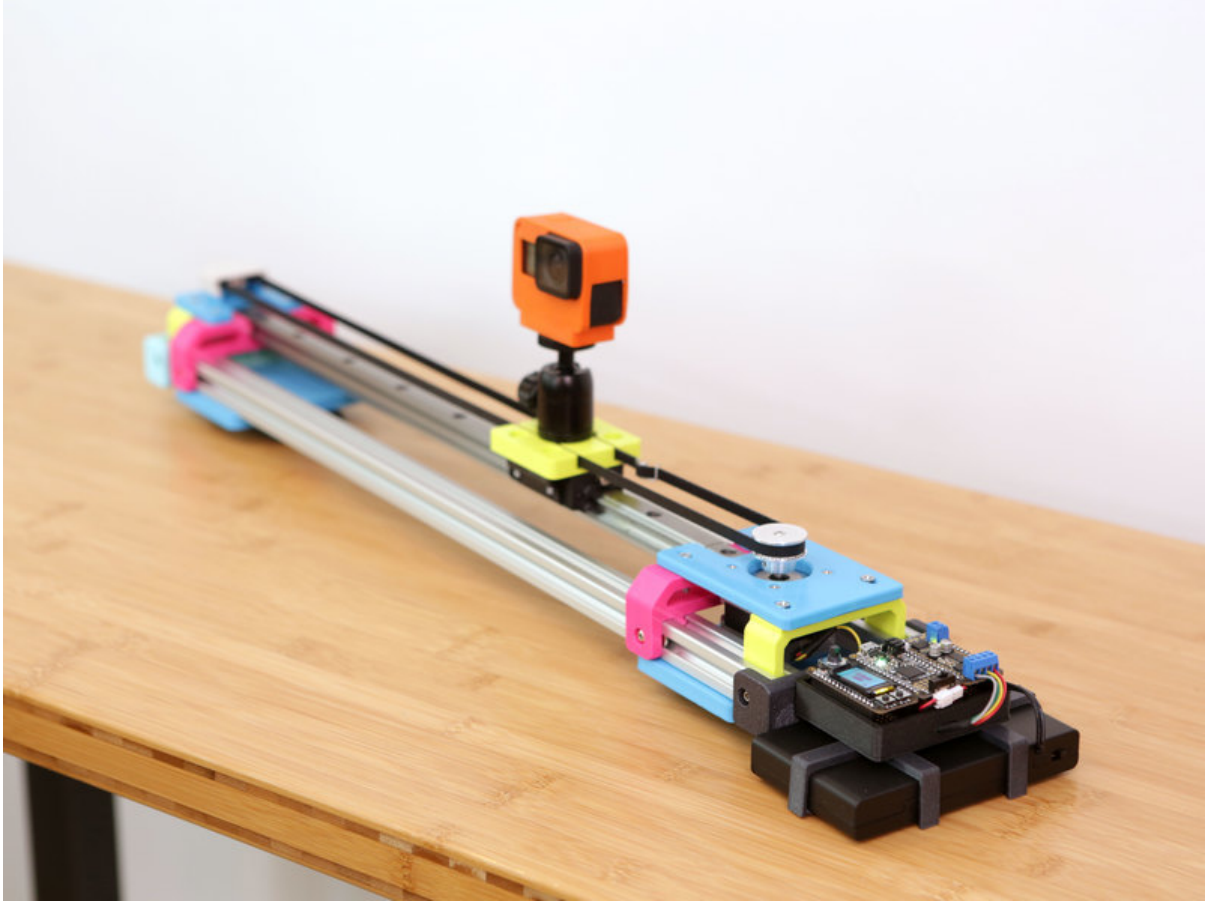
CircuitPython Motorized Camera Slider

Created by Ruiz Brothers



https://learn.adafruit.com/circuitpython-motorized-camera-slider

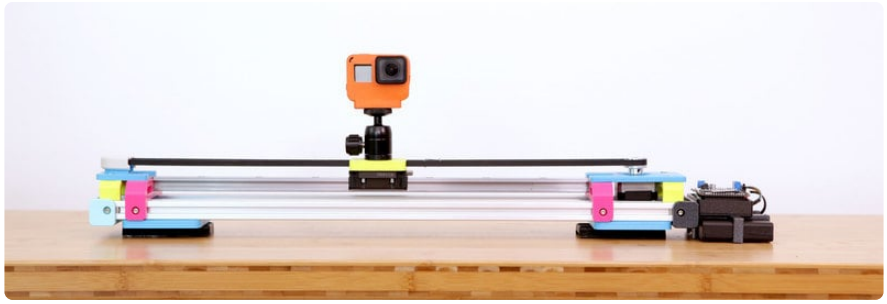
Last updated on 2024-06-03 02:58:54 PM EDT

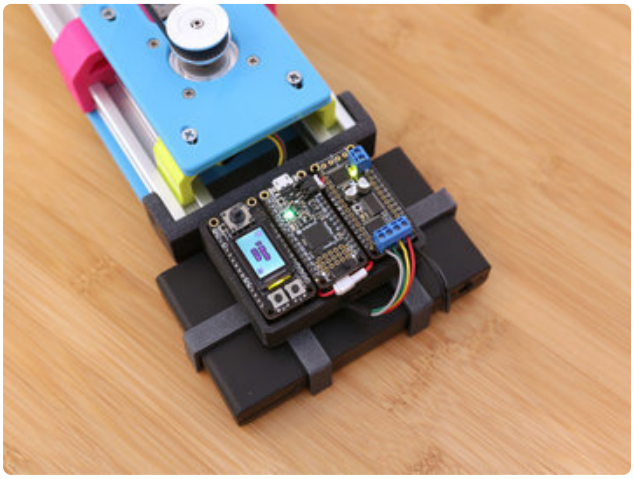
Table of Contents

Overview

Motorized Slider

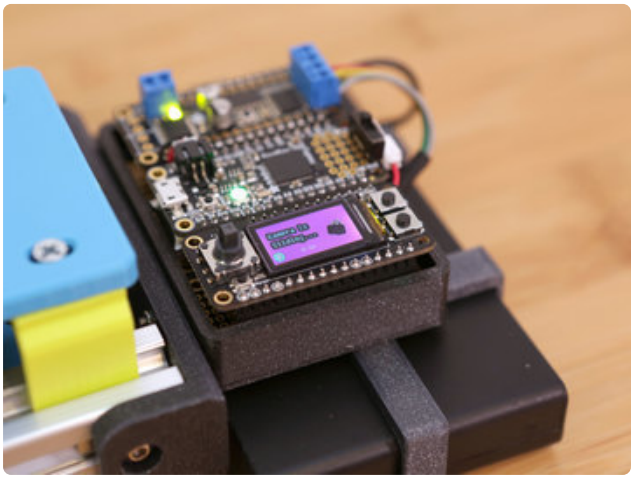
This project uses the **Adafruit Feather** platform and **CircuitPython** to make an easy to control motorized camera slider. The build uses aluminum extrusion to support a linear rail system with 3D printed parts. A camera can be mounted to the platform and slide along the rail. You may use the joystick and buttons to trigger different time settings presented on a mini TFT display.





FeatherWings

The **mini TFT FeatherWing** is perfect for selecting settings and displaying modes! The **MotorWing** provides the **Feather M4** with stepper motor control and makes it easy to connect. A **Tripler FeatherWing** keeps everything in place and makes it modular.



Features

The program features four timing presets to select from using the joystick and switch. **5-min, 10-min**, **20-min** and **60-min**. The status of the slider is displayed while it's in motion. Time remaining is also displayed alongside a nifty graphic and icon. When the slide is approaching the end, a STOP graphic is presented.

Reverse & Back Up

At the end of a slide, you can choose either to **back up** or go into **reverse mode**. This makes it really convenient to kick off a new timelapse!

Previous Build

This project was originally based off the **Adafruit Metro**, an Arduino compatible board. It used a motor shield and BLE module to control the slider using the Bluefruit LE Connect app. This project has the same mechanical design but with new parts utilizing an Adafruit Feather and FeatherWings.

Parts

Hardware

1 x Linear Bearing Supported Slide Rail — https://www.adafruit.com/product/1861
Linear Bearing Supported Slide Rail - 15mm wide - 500mm long

---

1 x Linear Bearing Pillow Block — https://www.adafruit.com/product/1866
15mm Diameter - Wider Version

---

2 x Slotted Aluminum Extrusion — https://www.adafruit.com/product/1221
Slotted Aluminum Extrusion - 20mm x 20mm - 610mm long

---

1 x Aluminum GT2 Timing Pulley — https://www.adafruit.com/product/1253
6mm Belt - 36 Tooth - 5mm Bore

---

1 x Ball Bearing — https://www.adafruit.com/product/1178
Radial Ball Bearing 608ZZ - Set of 4

---

1 x Timing Belt GT2 — https://www.adafruit.com/product/1184
Profile - 2mm pitch - 6mm wide 1164mm long

---

1 x Swivel-Head Pan Tilt — https://www.adafruit.com/product/2464
(PTZ) Shoe Mount Adapter

---

1 x GT2 Timing Belt Torsion Spring — https://www.amazon.com/gp/product/B01E913IJK/
6mm Width Belt Pack of 10

Screws

4 x M3 Screws
M3 x .5 x 6M (for Stepper Motor)
https://www.albanycountyfasteners.com/Phillips-Pan-Head-Machine-Screw-M3-x-5-p/1066-1008.htm

---

1 x M4 Screws
Button Hex Machine Screw - M4 thread - 8mm long - pack of 50 (for various mounts)
https://www.adafruit.com/product/1160

---

1 x M4 Screws (Longer)
M4 x .7 x 18M for various mounts
https://www.albanycountyfasteners.com/Phillips-Pan-Head-Machine-Screw-M4-x-7-p/1066-1009.htm

---

1 x M4 Hex Nuts
M4 x .7 (1.95M thick, 7.0M flat)
https://www.albanycountyfasteners.com/Metric-Hex-Jam-Nuts-A2-Stainless-Steel-p/5580000.htm

---

1 x M5 Screws
M5 x .8 x 16M for the Pillow Block
https://www.albanycountyfasteners.com/Phillips-Pan-Head-Machine-Screw-M4-x-7-p/1066-1010.htm

---

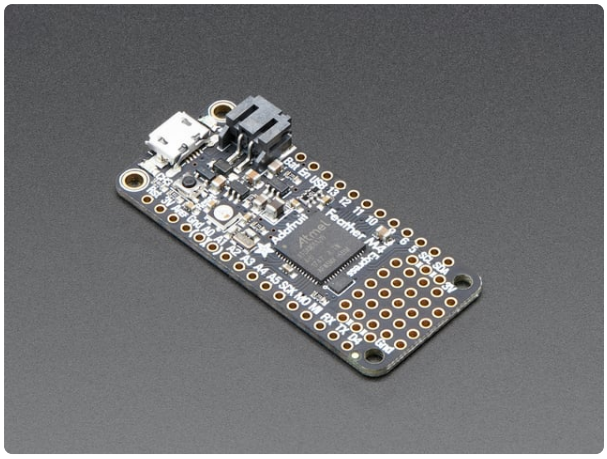
1 x T-NUT FOR 20X20 - M4 THREAD
Aluminum Extrusion Oval T-Nut for 20x20 - M4 Thread - pack of 50
https://www.adafruit.com/product/1158

---

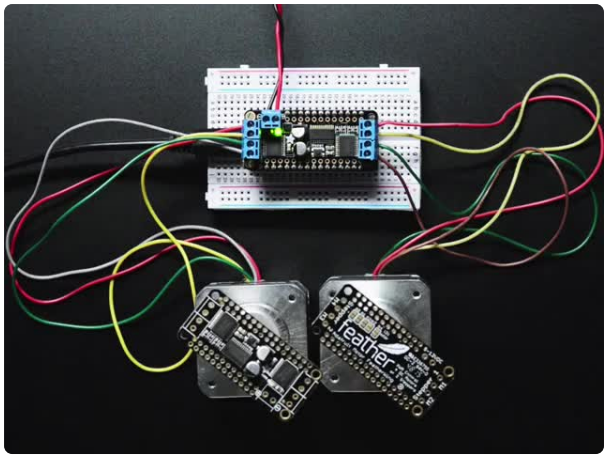
1 x Camera and Tripod
3/8" to 1/4" Adapter Screw
https://www.adafruit.com/product/2392

---

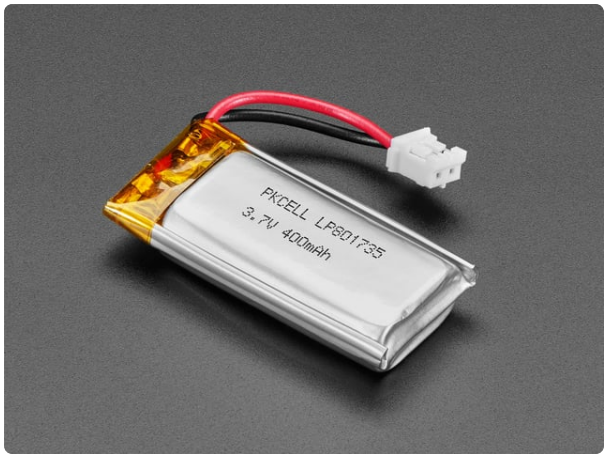
1 x 1/4" Screw with D-Ring
Cameras / Tripods / Photo / Video
https://www.adafruit.com/product/2629



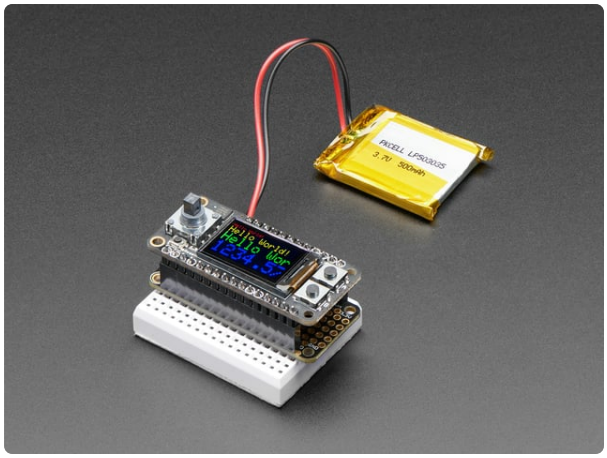
[Adafruit Feather M4 Express - Featuring ATSAMD51]
It's what you've been waiting for, the Feather M4 Express featuring ATSAMD51. This Feather is fast like a swift, smart like an owl, strong like a ox-bird (it's half ox,...
https://www.adafruit.com/product/3857



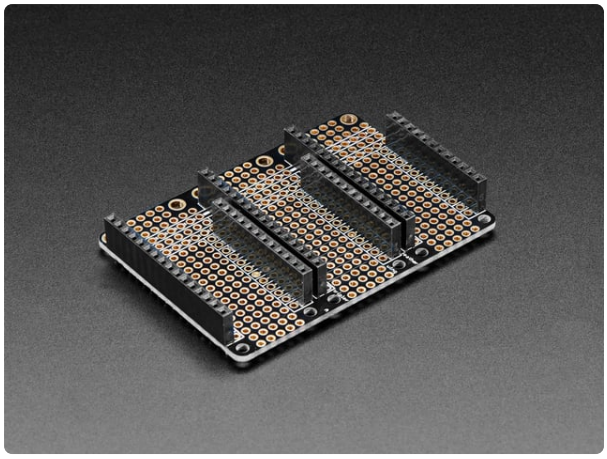
[DC Motor + Stepper FeatherWing Add-on For All Feather Boards]
A Feather board without ambition is a Feather board without FeatherWings! This is the DC Motor + Stepper FeatherWing which will let you use 2 x bi-polar...
https://www.adafruit.com/product/2927



[Lithium Ion Polymer Battery Ideal For Feathers - 3.7V 400mAh]
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...
https://www.adafruit.com/product/3898

