Last-Minute Halloween Accoutrements with HalloWing

Created by Phillip Burgess

https://learn.adafruit.com/last-minute-halloween-accoutrements-with-hallowing

Last updated on 2024-06-03 02:30:36 PM EDT

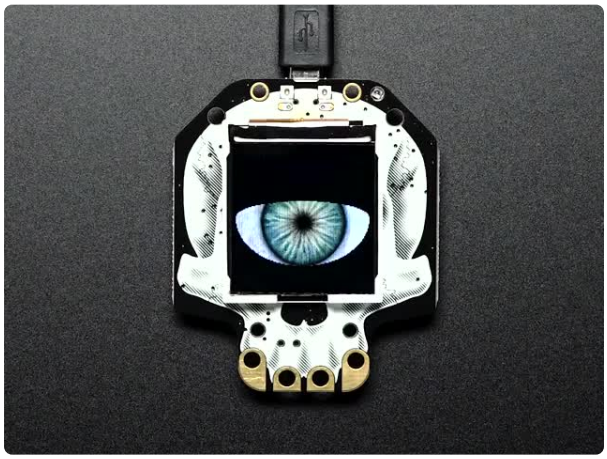
Table of Contents

Overview

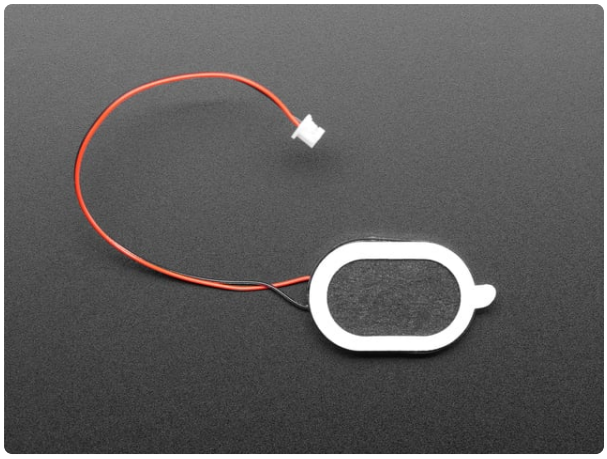
Oh noes! An 11th-hour invite to a Halloween function and you don't have a costume ready! Or maybe you want something discreet but fun when the work dress code doesn't permit a full werewolf getup.

HalloWing to the rescue! Our spooky little development board, plus a few extra tidbits from Adabox 009 (https://adafru.it/CEr) (or your parts collection) can quickly make an eye-catching accessory that no good host would turn you away for being under-dressed.

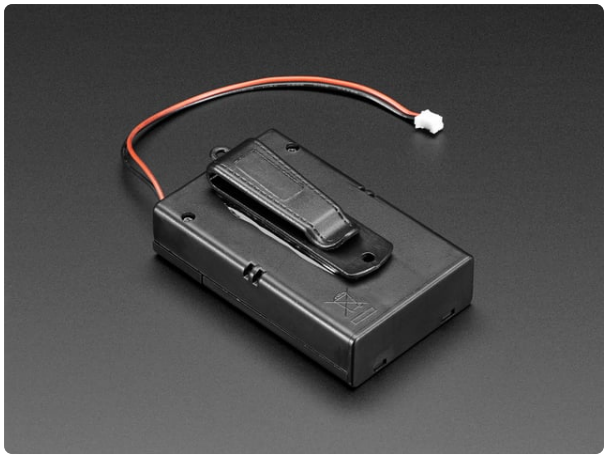
Parts



Adafruit HalloWing M0 Express
This is Hallowing..this is Hallowing... Hallowing! Hallowing! Are you the kind of person who doesn't...
https://www.adafruit.com/product/3900

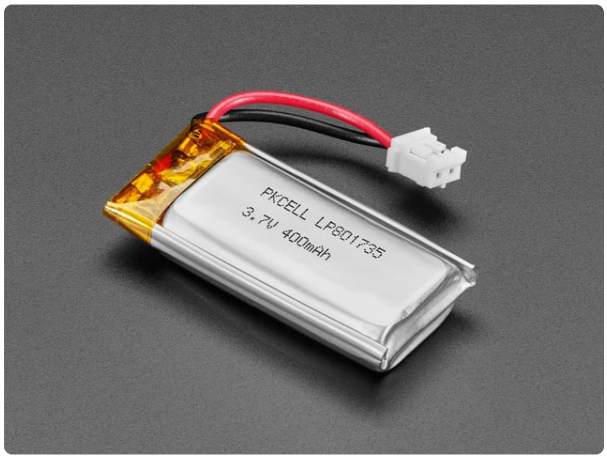


Mini Oval Speaker - 8 Ohm 1 Watt
Hear the good news! This wee speaker is a great addition to any audio project where you need 8 ohm impedance and 1W or less of power. We particularly like...
https://www.adafruit.com/product/3923



