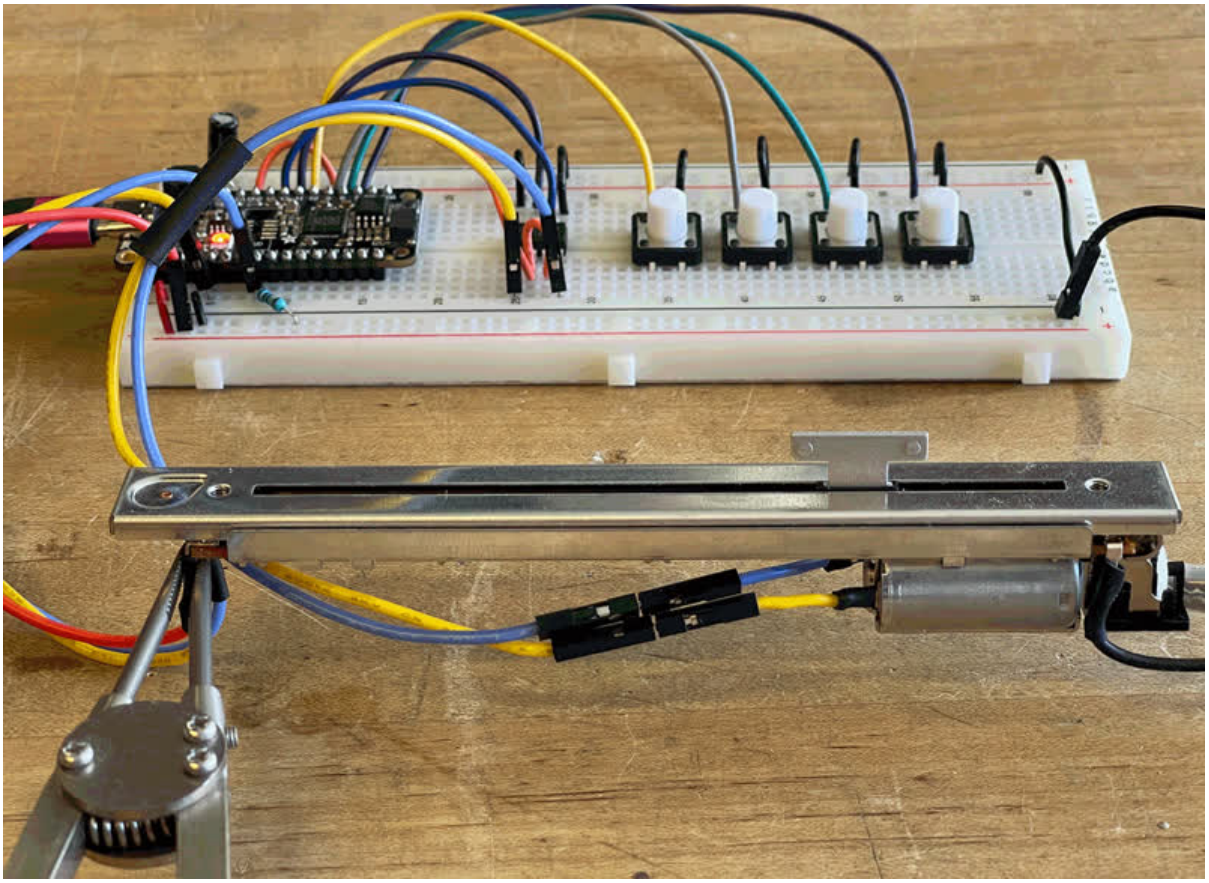
Flying Faders

Created by John Park

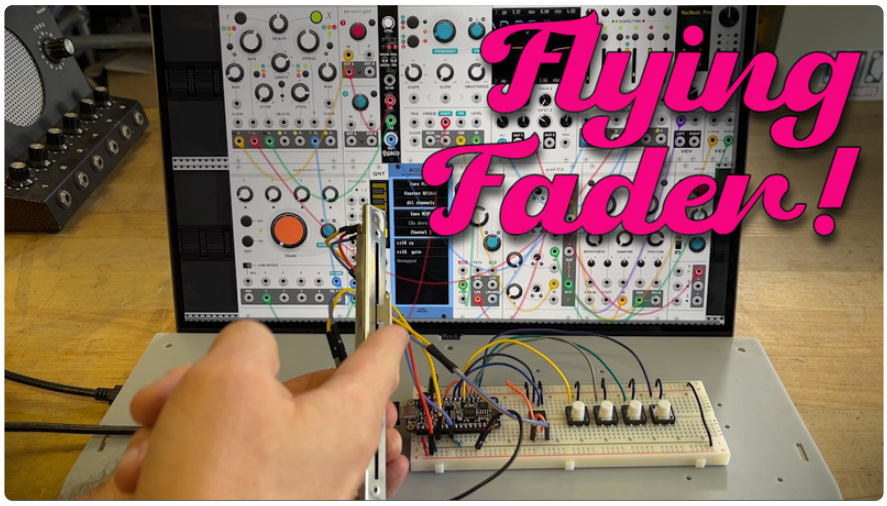


https://learn.adafruit.com/flying-faders

Last updated on 2024-06-03 03:40:03 PM EDT

Table of Contents

Overview

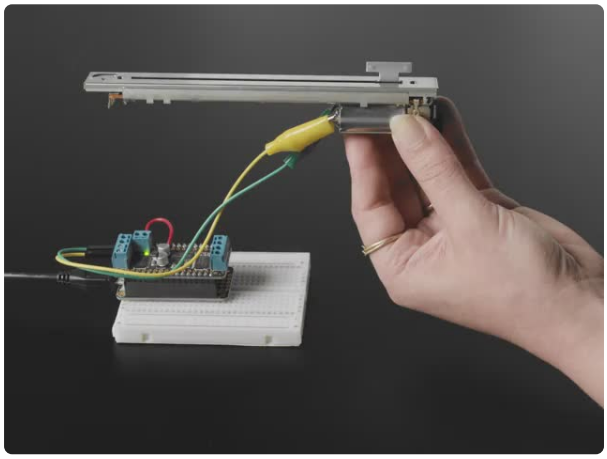


Motorized slide potentiometers, or "flying faders", are incredibly cool. They are most commonly found on large mixing boards at high end recording studios, used to allow fine control of audio levels while permitting pre-set positions to be recalled in real-time.