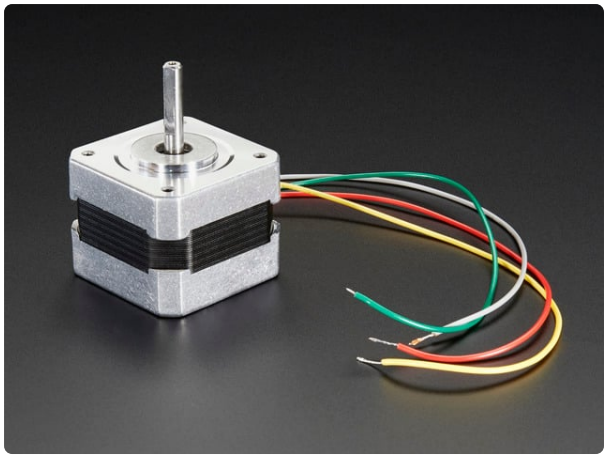


[Adafruit Mini Color TFT with Joystick FeatherWing]
Add a dazzling color display to your Feather project with this Adafruit Mini Color TFT with Joystick FeatherWing.It has so much stuff going on, we could...
https://www.adafruit.com/product/3321



FeatherWing Tripler Mini Kit - Prototyping Add-on For Feathers

This is the FeatherWing Tripler - a prototyping add-on and more for all Feather boards. This is similar to our https://www.adafruit.com/product/3417

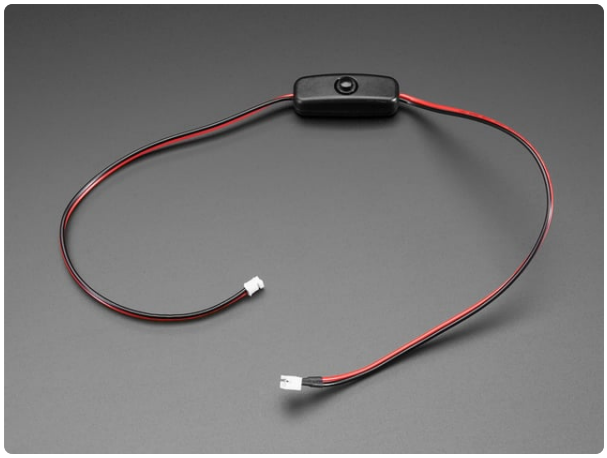


Stepper motor - NEMA-17 size - 200 steps/rev, 12V 350mA

A stepper motor to satisfy all your robotics needs! This 4-wire bipolar stepper has 1.8° per step for smooth motion and a nice holding torque. The motor was specified to have a max... https://www.adafruit.com/product/324

8 x AA battery holder with 5.5mm/2.1mm Plug and On/Off Switch

Make a portable power brick with plenty of juice! Use Alkaline AA's for a 12V 3000-4000mAh power supply, or rechargeable NiMH for 2000mAh 9.6V supply. Either one is good for running... https://www.adafruit.com/product/875



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

By popular request - we now have a way you can turn on-and-off Lithium Polymer batteries without unplugging them.This PH2 Female/Male JST 2-pin Extension... https://www.adafruit.com/product/3064

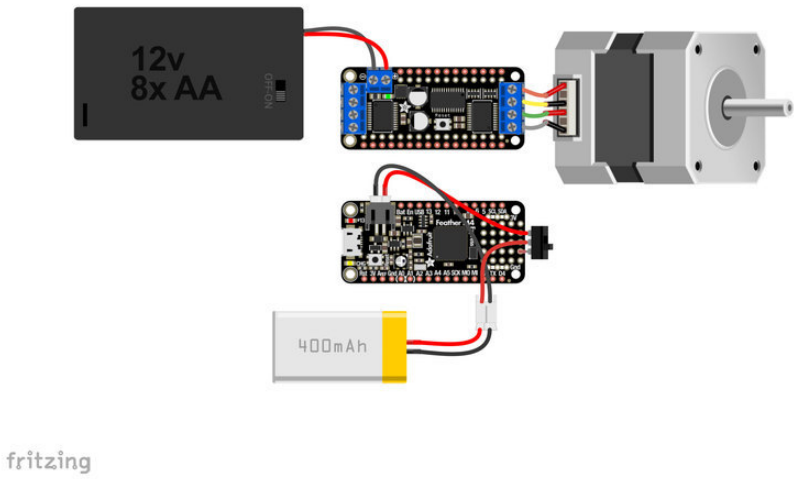
Circuit Diagram

Circuit Diagram

The diagram below provides a visual reference for wiring of the components. This diagrams was created using the software package Fritzing (https://adafru.it/oEP).

Adafruit Library for Fritzing

It is easy to use Adafruit's Fritzing parts library to create circuit diagrams for your projects. Download the library or just grab individual parts. Get the library and parts from GitHub - Adafruit Fritzing Parts (https://adafru.it/AYZ).



Feathers & Tripler Feather

The **Feather M4**, **Motor FeatherWing** and **mini TFT with joystick FeatherWing** get installed to the **Tripler FeatherWing** via Headers. This makes it easy to swap PCBs and features lots of extra pins for wiring additional components. The circuit diagram only shows the PCB that require wired connections.

Stepper Motor

The bipolar NEMA-17 stepper motor features 4-wired connections that can be connected via screw block terminals on the MotorFeatherWing. Connections can be

made to either the Motor 1 or Motor 2 ports, just make sure the connections are reflected in the code.

Power Supply

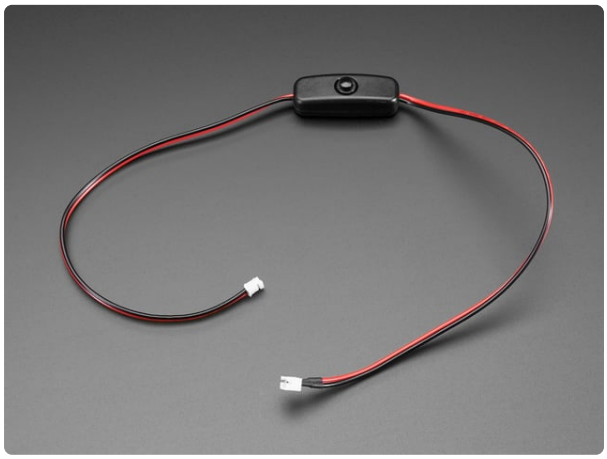
The Feather M4 is powered via 400mAh lipo battery. The Motor FeatherWing is powered separately with a 12V - 8x AA battery pack for providing enough voltage to power the stepper motor. If you'd like, the Motor FeatherWing can also be powered via a 12v wall adapter (http://adafru.it/798).

Slide Switches

A slide switch is connected to the Feather M4 via the pins on the prototyping area. To keep the Feather modular (easy to remove from the Tripler FeatherWing), a JST extension cable is wired in-line with the switch. This allows the battery to be turn off and on.

The 12v battery pack features a built-in slide switch.

If you'd like to avoid soldering a switch slide, you can use a pre-made on/of switch with JST extension cable.



JST 2-pin Extension Cable with On/Off Switch - JST PH2
By popular request - we now have a way you can turn on-and-off Lithium Polymer batteries without unplugging them.This PH2 Female/Male JST 2-pin Extension...
https://www.adafruit.com/product/3064

Software Setup

Setup Feather M4 with CircuitPython

We'll need to get our board setup so we can run CircuitPython code. Let's walk through these steps to get the latest version of CircuitPython onto your board.

The Mu Python Editor

Mu is a simple Python editor that works with Adafruit CircuitPython hardware. It's written in Python and works on Windows, MacOS, Linux and Raspberry Pi. The serial console is built right in so you get immediate feedback from your board's serial output! While you can use any text editor with your code, Mu makes it super simple.

Installing and Using the Mu Editor

https://adafru.it/ANO

Installing or upgrading CircuitPython

You should ensure you have CircuitPython 5.0 or greater on your board. Plug your board in with a known good data + power cable (not the cheesy USB cable that comes with USB power packs, they are power only). You should see a new flash drive pop up.

If the drive is **CIRCUITPY**, then open the **boot_out.txt** file to ensure the version number is 5.0 or greater.

```
Adafruit CircuitPython 5.0.0-beta.0 on 2019-11-19; Adafruit Feather M4 Express with
samd51j19
```

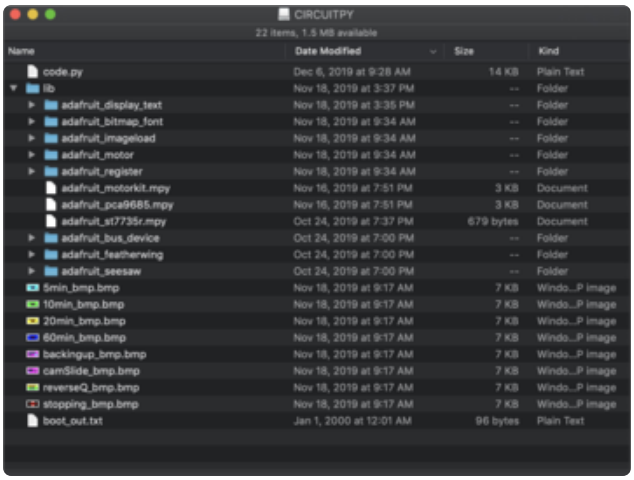
If the version is less than 5 -or- you only get a drive named **FEATHERBOOT** then follow the Feather M4 guide (https://adafru.it/CJN) on installing CircuitPython (https://adafru.it/CVs).

Download the Adafruit CircuitPython Library Bundle

In order to run the code, we'll need to download a few libraries. Libraries contain code to help interface with hardware a lot easier for us.

Use the Feather M4 page on Installing Libraries (https://adafru.it/HDo) to get the library that matches the major version of CircuitPython you are using noted above.

The green button below links to a file containing all the libraries available for CircuitPython. To run the code for this project, we need the large number of libraries in the Required Libraries list below. Unzip the library bundle and search for the libraries. Drag and drop the files into a folder named **lib** on the **CIRCUITPY** drive (create the folder if it is not already on the Feather M4).



Required Libraries

adafruit_display_text
adafruit_bitmap_font
adafruit_imageload
adafruit_motor
adafruit_register
adafruit_motorkit.mpy
adafruit_pca9685.mpy
adafruit_st7735r.mpy
adafruit_bus_device
adafruit_featherwing
adafruit_seesaw

Once we have all the files we need, a directory listing will look similar to below as far as files and directories.

Upload Code

Click on the Download: Project Zip link below to grab the project files in a zip file directly from GitHub. Place the **code.py** file and bitmap graphics files onto the **CIRCUITPY** main (root) directory. The code will run properly when all of the files have been uploaded including libraries.

Use any text editor or favorite IDE to modify the code. We suggest using Mu as noted above.

```
# SPDX-FileCopyrightText: 2019 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import displayio
import terminalio
import adafruit_imageload
from adafruit_display_text.label import Label
from adafruit_featherwing import minitft_featherwing
from adafruit_motorkit import MotorKit
from adafruit_motor import stepper

# setup stepper motor
kit = MotorKit()

# setup minitft featherwing
minitft = minitft_featherwing.MiniTFTFeatherWing()

# setup bitmap file locations
five_minBMP = "/5min_bmp.bmp"
ten_minBMP = "/10min_bmp.bmp"
twenty_minBMP = "/20min_bmp.bmp"
hourBMP = "/60min_bmp.bmp"
runningBMP = "/camSlide_bmp.bmp"
reverseqBMP = "/reverseQ_bmp.bmp"
```

```
backingUpBMP = "/backingup_bmp.bmp"
stopBMP = "/stopping_bmp.bmp"

# variables for state machines in loop
mode = 0
onOff = 0
pause = 0
stop = 0
z = 0

# image groups
five_minGroup = displayio.Group()
ten_minGroup = displayio.Group()
twenty_minGroup = displayio.Group()
hourGroup = displayio.Group()
reverseqGroup = displayio.Group()
backingUpGroup = displayio.Group()
stopGroup = displayio.Group()
progBarGroup = displayio.Group()

# bitmap setup for all of the menu screens
five_minBG, five_minPal = adafruit_imageload.load(
    five_minBMP, bitmap=displayio.Bitmap, palette=displayio.Palette
)
five_minDis = displayio.TileGrid(five_minBG, pixel_shader=five_minPal)
ten_minBG, ten_minPal = adafruit_imageload.load(
    ten_minBMP, bitmap=displayio.Bitmap, palette=displayio.Palette
)
ten_minDis = displayio.TileGrid(ten_minBG, pixel_shader=ten_minPal)
twenty_minBG, twenty_minPal = adafruit_imageload.load(
    twenty_minBMP, bitmap=displayio.Bitmap, palette=displayio.Palette
)
twenty_minDis = displayio.TileGrid(twenty_minBG, pixel_shader=twenty_minPal)
hourBG, hourPal = adafruit_imageload.load(
    hourBMP, bitmap=displayio.Bitmap, palette=displayio.Palette
)
hourDis = displayio.TileGrid(hourBG, pixel_shader=hourPal)
runningBG, runningPal = adafruit_imageload.load(
    runningBMP, bitmap=displayio.Bitmap, palette=displayio.Palette
)
runningDis = displayio.TileGrid(runningBG, pixel_shader=runningPal)
reverseqBG, reverseqPal = adafruit_imageload.load(
    reverseqBMP, bitmap=displayio.Bitmap, palette=displayio.Palette
)
reverseqDis = displayio.TileGrid(reverseqBG, pixel_shader=reverseqPal)
backingUpBG, backingUpPal = adafruit_imageload.load(
    backingUpBMP, bitmap=displayio.Bitmap, palette=displayio.Palette
)
backingUpDis = displayio.TileGrid(backingUpBG, pixel_shader=backingUpPal)
stopBG, stopPal = adafruit_imageload.load(
    stopBMP, bitmap=displayio.Bitmap, palette=displayio.Palette
)
stopDis = displayio.TileGrid(stopBG, pixel_shader=stopPal)

# setup for timer display when camera is sliding
text_area = Label(terminalio.FONT, text="      ")
text_area.x = 55
text_area.y = 65

# adding the bitmaps to the image groups so they can be displayed
five_minGroup.append(five_minDis)
ten_minGroup.append(ten_minDis)
twenty_minGroup.append(twenty_minDis)
hourGroup.append(hourDis)
progBarGroup.append(runningDis)
progBarGroup.append(text_area)
reverseqGroup.append(reverseqDis)
backingUpGroup.append(backingUpDis)
stopGroup.append(stopDis)
```

```
# setting button states on minitft featherwing to None
down_state = None
up_state = None
a_state = None
b_state = None
select_state = None

# arrays to match up with the different slide speeds
# graphics menu array
graphics = [five_minGroup, ten_minGroup, twenty_minGroup, hourGroup]
# delay for the stepper motor
speed = [0.0154, 0.034, 0.0688, 0.2062]
# time duration for the camera slide
slide_duration = [300, 600, 1200, 3600]
# beginning timer display
slide_begin = ["5:00", "10:00", "20:00", "60:00"]
# stepper motor steps that corresponds with the timer display
# fmt: off
slide_checkin = [  860,  1720,  2580,  3440,  4300,  5160,
                  6020,  6880,  7740,  8600,  9460, 10320,
                 11180, 12040, 12900, 13760, 14620, 15480,
                 16340, 17195]
# fmt: on
# variable that counts up through the slide_checkin array
check = 0

# start time
begin = time.monotonic()
print(begin)
# when feather is powered up it shows the initial graphic splash
minitft.display.root_group = graphics[mode]

while True:
    # setup minitft featherwing buttons
    buttons = minitft.buttons
    # define the buttons' state changes
    if not buttons.down and down_state is None:
        down_state = "pressed"
    if not buttons.up and up_state is None:
        up_state = "pressed"
    if not buttons.select and select_state is None:
        select_state = "pressed"
    if not buttons.a and a_state is None:
        a_state = "pressed"
    if not buttons.b and b_state is None:
        b_state = "pressed"
    # scroll down to change slide duration and graphic
    if buttons.down and down_state == "pressed":
        # blocks the button if the slider is sliding or
        # in an in-between state
        if pause == 1 or onOff == 1:
            mode = mode
            down_state = None
        else:
            mode += 1
            down_state = None
            if mode > 3:
                mode = 0
            print("Mode:,", mode)
            minitft.display.root_group = graphics[mode]
    # scroll up to change slide duration and graphic
    if buttons.up and up_state == "pressed":
        # blocks the button if the slider is sliding or
        # in an in-between state
        if pause == 1 or onOff == 1:
            mode = mode
            up_state = None
        else:
```

```
            mode -= 1
            up_state = None
            if mode < 0:
                mode = 3
            print("Mode: ", mode)
            minitft.display.root_group = graphics[mode]
    # workaround so that the menu graphics show after a slide is finished
    if mode == mode and pause == 0 and onOff == 0:
        minitft.display.root_group = graphics[mode]
    # starts slide
    if buttons.select and select_state == "pressed" or z == 2:
        # blocks the button if the slider is sliding or
        # in an in-between state
        if pause == 1 or onOff == 1:
            # print("null")
            select_state = None
        else:
            # shows the slider is sliding graphic
            minitft.display.root_group = progBarGroup
            # gets time of button press
            press = time.monotonic()
            print(press)
            # displays initial timer
            text_area.text = slide_begin[mode]
            # resets button
            select_state = None
            # changes onOff state
            onOff += 1
            # changes z state
            z = 0
            if onOff > 1:
                onOff = 0
            # number of steps for the length of the aluminum extrusions
            for i in range(17200):
                # for loop start time
                start = time.monotonic()
                # gets actual duration time
                real_time = start - press
                # creates a countdown from the slide's length
                end = slide_duration[mode] - real_time
                # /60 since time is in seconds
                mins_remaining = end / 60
                if mins_remaining < 0:
                    mins_remaining += 60
                # gets second(s) count
                total_sec_remaining = mins_remaining * 60
                # formats to clock time
                mins_remaining, total_sec_remaining = divmod(end, 60)
                # microstep for the stepper
                kit.stepper1.onestep(style=stepper.MICROSTEP)
                # delay determines speed of the slide
                time.sleep(speed[mode])
                if i == slide_checkin[check]:
                    # check-in for time remaining based on motor steps
                    print("0%d:%d" % (mins_remaining, total_sec_remaining))
                    print(check)
                    if total_sec_remaining < 10:
                        text_area.text = "%d:0%d" % (
                            mins_remaining,
                            total_sec_remaining,
                        )
                    else:
                        text_area.text = "%d:%d" % (mins_remaining,
total_sec_remaining)
                    check = check + 1
                if check > 19:
                    check = 0
                if end < 10:
                    # displays the stopping graphic for the last 10 secs.
```

```
                    minitft.display.root_group = stopGroup
            # changes states after slide has completed
            kit.stepper1.release()
            pause = 1
            onOff = 0
            stop = 1
            check = 0
            # delay for safety
            time.sleep(2)
            # shows choice menu
            minitft.display.root_group = reverseqGroup
    # b is defined to stop the slider
    # only active if the slider is in the 'stopped' state
    if buttons.b and b_state == "pressed" and stop == 1:
        # z defines location of the camera on the slider
        # 0 means that it is opposite the motor
        if z == 0:
            b_state = None
            time.sleep(1)
            minitft.display.root_group = backingUpGroup
            # delay for safety
            time.sleep(2)
            # brings camera back to 'home' at double speed
            for i in range(1145):
                kit.stepper1.onestep(direction=stepper.BACKWARD,
style=stepper.DOUBLE)
            time.sleep(1)
            kit.stepper1.release()
            # changes states
            pause = 0
            stop = 0
        # 1 means that the camera is next to the motor
        if z == 1:
            b_state = None
            time.sleep(2)
            # changes states
            pause = 0
            stop = 0
            z = 0
    # a is defined to slide in reverse of the prev. slide
    # only active if the slider is in the 'stopped' state
    if buttons.a and a_state == "pressed" and stop == 1:
        # z defines location of the camera on the slider
        # 1 means that the camera is next to the motor
        if z == 1:
            a_state = None
            time.sleep(2)
            stop = 0
            pause = 0
            # 2 allows the 'regular' slide loop to run
            # as if the 'select' button has been pressed
            z = 2
        # 0 means that the camera is opposite the motor
        if z == 0:
            a_state = None
            # same script as the 'regular' slide loop
            time.sleep(2)
            minitft.display.root_group = progBarGroup
            press = time.monotonic()
            print(press)
            text_area.text = slide_begin[mode]
            onOff += 1
            pause = 0
            stop = 0
            if onOff > 1:
                onOff = 0
            for i in range(17200):
                start = time.monotonic()
                real_time = start - press
```

```
                end = slide_duration[mode] - real_time
                mins_remaining = end / 60
                if mins_remaining < 0:
                    mins_remaining += 60
                total_sec_remaining = mins_remaining * 60
                mins_remaining, total_sec_remaining = divmod(end, 60)
                # only difference is that the motor is stepping backwards
                kit.stepper1.onestep(
                    direction=stepper.BACKWARD, style=stepper.MICROSTEP
                )
                time.sleep(speed[mode])
                if i == slide_checkin[check]:
                    print("0%d:%d" % (mins_remaining, total_sec_remaining))
                    if total_sec_remaining < 10:
                        text_area.text = "%d:0%d" % (
                            mins_remaining,
                            total_sec_remaining,
                        )
                    else:
                        text_area.text = "%d:%d" % (mins_remaining,
total_sec_remaining)
                    check = check + 1
                if check > 19:
                    check = 0
                if end < 10:
                    minitft.display.root_group = stopGroup
            # state changes
            kit.stepper1.release()
            pause = 1
            onOff = 0
            stop = 1
            z = 1
            check = 0
            time.sleep(2)
            minitft.display.root_group = reverseqGroup
```

