Table of Contents

Overview 3
• Using PyLeap
• Parts
• Additional Tools

Gathering the Components 6
• ESP32-S3 TFT display
• ADXL345 Accelerometer
• Battery

Building the Boxing Glove Tracker 8

Get Started with Adafruit IO 9
• I have an Adafruit.com Account already
• Create an Adafruit Account (for Adafruit IO)

Creating a Feed 12

CircuitPython 14
• CircuitPython Quickstart

WiFi Pairing 16
• Setup WiFi Credentials

PyLeap Boxing Glove Tracker Code 20
• Acquiring WiFi credentials
• Connecting to WiFi
• Creating a Socket Pool
• Initializing a MQTT Client
• Connecting to Adafruit IO
• Adding a threshold
• Accessing the Accelerometer
• Accessing the built-in battery monitor
• Setting the Current Time
• Storing Acceleration Data
• Displaying Collision Strength Data in Adafruit IO

Using PyLeap 26

Attaching the Components 27

Use 30
Overview

Track your punches with precision on Adafruit IO using this accelerometer-powered Boxing Glove tracker with an ESP32-S2 or ESP32-S3 TFT! In this guide we'll just refer to the ESP32-S3 TFT.

This project allows you to build your custom Boxing Glove Tracker using a Feather ESP32-S3 TFT running CircuitPython with an ADXL345 accelerometer. The ESP32-S3 TFT reads accelerometer data to measure the force of punches you throw.

The data collected by the accelerometer will then be sent to Adafruit IO, where you can visualize and analyze your training performance over time.
Using PyLeap

You can quickly transfer this project to your ESP32-S3 TFT via a WiFi connection using the PyLeap app.

PyLeap is a free app available for iOS, iPad and Android devices. It can be downloaded from the Apple App Store or Google Play Store. It allows users to easily download code files and assets and transfer them to their Adafruit devices using Bluetooth Low Energy (BLE) or WiFi.

This project is perfect for folks that want to enter the electronics and IoT world. It’s also handy for anyone who trains in boxing and wants to track their progress and improve their technique! Below are some parts you’ll need to have before starting this project.

Let's get started!
Parts

**Adafruit ESP32-S2 TFT Feather - 4MB Flash, 2MB PSRAM, STEMMA QT**
We've got a new machine here at Adafruit, it can uncover your deepest desires. Don't believe me? I'll turn it on right now to prove it to you! What, you want unlimited...
https://www.adafruit.com/product/5300

**ADXL345 - Triple-Axis Accelerometer (+-2g/4g/8g/16g) w/ I2C/SPI**
Filling out our accelerometer offerings, we now have the really lovely digital ADXL345 from Analog Devices, a triple-axis accelerometer with digital I2C and SPI interface breakout. We...
https://www.adafruit.com/product/1231

**STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long**
This 4-wire cable is a little over 100mm / 4" long and fitted with JST-SH female 4-pin connectors on both ends. Compared with the chunkier JST-PH these are 1mm pitch instead of...
https://www.adafruit.com/product/4210
Lithium Ion Battery - 3.7V 2000mAh
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...

Pink and Purple Woven USB A to USB C Cable - 1 meter long
This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also made for USB C for our modernized breakout boards, Feathers, and...
https://www.adafruit.com/product/5153

Additional Tools
Some additional supplies: an X-acto knife, scissors, and a sewing kit.

Gathering the Components
We will use a few components to build our Boxing Glove tracker:

1 x ESP32-S3 with a TFT display ()

1 x ADXL345 accelerometer ()

1 x Lithium-ion 3.7 volt battery ()
ESP32-S3 TFT display

The ESP32-S3 TFT will serve as the brain of the project, collecting data from the accelerometer and displaying it for visualization on the TFT display.

The TFT display will display the data from the accelerometer, battery percentage, and if the board is connected to the internet.

For more information on the ESP32-S2 TFT Feather, check out this Learn Guide!

ADXL345 Accelerometer

The ADXL345 accelerometer will be used to measure the acceleration and impact of the glove.

The accelerometer measures the acceleration of the gloves in all three dimensions (x, y, and z). This information can then be used to track the movement and impact of the gloves.