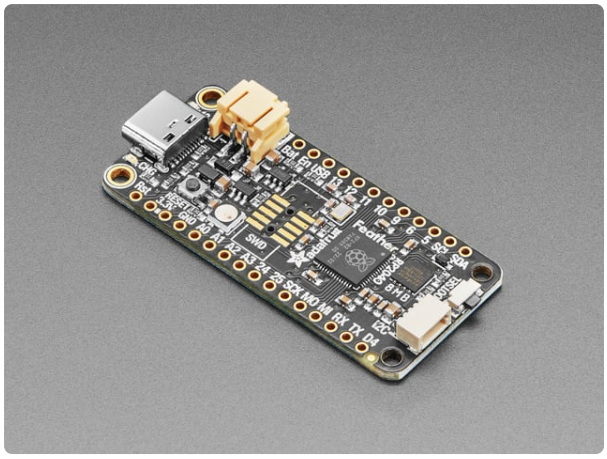
They consist of a slide potentiometer, and a motor that can convert rotational motion to linear motion, and either a capacitive touch sensor or current sensor to allow interaction with the fader while it is moving (the sensing prevents motor stalling or stripping of gears).

This guide will show you how to use the motorized slide pot with touch sensor in your projects -- this can be anything from lighting control, to MIDI synths, to animatronics, or home automation!

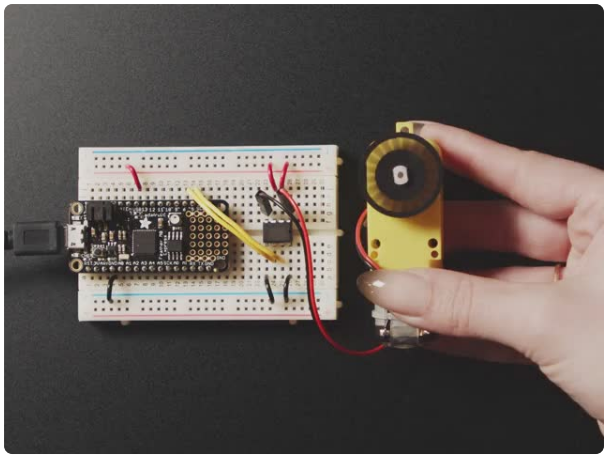
Parts



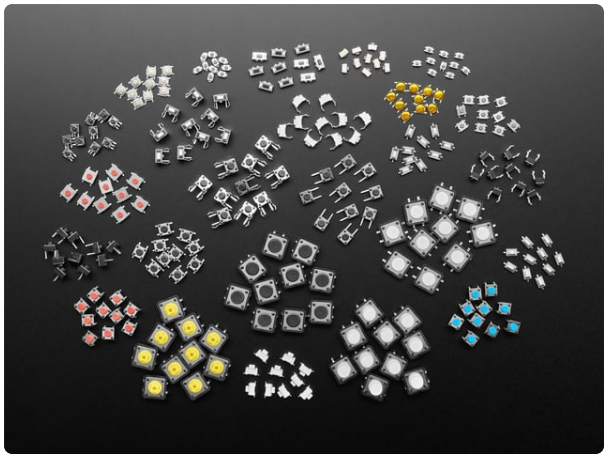
Motorized Slide Potentiometer - 10KΩ Linear with 5V DC Motor
This linear slide potentiometer comes with an added kick - its not just a plain old slide pot!...
https://www.adafruit.com/product/5466



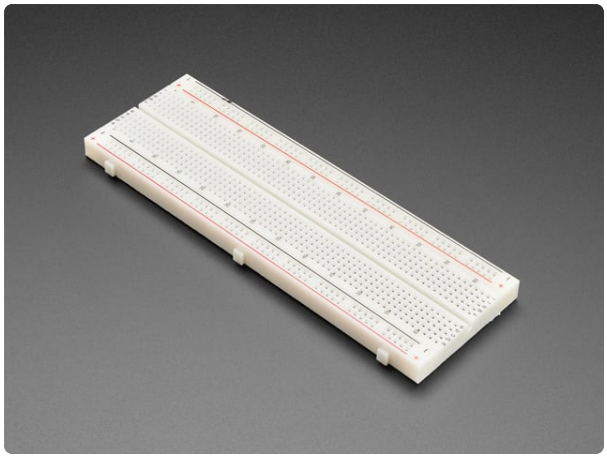
[Adafruit Feather RP2040]
A new chip means a new Feather, and the Raspberry Pi RP2040 is no exception. When we saw this chip we thought "this chip is going to be awesome when we give it the Feather...
https://www.adafruit.com/product/4884



[L9110H H-Bridge Motor Driver for DC Motors - 8 DIP]
Run two solenoids or a single DC motor with up to 800mA per channel using the super-simple L9110H H-bridge driver. This bridge chip is an 8 DIP package so it's easy to fit onto any...
https://www.adafruit.com/product/4489



[Tactile Switch Assortment - 25 Different Buttons - 10 pcs each]
When designing your next PCB, if you're not sure what sort of tactile switches you can fit, or how they would look, this Tactile Switch Assortment with 25 Different...
https://www.adafruit.com/product/5493

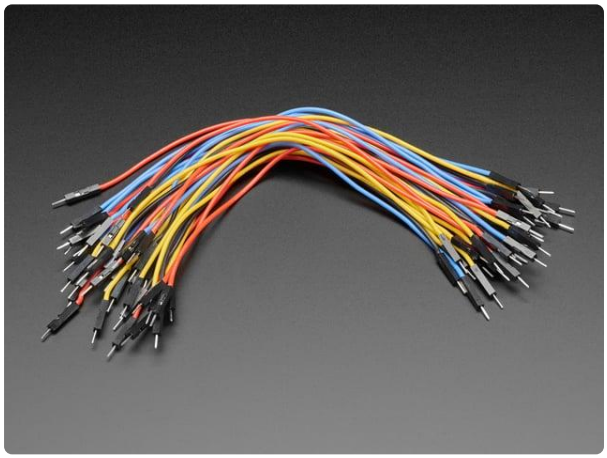


[Full Sized Premium Breadboard - 830 Tie Points]
This is a 'full-size' premium quality breadboard, 830 tie points. Good for small and medium projects. It's 2.2" x 7" (5.5 cm x 17 cm) with a standard double-strip...
https://www.adafruit.com/product/239