3 x AA Battery Holder with On/Off Switch, JST, and Belt Clip
This battery holder connects 3 AA batteries together in series for powering all kinds of projects. We spec'd these out because the box is compact, and 3 AA's add up to about...
https://www.adafruit.com/product/3287

- or -



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

- or -

HalloWing can run directly from a small **lithium-polymer battery** (such as the one included in Adabox 009 (http://adafru.it/3956)) for **an hour or two**, depending on what it's doing. If you want it to run for hours on end...all day or all night...it's real easy.

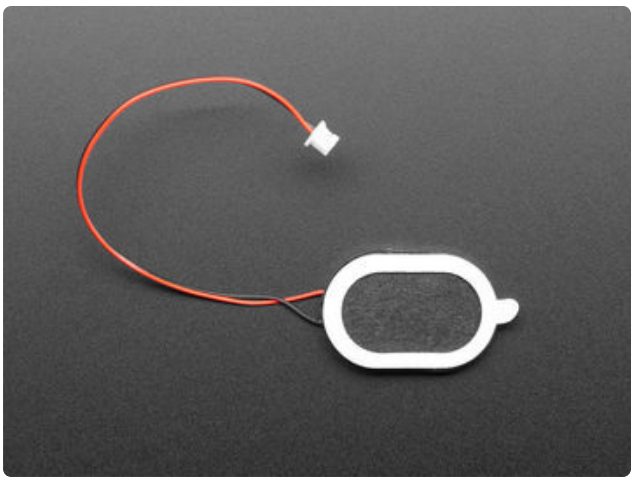
Simply use a pocket-sized **USB battery bank** (http://adafru.it/1959) (the sort you might already have for topping off your cell phone) and a USB A-to-microB cable. This is especially helpful for **NeoPixel** projects.

USB Battery Pack - 2200 mAh Capacity - 5V 1A Output
A smaller-sized rechargeable battery pack for your Raspberry Pi or Raspberry...
https://www.adafruit.com/product/1959

Making a Costume of Sound and Motion

It's natural to dwell on the visual elements of a costume. Consider for a moment though...the **surprise** of using **other senses**...sound, in this case...can provide a boost for a quickly-assembled outfit, or something that outwardly resembles conventional dress.

Here's a mini-project that shows HalloWing being used like a Star Trek communicator badge, or to make video game sounds any time you jump (or both, if you really want to confuse people).



This small 8-Ohm speaker is included in Adabox 009 or is available separately (http://adafru.it/3923). It features a tiny “PicoBlade” connector that plugs straight into Hallowing. You can use other small speakers but may need to solder wires directly to pads on the board.

There’s a tiny volume dial on HalloWing that can be adjusted with a screwdriver.

Our code uses **CircuitPython** (https://adafru.it/cpy-welcome) because it’s super easy to customize for different sounds or graphics.

If you’ve overwritten CircuitPython on your HalloWing board with another project (such as the eye or Minotaur Maze (https://adafru.it/CBj)), it’s easy to get it back. Follow the steps in this guide (https://adafru.it/CmJ) to get the standard CircuitPython firmware onto your board.

This project requires CircuitPython 4.0 or higher for loading BMP images (sound will work with 3.0 or higher) version of CircuitPython. Setup directions are the same as shown in that guide, only the .UF2 file is different:

