Two Way Telegraph with Analog Feedback Servos

Created by Ruiz Brothers

https://learn.adafruit.com/two-way-display-with-analog-feedback-servos

Last updated on 2024-06-03 03:39:49 PM EDT

Table of Contents

Overview 3

- IoT Telegraph
- Parts

Circuit Diagram 5

- Adafruit Library for Fritzing
- Wired Connections

CAD Files 7

- CAD Parts List
- CAD Assembly
- Build Volume
- Label & Graphics
- Design Source Files

CircuitPython 9

- CircuitPython Quickstart

Code the Two Way Telegraph 12

- Upload the Code and Libraries to the QT Py ESP32-S2
- secrets.py
- Analog Servo Calibration
- How the CircuitPython Code Works

Wiring 18

- QT Py and Servo Prep
- Tin Wires from Servo
- Prepped Servo Wires
- Solder Wires to QT Py
- Connected Servo
- Copper and Wire
- Solder Wire to Copper
- Connect Copper Tape
- Solder Copper Tape Wire
- Finished Circuit

Assembly 21

- Attach Copper Tape
- Wrapping Copper Tape
- Assemble Handle
- Attaching Servo
- Secure Servo
- Adhere Label
- Attach Bezel, Frame and Handle
- Secure QT Py to Holder
- Secure Base Plate and Holder

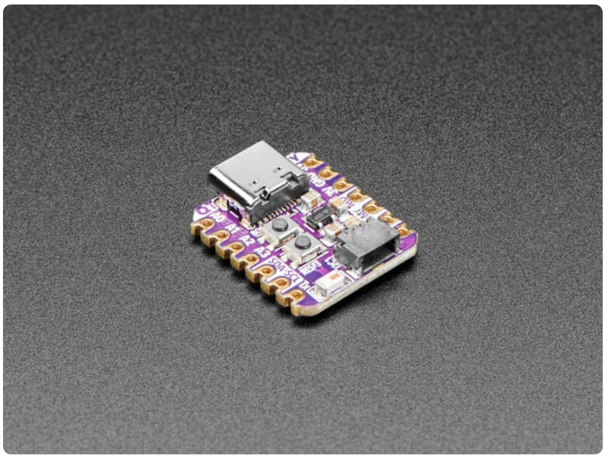
Usage 24

Overview

IoT Telegraph

Build an IoT telegraph using QT Py ESP32 S2, analog feedback servos and CircuitPython! 3D print the case inspired by ship engine telegraphs and use capacitive touch to detect when the handle is touched. Use Adafruit IO to make a WiFi enabled two-way communication system.

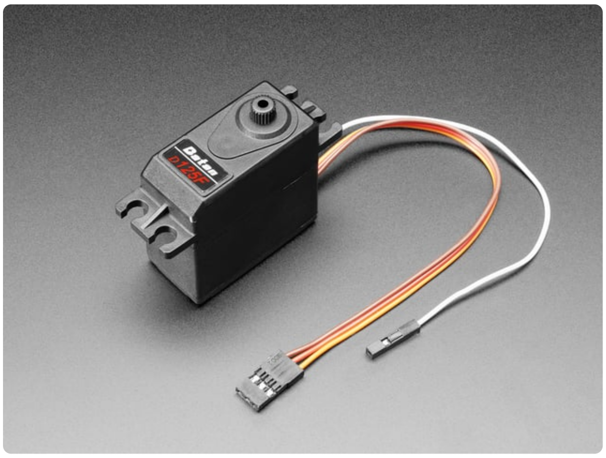
Parts



Adafruit QT Py ESP32-S2 WiFi Dev Board with STEMMA QT

What has your favorite Espressif WiFi microcontroller, comes with our favorite connector - the STEMMA QT, a chainable I2C port, and has...

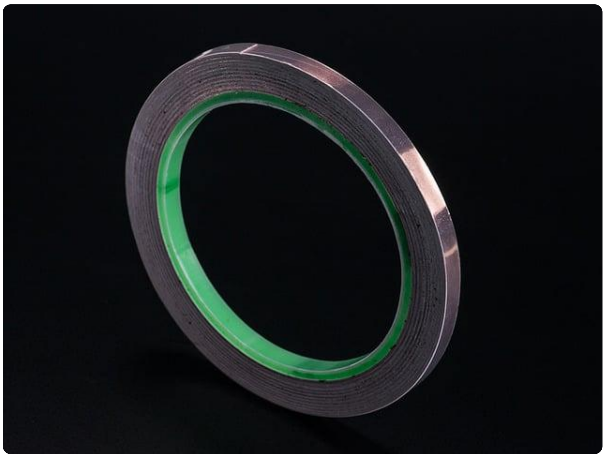
https://www.adafruit.com/product/5325



Analog Feedback Standard-Size Servo

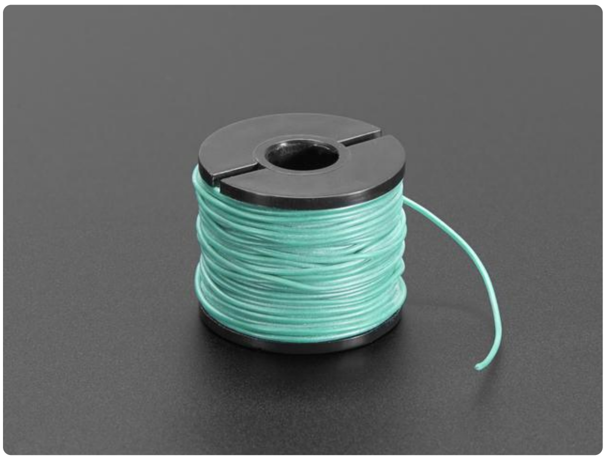
It looks like a servo; it acts like a servo, but it's more than just a servo! We got a factory to custom-make these classic 'standard' sized hobby servos with a twist - the...

https://www.adafruit.com/product/1404



[Copper Foil Tape with Conductive Adhesive - 6mm x 15 meter roll](https://www.adafruit.com/product/1128)
Copper tape can be an interesting addition to your toolbox. The tape itself is made of thin pure copper so its extremely flexible and can take on nearly any shape. You can easily...
https://www.adafruit.com/product/1128

[Button Hex Machine Screw - M4 thread - 8mm long - pack of 50](https://www.adafruit.com/product/1160)
Cute as a button, these button-head hex screws are what we suggest for putting together a project with our slotted extruded aluminum. Use a 2.5mm hex wrench to attach/detach. This...
https://www.adafruit.com/product/1160



[Silicone Cover Stranded-Core Wire - 50ft 30AWG Green](https://www.adafruit.com/product/3168)
Silicone-sheathing wire is super-flexible and...
https://www.adafruit.com/product/3168

[Black Nylon Machine Screw and Stand-off Set – M3 Thread](https://www.adafruit.com/product/4685)
Totaling 420 pieces, this M3 Screw Set is a must-have for your workstation. You'll have enough screws, nuts, and hex standoffs to fuel...
https://www.adafruit.com/product/4685

