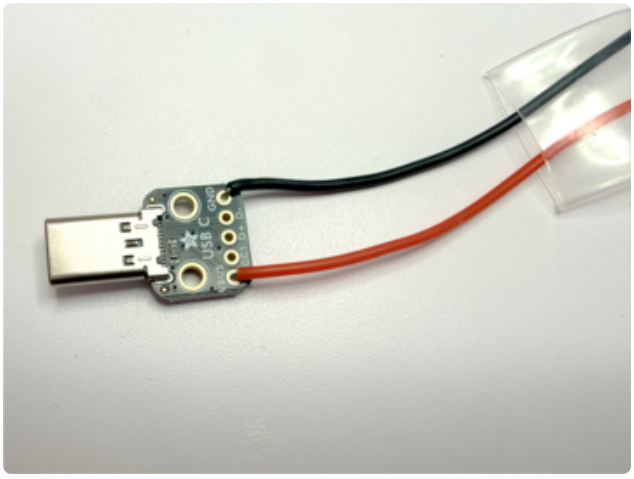
Make three 3-wire splice junctions. One wire from each set will go to the Feather (pins **A1**, **A2**, and **A3**) and the others will go to the **DI** pins on the LED strips.



Solder the wires onto the LED strips, making sure you're soldering to the IN end:

Red > VCC
Yellow > CI
Green > DI
Black > G

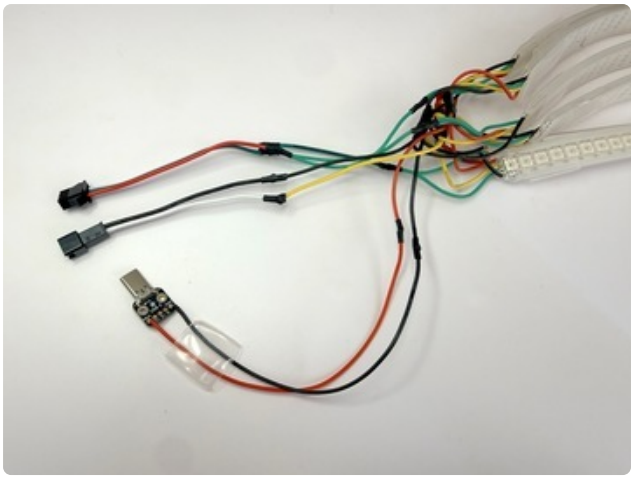
You should be left with three green wires, two black wires, one red wire and one yellow wire left unconnected.



Solder the red wire and one of the black wires to the USB breakout:

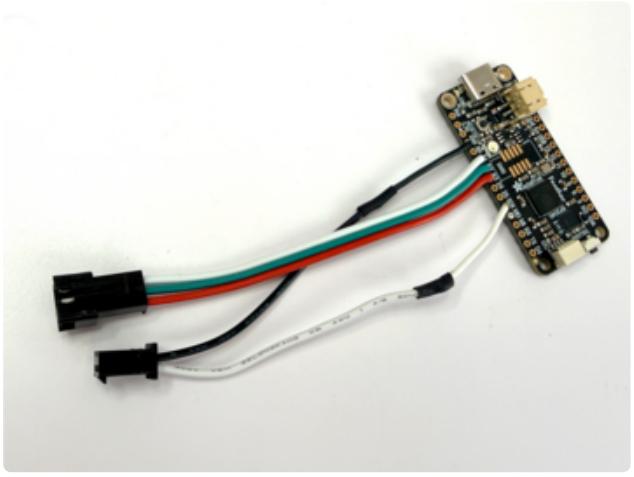
Red > VBUS
Black > G

Slide on some heat shrink, but again, don't seal anything yet.



Solder JST connectors onto the remaining wires: Black and yellow to the 2-pin connector, and the 3 green wires to the 3-pin connector.

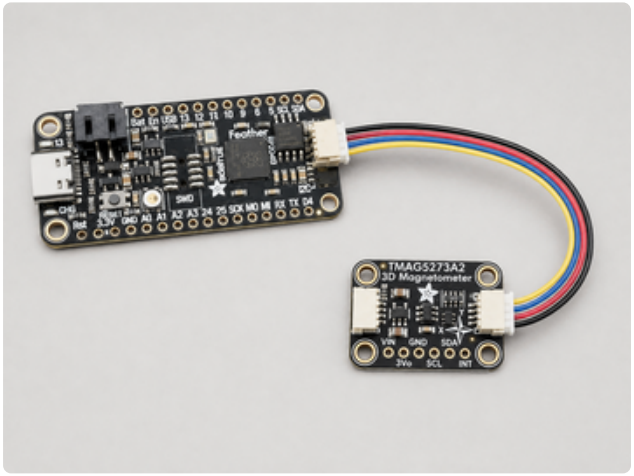
I also ended up extending the wires leading to the USB breakout a bit just to give myself more wiggle room during installation.