Double Check

See the directory listing above and double check that you have all the files listed to make this project function. If any are missing or in an incorrect directory, move them so they're in the right places.

Code Walkthrough

CircuitPython Code Walkthrough

There are a few (literally) moving parts that can make it seem a little overwhelming. We're going to go over what the different sections are doing so that it will make a little more sense.

First we'll import the libraries:

```
import time
import displayio
```

```
import terminalio
import adafruit_imageload
from adafruit_display_text.label import Label
from adafruit_featherwing import minitft_featherwing
from adafruit_motorkit import MotorKit
from adafruit_motor import stepper
```

Then we setup the DC Motor FeatherWing and MiniTFT FeatherWing. The MiniTFT FeatherWing helper has built-in functionality that sets up our ability to use TFT display without any additional lines of code! To access the TFT later, we'll call `minitft.display`

```
#setup stepper motor
kit = MotorKit()

#setup minitft featherwing
minitft = minitft_featherwing.MiniTFTFeatherWing()
```

These are our bitmap file locations. By doing this, we're telling the code where to look for what file we're referring to. All of the bitmaps are available on GitHub in the same folder as the code and can be dropped directly onto your Feather M4 to load properly.

```
#setup bitmap file locations
five_minBMP = "/5min_bmp.bmp"
ten_minBMP = "/10min_bmp.bmp"
twenty_minBMP = "/20min_bmp.bmp"
hourBMP = "/60min_bmp.bmp"
runningBMP = "/camSlide_bmp.bmp"
reverseqBMP = "/reverseQ_bmp.bmp"
backingUpBMP = "/backingup_bmp.bmp"
stopBMP = "/stopping_bmp.bmp"
```

All of these are variables for the state machines that we'll use later in the loop. To start, they'll be set to zero.

```
#variables for state machines in loop
mode = 0
onOff = 0
pause = 0
stop = 0
z = 0
```

Check out this Hacking Ikea Lamps Learn Guide by Kattni for more info on state machines

https://adafru.it/BIU

The `displayio` library utilizes image groups to hold your different images that you want to display with your project. Here groups are setup for all of the different bitmaps.

For more info on displayio, check out this Learn guide

https://adafru.it/HCB

```
#image groups
five_minGroup = displayio.Group()
ten_minGroup = displayio.Group()
twenty_minGroup = displayio.Group()
hourGroup = displayio.Group()
reverseqGroup = displayio.Group()
backingUpGroup = displayio.Group()
stopGroup = displayio.Group()
progBarGroup = displayio.Group()
```

The setup continues here for the images, with this portion setting up the files to be loaded into the program and letting the program know that they are bitmaps.

```
#bitmap setup for all of the menu screens
five_minBG, five_minPal = adafruit_imageload.load(five_minBMP,
                                                  bitmap=displayio.Bitmap,
                                                  palette=displayio.Palette)
five_minDis = displayio.TileGrid(five_minBG, pixel_shader=five_minPal)
ten_minBG, ten_minPal = adafruit_imageload.load(ten_minBMP,
                                                bitmap=displayio.Bitmap,
                                                palette=displayio.Palette)
ten_minDis = displayio.TileGrid(ten_minBG, pixel_shader=ten_minPal)
twenty_minBG, twenty_minPal = adafruit_imageload.load(twenty_minBMP,
                                                      bitmap=displayio.Bitmap,
                                                      palette=displayio.Palette)
twenty_minDis = displayio.TileGrid(twenty_minBG, pixel_shader=twenty_minPal)
hourBG, hourPal = adafruit_imageload.load(hourBMP,
                                          bitmap=displayio.Bitmap,
                                          palette=displayio.Palette)
hourDis = displayio.TileGrid(hourBG, pixel_shader=hourPal)
runningBG, runningPal = adafruit_imageload.load(runningBMP,
                                                bitmap=displayio.Bitmap,
                                                palette=displayio.Palette)
runningDis = displayio.TileGrid(runningBG, pixel_shader=runningPal)
reverseqBG, reverseqPal = adafruit_imageload.load(reverseqBMP,
                                                  bitmap=displayio.Bitmap,
                                                  palette=displayio.Palette)
reverseqDis = displayio.TileGrid(reverseqBG, pixel_shader=reverseqPal)
backingUpBG, backingUpPal = adafruit_imageload.load(backingUpBMP,
                                                    bitmap=displayio.Bitmap,
                                                    palette=displayio.Palette)
backingUpDis = displayio.TileGrid(backingUpBG, pixel_shader=backingUpPal)
stopBG, stopPal = adafruit_imageload.load(stopBMP,
                                          bitmap=displayio.Bitmap,
                                          palette=displayio.Palette)
stopDis = displayio.TileGrid(stopBG, pixel_shader=stopPal)
```

While the slider is sliding, the time remaining will be displayed and will update. This portion sets up the area on the TFT screen where that will be shown.

```
#setup for timer display when camera is sliding
text_area = Label(terminalio.FONT, text='      ')
text_area.x = 55
text_area.y = 65
```

This is the last of the `displayio` setup for our bitmaps. All of the setup that we just did in the above lines are now being added to the groups that we created. Notice that the `text_area` for the timer display is added to the `progBarGroup` along with the `runningDis` that holds the "in progress" bitmap so that they'll be shown at the same time on the screen.

```
# adding the bitmaps to the image groups so they can be displayed
five_minGroup.append(five_minDis)
ten_minGroup.append(ten_minDis)
twenty_minGroup.append(twenty_minDis)
hourGroup.append(hourDis)
progBarGroup.append(runningDis)
progBarGroup.append(text_area)
reverseqGroup.append(reverseqDis)
backingUpGroup.append(backingUpDis)
stopGroup.append(stopDis)
```

We'll be using state machines for the buttons on the MiniTFT FeatherWing too. Here the default states for all of the buttons are set to `None` to later be referenced in the loop.

```
# setting button states on minitft featherwing to None
down_state = None
up_state = None
a_state = None
b_state = None
select_state = None
```

Here are some arrays that hold the different settings for the different slider speed options. In the loop, they'll match up with the `mode` state. We have the bitmap that will show for the selection of the mode, the delay used in the `for` loop for the stepper motor, the duration of the slide in seconds and the beginning timer display when a slide starts.

```
# arrays to match up with the different slide speeds
# graphics menu array
graphics = [five_minGroup, ten_minGroup, twenty_minGroup, hourGroup]
# delay for the stepper motor
speed = [0.0154, 0.034, 0.0688, 0.2062]
# time duration for the camera slide
slide_duration = [300, 600, 1200, 3600]
# beginning timer display
slide_begin = ["5:00", "10:00", "20:00", "60:00"]
```

These are increments used to display the time remaining during a slide. Basically when the stepper reaches a number of microsteps that matches one of the twenty options below, it will grab the time remaining and update the display. The check is another state that will count up through the `slide_checkin` array in the loop.

```
# stepper motor steps that corresponds with the timer display
slide_checkin = [860, 1720, 2580, 3440, 4300, 5160,
                 6020, 6880, 7740, 8600, 9460, 10320,
```

```
                11180, 12040, 12900, 13760, 14620, 15480,
                16340, 17195]
#variable that counts up through the slide_checkin array
check = 0
```

Here are some final flight check items before the loop begins. We have a call for `time.monotonic()` which will give us the beginning time of the program and then the TFT displays the first bitmap, which on startup is the five minute slide option in the main menu.

```
# start time
begin = time.monotonic()
print(begin)
# when feather is powered up it shows the initial graphic splash
minitft.display.root_group = graphics[mode]
```

Here comes the loop! First, we setup the buttons on the MiniTFT FeatherWing.

```
while True:
    #  setup minitft featherwing buttons
    buttons = minitft.buttons
```

Here the rest of the button state machine is defined so that when a button is pressed it changes the state from `None`.

```
# define the buttons' state changes

if not buttons.down and down_state is None:
    down_state = "pressed"
if not buttons.up and up_state is None:
    up_state = "pressed"
if not buttons.select and select_state is None:
    select_state = "pressed"
if not buttons.a and a_state is None:
    a_state = "pressed"
if not buttons.b and b_state is None:
    b_state = "pressed"
```

We finally start to get into some action here with all of our prep paying off. The up and down buttons on the MiniTFT FeatherWing are setup to that when they're pressed they'll scroll through the different bitmap graphics that display the possible slider speeds available. When this is done the mode state is updated. There's also some logic built-in with the `pause` and `onOff` states that if the slider is either actively sliding or waiting to either reverse or go back to the main menu then the up and down buttons are blocked from triggering anything. Otherwise our states could get very confused.

```
# scroll down to change slide duration and graphic
        if buttons.down and down_state == "pressed":
            # blocks the button if the slider is sliding or
            # in an inbetween state
            if pause == 1 or onOff == 1:
```

```
                mode = mode
                down_state = None
            else:
                mode += 1
                down_state = None
                if mode &gt; 3:
                    mode = 0
                print("Mode:,", mode)
                minitft.display.root_group = graphics[mode]

        # scroll up to change slide duration and graphic
        if buttons.up and up_state == "pressed":
            # blocks the button if the slider is sliding or
            # in an inbetween state
            if pause == 1 or onOff == 1:
                mode = mode
                up_state = None
            else:
                mode -= 1
                up_state = None
                if mode &lt; 0:
                    mode = 3
                print("Mode: ", mode)
                minitft.display.root_group = graphics[mode]
```

This little snippet ensures that the slider options pop back up after you return to the main menu.

```
#workaround so that the menu graphics show after a slide is finished
        if mode == mode and pause == 0 and onOff == 0:
            minitft.display.root_group = graphics[mode]
```

Here is it, the moment you've been waiting for: the actual camera slide. A slide is initiated with the select button or if the state of `z` is `2`, but more on `z` later. First there's some built-in logic just like we have for the up and down buttons that if the slider is paused or actively sliding then nothing will happen.

```
# starts slide
if buttons.select and select_state == "pressed" or z == 2:
  # blocks the button if the slider is sliding or
  # in an inbetween state
  if pause == 1 or onOff == 1:
    #print("null")
    select_state = None
```

But, if all the states are properly aligned, then we'll begin a camera slide in the selected time limit. First, the graphic changes to show the `progBarGroup`, which if you remember contains the timer text as well. Then `time.monotonic()` is called and stored in press. The initial time from our array is shown, the button is reset, `onOff` is set to `1` to indicate that the slider is sliding and `z` is set to `0` (again, more on `z` later).

```
else:
  # shows the slider is sliding graphic
  minitft.display.root_group = progBarGroup
  # gets time of button press
```

```
  press = time.monotonic()
  print(press)
  # displays initial timer
  text_area.text = slide_begin[mode]
  # resets button
  select_state = None
  # changes onOff state
  onOff += 1
  # changes z state
  z = 0
  if onOff &gt; 1:
    onOff = 0
```

Now comes the stepper motor and most mathematical portion of the code. We begin with a for loop that has the range of microsteps required to run the length of the slider. `time.monotonic()` is called again and this time stored as `start`. Why call it again so soon? There is a slight delay between when the select button is pressed and when the stepper motor begins stepping, so in order to make sure everything is on the same page time-wise we're going to create a `real_time` which will equal `start` minus `press`, which should equal zero initially and then hold the actual time that the slider has been sliding. This means that we can use that to keep track of how much time has passed since the actual slide begin.