Avery Printable Sticker Paper, Matte White, 8.5 x 11 Inches

1 x Sticker Project Paper
Avery Printable Sticker Paper, Matte White, 8.5 x 11 Inches

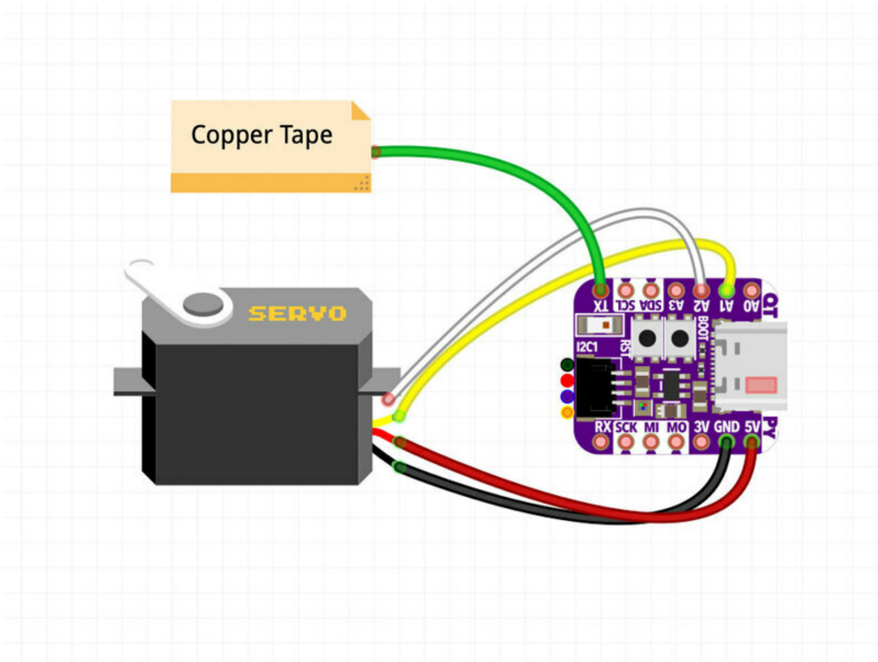
https://www.amazon.com/Avery-Sticker-Removable-Adhesive-53202/dp/B000XALDGM/

Circuit Diagram

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

Adafruit Library for Fritzing

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).



Wired Connections

Analog Feedback Servo

- Red/Orange wire to **5V** pin on QT Py
- Brown/Black wire to **GND** pin on QT Py
- Yellow wire to **A1** pin on QT Py
- White wire to **A2** pin on QT Py

Copper Tape – Capacitive Touch

- Signal wire to **TX** pin on QT Py

CAD Files

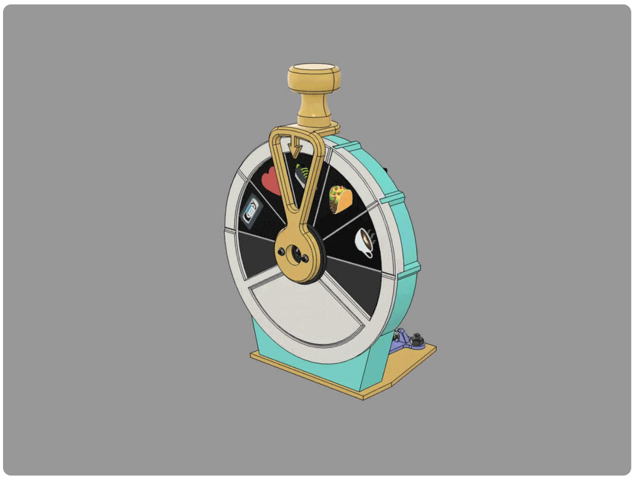
CAD Parts List

STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material. Original design source may be downloaded using the links below:

twt-handle.stl
twt-horn.stl
twt-front-plate.stl
twt-frame.stl
twt-front-ring.stl
twt-base.stl
qtpy-holder.stl

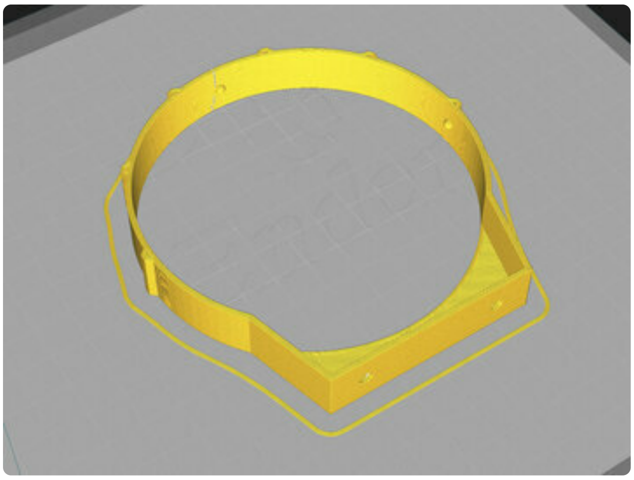
cad-source.zip
https://adafru.it/10td

Download STL files
https://adafru.it/10te



CAD Assembly

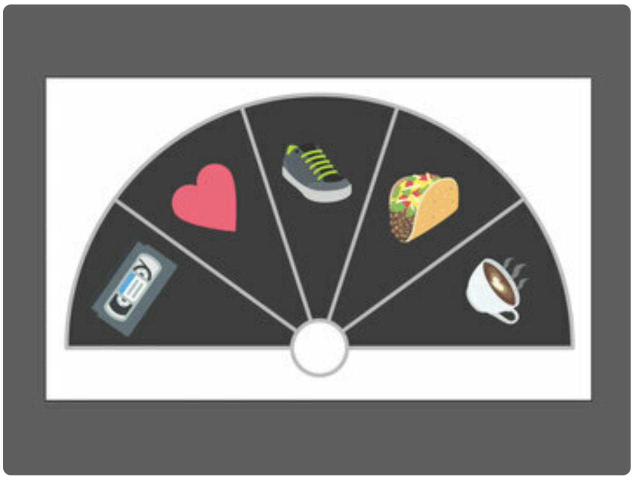
The servo is secured to the front plate with screws. The bezel snap fits over the frame. The front plate snap fits onto the frame. The base plate is secured to the frame with screws and hex nuts. The QT Py snap fits into the holder. The holder is secured to the base plate with screws and hex nuts. The handle is secured to the 3D printed horn with a single screw. The handle assembly is secured to a stock servo horn with screws. The stock servo horn is secured to the shaft of the motor with a single screw.



Build Volume

The parts require a 3D printer with a minimum build volume.

110mm (X) x 110mm (Y) x 100mm (Z)



Label & Graphics

Download the vector template and modify the graphics to your desired set of emoji's.