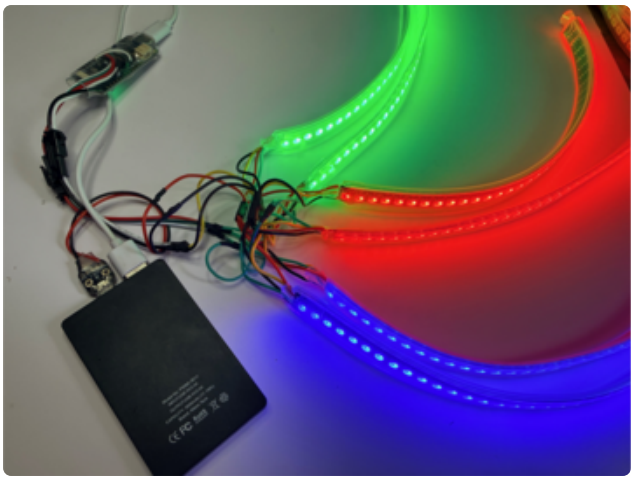
Connect the other sides of the JST connectors to the Feather:

3-pin connector to A1, A2, and A3
2-pin connector to G and SCK

The order isn't too important for the 3 pin connector. We'll sort them out during installation. But make sure you get the G wire and SCK wire consistent with the LED wiring for the 2-pin connector.



The hall effect sensor plugs into the Feather via the Stemma port.



Be sure you've uploaded the test code from the software page.

Plug the 3-pin and 2-pin connectors from the Feather into the LED strips. Plug the USB breakout into the battery. Also plug the Feather's USB port into the battery using a short USB cable.

If all goes well, your lights will turn on in red, green, and blue. Sort them out so the pairs are together.

Troubleshooting

This is a devilishly complicated wiring job. If all your lights come on in the right colors on the first try, I'd love to shake your hand! Mine sure didn't. If yours didn't either, here are some things to try:

- Look closely at the solder connections on the LED strips. These pads are SO tiny and SO close together that it's easy to bridge the solder, or have a stray wire from one pad touching another pad. With the power turned OFF, run a utility knife gently between the pads of the offending strip to see if there are any tiny stray wires.
- Make sure you soldered to the IN end of all the strips. If one is soldered on the OUT end (arrows pointing toward the solder joints), flip the strip around.
- If you had a hard time soldering, and the solder pads are burned or unsalvageable and you want to start again, cut off one LED from the front of the strip. If one or two of these strips have 35 instead of 36 in the length, the bike will still look great.
- Check all your connections again. Follow each wire through the connectors to make sure it goes where you think it goes. Some connectors have different color-coding order from other connectors and it's easy to accidentally plug the red wire into the black wire port on a connector from a different lot.
- Try re-uploading the test code. Open your serial monitor in Mu (or your favorite code editor) and see if the code is running correctly.
- Make sure both the USB breakout and the Feather are plugged into power, and that your battery is capable of running two ports at once. Try plugging them into a USB hub or wall socket to determine whether the battery is the problem.

Seal Your Connections. And Then Seal Them Some More.

I brought my bike to Burning Man on a very dusty and rainy year, and my project did not completely survive. Two of the strips started flickering on and off intermittently, and one of the silicone sleeves got torn somehow, either through riding or during transport - probably because I was too aggressive when tightening the zip tie.

When I pulled the JST connectors apart, there was playa dust inside both of them making the electrical connections much weaker. I also didn't seal the USB ports on either the battery or the controller. The bike did work for the first couple days of the festival but by the end I was experiencing problems.

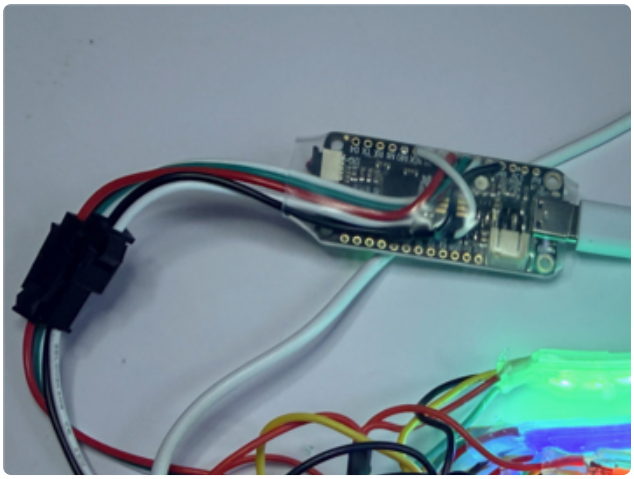
This build will survive normal city use and forest or beach festivals, but for windy, dusty, muddy desert festivals I recommend adding a LOT more sealing.