Pack of 10

1 x 47uF Electolytic Capacitors https://www.adafruit.com/product/2194

Pack of 10



Premium Silicone Covered Male-Male Jumper Wires - 200mm x 40

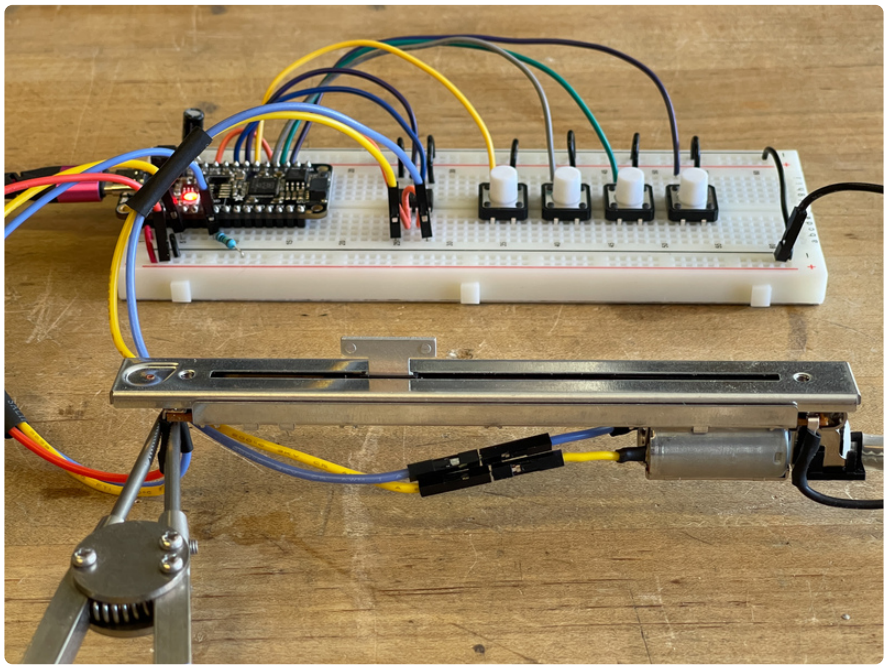
These premium male-male jumper wires are handy for making wire harnesses or jumpering between headers on PCBs. They're 200mm (~7.8") long and come loose as a pack of...

https://www.adafruit.com/product/4482

1 x Hook-up Wire Spool Set https://www.adafruit.com/product/3174

22AWG Solid Core - 10 x25ft

Build the Flying Fader Circuit



Wiring

The motorized slide pot has three subsystems to connect to the microcontroller/driver circuit:

- **Potentiometer**
- **Motor**
- **Touch sense**

The potentiometer is read as a voltage divider by the microcontroller. It has three pins -- positive voltage, wiper (the sliding, variable potentiometer part), and ground. These are connected as such to the Feather:

- **Potentiometer pin 1 to Feather GND**
- **Potentiometer pin 2 to Feather A0 (any analog ADC pin works)**
- **Potentiometer pin 3 to Feather 3V**

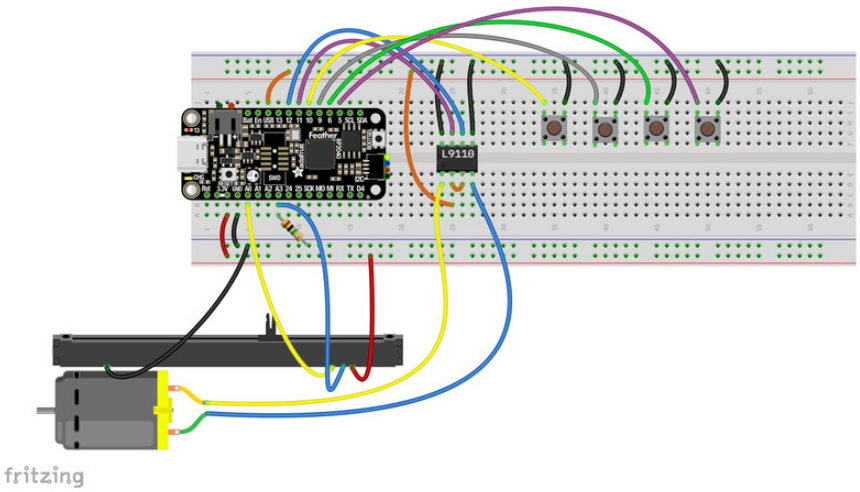
The **touch sense** strip is the pin labeled **"T"** in the datasheet and connects to any cap sense capable pin on the microcontroller.

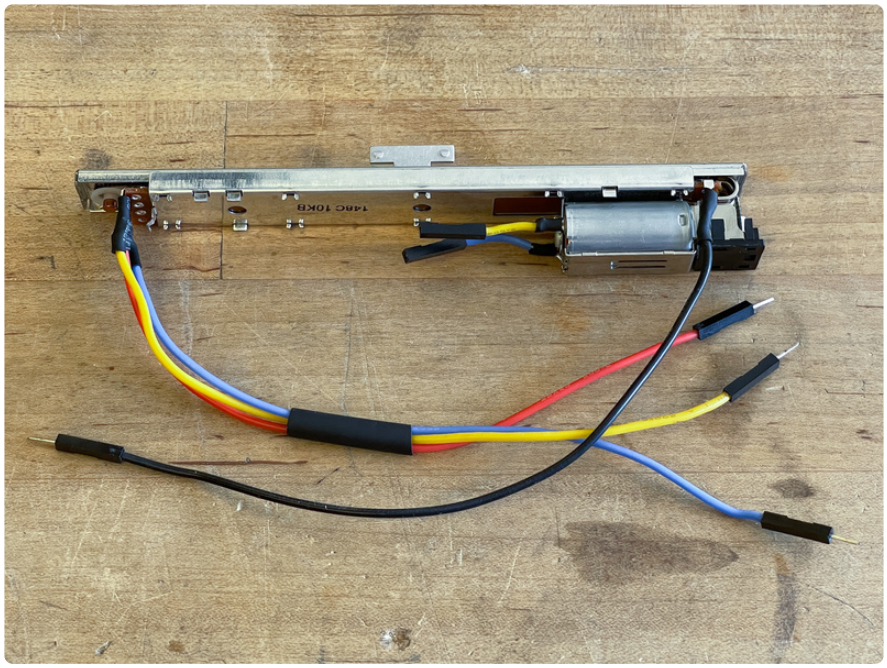
- **Touch sense pin to Feather A3**
- **Feather A3 to 1MΩ resistor to GND** (without the resistor, cap touch will not work on some microcontroller chips)