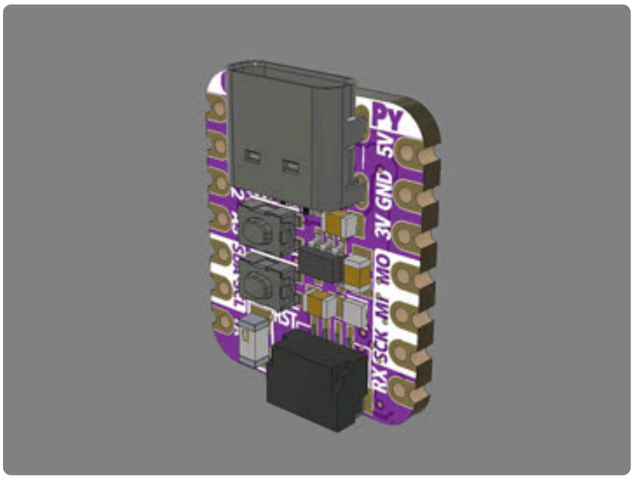
Use an inkjet color printer with sticker project paper to easily adhere the graphics to the 3D printed face plate.

Label Dimensions: 92mm x 46mm

Ensure the graphics are not scaled when printing. Reference the dimensions above for correct size and scaling.

Download Vector Templates

https://adafru.it/10tf



Design Source Files

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

CircuitPython

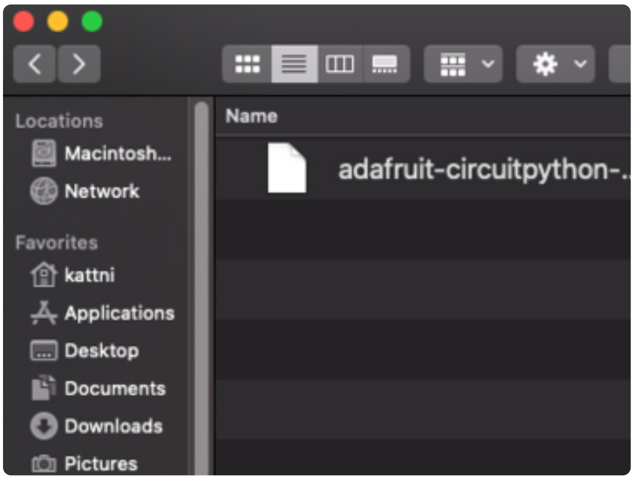
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

CircuitPython Quickstart

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

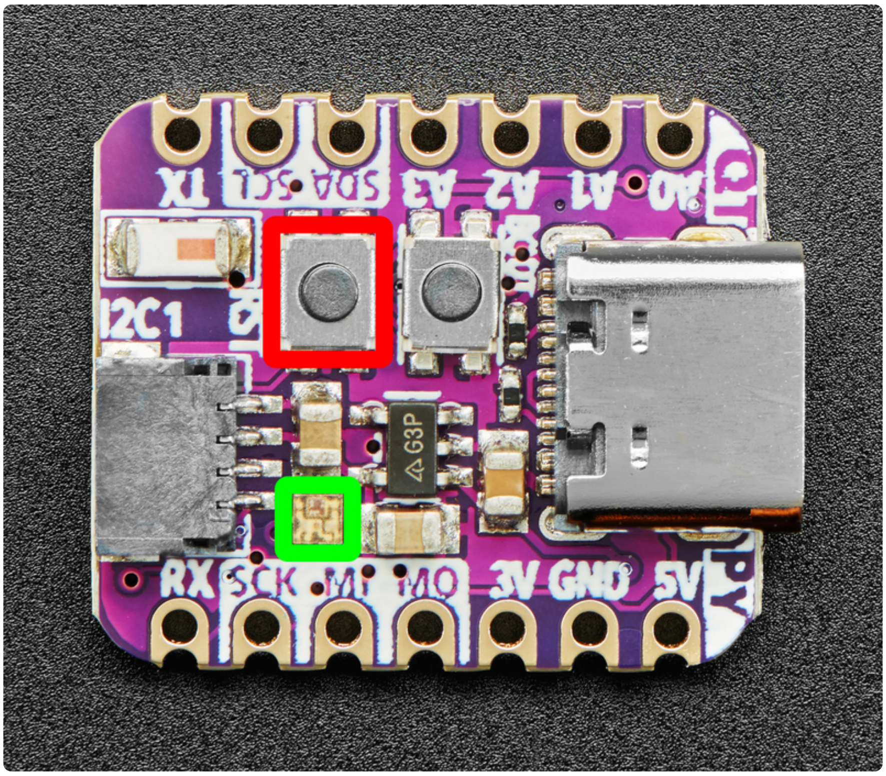
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/XCk



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

Save it wherever is convenient for you.



The board above has a chip antenna, not the u.Fl connector, but the process is the same.

Plug your board into your computer, using a known-good data-sync cable, directly, or via an adapter if needed.

Click the **reset** button once (highlighted in red above), and then click it again when you see the **RGB status LED(s)** (highlighted in green above) turn purple (approximately half a second later). Sometimes it helps to think of it as a "slow double-click" of the reset button.

If you do not see the LED turning purple, you will need to reinstall the UF2 bootloader. See the **Factory Reset** page in this guide for details.

On some very old versions of the UF2 bootloader, the status LED turns red instead of purple.

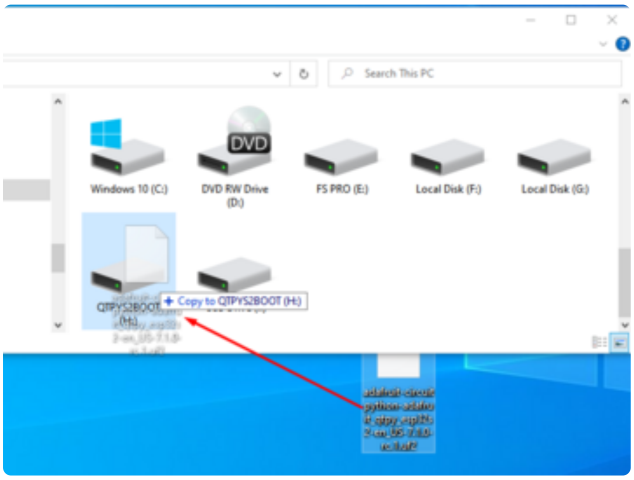
For this board, tap reset and wait for the LED to turn purple, and as soon as it turns purple, tap reset again. The second tap needs to happen while the LED is still purple.

Once successful, you will see the **RGB status LED(s)** turn green (highlighted in green above). If you see red, try another port, or if you're using an adapter or hub, try without the hub, or different adapter or hub.