Sealing Ideas

- Use a second layer of heat shrink over all wire connections. Squirt some hot glue inside the heat shrink before shrinking it down, or use heat shrink with adhesive already inside ().
- Use a lot of self-sealing silicone tape. Bring an extra roll and really pile it on over the USB battery port connections. You'll need to cut it off to charge the battery, but your electronics will work a lot longer. All the connections I covered with this stuff DID survive!
- Wrap the JST connectors in self-sealing silicone tape as well to keep the dust out
- Be sure to seal up both ends of the LED strips. Two of my strips had exposed wire on the OUT end that I hadn't noticed during the build, and those were definitely giving me trouble after the burn.
- Be careful when attaching the strips to the bike! I got a zip tie too tight and it rubbed its way right through the silicone sleeve.

Bike Assembly

Decide if you want the POV display on the front or back wheel. If you're using the back wheel, make sure there's plenty of room for the gear selector to move without hitting the strips or electronic components.



Once you're sure everything's working, seal up all the electronics to keep water and dust out.

Use hot glue and 3/4" clear heat shrink to seal both ends of all your LED strips.

Use 1" heat shrink and/or self-sealing silicone tape to enclose the Feather and hall effect sensor together into a little plastic-coated bundle, with the two JST connectors sticking out the end. Fill that end of the heat shrink bundle with a little more hot glue. Seal it up as much as you can while still keeping the USB port accessible.

Select three spokes that are placed at equidistant angles around the wheel. Use zip ties and clear silicone self-sealing tape to secure the three pairs of strips back-to-back along the spokes, in the order shown and matching colors in the back-to-back placement.

Manage the wires around the hub of the wheel with more zip ties. If these wires catch on the gear mechanism or get out of line in any way, they'll very likely break. Spend a little time doing some wire management for safety.

Use some strong zip ties to secure the controller to the battery, and the battery to two spokes. Tighten the zip tie just below a spot where the spokes cross, so the centrifugal force pulling the battery out toward the rim of the wheel as you ride pulls the zip tie against the widening spokes and keeps the battery firmly in place.

Make sure you can reach the USB ports and the on/off switch on the battery easily.

If you're taking this to a dusty, muddy desert, add self-sealing silicone tape around the USB port connection to seal it up once you're done testing.

Use 5 minute epoxy or another strong adhesive to glue the rare earth magnet to the inside of the frame, at the location where the magnet sensor will pass directly by it as the wheel turns.

Plug in the two JST connectors connecting the strips to the Feather. Plug the USB-C breakout into the USB-C port on the battery, and connect the Feather's USB port to another of the battery ports using a short USB cable.

Use the self-sealing silicone tape or more zip ties wherever needed to hold everything in place.

Code the POV Wheel

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

Click the **Download Project Bundle** button in the code window below. This will download a zipped folder containing the CircuitPython code, libraries, and sample images for the project.

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

"""Three-spoke CircuitPython bike wheel persistence-of-vision display."""

import gc
import os
import time
from array import array

import adafruit_imageload
import adafruit_pioasm
import adafruit_tmag5273
import board
import rp2pio


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

```
# DISPLAY SETTINGS
# ---------------------------------------------------------------------------

NUM_LEDS = 36
IMAGE_FOLDER = "/img"
IMAGE_SECONDS = 8.0

# Approximate angular resolution of the wheel display.
#
# Examples:
#   36 px wide  -> 4 repeats per revolution
#   72 px wide  -> 2 repeats per revolution
#   144 px wide -> 1 repeat per revolution
SCANLINES_PER_REVOLUTION = 144

# Keep playback fast enough for clean POV at lower wheel speeds.
MIN_SCANLINES_PER_SECOND = 400.0

# 1.0 corrects the measured phase error over roughly one revolution.
STABILIZATION_STRENGTH = 1.0

# Rotate the entire pattern relative to the Hall sensor magnet.
PATTERN_ROTATION_DEGREES = 30.0

# DotStar global brightness: 0-31.
DOTSTAR_BRIGHTNESS = 10

PIO_FREQUENCY = 8_000_000


# ---------------------------------------------------------------------------
# HALL SENSOR SETTINGS
# ---------------------------------------------------------------------------

MAG_TRIGGER = 800.0
MAG_RELEASE = 450.0

MAG_TRIGGER_SQ = MAG_TRIGGER * MAG_TRIGGER
MAG_RELEASE_SQ = MAG_RELEASE * MAG_RELEASE

MIN_REVOLUTION_MS = 120

HALL_CLEAR_MS = 50
HALL_CLEAR_NS = HALL_CLEAR_MS * 1_000_000

START_CONFIRM_SECONDS = 1.5
START_CONFIRM_NS = int(START_CONFIRM_SECONDS * 1_000_000_000)

SENSOR_POLL_MS = 4
SENSOR_POLL_NS = SENSOR_POLL_MS * 1_000_000

STOP_TIMEOUT_SECONDS = 2.0
STOP_REVOLUTION_MULTIPLIER = 2.5

STARTING_RPM = 110.0
STARTING_REV_NS = int(60_000_000_000 / STARTING_RPM)
```

```
76
77  REV_SMOOTHING = 1.0
78
79  PRINT_RPM = False
80
81
82  # ------------------------------------------------------------------------
83  # PIO
84  #
85  # A1, A2 and A3 drive the three physical spokes.
86  # Each data line feeds the matching strip on both sides of the wheel.
87  # All six DotStar strips share SCK.
88  # ------------------------------------------------------------------------
89
90  PIO_PROGRAM = """
91  .program dotstar_three_spokes
92  .side_set 1
93  .wrap_target
94      out pins, 3 side 0
95      nop         side 1
96  .wrap
97  """
98
99  assembled = adafruit_pioasm.assemble(PIO_PROGRAM)
100
101 state_machine = rp2pio.StateMachine(
102     assembled,
103     frequency=PIO_FREQUENCY,
104     first_out_pin=board.A1,
105     out_pin_count=3,
106     first_sideset_pin=board.SCK,
107     sideset_pin_count=1,
108     auto_pull=True,
109     pull_threshold=24,
110     out_shift_right=True,
111     initial_out_pin_state=0,
112     initial_out_pin_direction=0b111,
113     initial_sideset_pin_state=0,
114     initial_sideset_pin_direction=1,
115 )
116
117
118 # ------------------------------------------------------------------------
119 # TMAG5273 HALL SENSOR
120 # ------------------------------------------------------------------------
121
122 i2c = board.STEMMA_I2C()
123 tmag = adafruit_tmag5273.TMAG5273(i2c)
124
125 tmag.magnetic_channels = adafruit_tmag5273.MAG_CH_XYZ
126 tmag.conversion_average = adafruit_tmag5273.CONV_AVG_1X
127 tmag.operating_mode = adafruit_tmag5273.MODE_CONTINUOUS
128
129
130 # ------------------------------------------------------------------------
131 # DOTSTAR DATA PACKING
```