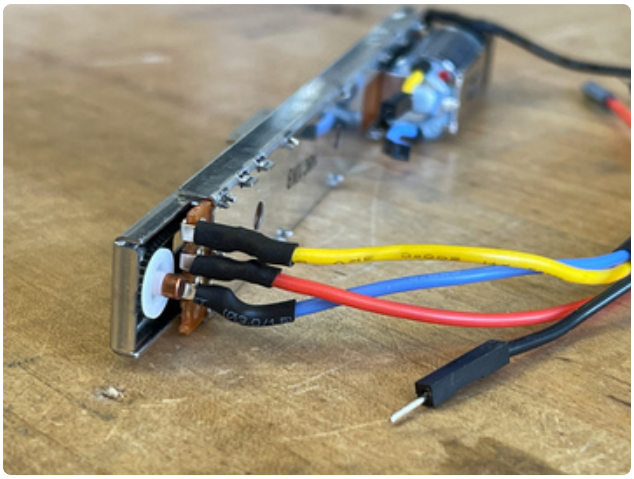
The **motor** is connected to an **h-bridge** driver. This allows the microcontroller to control the motor at variable speeds in both "forward" and "reverse" directions.

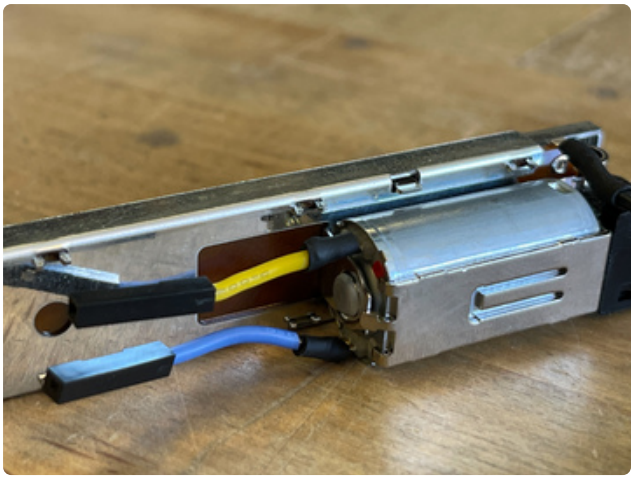
- **Motor A pin to L9110 Output A (pin 1)**
- **Motor B pin to L9110 Output B (pin 4)**
- **Feather D12 to L9110 Input A (pin 6)**
- **Feather D11 to L9110 Input B (pin 7)**
- **Feather USB 5V to L9110 VCC (pins 2, 3)**
- **L9110 GND (pins 5, 8) to Feather GND**

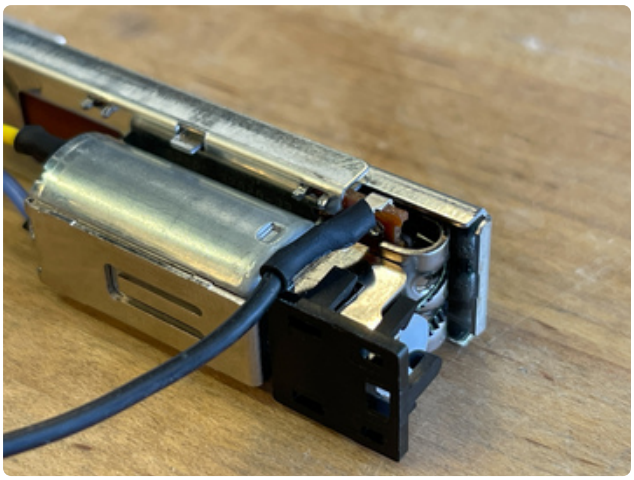
The motor is rated for 6V-10V but works fairly well at 5V. If you have trouble, particularly with touch sensitivity, make sure your USB port is providing a clean 5V, otherwise, you can hook up and external supply so long as you ground it. 8V really zips along nicely!


L9110
fritzing







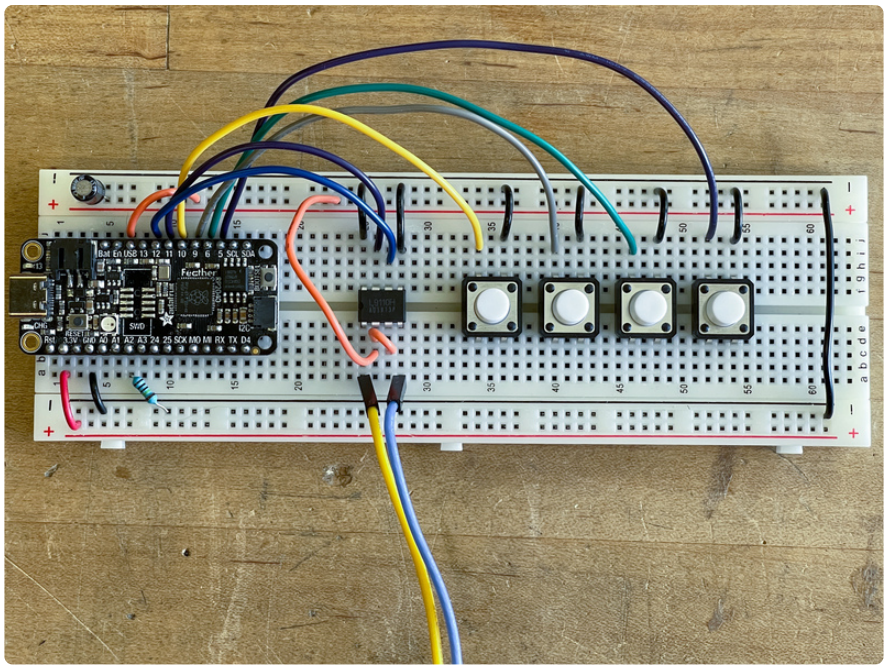


Wire the Fader

The motorized slide pot has solder lugs for connection points. Strip some insulation from your jumper wires and solder them to the lugs. You can also use heat shrink tubing to neaten up the wiring as shown here.