This accelerometer comes with STEMMA QT connectors so that we can plug the accelerometer into the ESP32-S2 TFT with a STEMMA QT / Qwiic JST SH 4-pin Cable ()

For more information on the accelerometer, check out this Learn Guide!
Battery

This project uses a thin and light Lithium Ion 3.7 volt battery to power the ESP32-S2 TFT board. I've decided to use this because it was easier to fit into the glove and prevented tearing the glove's leather.

For more information on Li-Ion batteries, check out this Learn Guide!

Building the Boxing Glove Tracker

Assembling the components

Here is a detailed, step-by-step guide for the assembling the components.

First, plug in your battery to the ESP32-S3 TFT Feather. ()
Then, plug your STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long () to the ESP32-S3 TFT then connect the other end of the cable to ADXL345 - Triple-Axis Accelerometer ().

Finally, setup your ESP32-S3 TFT by connecting it to your computer using a USB-C.

Now that all the components are connected, it's time to update the ESP32-S3 TFT to the latest version of CircuitPython.

Get Started with Adafruit IO

Adafruit IO is integrated with your adafruit.com account () so you don't need to create yet another online account! You need an Adafruit account to use Adafruit IO because we want to make sure the data you upload is available to only you (unless you decide to publish your data).

I have an Adafruit.com Account already

If you already have an Adafruit account, then you already have access to Adafruit IO. It doesn't matter how you signed up, your account will make all three available.
To access Adafruit IO, simply visit https://io.adafruit.com to start streaming, logging, and interacting with your data.

Create an Adafruit Account (for Adafruit IO)

An Adafruit account makes Adafruit content and services available to you in one place. Your account provides access to the Adafruit shop, the Adafruit Learning System, and Adafruit IO. This means only one account, one username, and one password are necessary to engage with the content and services that Adafruit offers.

If you do not have an Adafruit account, signing up for a new Adafruit account only takes a couple of steps.


Click the Sign Up button under the "Need An Adafruit Account?" title, below the Sign In section.

This will take you to the Sign Up page.

Fill in the requested information, and click the Create Account button.
SIGN UP

The best way to shop with Adafruit is to create an account which allows you to shop faster, track the status of your current orders, review your previous orders and take advantage of our other member benefits.

This takes you to your Adafruit Account home page. From here, you can access all the features of your account.

You can also access the Adafruit content and services right from this page. Along the top of the page, you'll see a series of links beginning with "Shop". To access any of these, simply click the link.

For example, to begin working with Adafruit IO, click the IO link to the right of the rest of the links. This is the same for the other links as well.

That's all there is to creating a new Adafruit account, and navigating to Adafruit IO.
Creating a Feed

Creating a Feed on Adafruit IO is a very simple process. Remember, if you are using WipperSnapper you do not need to create feeds - they are created for you automagically!

When you log in to your io.adafruit.com account, first visit the Overview page. This page lists information about your account, the data coming into IO from your devices, a list of WipperSnapper devices if you have any, and more.

To navigate to the feed page, click on the Feeds link on the header.
On the Feeds page, click the New Feed button.

A pop-up will appear, and you will be presented with two text inputs:

- **Name** - A short descriptive title of your data. Letters, numbers, dashes, underscores, or spaces are valid characters, and this field is required. The feed name must be unique to your account.
- **Description** - A long-form description of your data. This field is not required, but it’s useful to provide a detailed description of your feed.

After adding a feed name and description, click Create.
You'll see your new feed appear on the feed page, underneath the "Default" group.

CircuitPython

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

CircuitPython Quickstart

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

Download the latest version of CircuitPython for this board via circuitpython.org
Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.

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

Double-click the reset button (highlighted in red above), and 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.

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.

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.
You will see a new disk drive appear called FTHRS2BOOT.

Drag the adafruit_circuitpython_etc.uf2 file to FTHRS2BOOT.

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

That's it!

---

**WiFi Pairing**

**Setup WiFi Credentials**

PyLeap depends on adding WiFi credentials to detect Adafruit devices over WiFi.

Create a file with the name settings.toml in the root directory of the CIRCUITPY drive.
Add the following below:

The file contains the keys `CIRCUITPY_WIFI_SSID` and `CIRCUITPY_WIFI_PASSWORD`. Once these are defined, CircuitPython will automatically connect to the network and start the webserver used for the workflow.