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

https://adafru.it/Em7

Here’s the code we’ll be using. Click “**Download Project Bundle**” to download a ZIP file with this code and the project’s graphics and sound files and required libraries. After uncompressing, navigate to the folder matching your CircuitPython version, and copy all of the files (and the “lib” folder) to the **root directory** of the CIRCUITPY drive (i.e. not inside a folder...just **straight to the drive**).

```
# SPDX-FileCopyrightText: 2018 Phillip Burgess for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Jump & touch sound example for Adafruit Hallowing. Plays different sounds
in response to jumping and capacitive touch pads.
"""

import time
```

```
import board
import digitalio
import displayio
import audioio
import audiocore
import touchio
import neopixel

def load_wav(name):
    """
    Load a WAV audio file into RAM.
    @param name: partial file name string, complete name will be built on
                 this, e.g. passing 'foo' will load file 'foo.wav'.
    @return WAV buffer that can be passed to play_wav() below.
    """
    return audiocore.WaveFile(open(name + '.wav', 'rb'))

def play_wav(wav):
    """
    Play a WAV file previously loaded with load_wav(). This function
    "blocks," i.e. does not return until the sound is finished playing.
    @param wav: WAV buffer previously returned by load_wav() function.
    """
    AUDIO.play(wav)      # Begin WAV playback
    while AUDIO.playing: # Keep idle here as long as it plays
        pass
    time.sleep(1)        # A small pause avoids repeated triggering

TOUCH_WAV = load_wav('touch') # WAV file to play when capacitive pads touched
JUMP_WAV = load_wav('jump')   # WAV file to play when jumping
JUMP_THRESHOLD = 4.0          # Higher number = triggers more easily
IMAGEFILE = 'mario.bmp'       # BMP image to display

IS_HALLOWING_M4 = False

# Perform a couple extra steps for the HalloWing M4
try:
    if getattr(board, "CAP_PIN"):
        IS_HALLOWING_M4 = True
        # Create digitalio objects and pull low for HalloWing M4
        cap_pin = digitalio.DigitalInOut(board.CAP_PIN)
        cap_pin.direction = digitalio.Direction.OUTPUT
        cap_pin.value = False
    if getattr(board, "SPEAKER_ENABLE"):
        # Enable the Speaker
        speaker_enable = digitalio.DigitalInOut(board.SPEAKER_ENABLE)
        speaker_enable.direction = digitalio.Direction.OUTPUT
        speaker_enable.value = True
except AttributeError:
    pass

AUDIO = audioio.AudioOut(board.SPEAKER)  # Speaker

try:
    board.DISPLAY.auto_brightness = False
except AttributeError:
    pass
TOUCH1 = touchio.TouchIn(board.TOUCH1)  # Capacitive touch pads
TOUCH2 = touchio.TouchIn(board.TOUCH2)
TOUCH3 = touchio.TouchIn(board.TOUCH3)
TOUCH4 = touchio.TouchIn(board.TOUCH4)

# Set up accelerometer on I2C bus, 4G range:
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
if IS_HALLOWING_M4:
    import adafruit_msa301
    ACCEL = adafruit_msa301.MSA301(i2c)
```

```
else:
    import adafruit_lis3dh
    try:
        ACCEL = adafruit_lis3dh.LIS3DH_I2C(i2c, address=0x18) # Production board
    except ValueError:
        ACCEL = adafruit_lis3dh.LIS3DH_I2C(i2c, address=0x19) # Beta hardware
    ACCEL.range = adafruit_lis3dh.RANGE_4_G

try:
    board.DISPLAY.brightness = 0
    SCREEN = displayio.Group()
    board.DISPLAY.root_group = SCREEN

    # CircuitPython 7+ compatible
    BITMAP = displayio.OnDiskBitmap(IMAGEFILE)
    TILEGRID = displayio.TileGrid(BITMAP, pixel_shader=BITMAP.pixel_shader)

    SCREEN.append(TILEGRID)
    board.DISPLAY.brightness = 1.0   # Turn on display backlight
except (OSError, ValueError):
    pass

# If everything has initialized correctly, turn off the onboard NeoPixel:
PIXEL = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0)
PIXEL.show()

while True:
    # No freefall detect in LIS3DH library, but it's easily done manually...
    # poll the accelerometer and look for near-zero readings on all axes.
    X, Y, Z = ACCEL.acceleration
    A2 = X * X + Y * Y + Z * Z  # Acceleration^2 in 3space (no need for sqrt)
    if A2 < JUMP_THRESHOLD:
        # Freefall (or very close to it) detected, play a sound:
        play_wav(JUMP_WAV)
    elif TOUCH1.value or TOUCH2.value or TOUCH3.value or TOUCH4.value:
        # One of the capacitive pads was touched, play other sound:
        play_wav(TOUCH_WAV)
```

With the speaker, code and sound/graphics files all correctly installed, you'll see it does two (or three) things:

- When any of the capacitive pads (skull teeth) are touched, it emits a chirp sound like a Star Trek communicator. Add a magnet (http://adafru.it/1170) or pin back (http://adafru.it/2838), don a red shirt and black slacks, and ask your host for tea, Earl Grey, hot.
- If you jump while holding HalloWing (or hidden in a pocket or worn on a lanyard), it will make video game jumping sounds ... perhaps you have some overalls and a hat? Telling everyone "IT'S A ME!" never gets tired.
- If using a recent version of CircuitPython (4.0.0-alpha1 or later), the Hallowing display can show images ... easily switched to other themes if you'd like.

