Darth Faders
Created by John Park

https://learn.adafruit.com/darth-faders

Last updated on 2023-08-29 04:41:16 PM EDT
## Table of Contents

**Overview** 3
- Parts

**Build the Darth Faders** 6
- Wiring Connections
- Fader Wiring
- Terminal Block FeatherWing
- Feather RP2040 Prep
- Motor FeatherWing Prep
- Circuit Assembly

**3D Printed Enclosure** 11

**Assembly** 13
- Base
- Encoder Prep
- Mount Encoder
- Power Plug
- Connect Power
- Enclose
- Connect Encoder
- Cap It
- Encoder Knob
- Outside Connections

**Install CircuitPython** 24
- CircuitPython Quickstart
- Safe Mode
- Flash Resetting UF2

**Code the Darth Faders** 28
- Text Editor
- Download the Project Bundle
- How It Works
- Libraries
- Motors
- Rotary Encoder
- Fader Analog Read
- Position Function
- Animation List
- Main Loop
Overview

The machine is alive! Build this sci-fi prop/desk sculpture with motorized slide potentiometers, Feather RP2040, Motor FeatherWing, and CircuitPython.

Use the rotary encoder breakout to start and stop the motion, and to dial in specific fader poses.

Parts
3 x Motorized Slide Potentiometer
10KΩ Linear with 5V DC Motor

DC Motor + Stepper FeatherWing Add-on For All Feather Boards
A Feather board without ambition is a Feather board without FeatherWings! This is the DC Motor + Stepper FeatherWing which will let you use 2 x bi-polar...
https://www.adafruit.com/product/2927

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

Assembled Terminal Block Breakout FeatherWing for all Feathers
The Terminal Block Breakout FeatherWing kit is like the Golden Eagle of prototyping FeatherWings (eg. majestic, powerful, good-looking). To start, you get a nice prototyping area...
https://www.adafruit.com/product/2926
Adafruit I2C Stemma QT Rotary Encoder Breakout with NeoPixel
Rotary encoders are soooo much fun! Twist em this way, then twist them that way. Unlike potentiometers, they go all the way around and often have little detents for tactile feedback....
https://www.adafruit.com/product/4991

STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long
This 4-wire cable is a little over 100mm / 4" 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...
https://www.adafruit.com/product/4210

Panel Mount 2.1mm DC barrel jack
This power jack is designed to easily attach to a panel up to 8mm thick (0.315" or 5/16") and fit 2.1mm power plugs snugly and securely. Perfect for adding a power connector...
https://www.adafruit.com/product/610

1 x Anodized Aluminum Machined Knob
20mm Diameter
https://www.adafruit.com/product/5527

1 x 9 VDC 1000mA regulated switching power adapter
UL listed
https://www.adafruit.com/product/63

1 x Bolt-On Kit
M3 screws
https://www.adafruit.com/product/3816
Build the Darth Faders

The basic circuit for Darth Faders is pretty simple -- you'll drive the three faders using a Motor FeatherWing plugged into a Feather RP2040 running CircuitPython. A Terminal Block FeatherWing will help make the many wired connections.

A rotary encoder with push button will be used to change modes and flip between animation poses.

Power to the Feather comes from USB-C and a 9VDC wall wart will power the Motor FeatherWing.

An optional set of 1MΩ resistors will allow you to use the fader touch encoders should you decide to add code for such functionality.
Wiring Connections

These are the connections you’ll wire:

- Fader +V to Feather 3V
- Fader GND to Feather GND
- Fader motor pins to Motor FeatherWing pins (polarity doesn’t matter, you’ll set direction in code)
- DC plug center pin to Motor FeatherWing power + terminal
- DC plug sleeve pin to Motor FeatherWing power - terminal
- Rotary encoder breakout I2C via STEMMA QT to Feather STEMMA QT
Optional:

• Fader touch pin (per fader) to Feather D12, D11, D10, with 1MΩ resistor to Feather GND per pin

Fader Wiring
Wire the connections for all three faders as shown. You can use any wiring you like, the use of silicone wiring with Dupont connectors is the deluxe option!
Terminal Block FeatherWing
You can wire directly from the faders to the terminal blocks for the power, ground, potentiometer, and optional cap touch wires. Again, short Dupont cables were used here to simplify the connections.
Feather RP2040 Prep
Solder in a set of stacking headers on the Feather. This way, it can plug into the Terminal Block FeatherWing and then the Motor FeatherWing can plug into the Feather.