Buttons

For the demo here, we'll also use four momentary switch push-buttons. They are connected respectively to **Feather D5, D6, D9, D10** on one side and **GND** on the other.

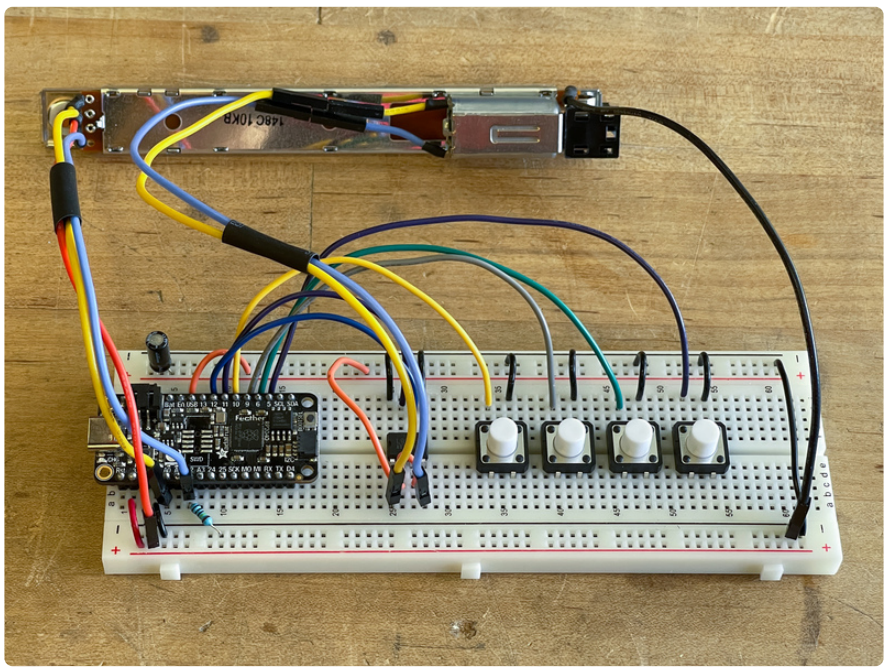


An optional electrolytic capacitor anywhere from 10uF-100uF can be connected to the power rails with the negative leg on Ground and positive leg on input voltage in order to smooth any voltage spikes or sags.

Breadboard basics link

https://adafru.it/18fA

Here's the whole kit and caboodle wired together for use.



Install CircuitPython

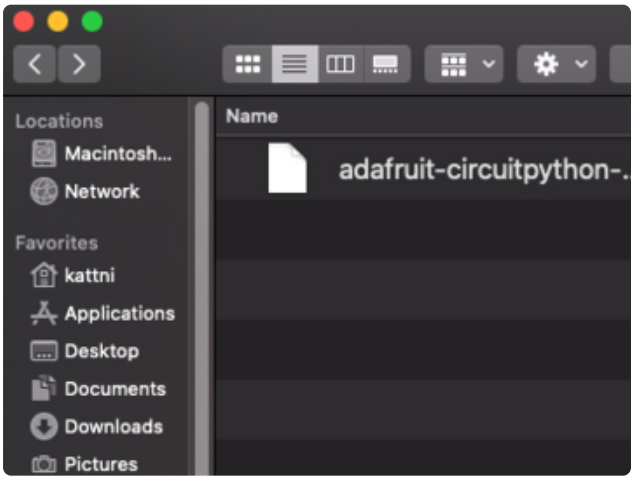
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

CircuitPython Quickstart

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

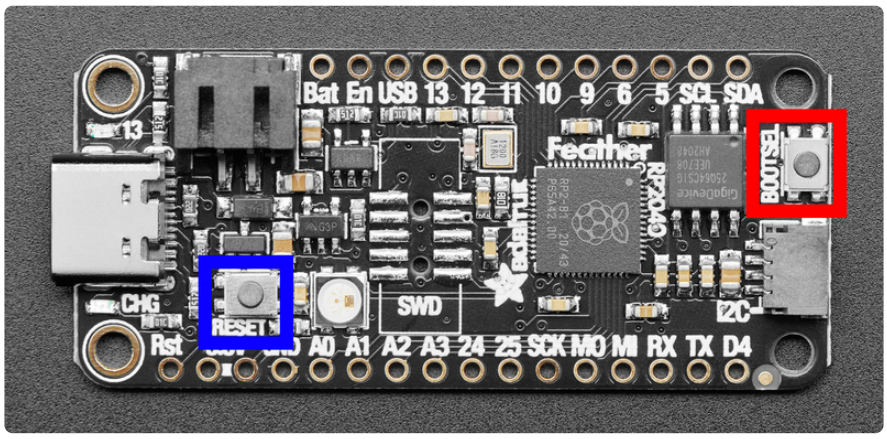
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/R1D



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

Save it wherever is convenient for you.