The web server is on port 80 unless overridden by `CIRCUITPY_WEB_API_PORT`. It also enables MDNS.

Here is an example settings.toml:

```toml
# To auto-connect to Wi-Fi
CIRCUITPY_WIFI_SSID="YOUR-WIFI-NETWORK-NAME"
CIRCUITPY_WIFI_PASSWORD="YOUR-WIFI-NETWORK-PASSWORD"

# To enable modifying files from the web. Change this too!
# Leave the User field blank in the browser.
CIRCUITPY_WEB_API_PASSWORD="passw0rd"

CIRCUITPY_WEB_API_PORT=80
```

Drag the settings.toml file to the CIRCUITPY drive.

After this step - Disconnect your device from your computer and power it via LiPoly or AAA battery pack.
Open the PyLeap app. Select WiFi and scan for Adafruit devices.

After seeing the spinning Blinka animation, you should see a screen with WiFi devices found. Once you’ve found your device in the peripheral device list, press the Connect button to continue.
WiFi Devices Found

Rescan

Feather ESP32-S2 TFT

Device ID: cpy-94ca7a
Device IP:

Connect

If WiFi devices were not found, either rescan for WiFi devices or double check your WiFi credentials.
PyLeap Boxing Glove Tracker Code

```python
# SPDX-FileCopyrightText: 2023 Trevor Beaton for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import os
import time
import ssl
import math
import board
import microcontroller
import wifi
import socketpool
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_io.adafruit_io import IO_MQTT
from adafruit_adxl34x import ADXL345
from adafruit_lc709203f import LC709203F, PackSize

aio_username = os.getenv('aio_username')
aio_key = os.getenv('aio_key')

# Wi-Fi
try:
    print("Connecting to %s" % os.getenv('CIRCUITPY_WIFI_SSID'))
    wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'),
                       os.getenv('CIRCUITPY_WIFI_PASSWORD'))
    print("Connected to %s!" % os.getenv('CIRCUITPY_WIFI_SSID'))
# Wi-Fi connectivity fails with error messages, not specific errors, so this except
# is broad.
except Exception as e:  # pylint: disable=broad-except
    print("Failed to connect to WiFi. Error:", e, "\nBoard will hard reset in 30
seconds.")
```

©Adafruit Industries
```python
time.sleep(30)
microcontroller.reset()

# Create a socket pool
pool = socketpool.SocketPool(wifi.radio)

# Initialize a new MQTT Client object
mqtt_client = MQTT.MQTT(
    broker="io.adafruit.com",
    username= aio_username,
    password= aio_key,
    socket_pool=pool,
    ssl_context=ssl.create_default_context(),
)

# Initialize Adafruit IO MQTT "helper"
io = IO_MQTT(mqtt_client)

try:
    if not io.is_connected:
        # Connect the client to the MQTT broker.
        print("Connecting to Adafruit IO...")
        io.connect()
        print("Connected to Adafruit IO!")
    except Exception as e:  # pylint: disable=broad-except
        print("Failed to get or send data, or connect. Error: ", e,
            "\nBoard will hard reset in 30 seconds./n")
        time.sleep(30)
        microcontroller.reset()

threshold = 20  # set threshold value here
time_interval = 0.5  # set the time interval in seconds

# create the I2C bus object
i2c = board.STEMMA_I2C()

# For ADXL345
accelerometer = ADXL345(i2c)

# To monitor the battery
battery_monitor = LC709203F(i2c)
battery_monitor.pack_size = PackSize.MAH400

t0 = time.monotonic()

while True:
    x, y, z = accelerometer.acceleration
    t1 = time.monotonic()
    dt = t1 - t0

    total_acceleration = math.sqrt(x**2 + y**2 + z**2)
    if total_acceleration >= threshold:
        print("Battery Percent: {:.2f} %".format(battery_monitor.cell_percent))
        print("Collision strength: %.2f" % total_acceleration)
        io.publish("punch-strength", total_acceleration)
        # add code here to trigger an event or alert the user
        t0 = t1
        time.sleep(time_interval)
```

This code is doing several things:

- Connecting to the internet
- Connecting to Adafruit IO
- Gathering accelerometer data and battery percentage data
• Displaying accelerometer and battery data on the TFT
• Sending accelerometer data to a feed in Adafruit IO

Acquiring WiFi credentials

These few lines of code extract the values for the Adafruit IO username and Adafruit IO API key from the settings.toml file mentioned here.