This is used further to display the time on the TFT. `end` is used to hold the time remaining, which is gathered by taking `real_time` and subtracting it from the total time of the slide duration. We then sort it into minutes and seconds to get our clock style display, which comes a little later.

With all this time talk though we can't forget the star of the show: the stepper motor. The stepper is using microsteps for the most accurate stepping and will step 17,200 microsteps to reach the end of the slide rail. The `time.sleep()` delay affects how fast or slow this is accomplished. The bigger the delay, the slower it slides. A lot of testing was done to ensure an accurate slide duration to match up with 5, 10, 20 and 60 minutes.

```
# number of steps for the length of the aluminum extrusions
for i in range(17200):
  # for loop start time
  start = time.monotonic()
  # gets actual duration time
  real_time = start - press
  # creates a countdown from the slide's length
  end = slide_duration[mode] - real_time
  # /60 since time is in seconds
  mins_remaining = end / 60
  if mins_remaining &lt; 0:
    mins_remaining += 60
    # gets second(s) count
    total_sec_remaining = mins_remaining * 60
    # formats to clock time
    mins_remaining, total_sec_remaining = divmod(end, 60)
    # microstep for the stepper
    kit.stepper1.onestep(style=stepper.MICROSTEP)
```

```
    # delay determines speed of the slide
    time.sleep(speed[mode])
```

This portion of the code is for the time remaining timer that is displayed on the TFT. If you remember earlier in the code we setup an array that divides the 17,200 steps by 20. Here when the variable `i` matches one of those numbers in the array, it triggers the displayed text on the TFT to update. It also increases the `check` state by one so that it can get ready to wait for the next step check-in.

```
if i == slide_checkin[check]:
  # check-in for time remaining based on motor steps
  print("0%d:%d" %
        (mins_remaining, total_sec_remaining))
  print(check)
  if total_sec_remaining &lt; 10:
    text_area.text = "%d:0%d" % (mins_remaining, total_sec_remaining)
    else:
      text_area.text = "%d:%d" % (mins_remaining, total_sec_remaining)
      check = check + 1
      if check &gt; 19:
        check = 0
```

All slides must come to an end and this final bit helps to make it a smooth one. First the graphic on the TFT is changed once there's ten seconds left to the "stopping" graphic. After the `for` loop for the stepper ends, the stepper is put into the `release` state and then our software states are updated. `pause` is set to `1` to indicate that the slider is in an in-between state, `onOff` is set to `0` since the slider is stopped, `stop` is set to `1` since we're stopped and `check` is set to `0` to reset for the next slide. There is then a 2 second delay as a safety measure and the TFT is updated again to show our next two options: reverse the slide or return to main menu.

```
if end &lt; 10:
# displays the stopping graphic for the last 10 secs.
    minitft.display.root_group = stopGroup
# changes states after slide has completed
kit.stepper1.release()
pause = 1
onOff = 0
stop = 1
check = 0
# delay for safety
time.sleep(2)
# shows choice menu
minitft.display.root_group = reverseqGroup
```

The choices presented on the TFT are selected with either the A or B button. First, we'll take a look at the B button which corresponds with the Main Menu option.

Here is the B button is only activated when pressed if the state of `stop` is `1`, which it is after a slide has ended. Inside this `if` statement are two other `if` statements tied to more state machine logic, this time using `z`. The state of `z` tracks where the camera is on the slider when it is not sliding; either "home" in the default position or

at the opposite end, like it would be after running one slide. The default state of `z` is `0` and it hasn't changed yet in our code, so `0` will mean that the camera has completed a slide and is not in the home position.

When the camera is not in the home position and the B button is pressed, the camera backs up quickly back to home using double steps on the stepper motor, which is much faster than the microsteps we've been using so far. While this is happening, the TFT's graphic is changed to show the Backing Up graphic. After the back up has completed, the states of `pause` and `stop` are reset to `0` so that the main menu options are displayed again and can be selected to start a new slide.

```
# b is defined to return to the menu
# only active if the slider is in the 'stopped' state
if buttons.b and b_state == "pressed" and stop == 1:
  # z defines location of the camera on the slider
  # 0 means that it is opposite the motor
  if z == 0:
    b_state = None
    time.sleep(1)
    minitft.display.root_group = backingUpGroup
    # delay for safety
    time.sleep(2)
    # brings camera back to 'home' at double speed
    for i in range(1145):
      kit.stepper1.onestep(direction=stepper.BACKWARD, style=stepper.DOUBLE)
      time.sleep(1)
      kit.stepper1.release()
      #changes states
      pause = 0
      stop = 0
```

If `z`'s state is `1`, that means that the camera is in the home position. You'll see shortly when the state of `z` is changed during a reverse slide. If the camera is in the home position and you select that you want to return to the main menu the stepper doesn't step at all and really the only thing that changes are that the states of `pause`, `stop` and `z` are all reset to `0` to prep for the next slide that you initiate via the main menu.

```
# 1 means that the camera is next to the motor
if z == 1:
  b_state = None
  time.sleep(2)
  # changes states
  pause = 0
  stop = 0
  z = 0
```

But what if instead of returning to the main menu you want to have more slider fun and slide back in reverse? Well you'll press the A button to initiate that reverse slide mode. Much like with the B button, what happens when you press A is determined by the state of `z`, or the position of the camera on the slider. You can see in our first part of the code that is controlled by the A button, if `z` is equal to `1`, or in the home position, then the states of `stop` and `pause` are set to `0` and `z` is set to `2`. If you

remember in our original portion of the code that steps the stepper, the `if` statement that starts the whole thing ends with `or if z == 2`. By having this in place, we can reuse that piece of code to work with our reverse option with the slider.

```
# a is defined to slide in reverse of the prev. slide
# only active if the slider is in the 'stopped' state
if buttons.a and a_state == "pressed" and stop == 1:
  # z defines location of the camera on the slider
  # 1 means that the camera is next to the motor
  if z == 1:
    a_state = None
    time.sleep(2)
    stop = 0
    pause = 0
    # 2 allows the 'regular' slide loop to run
    # as if the 'select' button has been pressed
    z = 2
```

But if A is pressed and `z` is set to `0`, or not at home, then the portion below runs. It is basically a copy of the original earlier piece of code but with a very important twist: the stepper steps in **reverse** but with all the same logic built-in along with the time options as well to display the timer on the TFT. The only other difference is that at the end when the slide is stopping, the state of `z` is set to `1`, to show that the camera is in the home position and the state logic will continue to work properly.

```
#0 means that the camera is opposite the motor
if z == 0:
  a_state = None
  # same script as the 'regular' slide loop
  time.sleep(2)
  minitft.display.root_group = progBarGroup
  press = time.monotonic()
  print(press)
  text_area.text = slide_begin[mode]
  onOff += 1
  pause = 0
  stop = 0
  if onOff &gt; 1:
    onOff = 0

    for i in range(17200):
      start = time.monotonic()
      real_time = start - press
      end = slide_duration[mode] - real_time
      mins_remaining = end / 60
      if mins_remaining &lt; 0:
        mins_remaining += 60
        total_sec_remaining = mins_remaining * 60
        mins_remaining, total_sec_remaining = divmod(end, 60)
        # only difference is that the motor is stepping backwards
        kit.stepper1.onestep(direction=stepper.BACKWARD, style=stepper.MICROSTEP)
        time.sleep(speed[mode])
        if i == slide_checkin[check]:
          print("0%d:%d" %
                (mins_remaining, total_sec_remaining))
          if total_sec_remaining &lt; 10:
            text_area.text = "%d:0%d" % (mins_remaining, total_sec_remaining)
            else:
              text_area.text = "%d:%d" % (mins_remaining, total_sec_remaining)
              check = check + 1
              if check &gt; 19:
```

```
                check = 0
                if end &lt; 10:
                  minitft.display.root_group = stopGroup
                  #state changes
                  kit.stepper1.release()
                  pause = 1
                  onOff = 0
                  stop = 1
                  z = 1
                  check = 0
                  time.sleep(2)
                  minitft.display.root_group = reverseqGroup
```

And that is the CircuitPython code! It's a bit long, but hopefully this walk-through helped things make sense and will also be helpful when writing your own code for future projects.

Usage

Supported Cameras

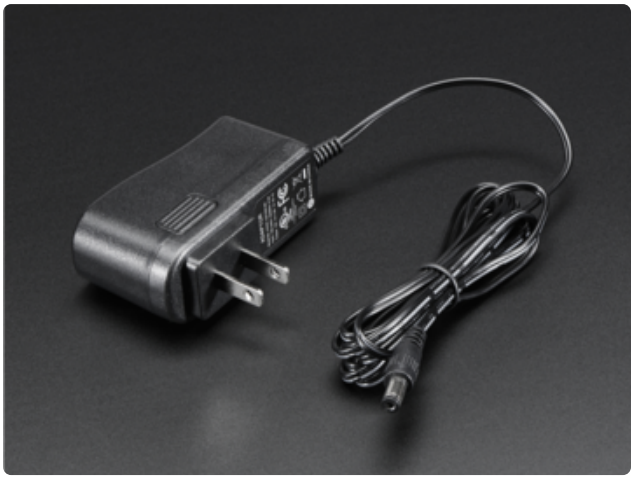
The slider works best with cameras that weigh under 15lb (6.8kg). We've tested mobile phones, action cameras and lightweight micro four cinema cameras.

Camera Support

A mini ball head tripod mount (https://adafru.it/HDY) can help support a small camera on the platform. For camera's that are around 15lb (6.8kg) a medium size ball head (https://adafru.it/HDZ) offers sturdier support. For small action cameras, a smaller tripod head (http://adafru.it/2464) can be used.

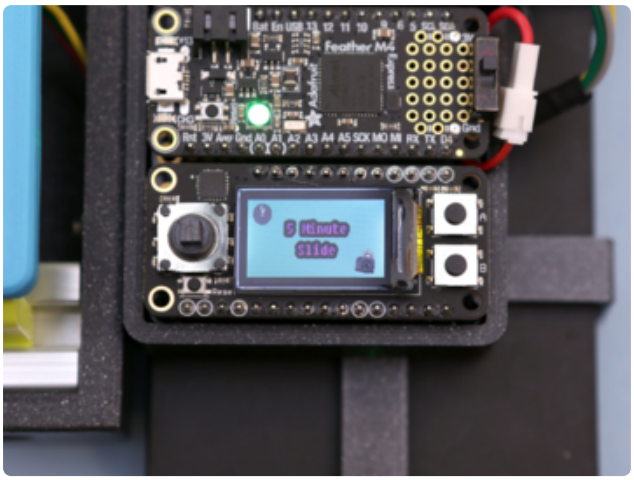
Battery Power

Turn on the Feather M4 using the slide switch. Use the built-in switch on the 12V battery pack to turn it on.



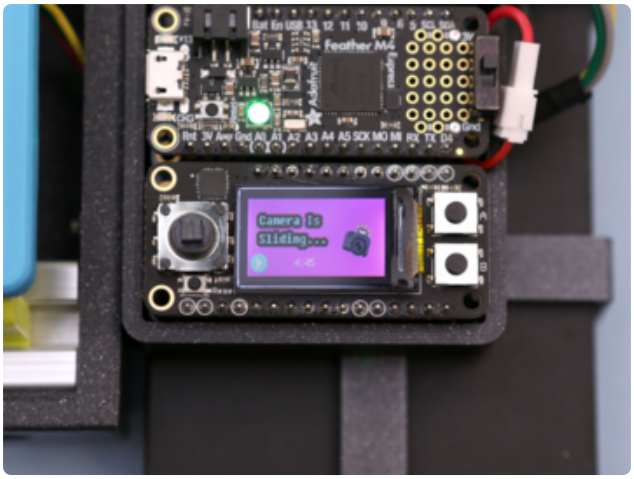
Power Supply

Plug-in the Feather M4 to a computer with USB port to power it on. Connect a 12V power adapter (http://adafru.it/798) to the motor power port on the Motor FeatherWing.



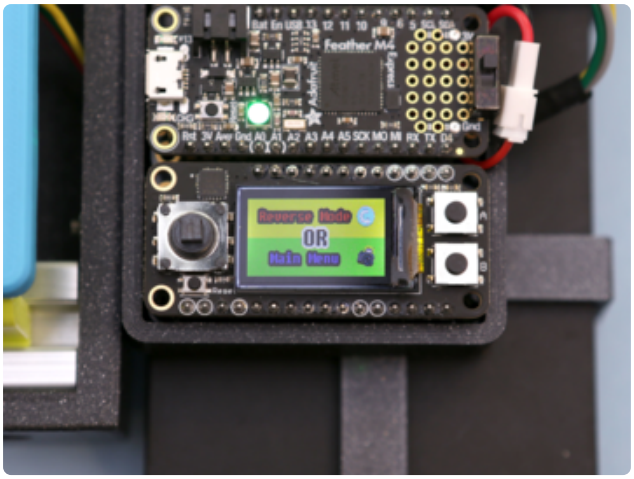
How to Start Sliding

Each of the four options are displayed on screen. The 5-min option appears first by default. Use the joystick to press up and down to cycle through the options. Make sure the camera platform is positioned on the starting point (close to the idler, away from the motor). Click to select the settings and start the stepper motor.



Camera Is Sliding

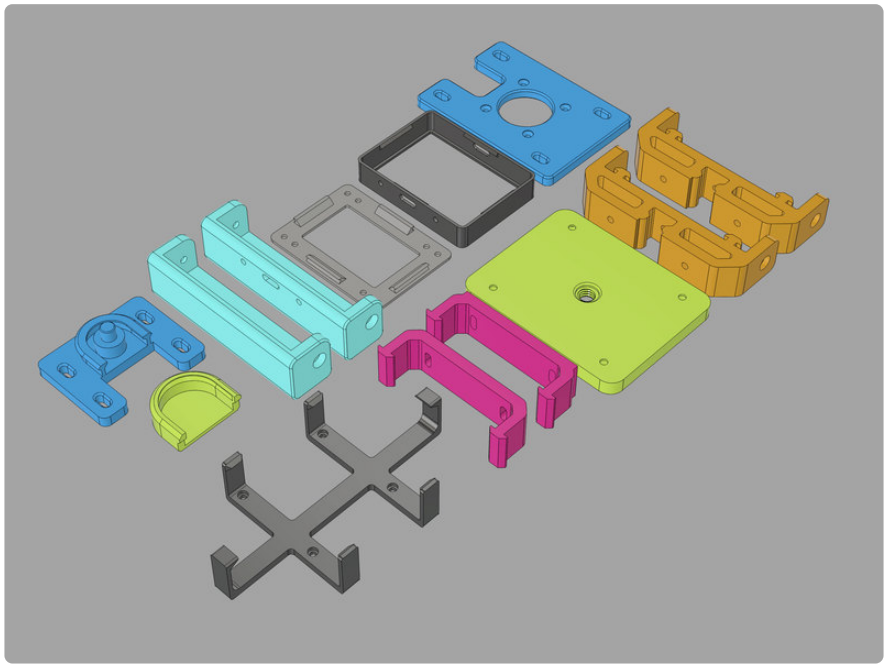
While the motor steps along, "camera is sliding" appears on screen with the time remaining. Currently, there is no pause or stop function. If the slider needs to be stopped, cutting the power is the easiest way to do so.



Reverse or Main Menu

At the end of a slide, two options are present on screen. Use **A** or **B** button to selection **Reverse mode** or **Main Menu**. Reverse mode will start the stepper motor with the same time setting, but in reverse. Main menu will back up the camera platform and display the time options again after it's been rewound.