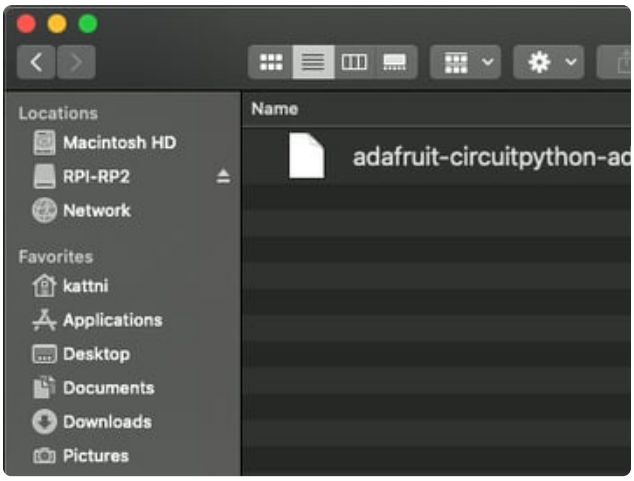
To enter the bootloader, hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset**

button (highlighted in blue above). **Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!**

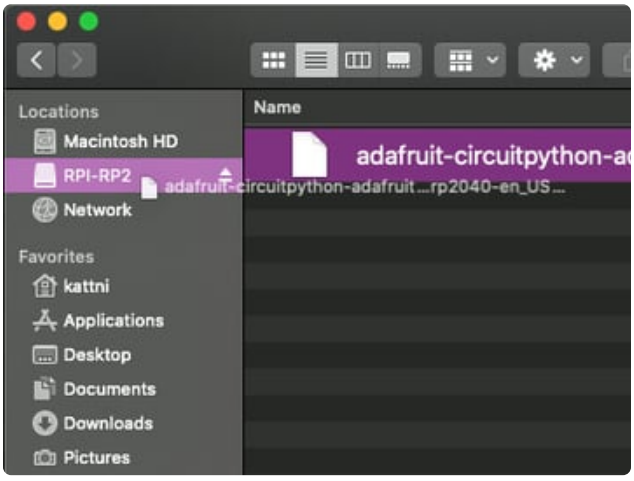
If the drive does not appear, release all the buttons, and then repeat the process above.

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.

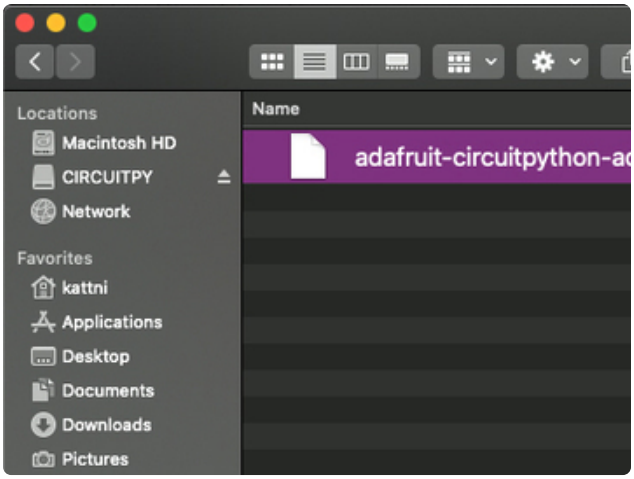
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



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 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.

Download flash erasing "nuke" UF2

https://adafru.it/RLE

Code the Flying Fader in CircuitPython

Text Editor

Adafruit recommends using the **Mu** editor for editing your CircuitPython code. You can get more info in this guide (https://adafru.it/ANO).

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 and 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 Feather board's **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.

```
# SPDX-FileCopyrightText: 2022 John Park for Adafruit Industries
# SPDX-License-Identifier: MIT
# Motorized fader demo
import time
import board
import pwmio
import analogio
import touchio
import neopixel
from digitalio import DigitalInOut, Pull
from adafruit_debouncer import Debouncer
from adafruit_motor import motor

MIDI_DEMO = False  # set to True to send MIDI CC

# optional MIDI setup
if MIDI_DEMO:
    import usb_midi
    import adafruit_midi
    from adafruit_midi.control_change import ControlChange
    midi = adafruit_midi.MIDI(midi_out=usb_midi.ports[1], out_channel=0)
    fader_cc_number = 16

# Button setup to store four saved values
button_pins = (board.D10, board.D9, board.D6, board.D5)
buttons = []
for button_pin in button_pins:
    tmp_pin = DigitalInOut(button_pin)
    tmp_pin.pull = Pull.UP
    buttons.append(Debouncer(tmp_pin))

saved_positions = (230, 180, 120, 60)  # pre-saved positions for the buttons to call

# Slide pot setup
fader = analogio.AnalogIn(board.A0)
fader_position = fader.value  # ranges from 0-65535
fader_pos = fader.value // 256  # make 0-255 range
last_fader_pos = fader_pos

# Motor setup
PWM_PIN_A = board.D12  # pick any pwm pins on their own channels
PWM_PIN_B = board.D11

# DC motor driver setup -- these pins go to h-bridge driver such as L9110
pwm_a = pwmio.PWMOut(PWM_PIN_A, frequency=50)
pwm_b = pwmio.PWMOut(PWM_PIN_B, frequency=50)
motor1 = motor.DCMotor(pwm_a, pwm_b)

# Touch setup  pin goes from touch pin on slide pot to touch capable pin and then
1MΩ to gnd
touch = touchio.TouchIn(board.A3)
```

```
touch.threshold = touch.raw_value + 30  # tune for fader knob cap

# NeoPixel setup
led = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2, auto_write=True)
led.fill(0xff0000)

def clamp(num, min_value, max_value):  # function for clamping motor throttle -1.0
to 1.0
    return max(min(num, max_value), min_value)

def go_to_position(new_position):
    global fader_pos  # pylint: disable=global-statement
    fader_pos = int(fader.value//256)
    while abs(fader_pos - new_position) > 2 :
        if fader_pos > new_position :
            speed = 2.25 * abs(fader_pos - new_position) / 256 + 0.12
            speed = clamp(speed, -1.0, 1.0)
            motor1.throttle = speed
            led[0] = (fader_pos, 0, 0)

            if MIDI_DEMO:
                global fader_cc  # pylint: disable=global-statement
                fader_cc = int(fader_pos / 2)  # cc is 0-127
                midi.send(ControlChange(fader_cc_number, fader_cc))

        if fader_pos < new_position:
            speed = -2.25 * abs(fader_pos - new_position) / 256 - 0.12
            speed = clamp(speed, -1.0, 1.0)
            motor1.throttle = speed
            led[0] = (fader_pos, 0, 0)

            if MIDI_DEMO:
                fader_cc = int(fader_pos / 2)  # cc is 0-127
                midi.send(ControlChange(fader_cc_number, fader_cc))

        fader_pos = int(fader.value//256)
    motor1.throttle = None
print("--__ Flying Fader Demo __--")
print("\n"*4)

go_to_position(saved_positions[3])  # boot up demo
go_to_position(saved_positions[0])
time.sleep(.6)

current_saved_position = 0  # state to store which is current position from the list

while True:
    for i in range(len(buttons)):
        buttons[i].update()
        if buttons[i].fell:  # if a button is pressed, update the position from list
            current_saved_position = i

    if touch.value:
        motor1.throttle = None  # idle
    else:
        go_to_position(saved_positions[current_saved_position])

    filter_amt = 0.1  # higher number will be a slower filter between 1.0 and 0.1
is good
    fader_pos = int((filter_amt * last_fader_pos) + ((1.0-filter_amt) *
fader.value//256))
    led[0] = (fader_pos, 0, 0)
    if abs(fader_pos - last_fader_pos) > 1 :  # do things in here, e.g. send MIDI CC
        fader_width = 90  # for text visualization in serial output
        print("-" * (fader_width - int(fader_pos/3)), fader_pos, "-" *
int(fader_pos/3), end='\r')
        last_fader_pos = fader_pos
        if MIDI_DEMO:
```

```
            fader_cc = int(fader_pos / 2)  # cc is 0-127
            midi.send(ControlChange(fader_cc_number, fader_cc))
```