Customization and Use

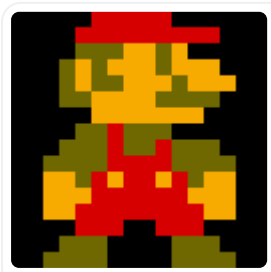
Customizing Images and Sounds

This line in the code (around line 40) specifies what image to load...it's **mario.bmp** by default:

```
IMAGEFILE = 'mario.bmp'       # BMP image to display
```

Change this to **badge.bmp** to use the other example image we provide:

```
IMAGEFILE = 'badge.bmp'       # BMP image to display
```



If you can create or convert (https://adafru.it/CED) other images to **128x128 pixel 24-bit color BMP** files, copy them to the **CIRCUITPY** drive and modify the value of `IMAGEFILE` to match the name. There are some other examples shown in this guide (https://adafru.it/CEC).

Sound files can also be customized. **16-bit mono PCM WAV** files (**22,050** sample rate **or less**) are ideal. These are specified just a few lines up in the code, around line 37:

```
TOUCH_WAV = load_wav('touch') # WAV file to play when capacitive pads touched
JUMP_WAV = load_wav('jump')   # WAV file to play when jumping
```

No need to specify the complete filename on these lines; the ".wav" is implied. (e.g. 'touch' will load the file touch.wav).

One line down, you can specify how **sensitive** the **jump detection** will be:

```
JUMP_THRESHOLD = 4.0          # Higher number = triggers more easily
```

This isn't any particular type of real-world units...just try higher or lower numbers if you even find it's necessary. Larger numbers will trigger more easily (i.e. you might

get some false triggers when walking or moving around), smaller numbers are more strict about jumps.

Changing Behavior

One last thing...you might find it silly that the code does both types of sounds... touching and jumping. Maybe with your own sound files you have an idea that could make use of both. But otherwise, it's unbecoming of a starship captain to be making beepy jumping noises, and vice-versa for Italian plumbers. Let's look at the last few lines of code...

```
    if A2 &lt; JUMP_THRESHOLD:
        # Freefall (or very close to it) detected, play a sound:
        play_wav(JUMP_WAV)
    elif TOUCH1.value or TOUCH2.value or TOUCH3.value or TOUCH4.value:
        # One of the capacitive pads was touched, play other sound:
        play_wav(TOUCH_WAV)
```

This first checks the **accelerometer** reading to see if we've jumped...and if not, it then proceeds to check the **capacitive touch** pads. If you just want **one or the other**, how might you change the code?

If you just want **jumping**, simply delete the latter three lines, so it's now just this:

```
    if A2 &lt; JUMP_THRESHOLD:
        # Freefall (or very close to it) detected, play a sound:
        play_wav(JUMP_WAV)
```

And if you only want the **touch pads**, delete the prior three lines, and change the "elif" to an "if":

```
    if TOUCH1.value or TOUCH2.value or TOUCH3.value or TOUCH4.value:
        # One of the capacitive pads was touched, play sound:
        play_wav(TOUCH_WAV)
```

You can probably see how the code might be modified to load multiple sounds and have each touch pad trigger a different one. I'll leave that to you as homework if you like. But I will recommend...don't over-do it! The sounds are funny but can quickly become tiresome...use your gags sparingly and get on with being a good party guest, don't drag it out into a whole PowerPoint slideshow (https://adafru.it/CEE).

Math (or the lack thereof)

I thought detecting jumps was going to be quite complicated and would lag well behind reality. But it turns out to be one of the simplest things. Using the onboard

accelerometer, we just have to check if the board is in free fall and play a sound accordingly.

Funny thing about free fall...the “fall” part makes us think of things on the way down, but that’s not necessarily true. Anything following a ballistic trajectory...that is, without propulsion and influenced only by gravity...is in free fall, even on the way up. Throw a ball, and the moment it leaves your hand (its source of propulsion), it’s on a ballistic trajectory. Jump, and the moment your feet leave the ground, it’s the same thing. (We can disregard air resistance as negligible here.) So we don’t have to sift through the accelerometer readings looking for spikes of acceleration...quite the opposite, just look for zero (or close to it):

```
    X, Y, Z = ACCEL.acceleration
    A2 = X * X + Y * Y + Z * Z  # Acceleration^2 in 3space (no need for sqrt)
    if A2 &lt; JUMP_THRESHOLD:
        # Freefall (or very close to it) detected, play a sound:
        play_wav(JUMP_WAV)
```

X, Y, Z refer to the **three axes** of the accelerometer. The orientation of the board **completely doesn’t matter** in this application, we just want the **magnitude** of the acceleration, in any direction...that’s what the multiplications and additions on the next line do...mostly...