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

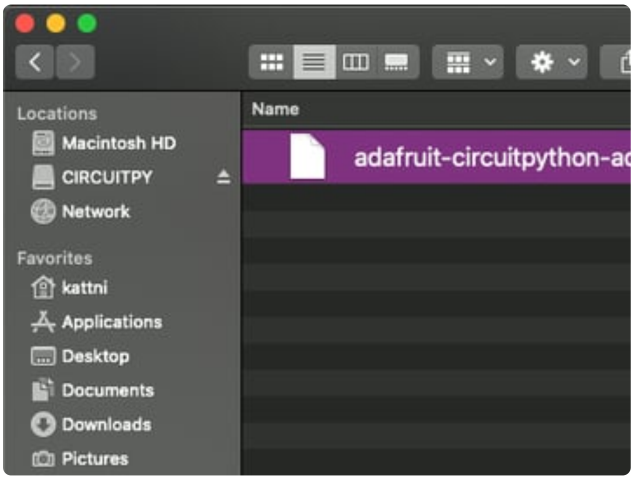
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**

If after several tries, and verifying your USB cable is data-ready, you still cannot get to the bootloader, it is possible that the bootloader is missing or damaged. Check out the Factory Reset page for details on resolving this issue.



You will see a new disk drive appear called **QTPYS2BOOT**.

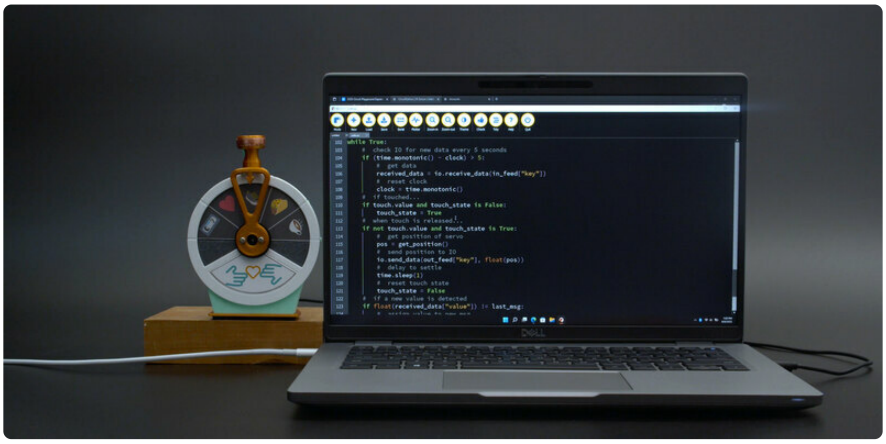
Drag the **adafruit_circuitpython_etc.uf2** file to **QTPYS2BOOT**.



The **BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it!

Code the Two Way Telegraph



Once you've finished setting up your QT Py ESP32-S2 with CircuitPython, you can access the code and necessary libraries by downloading the Project Bundle.

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

```
# SPDX-FileCopyrightText: 2022 Liz Clark for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import ssl
import board
import touchio
import pwmio
from analogio import AnalogIn
import adafruit_requests
import socketpool
import wifi
from adafruit_io.adafruit_io import IO_HTTP, AdafruitIO_RequestError
from simpleio import map_range
from adafruit_motor import servo

#  select which display is running the code
servo_one = True
#  servo_two = True

try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise
#  connect to adafruitio
aio_username = secrets["aio_username"]
aio_key = secrets["aio_key"]

print("Connecting to %s" % secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print("Connected to %s!" % secrets["ssid"])

pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())
# Initialize an Adafruit IO HTTP API object
io = IO_HTTP(aio_username, aio_key, requests)
```

```
#  pylint: disable=undefined-variable
#  disabling undefined-variable for ease of comment/uncomment
#  servo_one or servo_two at top for user

#  setup for display 1
if servo_one:
    #  servo calibration values
    CALIB_MIN = 15708
    CALIB_MAX = 43968
    #  create feeds
    try:
        # get feed
        out_feed = io.get_feed("touch-1")
        in_feed = io.get_feed("touch-2")
    except AdafruitIO_RequestError:
        # if no feed exists, create one
        out_feed = io.create_new_feed("touch-1")
        in_feed = io.create_new_feed("touch-2")
#  setup for display 2
if servo_two:
    CALIB_MIN = 15668
    CALIB_MAX = 43550
    try:
        # get feed
        out_feed = io.get_feed("touch-2")
        in_feed = io.get_feed("touch-1")
    except AdafruitIO_RequestError:
        # if no feed exists, create one
        out_feed = io.create_new_feed("touch-2")
        in_feed = io.create_new_feed("touch-1")

received_data = io.receive_data(in_feed["key"])

# Pin setup
SERVO_PIN = board.A1
FEEDBACK_PIN = board.A2
touch = touchio.TouchIn(board.TX)

#  angles for servo
ANGLE_MIN = 0
ANGLE_MAX = 180

# servo setup
pwm = pwmio.PWMOut(SERVO_PIN, duty_cycle=2 ** 15, frequency=50)
servo = servo.Servo(pwm)
servo.angle = None

# setup feedback
feedback = AnalogIn(FEEDBACK_PIN)

#  position finder function for servo
def get_position():
    return map_range(feedback.value, CALIB_MIN, CALIB_MAX, ANGLE_MIN, ANGLE_MAX)

#  touch debounce
touch_state = False
#  new_msg value
new_msg = None
#  last_msg value
last_msg = None
#  time.monotonic() holder for pinging IO
clock = 5

while True:
    #  check IO for new data every 5 seconds
    if (time.monotonic() - clock) > 5:
        #  get data
        received_data = io.receive_data(in_feed["key"])
```

```
        #  reset clock
        clock = time.monotonic()
    #  if touched...
    if touch.value and touch_state is False:
        touch_state = True
    #  when touch is released...
    if not touch.value and touch_state is True:
        #  get position of servo
        pos = get_position()
        #  send position to IO
        io.send_data(out_feed["key"], float(pos))
        #  delay to settle
        time.sleep(1)
        #  reset touch state
        touch_state = False
    #  if a new value is detected
    if float(received_data["value"]) != last_msg:
        #  assign value to new_msg
        new_msg = float(received_data["value"])
        #  set servo angle
        servo.angle = new_msg
        #  quick delay to settle
        time.sleep(1)
        #  release servo
        servo.angle = None
        #  log msg
        last_msg = new_msg
```

Upload the Code and Libraries to the QT Py ESP32-S2

After downloading the Project Bundle, plug your QT Py ESP32-S2 into the computer's USB port with a known good USB data+power cable. You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called **CIRCUITPY**. Unzip the folder and copy the following items to the QT Py ESP32-S2's **CIRCUITPY** drive.