3D Printing



3D Printed Parts

The parts for this project are designed to be 3D printed with FDM based machines. STL files are oriented to print "as is". Parts require tight tolerances that might need adjustments to the slice settings. Reference the suggested settings below. Parts do not require any support material.

CAD Files

The parts can further be separated into small pieces for fitting on printers with smaller build volumes. Note: a STEP file is included for other 3D surface modeling programs such as Solidworks, Maya and Rhino.

Download from Thingiverse

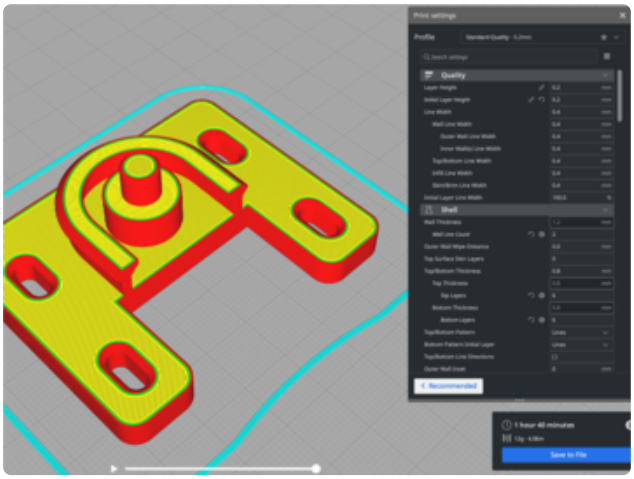
https://adafru.it/HDp

Download from PrusaPrinters

https://adafru.it/HDq

Download from AutoDesk A360

https://adafru.it/B2I



Slice Parts

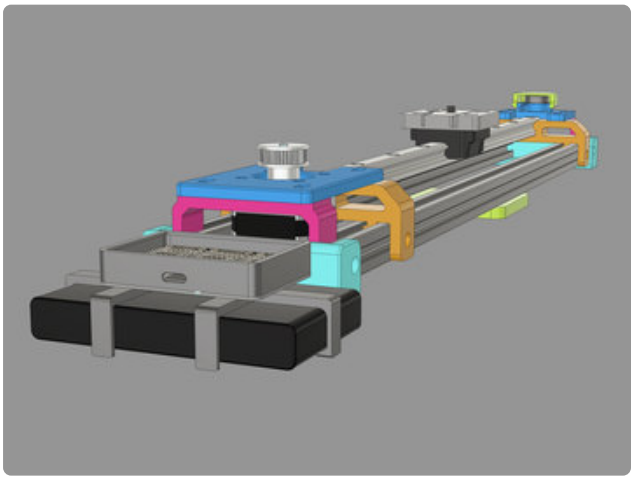
The parts were printed and tested in PLA filament. For parts with more strength or usage outdoors we suggest using PETG filament. The parts were sliced using Ultimaker CURA 4.x software on a Creatly CR-10S 3D printer.

Parts List

List of all the 3d printed parts with filenames. Note the slider-rail-mount need to be printed twice.

- slider-featherwing-case.stl

- slider-featherwing-case-bottom.stl
- slider-bat-mount.stl
- slider-bearing-cap.stl
- slider-bar-mount-motor.stl
- slider-bar-mount-bearing.stl
- slider-tripod-mount.stl
- slider-cam-mount.stl
- 2x slider-rail-mount.stl
- 2x slider-feet.stl
- slider-bearing-mount.stl
- slider-motor-mount.stl
- slider-motor-support.stl
- slider-bearing-support.stl



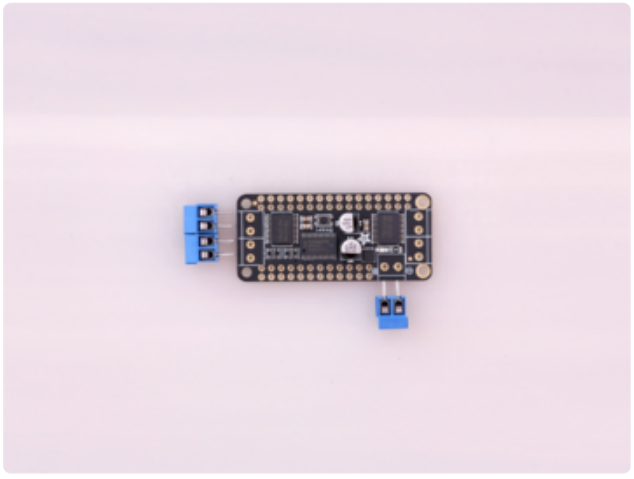
Design Source Files

The project assembly was designed in Fusion 360. This can be downloaded in different formats like STEP, SAT and more. Electronic components like Adafruit's board, displays, connectors and more can be downloaded from our Adafruit CAD parts GitHub Repo (https://adafru.it/AW8).

Adafruit CAD Parts on Github

https://adafru.it/AW8

Motor FeatherWing Prep



Terminal Blocks

The Motor FeatherWing includes screw block terminals for connecting stepper motor and the 12v battery pack. The project uses a single stepper motor.

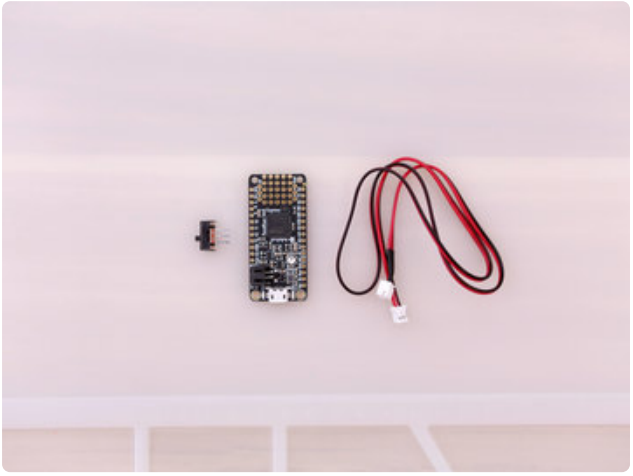
Solder Screw Block Terminals

Insert two 2-pin screw block terminals into the M3 and M4 motor pins. Then insert one 2-pin screw block terminal into the power port. Solder the pins in place. You can optionally use M1 and M2 ports, just remember to reflect that in the code.

Install Headers

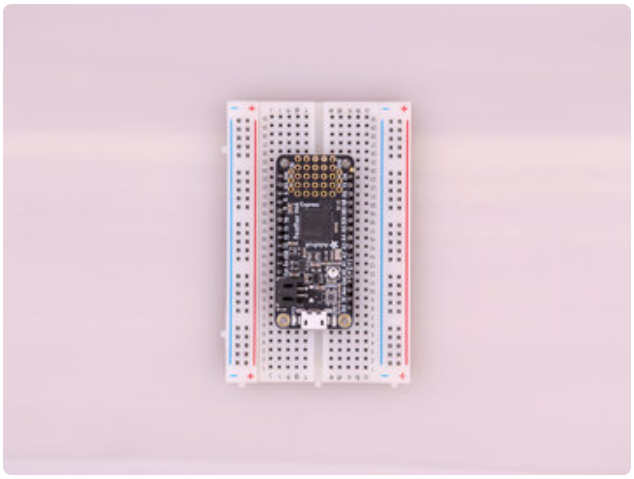
Insert the 12-pin and 16-pin male headers into the pins on the Motor FeatherWing. A breadboard can be used to assist in soldering by holding the headers in place.

Feather Prep



Feather Switch JST Cable

The slide switch and JST extension cable will need to be soldered to the Feather M4.



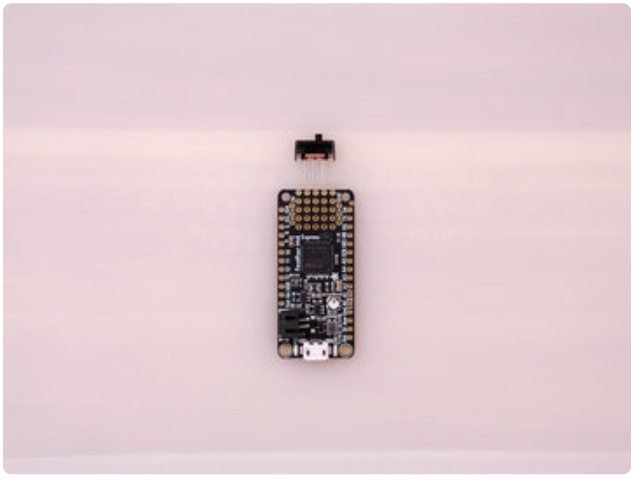
Install Headers

Insert the 12-pin and 16-pin headers into the pins on the Feather M4. A breadboard can be used to assist in soldering by holding the headers in place.

If you'd like to avoid soldering a switch slide, you can use a pre-made on/off switch with JST extension cable.

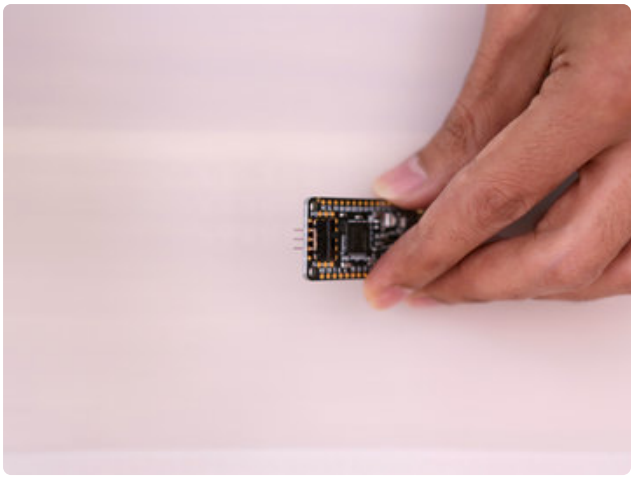
JST 2-pin Extension Cable with On/Off Switch - JST PH2

By popular request - we now have a way you can turn on-and-off Lithium Polymer batteries without unplugging them.This PH2 Female/Male JST 2-pin Extension...
https://www.adafruit.com/product/3064



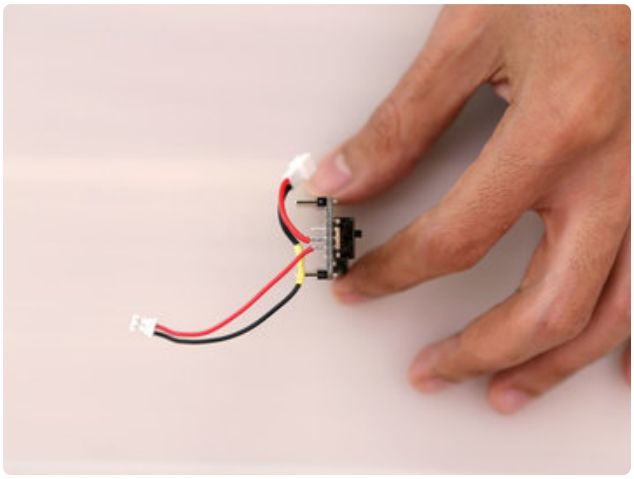
Installing Feather Switch

The slide switch can soldered to the prototyping area on the Feather M4. This will keep the slide switch secured to the Feather PCB.



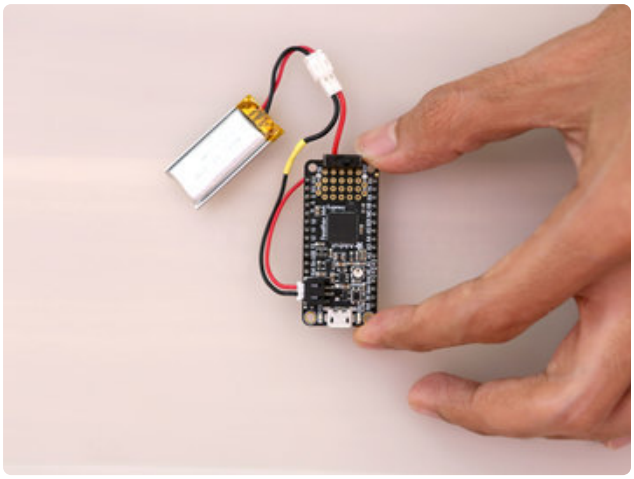
Switch Placement

The slide switch can be placed on the far end of the prototyping area on the Feather M4. This gives easy access to the switch. Solder the pins to the Feather. Mounting tack can help keep the body of the slide switch in place while soldering.



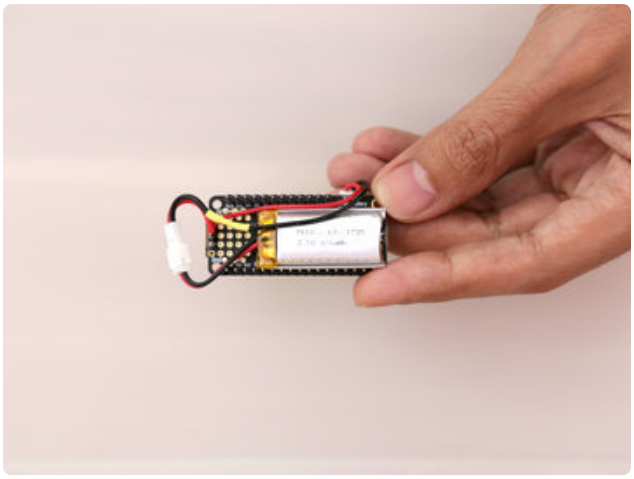
Solder JST Cable to Switch

Using wire cutters, trim the ends of the JST extension cable short. The voltage wire is soldered to the middle and either far left or right pin on the slide switch. The ground wire is tied together. Pieces of heat shrink tubing are used to insulate the exposed wire.



Connect Battery to Feather

The battery is connected to the JST connected that is wired to the slide switch. The female JST connector is plugged directly into the power port on the Feather M4.



Battery Placement

The 400mAh battery is designed to fit in between the headers on the Feather M4. Place the battery under the PCB and adjust the wiring so it's nice and neat.

Battery Prep

Wiring Battery

The ground and voltage wires will be connected directly to the screw block-terminals on the Motor FeatherWing. Using wire cutters, snip off the 2.1mm barrel jack from the 12V battery pack.

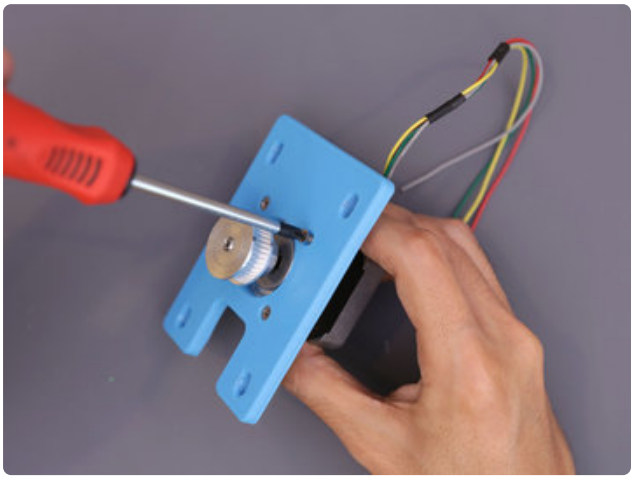
Tinning Wires

Using wire stripper, remove a bit of insulation from the tips of each wire. A helping hand stand can help keep the wires in place while soldering. Tin the ends of the exposed wire by adding a bit of solder. This will help prevent the strands of wire from fraying. Tinned wires will be easier to connect to the screw block terminals on the Motor FeatherWing.

Stepper Motor Assembly

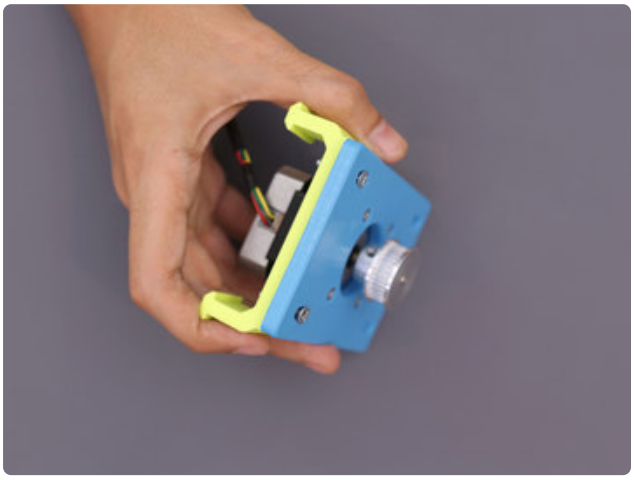
Pre-fasten Motor Mount

Fasten 4x M3 (6mm long) screws into the top of the motor mount – The top is the surface with the chamfered mounting holes. The screws should have a semi-loose tolerance.