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


def make_pin_table(pin_number):
    """Build packed PIO values for one DotStar data pin."""
    table = array("I", [0] * 256)

    for value in range(256):
        packed = 0

        for position in range(8):
            source_bit = 7 - position

            if value & (1 << source_bit):
                packed |= 1 << ((position * 3) + pin_number)

        table[value] = packed

    return table


PIN_A_BYTES = make_pin_table(0)
PIN_B_BYTES = make_pin_table(1)
PIN_C_BYTES = make_pin_table(2)


def pack_three_bytes(value_a, value_b, value_c):
    """Pack one byte for each of the three DotStar outputs."""
    return (
        PIN_A_BYTES[value_a]
        | PIN_B_BYTES[value_b]
        | PIN_C_BYTES[value_c]
    )


BRIGHTNESS_BYTE = 0xE0 | DOTSTAR_BRIGHTNESS

PACKED_BRIGHTNESS = pack_three_bytes(
    BRIGHTNESS_BYTE,
    BRIGHTNESS_BYTE,
    BRIGHTNESS_BYTE,
)

PACKED_END = pack_three_bytes(0xFF, 0xFF, 0xFF)

FRAME_WORDS = 4 + (NUM_LEDS * 4) + 4


# ------------------------------------------------------------------------
# IMAGE HELPERS
# ------------------------------------------------------------------------


def find_images():
    """Return all GIF and BMP files found in the image folder."""
    try:
```

```
        filenames = os.listdir(IMAGE_FOLDER)
    except OSError:
        return []

    images = []

    for filename in filenames:
        if filename.startswith("."):
            continue

        lower_name = filename.lower()

        if lower_name.endswith((".gif", ".bmp")):
            images.append(filename)

    images.sort()

    return images


def pixel_color(bitmap, shader, x_position, y_position):
    """Return one source pixel as 0xRRGGBB."""
    pixel_value = bitmap[x_position, y_position]

    try:
        return int(shader[pixel_value]) & 0xFFFFFF
    except TypeError:
        pass

    try:
        return int(shader.convert(pixel_value)) & 0xFFFFFF
    except (AttributeError, TypeError):
        pass

    return int(pixel_value) & 0xFFFFFF


def split_color(color):
    """Return red, green and blue values from a packed RGB color."""
    return (
        (color >> 16) & 0xFF,
        (color >> 8) & 0xFF,
        color & 0xFF,
    )


def pack_pixel_colors(color_a, color_b, color_c):
    """Pack three RGB colors into DotStar blue, green and red words."""
    rgb_a = split_color(color_a)
    rgb_b = split_color(color_b)
    rgb_c = split_color(color_c)

    return (
        pack_three_bytes(
            rgb_a[2],
            rgb_b[2],
```