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

Your QT Py ESP32-S2 **CIRCUITPY** drive should look like this after copying the **lib** folder and the **code.py** file.

secrets.py

You will need to create and add a **secrets.py** file to your **CIRCUITPY** drive. Your **secrets.py** file will need to include the following information:

```
secrets = {
    'ssid' : 'YOUR-SSID-HERE',
    'password' : 'YOUR-SSID-PASSWORD-HERE',
    'aio_username' : 'YOUR-AIO-USERNAME-HERE',
    'aio_key' : 'YOUR-AIO-KEY-HERE',
    }
```

Your **secrets.py** file will have your Adafruit IO username and key. Reference this guide page (https://adafru.it/10tA) for steps on how to grab this information from your Adafruit IO account.

Analog Servo Calibration

Analog Servo Calibration with CircuitPython

https://adafru.it/10tB

Analog servo motors make this project possible, but they work best after they've been calibrated. This CircuitPython calibration code (https://adafru.it/10tB) from the Analog Feedback Servos Learn Guide (https://adafru.it/dtE) will tell you your analog servo's minimum and maximum values.

Run this code with both of your analog servos and record the results for the **code.py** file for this project as the `CALIB_MIN` and `CALIB_MAX` values on lines 46 and 47 for the first display and lines 59 and 60 for the second display.

```
#  setup for display 1
if servo_one:
    #  servo calibration values
    CALIB_MIN = 15708
    CALIB_MAX = 43968
...
#  setup for display 2
if servo_two:
    CALIB_MIN = 15668
    CALIB_MAX = 43550
```

How the CircuitPython Code Works

After importing the libraries, you'll leave either `servo_one = True` or `servo_two = True` uncommented, depending on which of the two displays the code is running for.

```
#  select which display is running the code
servo_one = True
#  servo_two = True
```

Next, depending on which variable is `True`, the analog servo's `CALIB_MIN` and `CALIB_MAX` calibration values are set. Then, the `out_feed` and `in_feed` are setup. If the feeds do not exist in your Adafruit IO account yet, then they will be created with `io.create_new_feed()`.

```
#  setup for display 1
if servo_one:
    #  servo calibration values
    CALIB_MIN = 15708
    CALIB_MAX = 43968
    #  create feeds
    try:
        # get feed
        out_feed = io.get_feed("touch-1")
        in_feed = io.get_feed("touch-2")
    except AdafruitIO_RequestError:
        # if no feed exists, create one
        out_feed = io.create_new_feed("touch-1")
        in_feed = io.create_new_feed("touch-2")
#  setup for display 2
if servo_two:
    CALIB_MIN = 15668
    CALIB_MAX = 43550
    try:
        # get feed
        out_feed = io.get_feed("touch-2")
        in_feed = io.get_feed("touch-1")
    except AdafruitIO_RequestError:
        # if no feed exists, create one
        out_feed = io.create_new_feed("touch-2")
        in_feed = io.create_new_feed("touch-1")
```

In the loop, Adafruit IO is polled every `5` seconds to check the current value in the `in_feed`. This is the feed that the opposing servo is sending data to.

```
#  check IO for new data every 5 seconds
    if (time.monotonic() - clock) &gt; 5:
        #  get data
        received_data = io.receive_data(in_feed["key"])
        #  reset clock
        clock = time.monotonic()
```

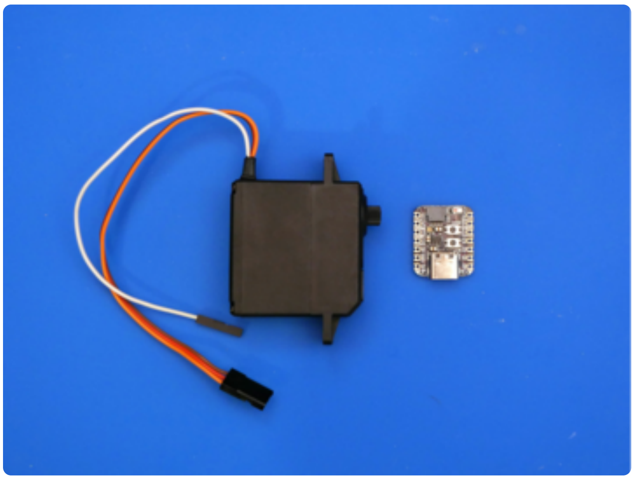
If the incoming data has changed from the previous value, then the servo's position updates to the new angle.

```
#  if a new value is detected
    if float(received_data["value"]) != last_msg:
        #  assign value to new_msg
        new_msg = float(received_data["value"])
        #  set servo angle
        servo.angle = new_msg
        #  quick delay to settle
        time.sleep(1)
        #  release servo
        servo.angle = None
        #  log msg
        last_msg = new_msg
```

When the telegraph's touch input is released, the position of the servo is sent to the `out_feed` and the `touch_state` for debouncing is reset. By having two feeds, each telegraph build has one feed it is publishing to and one feed it is listening to.

```
#  if touched...
    if touch.value and touch_state is False:
        touch_state = True
    #  when touch is released...
    if not touch.value and touch_state is True:
        #  get position of servo
        pos = get_position()
        #  send position to IO
        io.send_data(out_feed["key"], float(pos))
        #  delay to settle
        time.sleep(1)
        #  reset touch state
        touch_state = False
```

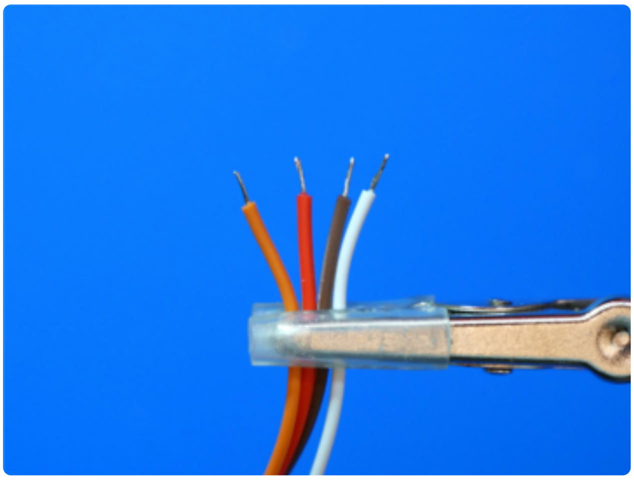
Wiring



QT Py and Servo Prep

Get the QT Py and analog feedback servo ready to wire.

The jumper wires feature connectors that will be removed using wire cutters.

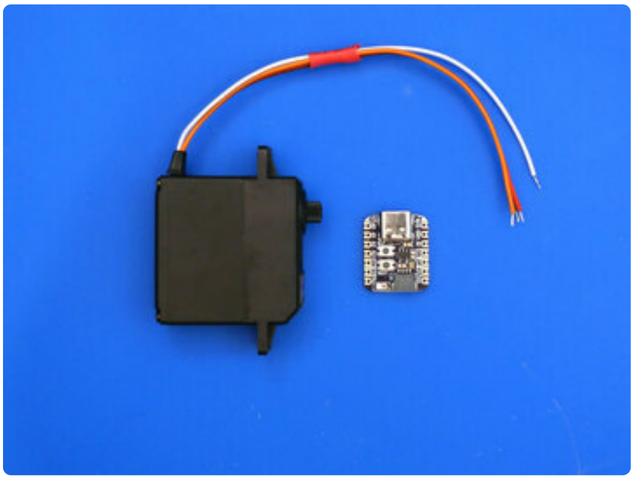


Tin Wires from Servo

Using wire stripper, remove a bit of insulation from each wire from the analog feedback servo.