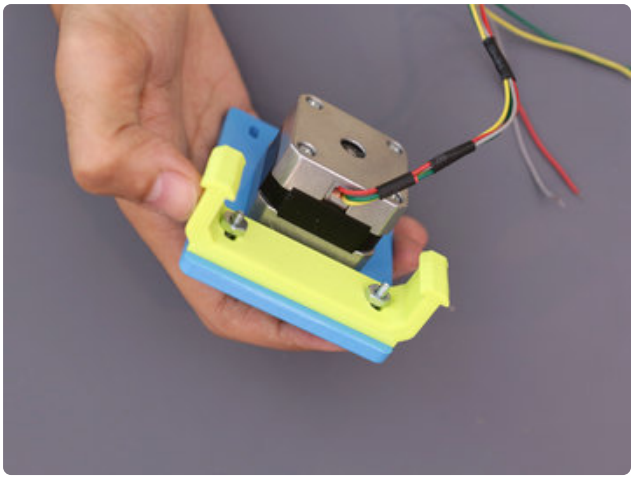
Secure Stepper Motor to Mount

Fit the mount over the stepper motor. Orient the motor so the wires are positioned like it is in the photo. Fasten the 4x M3 screws until fully tightened to secure the mount to the stepper motor.



Install Motor Mount Support

Grab the **slider-motor-support.stl** part and place it under the motor mount. Position the mount so the mounting holes line up to match the photo. Insert and fasten 2x M4 machine screws into the mounting holes.



Secure Motor Mount Support

While holding the support mount in place, insert 2x hex nuts and fasten until fully tightened. The mounting holes are slotted and adjustable for adding more tension to the belt.

Install Rail Mount to Motor Mount

The rail mount features a recessed area for a hex nut. Insert a hex nut and hold it in place. The hex nut is loose, so you'll need to hold it up right so it doesn't fall out while fastening to the motor mount.

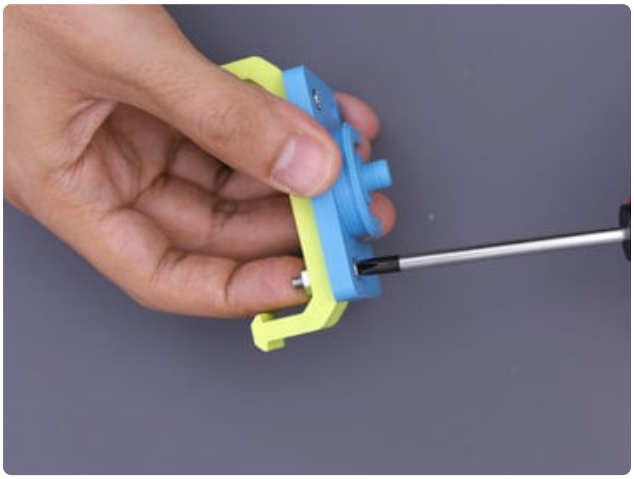
Secure Rail Mount

Place the rail mount onto the motor mount. Insert and fasten an M4 screw through the top of the motor mount. The screw should go through the two mounts with the hex nut securing them together. Repeat this process for the second mounting hole.

Motor Subassembly

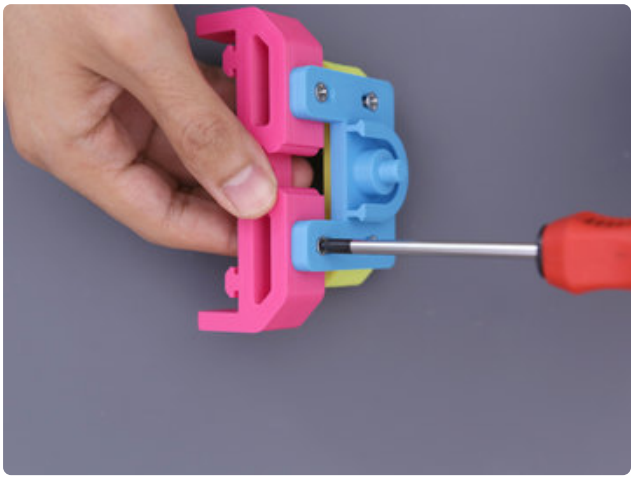
The stepper motor subassembly should look like this. Both the rail mount and the support mount feature slotted holes for adjusting the tension of the belt – This can be done later on the assembly.

Bearing Mount Assembly



Install Mount Support

The bearing mount assembly is similar to the motor assembly. They both need a rail mount and a support mount. Place the support mount under the bearing mount. Insert an M4 screw through the top of the bearing mount. Use an M4 hex nut to secure the two mounts together.



Install Rail Mount

Place the rail mount under the bearing mount. Install hex nuts into the recessed areas of the rail mount. Insert and fasten M4 screws to secure the bearing mount to the rail mount.

Bearing Mount Subassembly

The bearing mount should look like this. Again, the slotted mounting holes are there so you can adjust the belt tension later on in the build.

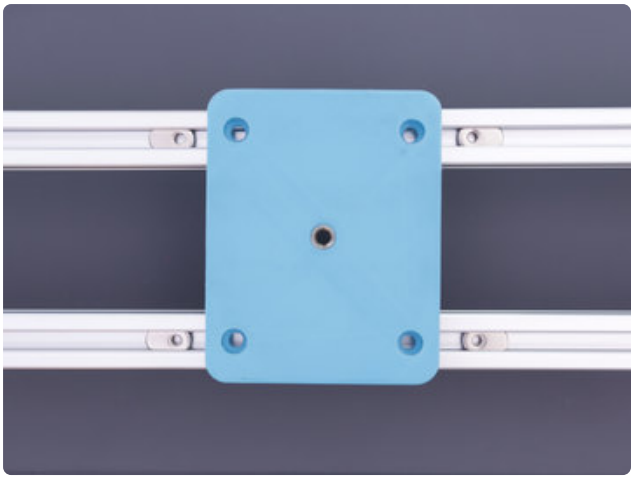
Tripod Mount Assembly

Install Tripod Screw

The tripod mount needs a 3/8" to 1/4" screw adapter so it can be secured to a camera tripod. Insert and fasten the tripod screw adapter to the center of the tripod mount. Use a large flat screwdriver to fully tighten the screw.

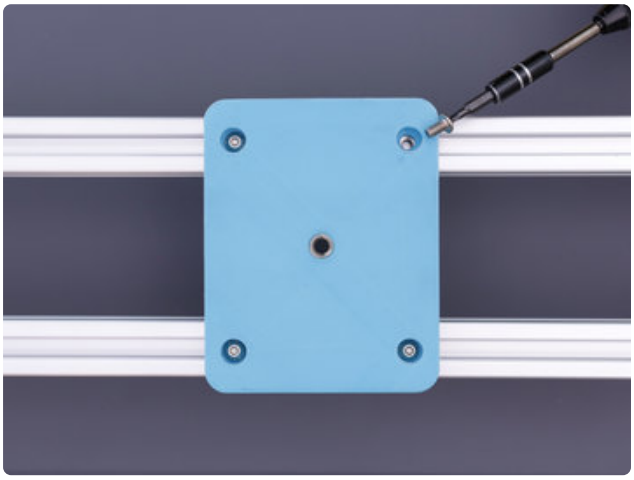
Install T-Nuts to Aluminum Extrusions

The oval T-Nuts are designed to be slotted into the aluminum extrusion. Grab both aluminum extrusion bars and insert two t-nuts. These will be used to secure the tripod mount to the aluminum bars.



Tripod Mount Pre-Install

Position the two aluminum extrusion bars so the tripod mount can be rested over them. Orient the parts so the mounting holes are positioned over the extrusion bars. Slide the T-Nuts so the mounting holes line up with the ones in the Tripod Mount.



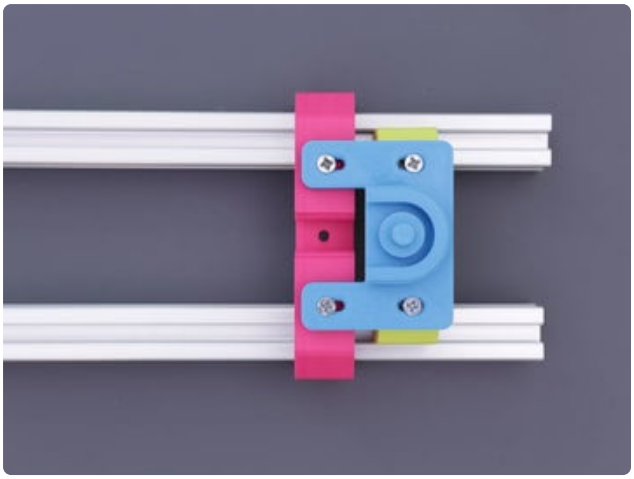
Secure Tripod Mount

Insert and fasten 4x M4 screws to fix the Tripod Mount to the two aluminum extrusion bars. Do not fully tighten the screws, leave them loose. The M4 screw should be threaded through the T-Nuts. The tripod mount should be able to slide along the slotted aluminum extrusions. We will fully tighten the screws later on in the build.

Extrusion Assembly

Install Side T-Nuts

Insert two T-Nuts to both of the sides of the two aluminum extrusion bars. Make sure to follow the orientation of the T-Nuts and reference the photo.

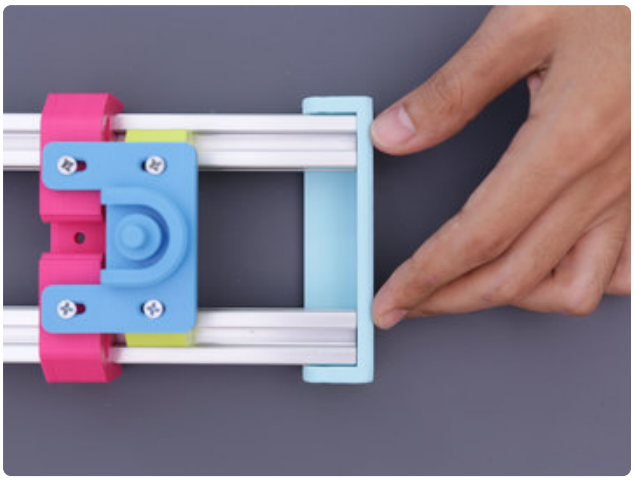


Install Bearing Mount

Grab the bearing mount assembly and slide the T-Shaped nubs through the slotted aluminum extrusions. This should be installed to the opposite side of the tripod mount. The bearing mount assembly should be able to freely slide through the slotted aluminum extrusion.

Insert Side Screws

Slide the bearing mount assembly so the mounting holes line up with the T-Nuts. Insert and fasten M4 screws to secure the mount to the T-Nuts. Leave them loose – The assembly need to be able to slide along the t-slotted aluminum extrusions.



Install Bearing Bar Mount

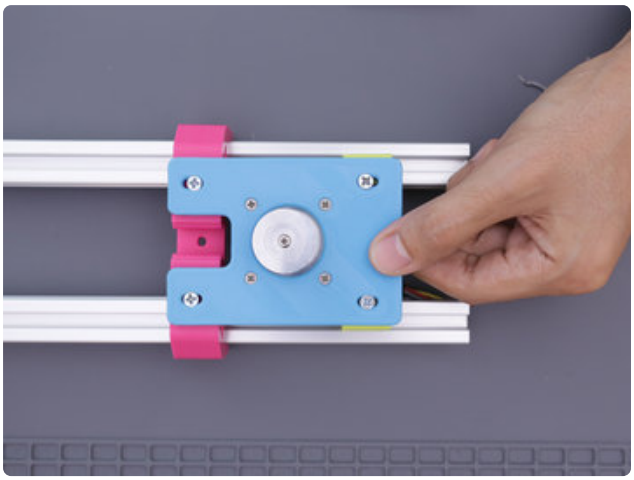
Grab the bearing bar mount and fit it onto the end cap of the two aluminum extrusions. The bar mount features t-shaped nubs that should slot into the aluminum extrusions.

Secure Bar Mount

Slide the remaining T-Nut so it's line up with the mounting hole in the bar mount. Insert and fasten M4 screws to fix the bar mount to the T-Nuts. Again, you'll want to leave them a bit loose so we can adjust them later in the build.

More T-Nuts

Insert two more T-Nuts on to the other side of the aluminum extrusions. These will secure the Motor Mount assembly to the extrusion bars. Again, make sure the orientation matches the photo.



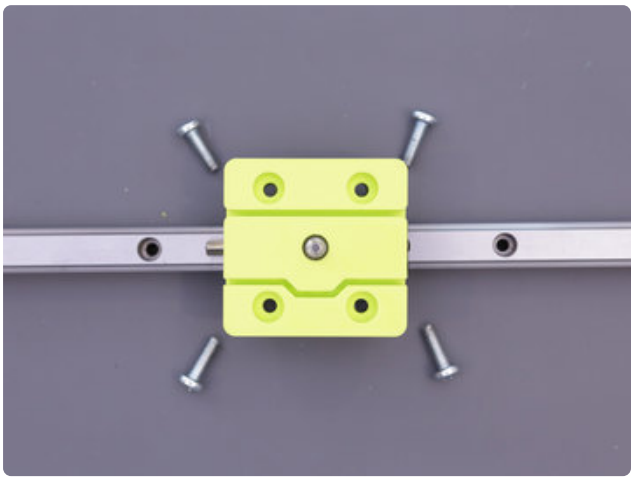
Install Motor Mount Assembly

Insert and slide the motor mount assembly onto the two slotted aluminum extrusions. Both the rail mount and the support mount should be able to fitted and slide along the slotted aluminum extrusion bars.

Camera Mount Assembly

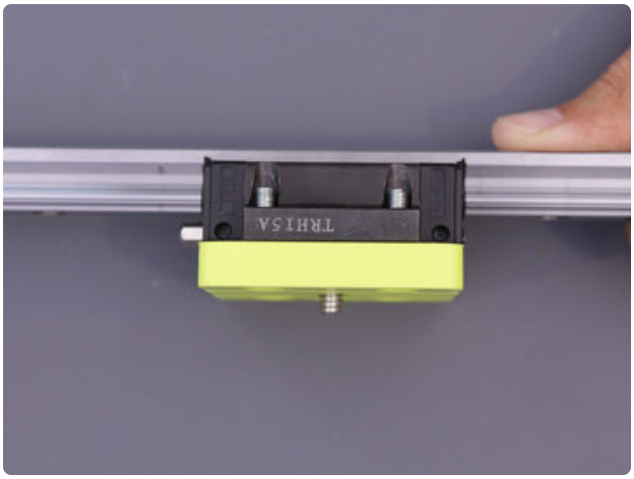
Install D-Ring

Grab the D-Ring tripod screw and insert it into the camera mount. The screw will be very loose and fall out easily. Hold the screw in place while installing it to the Pillow Block.



Install Camera Mount

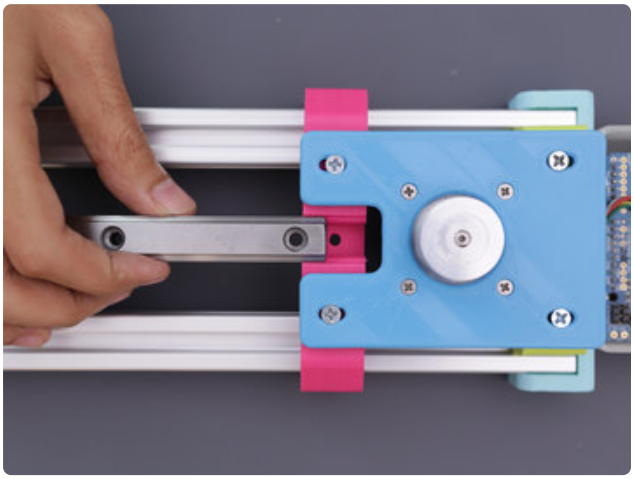
While holding the tripod screw, place the camera mount on top of the linear bearing pillow block. Orient the mount so the mounting holes line up. M5 x 16mm long screws will secure the camera mount to the linear bearing pillow block.