```
            rgb_c[2],
        ),
        pack_three_bytes(
            rgb_a[1],
            rgb_b[1],
            rgb_c[1],
        ),
        pack_three_bytes(
            rgb_a[0],
            rgb_b[0],
            rgb_c[0],
        ),
    )


def calculate_repeats(width):
    """Return the number of image repeats for one wheel revolution."""
    repeats = int((SCANLINES_PER_REVOLUTION / width) + 0.5)

    return max(1, repeats)


def calculate_spoke_offsets(width, repeats):
    """Return source-image offsets for the three 120-degree spokes."""
    spoke_offset = int(round((repeats * width) / 3))

    return (
        0,
        spoke_offset % width,
        (spoke_offset * 2) % width,
    )


# ---------------------------------------------------------------------------
# IMAGE PREPACKING
# ---------------------------------------------------------------------------


def make_scanline_frame(bitmap, shader, x_position, spoke_offsets):
    """Build one complete DotStar frame for all three spokes."""
    frame = array("I", [0] * FRAME_WORDS)

    x_positions = (
        x_position,
        (x_position + spoke_offsets[1]) % bitmap.width,
        (x_position + spoke_offsets[2]) % bitmap.width,
    )

    index = 4

    for led_number in range(NUM_LEDS):
        source_y = (NUM_LEDS - 1) - led_number

        color_a = pixel_color(
            bitmap,
            shader,
```

```
            x_positions[0],
            source_y,
        )

        color_b = pixel_color(
            bitmap,
            shader,
            x_positions[1],
            source_y,
        )

        color_c = pixel_color(
            bitmap,
            shader,
            x_positions[2],
            source_y,
        )

        blue_word, green_word, red_word = pack_pixel_colors(
            color_a,
            color_b,
            color_c,
        )

        frame[index] = PACKED_BRIGHTNESS
        frame[index + 1] = blue_word
        frame[index + 2] = green_word
        frame[index + 3] = red_word

        index += 4

    frame[index] = PACKED_END
    frame[index + 1] = PACKED_END
    frame[index + 2] = PACKED_END
    frame[index + 3] = PACKED_END

    return frame


def load_image(filename):
    """Load one image and prepack it for fast POV playback."""
    path = IMAGE_FOLDER + "/" + filename

    gc.collect()

    bitmap, shader = adafruit_imageload.load(path)

    width = bitmap.width
    height = bitmap.height

    if height != NUM_LEDS:
        del bitmap
        del shader

        gc.collect()

```

```
356             raise ValueError(
357                 f"Image must be exactly {NUM_LEDS} pixels high."
358             )
359
360         repeats = calculate_repeats(width)
361         spoke_offsets = calculate_spoke_offsets(width, repeats)
362
363         frames = []
364
365         try:
366             for x_position in range(width):
367                 frames.append(
368                     make_scanline_frame(
369                         bitmap,
370                         shader,
371                         x_position,
372                         spoke_offsets,
373                     )
374                 )
375
376         except MemoryError as error:
377             del frames
378             del bitmap
379             del shader
380
381             gc.collect()
382
383             raise MemoryError(
384                 f"Not enough RAM to prepack {filename}."
385             ) from error
386
387         del bitmap
388         del shader
389
390         gc.collect()
391
392         return frames, width, repeats
393
394
395     # ---------------------------------------------------------------------------
396     # BLANK FRAME
397     # ---------------------------------------------------------------------------
398
399
400     def make_blank_frame():
401         """Create one all-black DotStar frame."""
402         frame = array("I", [0] * FRAME_WORDS)
403
404         index = 4
405
406         for _ in range(NUM_LEDS):
407             frame[index] = PACKED_BRIGHTNESS
408             frame[index + 1] = 0
409             frame[index + 2] = 0
410             frame[index + 3] = 0
411
```

```
412             index += 4
413
414         frame[index] = PACKED_END
415         frame[index + 1] = PACKED_END
416         frame[index + 2] = PACKED_END
417         frame[index + 3] = PACKED_END
418
419         return frame
420
421
422     BLANK_FRAME = make_blank_frame()
423
424
425     def blank_all():
426         """Turn off all LED strips."""
427         state_machine.write(BLANK_FRAME)
428
429
430     # --------------------------------------------------------------------------
431     # IMAGE PLAYLIST
432     # --------------------------------------------------------------------------
433
434     image_files = find_images()
435
436     if not image_files:
437         raise RuntimeError("No GIF or BMP images found in /img.")
438
439     image_number = -1
440     scanline_frames = None
441     image_width = 0
442     image_repeats = 1
443
444
445     def load_next_usable_image():
446         """Load the next image that can be successfully prepacked."""
447         # pylint: disable=global-statement
448         global image_number
449         global scanline_frames
450         global image_width
451         global image_repeats
452
453         if scanline_frames is not None:
454             old_frames = scanline_frames
455             scanline_frames = None
456
457             del old_frames
458
459             gc.collect()
460
461         attempts = 0
462
463         while attempts < len(image_files):
464             image_number = (image_number + 1) % len(image_files)
465
466             filename = image_files[image_number]
467
```