Third helping hands can help keep the wires steady while soldering.

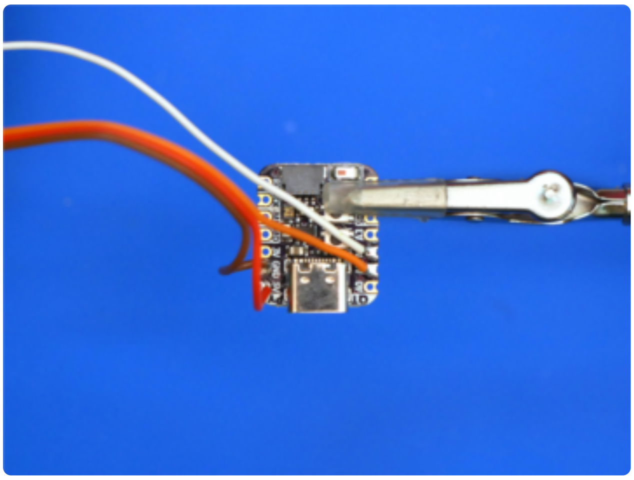
Tin each wire by adding a bit of solder to prevent the strands of wire from fraying.



Prepped Servo Wires

The wires from the analog feedback servo will be soldered to the pin on the QT Py.

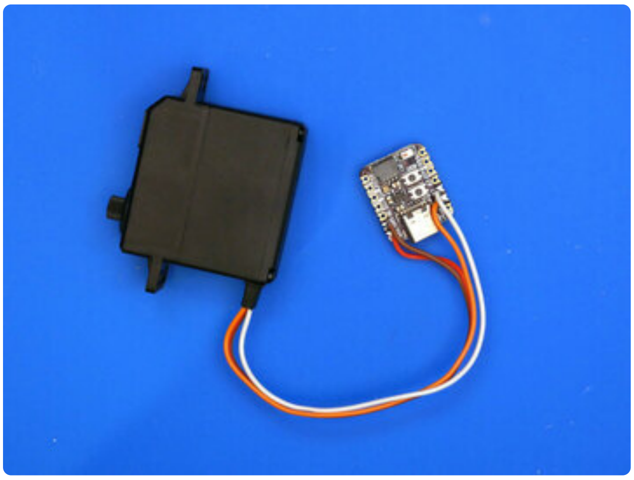
An optional bit of heat shrink tubing can be used to keep the four wires from the analog feedback servo together.



Solder Wires to QT Py

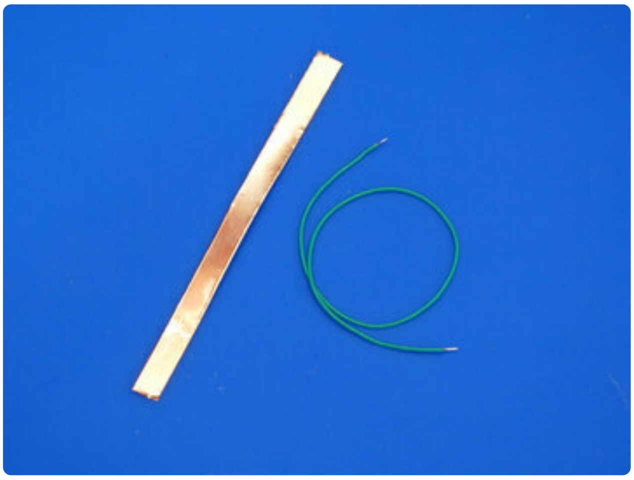
Connect the four wires from analog feedback servo to the following pins on the QT Py.

Red wire to 5V pin
Brown wire to GND pin
Orange wire to A1 pin
White wire to A2 pin



Connected Servo

Take a moment to inspect the wiring to ensure proper connections.

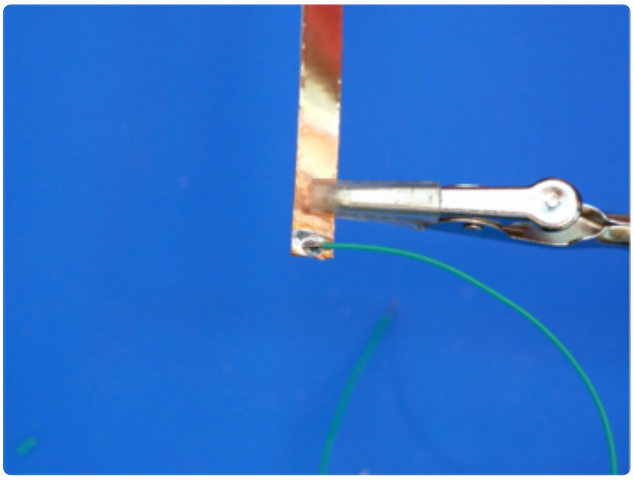


Copper and Wire

Get the copper tape and a piece of 30AWG wire ready to connect together.

Measure and cut a piece of copper tape about 3in(76mm) in length.

Measure and cut a piece of wire to about 5in (127mm) in length.



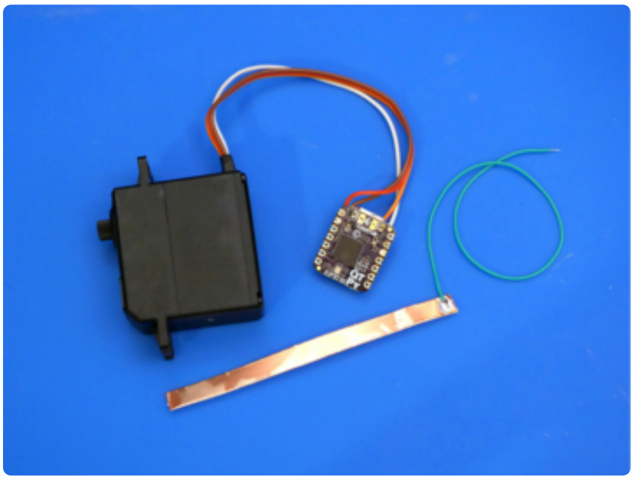
Solder Wire to Copper

Using wire strippers, remove a bit of insulation from both ends of the wire.

Tin both ends of the wire by adding a bit of solder.