Use the Flying Fader

Once the code is up and running, allow it to run through the boot up sequence that will move the fader to a couple of position presets, and it will also calibrate the touch sensor. Be sure not to touch the potentiometer during this start up.

Then, you can press any of the four buttons to go to a preset location, or move the fader -- you'll feel the motor is disengaged until you let go, then it will move back to the last preset position!

The default may be fine, but you can fine tune the touch sensitivity by adjusting the value of this line: touch.threshold = touch.raw_value + 30

How It Works

Libraries

First, the code imports the necessary libraries. In particular:

- `pwmio` along with `adafruit_motor` are used to control the motor speed and direction via the H-bridge driver
- `analogio` is used to read the slide potentiometer as an analog input
- `touchio` is used to detect when the potentiometer lever is being touched

```
import time
import board
import pwmio
import analogio
import touchio
import neopixel
from digitalio import DigitalInOut, Pull
from adafruit_debouncer import Debouncer
from adafruit_motor import motor
```

Button Setup

Button setup is optional (you can use the motorized slide pot without buttons) but it's nice to have some buttons to press to go to preset positions. You could also repurpose these to record new presets if you like.

```
button_pins = (board.D10, board.D9, board.D6, board.D5)
buttons = []
for button_pin in button_pins:
    tmp_pin = DigitalInOut(button_pin)
    tmp_pin.pull = Pull.UP
    buttons.append(Debouncer(tmp_pin))
```

The `saved_positions` list is a set of integer numbers that can be recalled by pressing buttons. Change these to whatever you like.

```
saved_positions = (230, 180, 120, 60)
```

Slide Potentiometer Setup

The slide potentiometer is set up as an analog input using **analogio** on pin **A0** (an ADC pin is required) named `fader`.

By reading the pin value (`fader.value`) a 16-bit value is returned -- we then convert this to a 0-255 range for use in the code (the ADC is usually 12- or 10-bit, so the 16-bit value tends to be too noisy in practice. The 8-bit works very well.)

A state variable called `last_fader_pos` is created so we can detect when the pot position is changed.

```
fader = analogio.AnalogIn(board.A0)
fader_position = fader.value  # ranges from 0-65535
fader_pos = fader.value // 256  # make 0-255 range
last_fader_pos = fader_pos
```

Motor Setup

The motor is a DC motor that can be controlled to go in either direction and at varying speed by using a full H-bridge driver, in this case the L9910.

To do this, we'll use two **PWM** (pulse width modulation) pins from the microcontroller which act as a sort of pseudo analog output.

The two PWM objects are created with a frequency of 50Hz (more on tuning PWM for DC motors in this guide (https://adafru.it/10hD)), and then the **motor1** object is created using `motor.DCMotor()` .

```
PWM_PIN_A = board.D12
PWM_PIN_B = board.D11

pwm_a = pwmio.PWMOut(PWM_PIN_A, frequency=50)
```

```
pwm_b = pwmio.PWMOut(PWM_PIN_B, frequency=50)
motor1 = motor.DCMotor(pwm_a, pwm_b)
```

Touch Setup

Next, the capacitive touch sense for the pot lever is set up. This requires a touch-capable pin.

You can fine-tune the touch threshold if you want, or use the setting that is self-calibrated during setup.

```
touch = touchio.TouchIn(board.A3)
touch.threshold = touch.raw_value + 30
```

Clamp Function

This function is created to simply clamp the motor throttle values between -1.0 and 1.0.

```
def clamp(num, min_value, max_value):
    return max(min(num, max_value), min_value)
```

Go_to_position Function

This function is called whenever the fader needs to fly to a new position!

The `new_position` argument can be a value from 0-255.

The way this works is a sort of PD (position-derivative) feedback loop. Essentially, the motor moves a tiny bit, then the slide potentiometer value is read. The difference between the current position and the new position will continue to decrease, approaching zero, as the motor moves the slide to the goal.