```
        try:
            (
                scanline_frames,
                image_width,
                image_repeats,
            ) = load_image(filename)

            print(
                "Now playing:",
                filename,
                "| width:",
                image_width,
                "| repeats:",
                image_repeats,
            )

            return

        except (
            MemoryError,
            ValueError,
            OSError,
            RuntimeError,
        ) as error:
            print()
            print("IMAGE SKIPPED")
            print("File:", filename)
            print("Error:", type(error).__name__)
            print("Reason:", error)
            print()

            attempts += 1

    raise RuntimeError("None of the images could be loaded.")


load_next_usable_image()


def send_scanline(line_number):
    """Send one prepacked scanline to the three physical spokes."""
    state_machine.write(scanline_frames[line_number])


# ---------------------------------------------------------------------
# WHEEL TIMING
# ---------------------------------------------------------------------

moving = False
magnet_active = False

last_sensor_ns = 0
last_magnet_ns = 0
last_revolution_ns = 0
magnet_clear_since_ns = 0
previous_valid_pass_ns = 0
```

```
524
525     valid_passes = 0
526
527     revolution_ns = STARTING_REV_NS
528
529     image_line = 0
530
531     scanline_period_ns = int(
532         revolution_ns / (image_width * image_repeats)
533     )
534
535     now = time.monotonic_ns()
536
537     next_scanline_ns = now
538     image_change_ns = now
539
540
541     def update_scanline_timing():
542         """Update POV scanline speed using the measured wheel revolution."""
543         # pylint: disable=global-statement
544         global scanline_period_ns
545
546         total_lines = image_width * image_repeats
547
548         target_line = int(
549             round(
550                 (PATTERN_ROTATION_DEGREES / 360.0)
551                 * total_lines
552             )
553         ) % image_width
554
555         phase_error = image_line - target_line
556
557         if phase_error > image_width / 2:
558             phase_error -= image_width
559
560         elif phase_error < -(image_width / 2):
561             phase_error += image_width
562
563         correction = phase_error * STABILIZATION_STRENGTH
564
565         corrected_lines = max(
566             total_lines - correction,
567             1,
568         )
569
570         hall_period_ns = int(
571             revolution_ns / corrected_lines
572         )
573
574         minimum_rate_period_ns = int(
575             1_000_000_000 / MIN_SCANLINES_PER_SECOND
576         )
577
578         scanline_period_ns = min(
579             hall_period_ns,
```

```
        minimum_rate_period_ns,
    )


def update_magnet_state(now_ns, magnitude_sq):
    """Update magnet re-arm state."""
    # pylint: disable=global-statement
    global magnet_active
    global magnet_clear_since_ns

    if magnitude_sq <= MAG_RELEASE_SQ:
        if magnet_clear_since_ns == 0:
            magnet_clear_since_ns = now_ns

        if (
            magnet_active
            and now_ns - magnet_clear_since_ns
            >= HALL_CLEAR_NS
        ):
            magnet_active = False

    else:
        magnet_clear_since_ns = 0


def handle_magnet_pass(now_ns):
    """Process one valid Hall sensor magnet pass."""
    # pylint: disable=global-statement
    global last_magnet_ns
    global last_revolution_ns
    global previous_valid_pass_ns
    global valid_passes
    global revolution_ns
    global moving
    global next_scanline_ns
    global image_change_ns

    if (
        last_magnet_ns
        and now_ns - last_magnet_ns
        < MIN_REVOLUTION_MS * 1_000_000
    ):
        return

    last_magnet_ns = now_ns

    if last_revolution_ns:
        measured_ns = now_ns - last_revolution_ns

        revolution_ns = int(
            revolution_ns * (1.0 - REV_SMOOTHING)
            + measured_ns * REV_SMOOTHING
        )

        update_scanline_timing()

```

```
636             if PRINT_RPM:
637                 rpm = 60_000_000_000.0 / revolution_ns
638                 print(f"RPM {rpm:.1f}")
639
640         last_revolution_ns = now_ns
641
642         if moving:
643             return
644
645         if (
646             valid_passes == 0
647             or now_ns - previous_valid_pass_ns
648             > START_CONFIRM_NS
649         ):
650             valid_passes = 1
651
652         else:
653             valid_passes += 1
654
655         previous_valid_pass_ns = now_ns
656
657         if valid_passes >= 2:
658             moving = True
659             image_change_ns = now_ns
660             next_scanline_ns = now_ns
661
662             print("Wheel moving")
663
664
665     def check_hall(now_ns):
666         """Poll the Hall sensor and update wheel speed."""
667         # pylint: disable=global-statement
668         global last_sensor_ns
669         global magnet_active
670
671         if now_ns - last_sensor_ns < SENSOR_POLL_NS:
672             return
673
674         last_sensor_ns = now_ns
675
676         x_axis, y_axis, z_axis = tmag.magnetic
677
678         magnitude_sq = (
679             (x_axis * x_axis)
680             + (y_axis * y_axis)
681             + (z_axis * z_axis)
682         )
683
684         update_magnet_state(
685             now_ns,
686             magnitude_sq,
687         )
688
689         if (
690             not magnet_active
691             and magnitude_sq >= MAG_TRIGGER_SQ
```