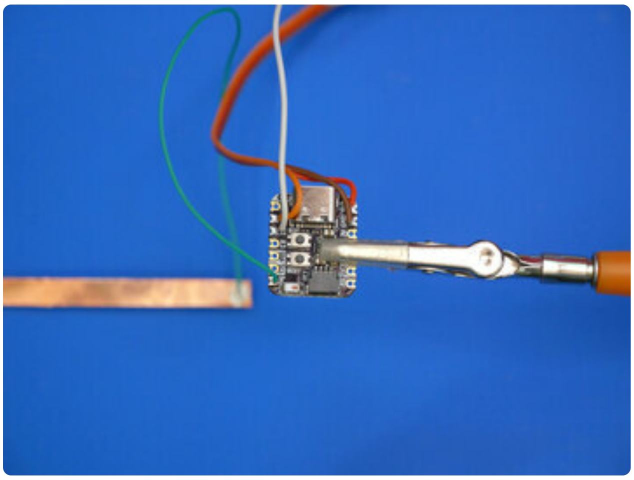
Tin one end of the copper tape by adding a small amount of solder.

Solder the wire to the piece of copper tape by heating up the tinned area.



Connect Copper Tape

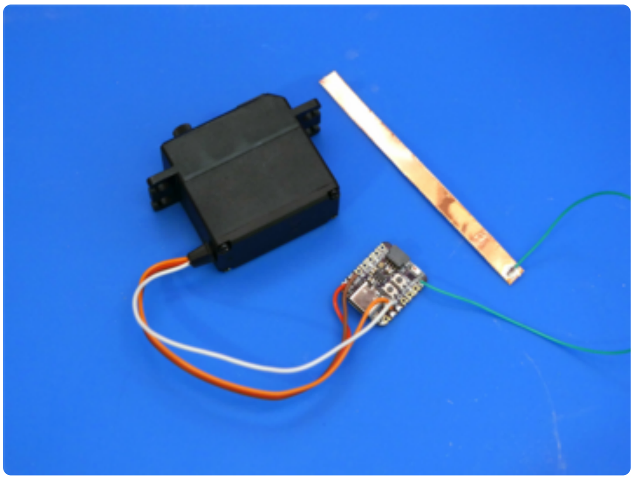
Take a moment to inspect the copper tape and wire to ensure they have been properly soldered together.



Solder Copper Tape Wire

Connect the wire from the piece of copper tape to the TX pin on the QT Py.

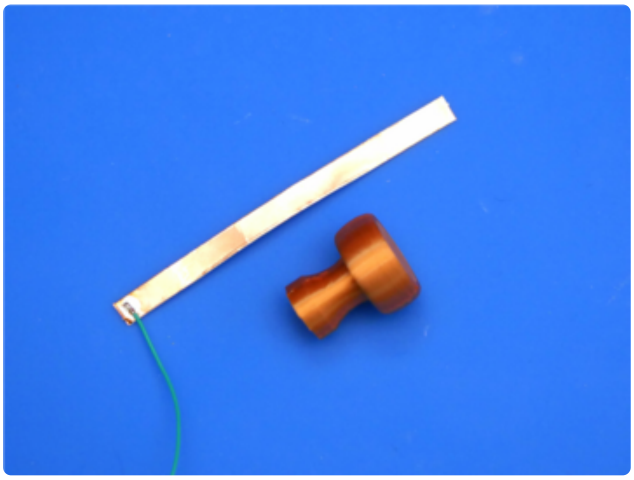
A helping hands tool can keep things sturdy while soldering.



Finished Circuit

Take a moment to inspect all of the wires in the circuit to ensure everything has been properly soldered.

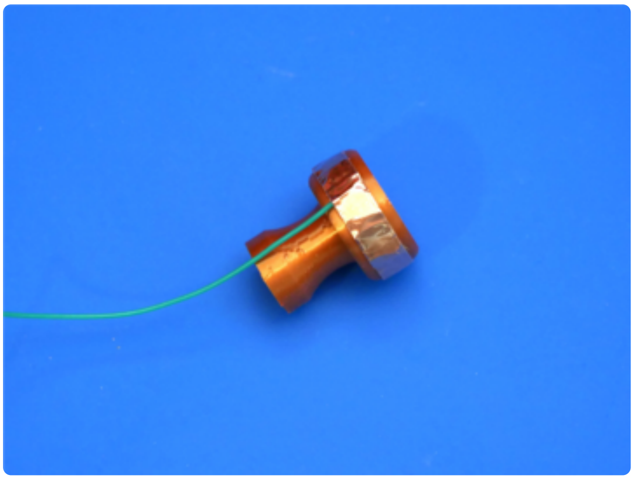
Assembly



Attach Copper Tape

The piece of copper tape will be wrapped around the edge of the 3D printed handle.

Remove the protective backing from the piece of copper tape by carefully peeling it off.



Wrapping Copper Tape

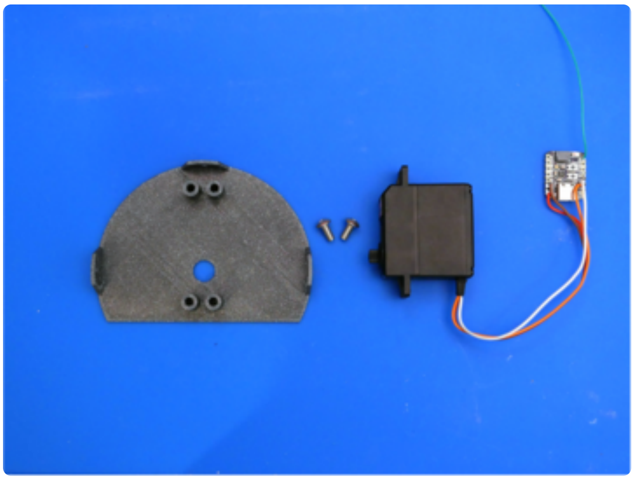
Place the piece of copper tape over the edge of the 3D printed handle.

Slowly wrap the tape around the handle so it overlaps the wire and solder joint.

Assemble Handle

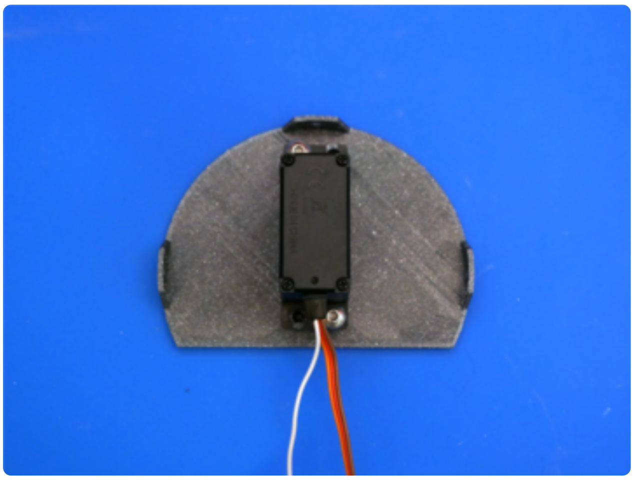
Use an M3x6mm screw to secure the 3D printed handle to the arrow attachment.