To avoid overshoot, the throttle value is decreased as the slider gets closer to the goal, sort of like slowing a car by letting off the gas before hitting the brakes at a stop sign.

```
def go_to_position(new_position):
    global fader_pos
    fader_pos = int(fader.value//256)
    while abs(fader_pos - new_position) > 2 :
        if fader_pos > new_position :
            speed = 2.25 * abs(fader_pos - new_position) / 256 + 0.12
            speed = clamp(speed, -1.0, 1.0)
            motor1.throttle = speed
```

```
            led[0] = (fader_pos, 0, 0)

        if fader_pos < new_position:
            speed = -2.25 * abs(fader_pos - new_position) / 256 - 0.12
            speed = clamp(speed, -1.0, 1.0)
            motor1.throttle = speed
            led[0] = (fader_pos, 0, 0)

        fader_pos = int(fader.value//256)
    motor1.throttle = None
```

Main Loop

The main loop of the program does three essential things:

- Check for button presses, which change the `current_saved_position` value
- Check for touch sense on the potentiometer lever, which disables the motor throttle to allow free movement. When the touch is removed, the `go_to_positon()` function is called to return the fader to the last position
- Watch for changes to the potentiometer value -- this can be both when manually moving the fader or when the motor is driving it. When the fader is slid, the analog values are filtered to remove noise/jitter and then anything you want to happen, happens! This can be LED fading, MIDI output, etc.

```
for i in range(len(buttons)):
        buttons[i].update()
        if buttons[i].fell:  # if a button is pressed, update the position from list
            current_saved_position = i

    if touch.value:
        motor1.throttle = None  # idle
    else:
        go_to_position(saved_positions[current_saved_position])

    filter_amt = 0.1  # higher number will be a slower filter between 1.0 and 0.1
is good
    fader_pos = int((filter_amt * last_fader_pos) + ((1.0-filter_amt) *
fader.value//256))
    led[0] = (fader_pos, 0, 0)
    if abs(fader_pos - last_fader_pos) > 1 :  # do things in here, e.g. send MIDI CC
        fader_width = 90  # for text visualization in serial output
        print("-" * (fader_width - int(fader_pos/3)), fader_pos, "-" *
int(fader_pos/3), end='\r')
        last_fader_pos = fader_pos
```

Code Customization

The demo code includes an option to send MIDI CC values. To enable this, simply change the line:

`MIDI_DEMO = False`

to

`MIDI_DEMO = True`

This will allow the code to load the necessary MIDI libraries and then send 0-127 values over MIDI channel 1 CC number 16. (You can change those settings as well if you like.)

Try firing up a software synthesizer and set a synth parameter to be controlled by your flying fader!

Code the Flying Fader in Arduino

The Arduino version works in nearly the same way as the CircuitPython version. First, set up the Arduino IDE for use with Feather RP2040 as shown in this guide (https://adafru.it/-tf).

Then, copy the code below and paste it into a new Arduino document. Save it in your Arduino project folder, and then upload it to the Feather, using the settings shown in the guide linked above.

```
// SPDX-FileCopyrightText: 2022 John Park for Adafruit Industries
// SPDX-License-Identifier: MIT
// Motorized fader demo
// capsense implementation by @todbot / Tod Kurt

#include <Bounce2.h>

const int pwmA = 12;  // motor pins
const int pwmB = 11;

const int fader = A0;  // fader pin
int fader_pos = 0;
float filter_amt = 0.75;
float speed = 1.0;

int saved_positions[] = { 230, 180, 120, 60 } ;
int current_saved_position = 1 ;

const int num_buttons = 4;
const int button_pins[num_buttons] = {10, 9, 8, 7};  // feather silk != arduino pin
number. 10, 9, 6, 5 on board
Bounce buttons[num_buttons];
bool motor_release_state = false;  // to handle motor release

class FakeyTouch
{
  public:
  FakeyTouch( int apin, int athreshold = 300 ) {  // tune the threshold value to
your hardware
    pin = apin;
    thold = athreshold;
  }
  void begin() {
```

```
    baseline = readTouch();
  }
  int readTouch() {
    pinMode(pin, OUTPUT);
    digitalWrite(pin,HIGH);
    pinMode(pin,INPUT);
    int i = 0;
    while( digitalRead(pin) ) { i++; }
    return i;
  }
  bool isTouched() {
    return (readTouch() > baseline + thold);
  }
  int baseline;
  int thold;
  int pin;
};

const int touchpin_F = A3;
FakeyTouch touchF = FakeyTouch( touchpin_F );


void setup() {
  Serial.begin(9600);
  delay(1000);
  pinMode (pwmA, OUTPUT);
  pinMode (pwmB, OUTPUT);
  analogWriteFreq(100);
  analogWrite(pwmA, 0);
  analogWrite(pwmB, 0);
  for (uint8_t i=0; i< num_buttons; i++){
    buttons[i].attach( button_pins[i], INPUT_PULLUP);
  }
}

int last_fader_pos = fader_pos;


void loop() {
  for (uint8_t i=0; i< num_buttons; i++){
    buttons[i].update();
    if( buttons[i].fell()) {
      current_saved_position = i;
      go_to_position(saved_positions[current_saved_position]);
    }
  }

  if( touchF.isTouched()){
    motor_release_state = true;
    analogWrite(pwmA, 0);
    analogWrite(pwmB, 0);
    delay(60);
  }
  else{
    motor_release_state = false;
    go_to_position(saved_positions[current_saved_position]);
  }

  fader_pos = int( (filter_amt * last_fader_pos) + ( (1.0-filter_amt) *
int(analogRead(fader) / 4 )) );
  if (abs(fader_pos - last_fader_pos) > 1) {
    Serial.println(fader_pos);
    if (motor_release_state==false){
      go_to_position(saved_positions[current_saved_position]);
    }
    last_fader_pos = fader_pos;
  }
}
```

```
void go_to_position(int new_position) {
  fader_pos = int(analogRead(fader) / 4);
  while (abs(fader_pos - new_position) > 4) {
   if (fader_pos > new_position) {
    speed = 2.25 * abs(fader_pos - new_position) / 256 + 0.2;
    speed = constrain(speed, -1.0, 1.0);
      if (speed > 0.0) {
        analogWrite(pwmA, 255);
        analogWrite(pwmB, 0);
      }
   }
   if (fader_pos < new_position) {
      speed = 2.25 * abs(fader_pos - new_position) / 256 - 0.2;
      speed = constrain(speed, -1.0, 1.0);
        if (speed > 0.0) {
          analogWrite(pwmA, 0);
          analogWrite(pwmB, 255);
        }
      }

    fader_pos = int(analogRead(fader) / 4);

  }
  analogWrite(pwmA, 0);
  analogWrite(pwmB, 0);
}
```