```
    ):
        magnet_active = True

        handle_magnet_pass(now_ns)


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

blank_all()

print()
print("POV bike wheel ready.")
print("Images:", len(image_files))
print("Scanlines/revolution:", SCANLINES_PER_REVOLUTION)

while True:
    now = time.monotonic_ns()

    check_hall(now)

    # Stop after several missed Hall passes.
    if moving and last_magnet_ns:
        stop_timeout_ns = max(
            int(STOP_TIMEOUT_SECONDS * 1_000_000_000),
            int(
                revolution_ns
                * STOP_REVOLUTION_MULTIPLIER
            ),
        )

        if now - last_magnet_ns >= stop_timeout_ns:
            moving = False
            valid_passes = 0
            previous_valid_pass_ns = 0
            image_line = 0

            blank_all()

            print("Wheel stopped")

    if not moving:
        continue

    # Advance to the next image.
    if (
        len(image_files) > 1
        and now - image_change_ns
        >= int(IMAGE_SECONDS * 1_000_000_000)
    ):
        load_next_usable_image()

        image_line = 0

        update_scanline_timing()
```

```
748
749             now = time.monotonic_ns()
750             image_change_ns = now
751             next_scanline_ns = now
752
753             continue
754
755         # Draw the next angular slice.
756         if now >= next_scanline_ns:
757             send_scanline(image_line)
758
759             image_line += 1
760
761             if image_line >= image_width:
762                 image_line = 0
763
764             next_scanline_ns += scanline_period_ns
765
766             # Skip stale frames instead of blasting through them after a delay.
767             if now - next_scanline_ns > scanline_period_ns * 4:
768                 next_scanline_ns = now + scanline_period_ns
```

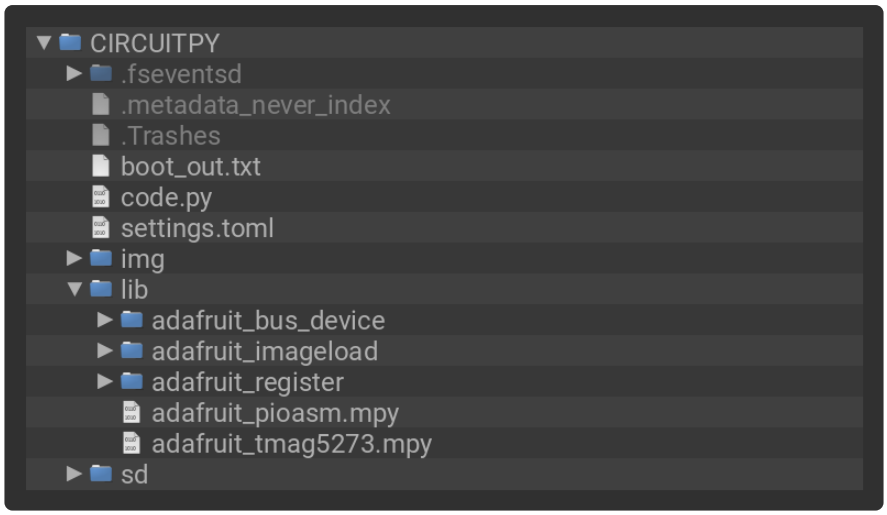
https://github.com/adafruit/Adafruit_Learning_System_Guides/blob/main/CircuitPython_Bike_Wheel_POV/final/code.py

After downloading the bundle, plug the Feather RP2040 into your computer with a known good USB data+power cable. A drive named **CIRCUITPY** should appear in Finder or File Explorer.

Copy the following items to the CIRCUITPY drive:

- **/lib** folder
- **code.py**
- create a folder called **/img** and put all images inside

The **/img** folder contains the artwork displayed by the wheel. Add your own images to this folder to customize the artwork that shows.



The code watches the Hall effect sensor to determine how quickly the wheel is turning, then adjusts the LED timing to keep the persistence-of-vision images looking consistent as your speed changes. The three spokes work together to draw different sections of the same image around the wheel.

Configuration

The main settings are gathered near the top of **code.py** so you can customize the display without digging through the rest of the program.

```
NUM_LEDS = 36
IMAGE_FOLDER = "/img"
IMAGE_SECONDS = 8.0

SCANLINES_PER_REVOLUTION = 144
MIN_SCANLINES_PER_SECOND = 400.0

STABILIZATION_STRENGTH = 1.0
PATTERN_ROTATION_DEGREES = 30.0

DOTSTAR_BRIGHTNESS = 10
```

`NUM_LEDS` should match the number of DotStar LEDs on each spoke. This build uses 36 LEDs per strip.

`IMAGE_SECONDS` controls how long each image is displayed before the next image in the **/img** folder is loaded.

`DOTSTAR_BRIGHTNESS` controls the overall LED brightness. DotStars use a global brightness value from `0` to `31`. Lower values save power and can actually make POV images easier to photograph at night.

Image Size and Repeats

All images must be exactly **36 pixels high** to match the 36 LEDs running from the hub toward the rim.

The image width can vary.