Use the circular servo horn (included with the analog feedback servo) and secure it to the 3D printed arrow attachment using the included hardware screws.



Attaching Servo

The analog feedback servo is secured to the 3D printed plate using two (or four) M4 x 6mm long screws.

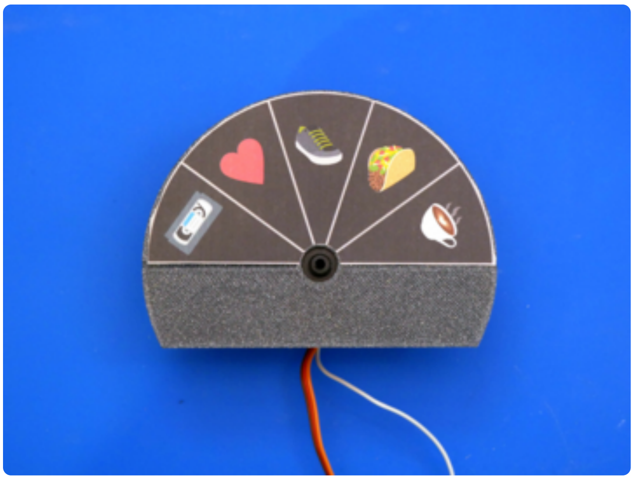


Secure Servo

Orient the analog feedback servo with the 3D printed plate so the center hole is lined up with the shaft of the servo.

Place the analog feedback servo over the 3D printed plate with the mounting tabs aligned to the built-in standoffs.

Insert and fasten the M4 screws to secure the servo to the 3D printed plate.



Adhere Label

Using scissors, carefully cut out the sticker label and remove the protective backing.

Begin attaching the label to the front of the 3D printed plate making sure the edges line up properly.

Attach Bezel, Frame and Handle

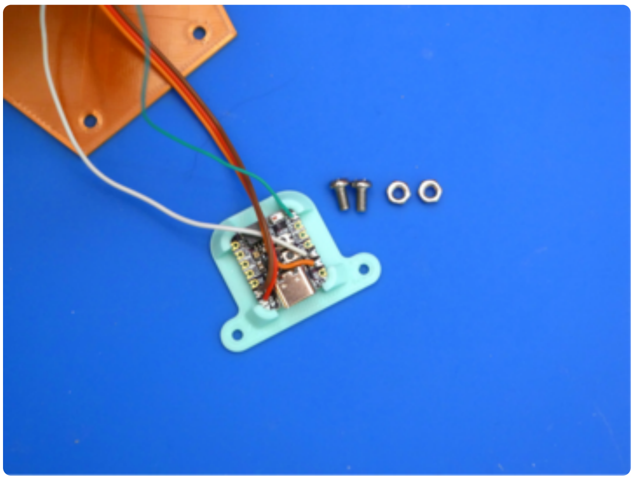
The 3D printed frame snap fits over the 3D printed plate with the tabs lined up to the bumps.

The 3D printed bezel snap fits over the 3D printed frame with the notches lined up.

The handle assembly is fitted over the shaft of the analog feedback servo.

Secure the servo horn to the shaft using the included hardware screw.

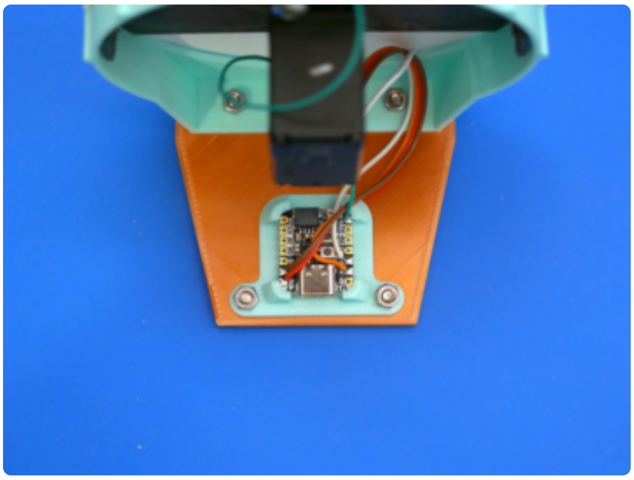
The servo horn will need to be refitted onto the shaft a few times in order to properly calibrate the position of the handle.



Secure QT Py to Holder

The QT Py is snap fitted onto the 3D printed holder. Use flush snips to remove any bits of wire underneath the pins on the QT Py.

Insert the QT Py PCB at an angle under two of the clips near the back and then slightly bend the holder to snap fit the front clips over the PCB.

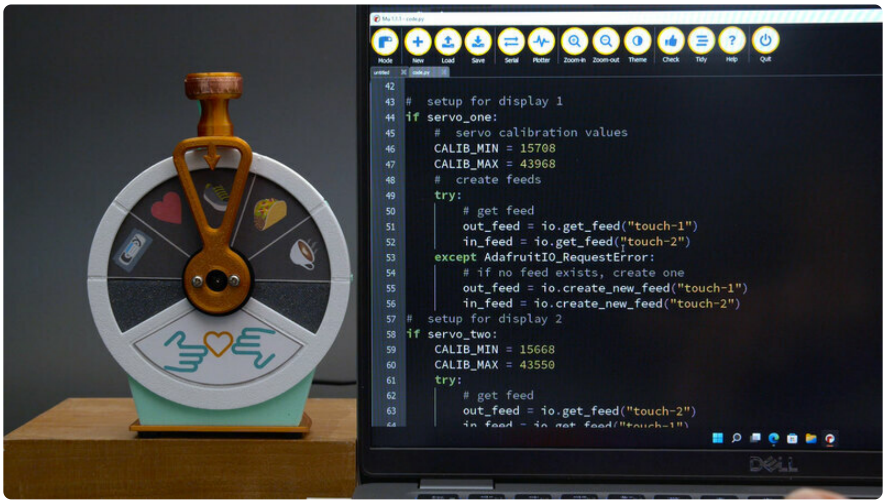


Secure Base Plate and Holder

Use 2x M3x8mm screws and hex nuts to secure the 3D printed base plate to the 3D printed frame.

Place the QT Py holder over the base plate and use two M3x8mm screws and hex nuts to secure them together.

Usage



After building two of the telegraphs, calibrate the analog servos by running the CircuitPython calibration code (https://adafru.it/10tB). Make note of each servo's minimum and maximum offset values. Then, update the **code.py** file with the values as noted on the Code the Two Way Telegraph page of this guide (https://adafru.it/10tC).

Power up the telegraphs and you're ready to start sending messages!

On one of the telegraphs, pull the servo's lever to an emoji while touching the copper tape capacitive touch point.

After releasing, the other telegraph will move its servo lever to point at the same emoji.

This can be a fun way to communicate quietly within the same household or long distance to let someone know you're thinking of them. You can also update the graphics to better suit your needs.