You can also plug the STEMMA QT cable into the Feather at this time. This will be used to connect the rotary encoder breakout.

Motor FeatherWing Prep
Solder in the terminal blocks as shown in the Motor FeatherWing instructions here.

If you plan to use Dupont cables, you can screw them in as shown -- this will simplify the connections later during assembly.
Circuit Assembly
Plug the Feather into the Terminal Block FeatherWing.

Then, plug all of your fader wiring into the Motor FeatherWing and the Terminal Block FeatherWing as shown in the Fritzing diagram (and these photos).

Finally, plug the Motor FeatherWing onto the Feather.

3D Printed Enclosure
The Darth Faders enclosure is made of three models you'll need to 3D print. There is a base to hold the electronics, wall body to house the faders, power plug, and rotary encoder, and the cap.

They are straightforward prints, requiring no support.
The model files you'll need to build your Darth Faders are linked here:

Darth Faders 3D model files
Assembly

Base

Fit the FeatherWing/Feather/FeatherWing sandwich onto the pegs on the base as shown.

Loosely fit the three faders into their slots in the base as shown, then screw three M3 screws into the front of the base and into their respective threaded holes in the three faders.
Note: you can and should screw the base/fader screws in place before adding the walls, despite the fact that the walls are shown in this photo.
Encoder Prep

Screw on the spacers and nuts as shown. This will provide proper spacing to attach to the inside of the walls.
Mount Encoder
Attach the encoder from the inside of the printed enclosure as shown.
Power Plug

You'll provide 9VDC power to the Motor FeatherWing using a wall wart. Prep a DC jack with short wires as shown to connect to the terminals on the FeatherWing.

Push the plug through the enclosure and then fasten with the provided nut.
Connect Power
Mind the polarity and connect the DC power connector to the Motor FeatherWing's power terminals.
Enclose
Slide the walls over the wiring and into the base. If you hadn't already screwed in the faders, do so now.
Connect Encoder
Plug the STEMMA QT cable from the Feather into the rotary encoder.
Cap It
Place the cap over the enclosure, then screw in the remaining three screws to connect the faders to the cap.
Encoder Knob

Place the knob over the encoder shaft, making sure to leave enough space between the knob and the enclosure for the push encoder shaft to travel when pressed.

Tighten the grub screw with a hex wrench.
Outside Connections

Plug in USB-C to the Feather and DC power. You are ready to code and use the Darth Faders.

Note the power switch on the Terminal Block FeatherWing which is used to turn on and off the Feather via its Enable pin.
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.

[Download the latest version of CircuitPython for this board via circuitpython.org](circuitpython.org)

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 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 exit safe mode.
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 doesn't even show up as a disk drive when installing CircuitPython, try loading this 'nuke' UF2 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.

Download flash erasing "nuke" UF2

---

**Code the Darth Faders**

**Text Editor**

Adafruit recommends using the Mu editor for editing your CircuitPython code. You can get more info in [this guide](#).

Alternatively, you can use any text editor that saves simple text files.

**Download the Project Bundle**

Your project will use a specific set of CircuitPython libraries, plus the code.py file. To get everything you need, click on the Download Project Bundle link below, and uncompress the .zip file.

Drag the contents of the uncompressed bundle directory onto your board's CIRCUITPY drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.

```python
# SPDX-FileCopyrightText: 2022 John Park and Tod Kurt for Adafruit Industries
# SPDX-License-Identifier: MIT

# Darth Faders motorized slide potentiometer sculpture/prop

import time
import board
import analogio
from adafruit_debouncer import Debouncer
from adafruit_motorkit import MotorKit
from adafruit_seesaw import seesaw, rotaryio, digitalio

num_faders = 3

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a microcontroller
motorwing = MotorKit(i2c=i2c)
motorwing.frequency = 122  # tune this 50 - 200 range
max_throttle = 0.18  # tune this 0.2 - 1 range

# make arrays for all the things we care about
motors = [None] * num_faders
faders = [None] * num_faders
faders_pos = [None] * num_faders
```
last_faders_pos = [None] * num_faders