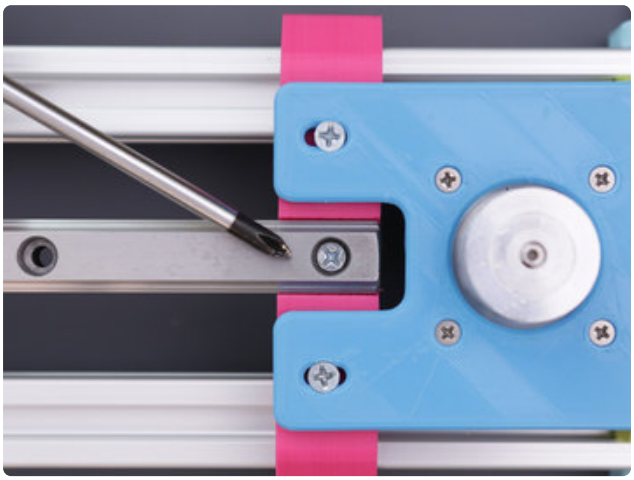


Secure Camera Mount

Insert and fasten the M5 screws into the mounting holes to secure the camera mount to the pillow block. Fully tighten the secures to join the two parts together. The mounting holes in the pillow block is threaded, so it doesn't need hex nuts.

Rail Assembly



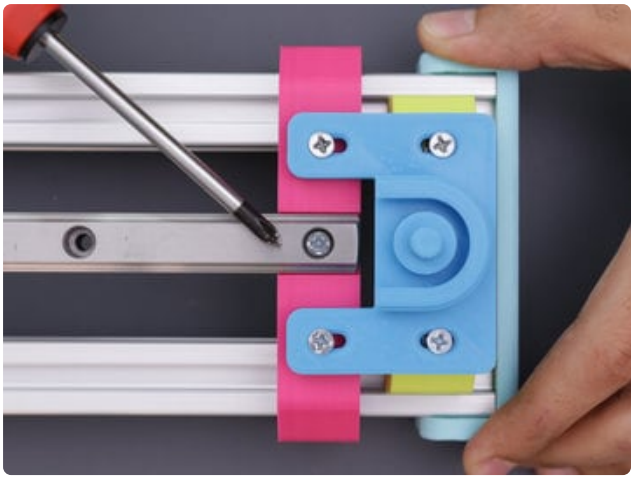


Install Slide Rail

Grab the linear rail and slide it into the motor mount assembly. The rail should slot into the motor rail mount loosely. The bearing assembly should be able to slide out of the extrusion bars to allow space for the linear rail. Line up the mounting hole on the rail with the mount. Insert an M4 x 18mm long screw into the mounting hole.

Secure Rail to Motor Assembly

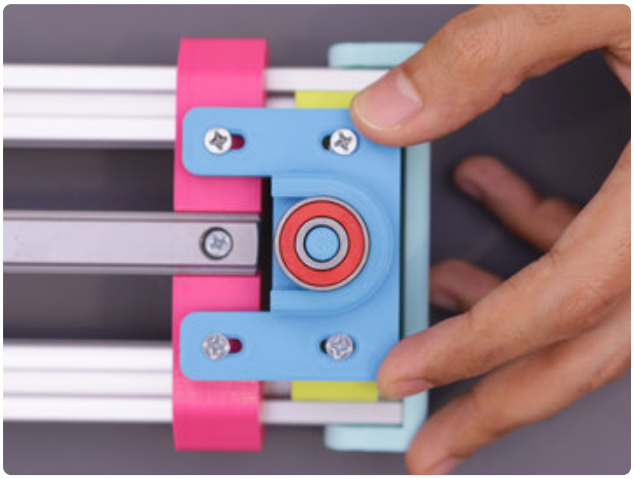
Flip the entire assembly over and place a hex nut onto the threading of the M4 screw. Fasten the screw and hex nut to tightly secure the parts together.



Secure Rail to Bearing Assembly

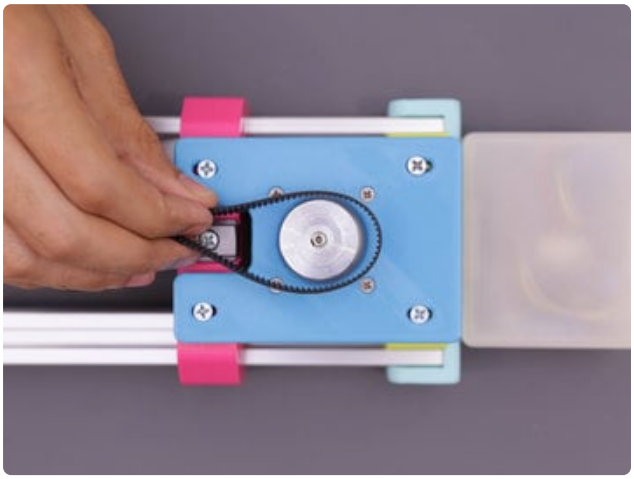
Following a similar process, slide the rail onto the bearing mount assembly. Line up the mounting holes and insert another M4 x 18mm long screw. Use a hex nut to secure the rail to the rest of the assembly.

Belt Assembly



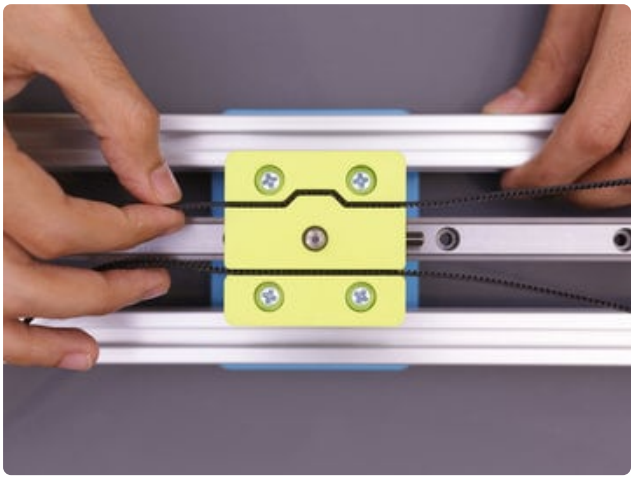
Install Bearing to Mount

Place the radial ball bearing over the post in the bearing mount.



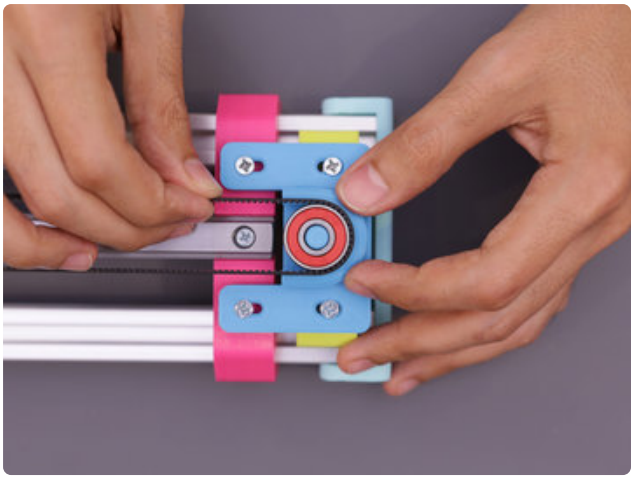
Install Belt to Pulley

Grab the timing belt and place it over the pulley of the stepper motor. The teeth in the belt should grab onto the teeth of the timing pulley.



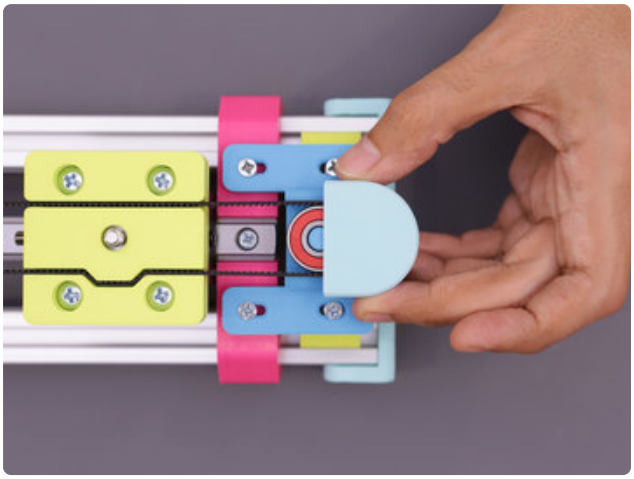
Install Belt to Camera Mount

Work the timing belt into the notches of the camera mount. Press the belt into the slotted features of the camera mount to secure it in place.



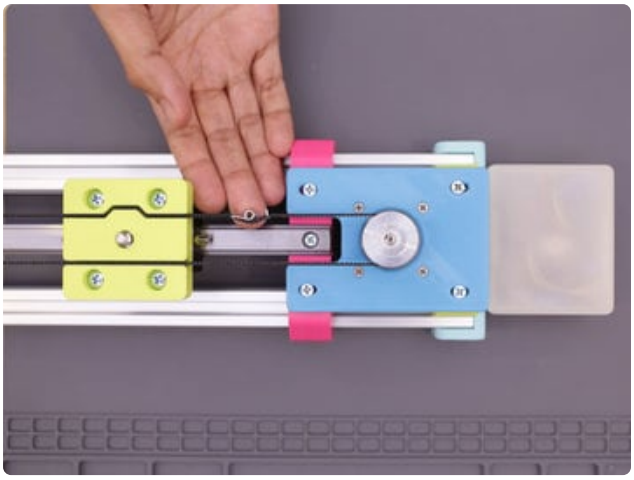
Install Belt to Bearing

Grab the other end of the timing belt and work it on top of the radial ball bearing. Pull the belt and press it down so it grabs onto the ball bearing.



Install Bearing Cover

The bearing cover slides into the grove on the edge of the bearing mount. This rests in place and doesn't need any further securing. The tolerances will be loose or tight, depending on the printed part. That should not effect the performance.



Belt Tensioner

You can optionally install a belt tensioner (https://adafru.it/B2p) to tighten the timing belt.

Case Assembly

Secure Case to Mount

The Tripler FeatherWing case is attached to the left bar mount using machine screws and hex nuts. Use the following hardware to secure these parts together.

2x M3 x 8mm screws
2x M3 nylon hex lock nuts

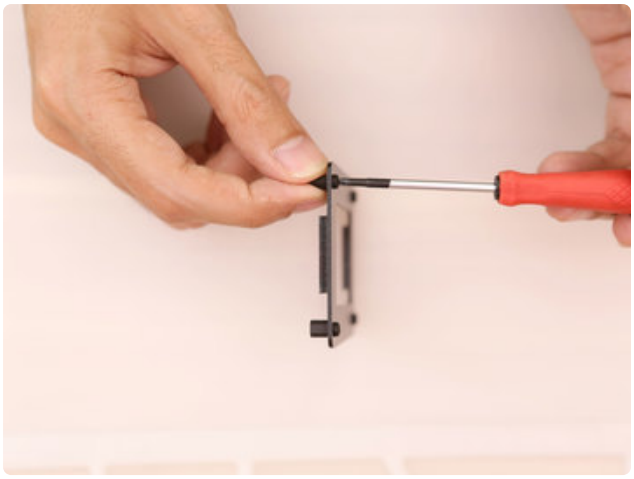
Secured Case Mount

The M3 nylon insert hex nuts are press fitted into the holes on the inside of the left bar mount. Fasten the screws until they're fully tightened.

Bottom Cover Screws

The bottom cover requires the following hardware to secure the Tripler FeatherWing.

8x M2.5 x 4mm machine screws
4x M2.5 x 6mm female-female standoffs



Install Standoffs

Insert the M2.5 x 4mm screws through the bottom. Press fit them through the outer four holes on the cover. While holding the screw in place, add a standoff and fasten it tightly.

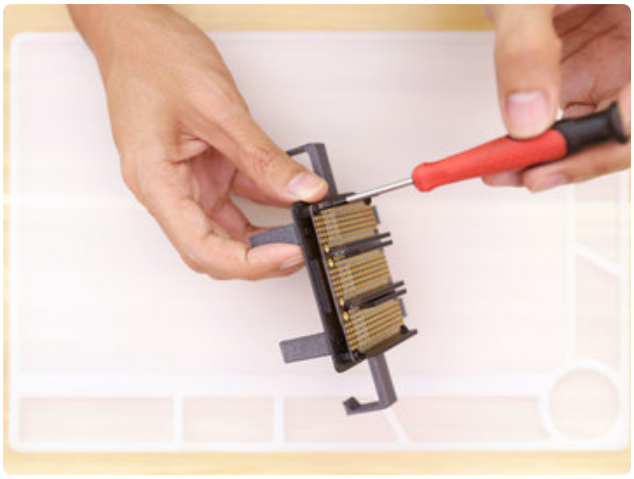
Secure Battery Holder to Cover

Place the battery holder under the cover and line up the mounting holes. Press fit screws through the remaining mounting holes. Use the following hardware to secure the parts together.

4x M2.5 x 6mm screws
4x M2.5 hex nuts

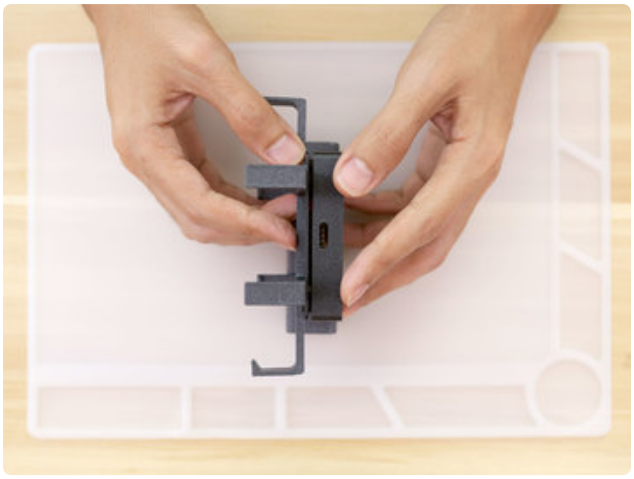
Secured Battery Holder

Here's the battery holder secured to the bottom cover. The mounting holes are symmetrical so it orientation of the parts doesn't matter.



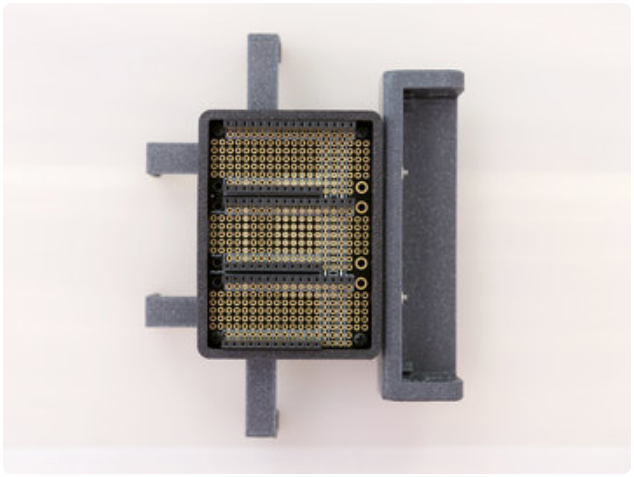
Secure Tripler FeatherWing

Place the Tripler FeatherWing over the standoffs and line up the mounting holes. Insert and fasten the M2.5 screws to secure the PCB to the standoffs.



Install Case to Botton Cover

Position the case over the bottom cover and press down to snap them together. The case is also symmetrical so orientation doesn't matter.