The values for `aio_username` and `aio_key` are in the settings.toml file.

```python
aio_username = os.getenv('aio_username')
aio_key = os.getenv('aio_key')
```

Have you set up your Adafruit IO account yet? If not, you can check out the previous "Getting Started with Adafruit IO" page to get started!

Connecting to WiFi

This code block connects to a WiFi network using the `wifi.radio.connect` function and passes in the network’s SSID and password as arguments. The values of the SSID and password are read from environment variables `CIRCUITPY_WIFI_SSID` and `CIRCUITPY_WIFI_PASSWORD`.

```python
# Wi-Fi
try:
    print("Connecting to %s" % os.getenv('CIRCUITPY_WIFI_SSID'))
    wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'),
                       os.getenv('CIRCUITPY_WIFI_PASSWORD'))
    print("Connected to %s!" % os.getenv('CIRCUITPY_WIFI_SSID'))
# Wi-Fi connectivity fails with error messages, not specific errors, so this except is broad.
except Exception as e:  # pylint: disable=broad-except
    print("Failed to connect to WiFi. Error:", e, "\nBoard will hard reset in 30 seconds.")
    time.sleep(30)
    microcontroller.reset()
```

Creating a Socket Pool

Next, the code initializes a socket pool and an MQTT client for communicating with the Adafruit IO cloud service.
The `socketpool.SocketPool` function creates a pool of sockets for management of network connections. It takes the `wifi.radio` object as an argument to allow for network communication over the WiFi connection.

```python
pool = socketpool.SocketPool(wifi.radio)
```

**Initializing a MQTT Client**

The `MQTT.MQTT` function is then used to initialize a new MQTT client object, which will be used to send and receive messages with the Adafruit IO cloud service. This function takes several arguments:

- `broker`: the hostname of the MQTT broker (Adafruit IO in this case).
- `username` and `password`: the credentials for authenticating with the Adafruit IO service are stored in the settings.toml file.
- `socket_pool`: the socket pool created in the previous step.
- `ssl_context`: an SSL context created using the `create_default_context` method from the `ssl` module. This provides secure communication between the device and the Adafruit IO service.

Finally, the `IO_MQTT` function is used to initialize an "Adafruit IO MQTT helper", a convenient wrapper around the MQTT client that provides a simple API for working with Adafruit IO.

```python
mqtt_client = MQTT.MQTT( 
    broker="io.adafruit.com", 
    username= aio_username, 
    password= aio_key, 
    socket_pool=pool, 
    ssl_context=ssl.create_default_context(),
)

# Initialize Adafruit IO MQTT "helper"
io = IO_MQTT(mqtt_client)
```

**Connecting to Adafruit IO**

Now to check the connection status of Adafruit IO, and if it is not connected, it is trying to connect to the MQTT broker using the `io.connect()` method.

If the connection to Adafruit IO fails, an error message is printed, and the board will hard reset after 30 seconds.
The `except` block is broad to catch all exceptions and the `broad-except`.

```python
try:
    # If Adafruit IO is not connected...
    if not io.is_connected:
        # Connect the client to the MQTT broker.
        print("Connecting to Adafruit IO...")
        io.connect()
        print("Connected to Adafruit IO!")
    except Exception as e:  # pylint: disable=broad-except
        print("Failed to get or send data, or connect. Error:", e,
        "\nBoard will hard reset in 30 seconds.")
        time.sleep(30)
        microcontroller.reset()
```

Adding a threshold

These lines of code set two variables, `threshold` and `time_interval`.

- **`threshold`** is set to 20; this value filters out movement that might trigger punch detection.
- **`time_interval`** is set to 0.5, which is the time interval in seconds; I added this to limit the amount of data points created. You can change this to cater to your Adafruit IO plan.

```python
threshold = 20 # set threshold value here
time_interval = 0.5 # set the time interval in seconds
```

Accessing the Accelerometer

Now to create an I2C bus object using the `board.STEMMA_I2C()` method, and then create an instance of an ADXL345 accelerometer using that I2C bus object.

The `ADXL345` class is imported from an external library and represents an ADXL345 3-axis accelerometer, which can measure acceleration along the X, Y, and Z axes.