```
SCANLINES_PER_REVOLUTION = 144
```

This setting determines approximately how much image data is drawn during one wheel revolution. Shorter images automatically repeat around the wheel.

```
36 pixels wide  = about 4 repeats
72 pixels wide  = about 2 repeats
144 pixels wide = about 1 repeat
```

Narrow geometric patterns tend to work especially well for POV. Wider images can display more complex artwork, but very long images may be harder to read clearly while riding.

Add your own `.bmp` or `.gif` files to the **/img** folder. The code will cycle through the files alphabetically.

If an image cannot be loaded — for example, if it is the wrong height or is too large to fit in memory — the code will skip it and continue to the next image.

Wheel Speed

The Hall effect sensor measures each wheel revolution and uses that timing to adjust the image playback speed.

```
MIN_SCANLINES_PER_SECOND = 400.0
```

This sets the minimum playback speed. At very slow wheel speeds, allowing the LEDs to update too slowly can make the image look incomplete or broken up. The default value of `400` keeps the animation moving fast enough to maintain a convincing POV effect.

For this build, I found the default value worked well and generally wouldn't recommend changing it unless you're experimenting with a different wheel size or LED layout.

Image Position

The Hall sensor also gives the code a reference point for the wheel position.

```
STABILIZATION_STRENGTH = 1.0
PATTERN_ROTATION_DEGREES = 30.0
```

`STABILIZATION_STRENGTH` controls how strongly the image timing corrects itself relative to the Hall sensor. A value of `1.0` works well for this build.

`PATTERN_ROTATION_DEGREES` rotates the displayed artwork around the wheel relative to the magnet. Change this value if your images consistently appear in an awkward part of the wheel.

Try values such as:

```
PATTERN_ROTATION_DEGREES = 0.0
PATTERN_ROTATION_DEGREES = 30.0
PATTERN_ROTATION_DEGREES = 90.0
```

This does not rotate the bitmap itself. It shifts where the repeating POV pattern appears around the wheel.

Hall Sensor Settings

The Hall effect sensor settings are also near the top of the file:

```
MAG_TRIGGER = 800.0
MAG_RELEASE = 450.0
```

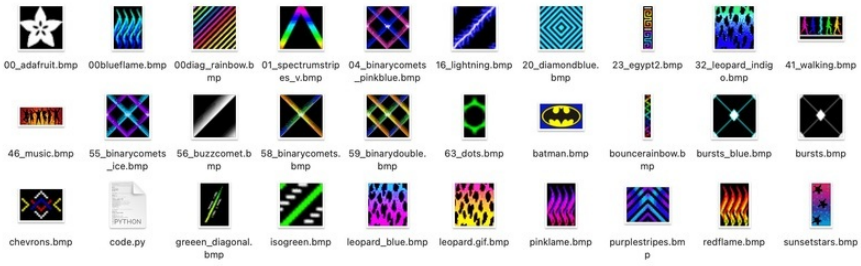
These values determine when the code recognizes that the wheel magnet has passed the sensor.

The defaults were tuned for the magnet and sensor placement used in this build. You probably won't need to change them unless you're using a different magnet or mounting the sensor farther away.

The LEDs remain off while the wheel is stationary. After the code detects two valid magnet passes, the POV display starts automatically. When the wheel stops, the LEDs turn off again.

I think that’s about the right level for this guide: enough for someone to confidently change the artwork, timing, brightness, and image position, without forcing them through the PIO packing or Hall-state logic.

Create Images



The code bundle includes around 30 starter images. If you want to add your own artwork, here are the specs:

- .gif or .bmp format - be sure .gif images are still, not animated, and .bmp images are saved as 8-bit files
- Images should have a height equal to the number of pixels in your strip, so mine are 36 pixels high
- Image width can vary, but images look best if the width is fairly narrow. Images that are 36 or 72 pixels wide will lay out beautifully on the wheel. Images that are narrower than 36 pixels also look great. For the most precise looking images, scale down in factors of 36 (18, 3, 6, etc).

36 pixels is not very many pixels! Your images will look tiny on your computer screen. There's an art to getting small images to look clear. Think of an icon - they're pixelated when you zoom in but clear from a distance. Icons or geometric shapes look great.

It's also effective to use the same shape but color-shifted into different hues. That gives a cohesive look to your bike.

If you want to play them in a certain order, use a naming convention that sorts them numerically or alphabetically. 00_adafruit.bmp plays first since it's listed first in the **/img** folder.