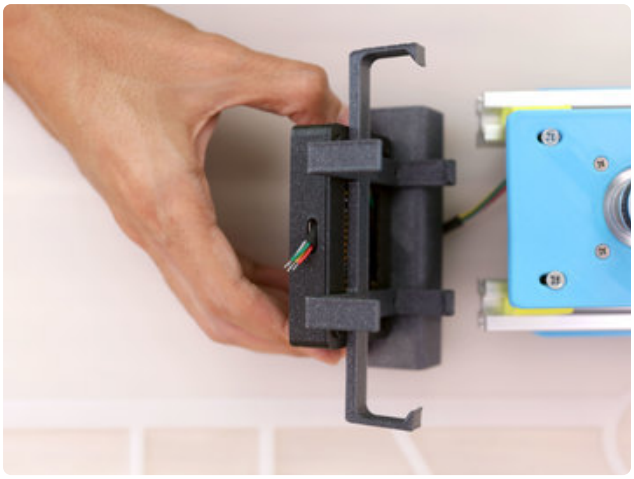


Installed Case

The bottom cover features snaps that lock onto the nubs on the inside of the case. This allows the two parts to snap fit together.

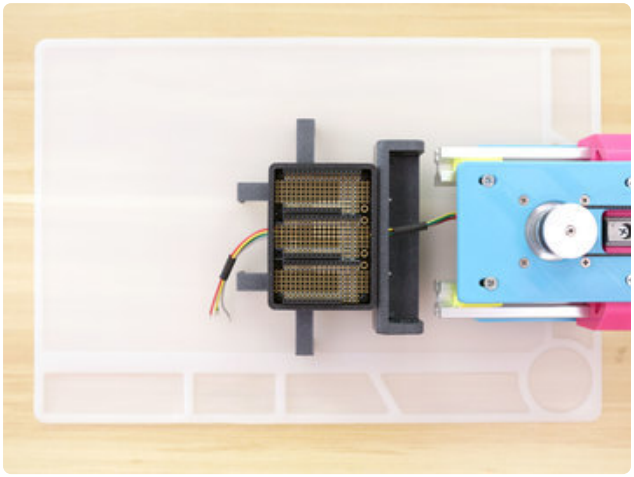
Install Motor Wires

Grab the four wires from the stepper motor and begin to insert them through the hole on the inside of the left bar mount. Use pieces of heat shrink tubing to keep the wires bundled together.



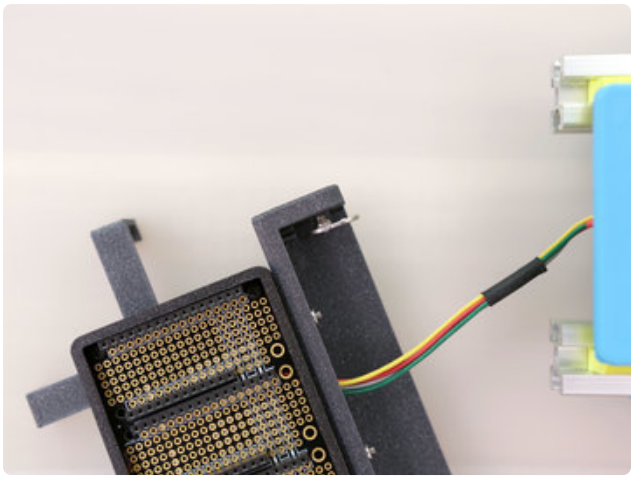
Motor Wiring

Push and thread the wiring through the case and out the hole on the other end of the case. You can optionally remove the Triper FeatherWing to gain easier access to the wiring.



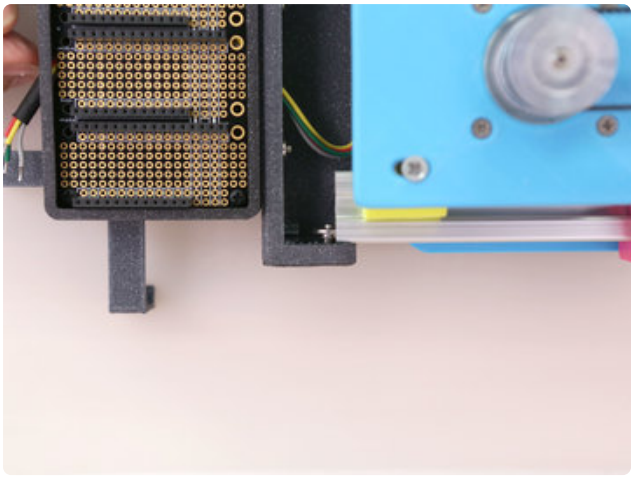
Stepper Motor Wires

Pull out the wires from the stepper motor through the hole on the end of the case. If the wires are too long, you can optionally trim them short.



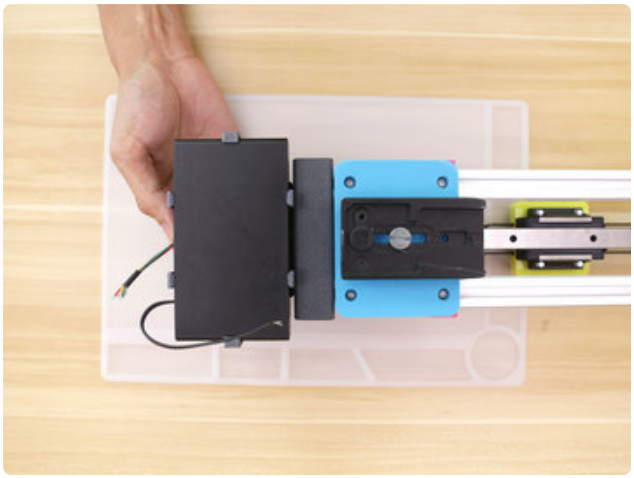
Installing T-Nuts

Insert and fasten two M4 x 8mm machine screws to the tabs on the side of the left bar mount. Fit the M4 oval t-nuts onto the threads of the screws. Fasten the screws so the oval t-nuts are secured to the threads. Don't fasten them all the way just yet.



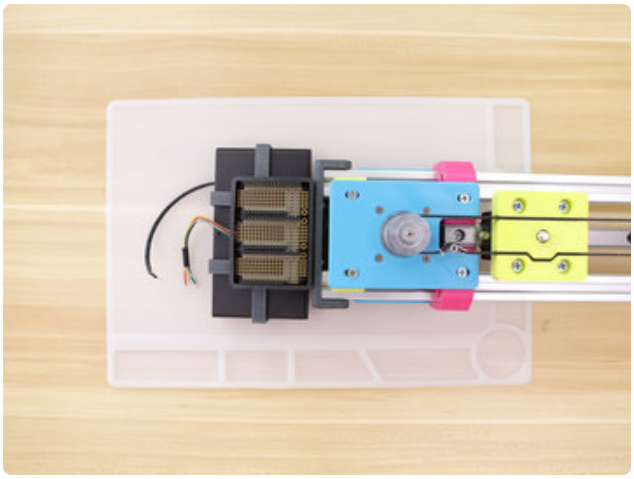
Install Mount

With the oval t-nuts installed to the screws, start to fit the left bar mount onto the aluminum extrusions. Adjust the oval t-nuts so they slide into the profiles of the aluminum extrusions.



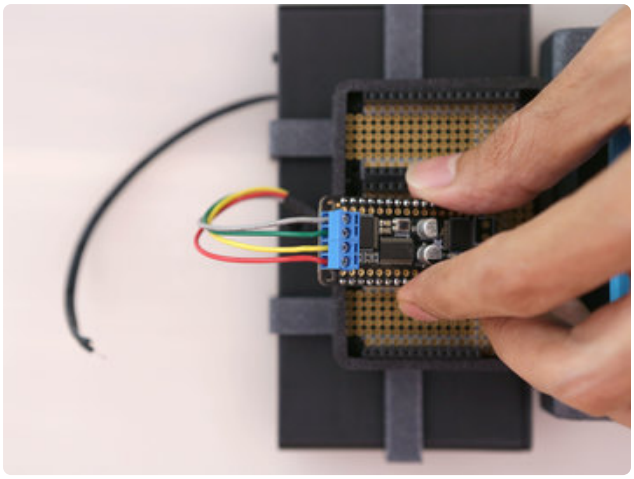
Install Battery

Carefully begin to flip the whole build upside down. This will make installing the battery pack to the battery holder much easier. Grab the 12V battery pack and start to fit it into the holder. The clips can flex slightly so they grasp onto the battery pack.



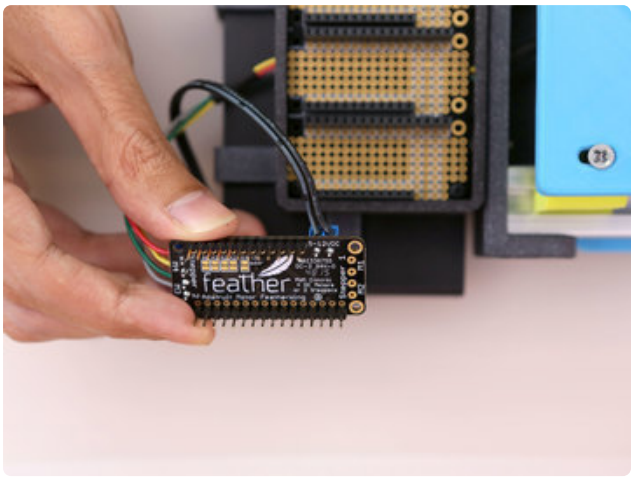
Battery Orientation

The battery is oriented such that the built-in slide switch is accessible and wiring is able to reach the Motor FeatherWing.



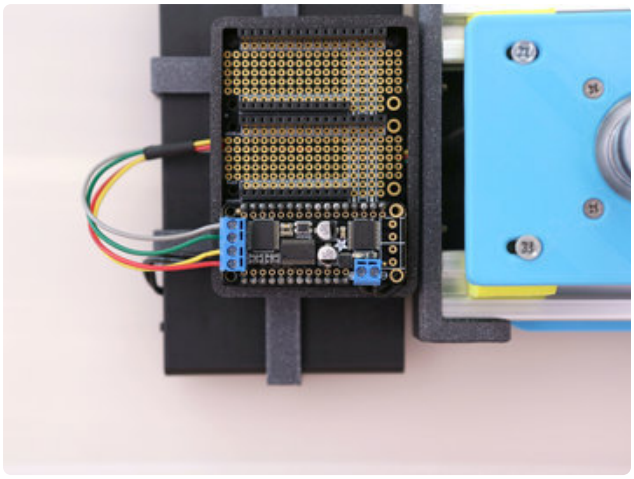
Connect Stepper Motor to FeatherWing

Grab the Motor FeatherWing and connect the wires from the stepper motor to the screw block terminals. Reference the photo for the correct placement of the wires – Polarity matters here so follow it carefully. Use a screw driver to tighten the screws and secure the wires in place.



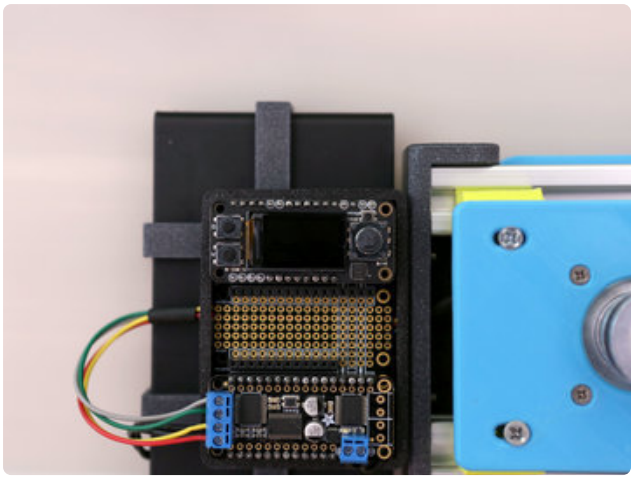
Connect Battery Wires to FeatherWing

Insert the voltage and ground wire from the 12V battery pack to the screw block terminals. The power port is located on the side of the motor FeatherWing. Fasten the screws to secure the voltage and ground wires from the battery pack.



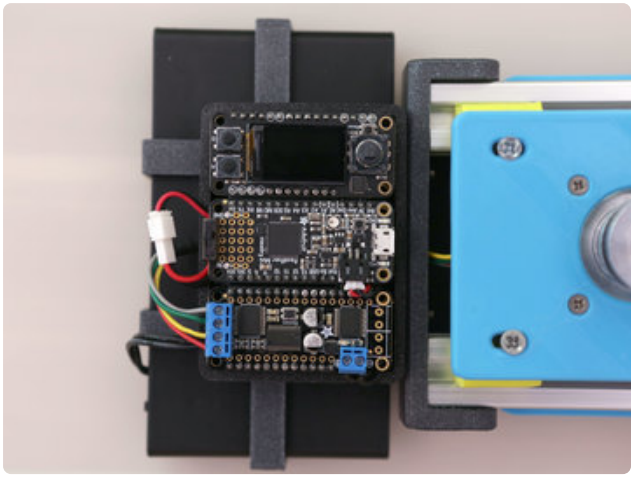
Install Motor FeatherWing to Tripler

Press the Motor FeatherWing onto the headers on the Tripler FeatherWing. You can choose to install it on any of the three available sets of headers on Tripler FeatherWing.



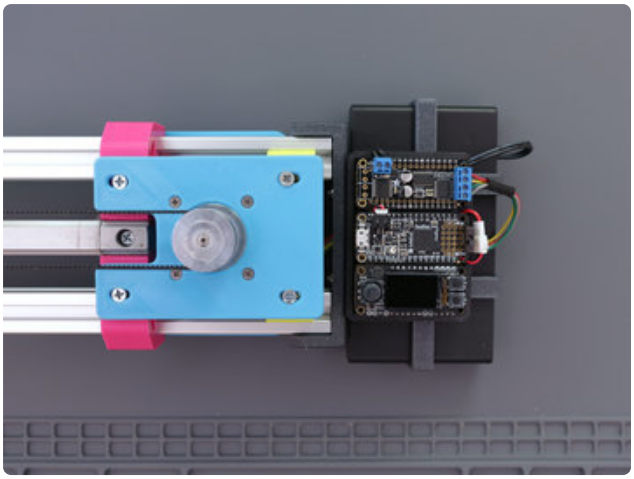
Install Mini TFT FeatherWing to Tripler.

Press the mini TFT FeatherWing onto the headers on the Tripler FeatherWing. You can choose to install it on any of the remaining sets of headers.



Install Feather M4 to Tripler

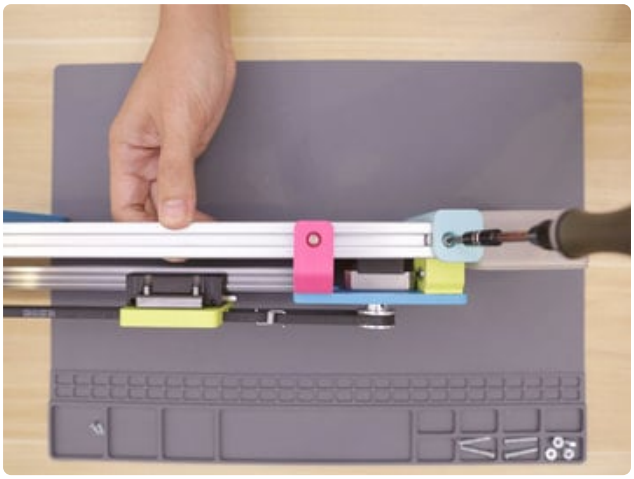
Finally, press fit the Feather M4 on to the remaining set of headers on the Tripler FeatherWing. Again, you can rearrange the FeatherWings however you see fit. Make sure the 400mAh lipo battery and JST cabling is not being kinked or pinched by the headers.



Installed Case Assembly

Thoroughly inspect the wiring and make sure all of the wired connections are properly placed and secured. Remember to turn on both switches, the one built into the 12V battery pack and the slide switch soldered to the Feather M4.

Final Assembly



Motor SubAssembly

Ensure the timing belt is tight. Proceed to tighten all of the screws in the motor assembly.

Bearing SubAssembly

Hold the entire assembly while tightening all of the screws in the bearing assembly.

Tripod Mount SubAssembly

Slide the tripod mount so it's in the center of the sliding rail. Then, proceed to tighten all of the screws.