Many situations would require taking the **square root** of that result to get the actual acceleration magnitude, in actual meters-per-second-squared real-world units. Square roots are relatively slow to calculate. Since we’re just comparing against a number, we can speed things up by leaving out the square root, and instead square the value we’re comparing against (JUMP_THRESHOLD). The default value of JUMP_THRESHOLD is 4.0, which means we’re really looking for an acceleration magnitude less than 2.0 meters/second^2 (normal acceleration due to gravity is 9.8 m/s^2, and perfect free fall would be 0.0...given our readings and our jumping may be a little sloppy, this is “close enough” to free fall with hopefully not too many false positives from just energetic walking around). It’s that simple.

Stomp and Roar!

A variation on the code makes it respond to footsteps...

In the video above, so we can hear it outside, the HalloWing’s speaker output has been wired into a portable amplified speaker worn on a belt...but for smaller indoor gatherings, the little oval speaker from Adabox 009 may suffice.

HalloWing can be worn on a lanyard, tucked in a pocket, or even ride along inside one's trick-or-treat candy bucket...it still senses steps and jumps!

It's not a commercial-grade step counter...it might miss some steps, or register false positives...quite fine though for casual Halloween shenanigans, for something that can be made and customized very quickly. Try it out!

As with the prior example, click "**Download Project Bundle**" to download a ZIP file with this code and the project's graphics and sound files and required libraries. After uncompressing, navigate to the folder matching your CircuitPython version, and copy all of the files (and the "lib" folder) to the **root directory** of the CIRCUITPY drive (i.e. not inside a folder...just **straight to the drive**).

```
# SPDX-FileCopyrightText: 2018 Phillip Burgess for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Stomp & roar sound example for Adafruit Hallowing. Functions as a crude
pedometer, plays different sounds in response to steps & jumps. Step
detection based on "Full-Featured Pedometer Design Realized with 3-Axis
Digital Accelerometer" by Neil Zhao, Analog Dialogue Technical Journal,
June 2010.
"""

import time
import math
import digitalio
import displayio
import board
import audioio
import audiocore
import neopixel

def load_wav(name):
    """
    Load a WAV audio file into RAM.
    @param name: partial file name string, complete name will be built on
                 this, e.g. passing 'foo' will load file 'foo.wav'.
    @return WAV buffer that can be passed to play_wav() below.
    """
    return audiocore.WaveFile(open(name + '.wav', 'rb'))

STOMP_WAV = load_wav('stomp') # WAV file to play with each step
ROAR_WAV = load_wav('roar')   # WAV when jumping
IMAGEFILE = 'reptar.bmp'      # BMP image to display

IS_HALLOWING_M4 = False

# Perform a couple extra steps for the HalloWing M4
try:
    if getattr(board, "CAP_PIN"):
        IS_HALLOWING_M4 = True
    if getattr(board, "SPEAKER_ENABLE"):
        # Enable the Speaker
        speaker_enable = digitalio.DigitalInOut(board.SPEAKER_ENABLE)
        speaker_enable.direction = digitalio.Direction.OUTPUT
        speaker_enable.value = True
except AttributeError:
    pass
```

```
AUDIO = audioio.AudioOut(board.SPEAKER)  # Speaker

try:
    board.DISPLAY.auto_brightness = False
except AttributeError:
    pass

# Set up accelerometer on I2C bus, 4G range:
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
if IS_HALLOWING_M4:
    import adafruit_msa301
    ACCEL = adafruit_msa301.MSA301(i2c)
else:
    import adafruit_lis3dh
    try:
        ACCEL = adafruit_lis3dh.LIS3DH_I2C(i2c, address=0x18) # Production board
    except ValueError:
        ACCEL = adafruit_lis3dh.LIS3DH_I2C(i2c, address=0x19) # Beta hardware