# set up motors in motor array
motors[0] = motorwing.motor1
motors[1] = motorwing.motor2
motors[2] = motorwing.motor3

# set motors to "off"
for i in range(num_faders):
    motors[i].throttle = None

# STEMMA QT Rotary encoder setup
seesaw = seesaw.Seesaw(i2c, addr=0x36)  # default address is 0x36
seesaw.pin_mode(24, seesaw.INPUT_PULLUP)
button_in = digitalio.DigitalIO(seesaw, 24)
button = Debouncer(button_in)
encoder = rotaryio.IncrementalEncoder(seesaw)
last_encoder_pos = 0

# set up faders
fader_pins = (board.A0, board.A1, board.A2)
for i in range(num_faders):
    faders[i] = analogio.AnalogIn(fader_pins[i])
    faders_pos[i] = faders[i].value // 256  # make it 0-255 range
    last_faders_pos[i] = faders_pos[i]

def update_position(fader, new_position, speed):
    global faders_pos  # pylint: disable=global-statement
    faders_pos[fader] = int(faders[fader].value//256)
    if abs(faders_pos[fader] - new_position) > 2 :
        if faders_pos[fader] > new_position :
            motors[fader].throttle = speed
        if faders_pos[fader] < new_position:
            motors[fader].throttle = speed * -1
        faders_pos[fader] = int(faders[fader].value//256)
    if faders_pos[fader] == new_position:
        motors[fader].throttle = None

# pre-saved positions for the buttons to call
H = 240  # high
M = 127  # mid
L = 10  # low

# create custom animation patterns here:
saved_positions = (
    (M, H, L, L, H, M, L),  # fader 1
    (M, H, L, H, M, L),  # fader 2
    (M, L, L, H, H, M, L)  # fader 3
)

num_positions = len(saved_positions[0])  # how many moves in our move list
position = 0  # which column of our 'saved_position' move list we're currently on
last_time = time.monotonic()
period = 4  # time in seconds between new desitination picks
print('Darth Fader will see you know.')

move_state = False
encoder_delta = 0

while True:
    button.update()
    if button.fell:
        move_state = not move_state
print("move_state is " + str(move_state))
if not move_state:
    for i in range(num_faders):
        motors[i].throttle = None
# always move all motors toward destinations
for i in range(num_faders):
    update_position(i, saved_positions[i][position], max_throttle)
# has encoder been turned?
encoder_pos = -encoder.position
if encoder_pos != last_encoder_pos:
    encoder_delta = encoder_pos - last_encoder_pos
    last_encoder_pos = encoder_pos
# if we are not moving automatically, allow encoder tuning animation frames
if not move_state:
    if encoder_delta:  # encoder was turned
        direction = 1 if (encoder_delta > 0) else -1  # which direction encoder was turned
        position = (position + direction) % num_positions  # increment/decrement
        encoder_delta = 0
# else we are moving automatically
if move_state:
    # if it is time to go to a new destination, do it
    if time.monotonic() - last_time > period:  # after 'period' seconds, change
        last_time = time.monotonic()
        position = (position + 1) % num_positions

How It Works

The Darth Faders have two modes -- free running where the faders will move to different predefined positions continuously, and per-pose mode where the rotary encoder switches from pose to pose.

The motorized slide pots act as linear actuators with servo control -- there is a feedback loop where the motor moves the slide which changes the analog read value of the potentiometer's resistance. This is what allows you to send a fader to a specific position.

Libraries

You'll import libraries to provide functionality in the code:

- time
- board
- analogio
- adafruit_debouncer
- adafruit_motorkit
- adafruit_seesaw
import time
import board
import analogio
from adafruit_debouncer import Debouncer
from adafruit_motorkit import MotorKit
from adafruit_seesaw import seesaw, rotaryio, digitalio

Motors

The fader motors are controlled by the motor FeatherWing via I2C. Here’s the setup:

```python
num_faders = 3
motorwing = MotorKit(i2c=board.I2C())
motorwing.frequency = 122  # tune this 50 - 200 range
max_throttle = 0.18  # tune this 0.2 - 1 range

# make arrays for all the things we care about
motors = [None] * num_faders
faders = [None] * num_faders
faders_pos = [None] * num_faders
last_faders_pos = [None] * num_faders

# set up motors in motor array
motors[0] = motorwing.motor1
motors[1] = motorwing.motor2
motors[2] = motorwing.motor3

# set motors to "off"
for i in range(num_faders):
    motors[i].throttle = None
```

The PWM frequency and throttle values allow you to find the right balance between smoothness and speed (more on tuning PWM for DC motors in this guide).