The accelerometer is being initialized using the I2C bus object so that the accelerometer can communicate with the microcontroller board over the I2C bus.

```python
# create the I2C bus object
i2c = board.STEMMA_I2C()
# For ADXL345
accelerometer = ADXL345(i2c)
```
Accessing the built-in battery monitor

Next to create an instance of the LC709203F class named `battery_monitor` and set the `pack_size` attribute of the `battery_monitor` object to `PackSize.MAH400`.

The `pack_size` attribute is used to set the capacity of the battery being monitored by the LC709203F. In this case, `PackSize.MAH400` is used to specify a battery capacity of 400 milliampere-hours (mAh).

```python
battery_monitor = LC709203F(i2c)
battery_monitor.pack_size = PackSize.MAH400
```

Setting the Current Time

This line `t0 = time.monotonic()` sets the variable `t0` to the current time as returned by the `monotonic` method of the `time` module. The `monotonic` method returns the number of seconds that have passed since an unspecified time in the past, guaranteed never to go back even if the system time is changed.

```python
t0 = time.monotonic()
```

Storing Acceleration Data

These few lines use a `while` loop to read the `x`, `y`, and `z` components of the accelerometer’s acceleration and then store the variables.

```python
while True:
    x, y, z = accelerometer.acceleration
    t1 = time.monotonic()
    dt = t1 - t0
    total_acceleration = math.sqrt(x**2 + y**2 + z**2)
```

The sleep interval between two acceleration readings is determined by the `time_interval` variable. The code runs in a continuous loop until manually stopped.

```python
if total_acceleration >= threshold:
    print("Battery Percent: {:.2f} \%").format(battery_monitor.cell_percent)
    print("Collision strength: %.2f \% total_acceleration")
    io.publish("punch-strength", total_acceleration)
    # add code here to trigger an event or alert the user
    t0 = t1
    time.sleep(time_interval)
```
Displaying Collision Strength Data in Adafruit IO

In this `if`-statement, if the total acceleration is greater than or equal to the threshold, the battery percent and total acceleration will be displayed on the TFT, and the total acceleration value will be published to a feed named "punch-strength" in Adafruit IO.

Great! Let's attach components to the glove!

Using PyLeap

To upload this project to your ESP32-S2 TFT, select the project in the project list.

Browse available WiFi PyLeap Projects

- Boxing Glove Tracker
- ESP32-S2 NeoPixel LED
- I2C: On Board

Then, once the project cell has collapsed, press the "Run it" button to download and transfer the project over to your ESP32-S2 TFT.
It's that easy! Now, let's set up an Adafruit IO account.

Attaching the Components

Disconnect the Battery from the ESP32-S3 TFT before beginning

Before starting to pack the electronics into the glove, it is important to disconnect the battery from the ESP32-S3 TFT.

This is because the needle used to sew the ESP32-S3 TFT over the opening in the glove may hit the electronics, and it would not be safe to do so while the board is powered on.
Make a small incision in the wrist area of your boxing glove with an X-acto knife or pair of scissors. It should be 1 inch wide.

Prepare the Components

Once the battery is disconnected, you will want to pack the battery to the left and the accelerometer on the right into the incision you’ve made in the wrist area of the glove.

Check the diagram below for placement.
Make sure to add padding behind both electronics to secure them in place.

Also, leave some battery wire along with the connected JST sticking out of the opening for easy reconnection later.

Secure the Electronics

Now it's time to secure the electronics in place. With a needle and thread, sew the ESP32-S3 TFT over the opening, ensuring that the TFT covers the entire opening and that the edges are securely sewn to the glove.
This will keep the electronics in place and prevent them from shifting or coming loose during use. When your board is secure, plug the battery into the ESP32-S3 TFT.

Use

After making sure you are connected to the internet and Adafruit IO on your TFT display, throw a few air punches to trigger the collision detection.

If you are not able to activate the collision detection, try lowering the threshold in the code.py file shown below in the code:

```
threshold = 25 # set threshold value here
```

When a punch is thrown, and collision(s) are detected, you should see the accelerometer data along with the battery percentage shown on the TFT display.
After receiving confirmation that you are connected to Adafruit IO, you should see new data points saved and added to a line graph on your feed.

Be sure to customize this project however you'd like!