Rotary Encoder

The STEMMA QT Rotary Encoder breakout runs on a seesaw chip and communicates via I2C. The push encoder button is set up with debouncer.

```python
# STEMMA QT Rotary encoder setup
seesaw = seesaw.Seesaw(board.I2C(), addr=0x36)  # default address is 0x36
seesaw.pin_mode(24, seesaw.INPUT_PULLUP)
button_in = digitalio.DigitalIO(seesaw, 24)
button = Debouncer(button_in)
encoder = rotaryio.IncrementalEncoder(seesaw)
last_encoder_pos = 0
```

Fader Analog Read

The three slide potentiometers are set up on analog read pins and their raw values are re-mapped to 0-255.
# set up faders
fader_pins = (board.A0, board.A1, board.A2)
for i in range(num_faders):
faders[i] = analogio.AnalogIn(fader_pins[i])
faders_pos[i] = faders[i].value // 256  # make it 0-255 range
last_faders_pos[i] = faders_pos[i]

**Position Function**

Whenever the faders need to move, this `update_positions()` function is called with the arguments for which fader, what position, and the speed.

```python
def update_position(fader, new_position, speed):
global faders_pos  # pylint: disable=global-statement
    faders_pos[fader] = int(faders[fader].value//256)
    if abs(faders_pos[fader] - new_position) > 2:
        if faders_pos[fader] > new_position:
            motors[fader].throttle = speed
        if faders_pos[fader] < new_position:
            motors[fader].throttle = speed * -1
        faders_pos[fader] = int(faders[fader].value//256)
    if faders_pos[fader] == new_position:
        motors[fader].throttle = None
```

**Animation List**

We can treat the positions sort of like poses in an animation or values in a sequence. Each fader has a list of positions, these are used when calling the `update_position()` function.

```python
# pre-saved positions for the buttons to call
H = 240  # high
M = 127  # mid
L = 10   # low

# create custom animation patterns here:
saved_positions = (
    (M, H, L, L, H, M, L),  # fader 1
    (M, H, L, H, H, M, L),  # fader 2
    (M, L, L, H, H, M, L)   # fader 3
)

num_positions = len(saved_positions[0])  # how many moves in our move list
position = 0  # which column of our 'saved_position' move list we're currently on
last_time = time.monotonic()
period = 4  # time in seconds between new desitnation picks
```
Main Loop

The main loop checks for encoder button presses to change between the two modes, then it checks to see if the encoder has been turned to switch from pose-to-pose.

If the move state is true, the `update_position()` function is called, and after the pre-set period is complete it moves on to the next one in the list.

```python
while True:
    button.update()
    if button.fell:
        move_state = not move_state
        print("move_state is " + str(move_state))
        if not move_state:
            for i in range(num_faders):
                motors[i].throttle = None

    # always move all motors toward destinations
    for i in range(num_faders):
        update_position(i, saved_positions[i][position], max_throttle)

    # has encoder been turned?
    encoder_pos = -encoder.position
    if encoder_pos != last_encoder_pos:
        encoder_delta = encoder_pos - last_encoder_pos
        last_encoder_pos = encoder_pos

    # if we are not moving automatically, allow encoder tuning animation frames
    if not move_state:
        if encoder_delta:  # encoder was turned
            direction = 1 if (encoder_delta > 0) else -1  # which direction encoder was turned
            position = (position + direction) % num_positions  # increment/decrement
            encoder_delta = 0

    # else we are moving automatically
    if move_state:
        # if it is time to go to a new destination, do it
        if time.monotonic() - last_time > period:  # after 'period' seconds, change
            last_time = time.monotonic()
            position = (position + 1) % num_positions
```

©Adafruit Industries