    ACCEL.range = adafruit_lis3dh.RANGE_4_G

STEP_INTERVAL_MIN = 0.3 # Shortest interval to walk one step (seconds)
STEP_INTERVAL_MAX = 2.0 # Longest interval to walk one step (seconds)
SAMPLE_RATE_HZ = 50     # Accelerometer polling frequency (per second)
WINDOW_INTERVAL = 1.0   # How often to reset window min/max range (seconds)
PRECISION = 2.0         # Lower numbers = more sensitive to steps
SAMPLE_INTERVAL = 1.0 / SAMPLE_RATE_HZ

FILTER_SIZE = 4         # Number of accelerometer readings to average
FILTER_BUF = [0] * FILTER_SIZE
FILTER_SUM = 0          # Initial average value
FILTER_INDEX = 0        # Current position in sample-averaging buffer

# Display BMP image.
try:
    board.DISPLAY.brightness = 0
    SCREEN = displayio.Group()
    board.DISPLAY.root_group = SCREEN

    # CircuitPython 7+ compatible
    BITMAP = displayio.OnDiskBitmap(IMAGEFILE)
    TILEGRID = displayio.TileGrid(BITMAP, pixel_shader=BITMAP.pixel_shader)

    SCREEN.append(TILEGRID)
    board.DISPLAY.brightness = 1.0   # Turn on display backlight
except (OSError, ValueError):
    pass

# If everything has initialized correctly, turn off the onboard NeoPixel:
PIXEL = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0)
PIXEL.show()

# Read initial accelerometer state and assign to various things to start
X, Y, Z = ACCEL.acceleration
MAG = math.sqrt(X * X + Y * Y + Z * Z) # 3space magnitude
WINDOW_MIN = MAG # Minimum reading from accel in last WINDOW_INTERVAL seconds
WINDOW_MAX = MAG # Maximum reading from accel in last WINDOW_INTERVAL seconds
THRESHOLD = MAG  # Midpoint of WINDOW_MIN, WINDOW_MAX
SAMPLE_OLD = MAG
SAMPLE_NEW = MAG

LAST_STEP_TIME = time.monotonic() # Time of last step detect
LAST_WINDOW_TIME = LAST_STEP_TIME # Time of last min/max window reset

while True:

    TIME = time.monotonic() # Time at start of loop
```

```
    X, Y, Z = ACCEL.acceleration           # Read accelerometer
    MAG = math.sqrt(X * X + Y * Y + Z * Z) # Calc 3space magnitude

    # Low-pass filter: average the last FILTER_SIZE magnitude readings
    FILTER_SUM -= FILTER_BUF[FILTER_INDEX]   # Subtract old value from sum
    FILTER_BUF[FILTER_INDEX] = MAG           # Store new value in buffer
    FILTER_SUM += MAG                        # Add new value to sum
    FILTER_INDEX += 1                        # Increment position in buffer
    if FILTER_INDEX >= FILTER_SIZE:          # and wrap around to start
        FILTER_INDEX = 0
    SAMPLE_RESULT = FILTER_SUM / FILTER_SIZE # Average buffer value

    if SAMPLE_RESULT < 2: # Jump detected (freefall, or close to it)
        while MAG < 10 and time.monotonic() - TIME < 1: # Wait for landing
            X, Y, Z = ACCEL.acceleration
            MAG = math.sqrt(X * X + Y * Y + Z * Z)
        AUDIO.play(ROAR_WAV)
        while AUDIO.playing:
            pass
        continue # Back to top of loop

    # Every WINDOW_INTERVAL seconds, calc new THRESHOLD, reset min and max
    if TIME - LAST_WINDOW_TIME >= WINDOW_INTERVAL: # Time for new window?
        THRESHOLD = (WINDOW_MIN + WINDOW_MAX) / 2  # Average of min & max
        WINDOW_MIN = SAMPLE_RESULT                 # Reset min and max to
        WINDOW_MAX = SAMPLE_RESULT                 # the last value read
        LAST_WINDOW_TIME = TIME                    # Note time of reset
    else:                                          # Not WINDOW_INTERVAL yet,
        if SAMPLE_RESULT < WINDOW_MIN:             # keep adjusting min and
            WINDOW_MIN = SAMPLE_RESULT             # max to accel data.
        if SAMPLE_RESULT > WINDOW_MAX:
            WINDOW_MAX = SAMPLE_RESULT

    # Watch for sufficiently large changes in accelerometer readings...
    SAMPLE_OLD = SAMPLE_NEW
    if abs(SAMPLE_RESULT - SAMPLE_OLD) > PRECISION:
        SAMPLE_NEW = SAMPLE_RESULT
        # If crossing the threshold in the + direction...
        if SAMPLE_OLD <= THRESHOLD <= SAMPLE_NEW:
            # And if within reasonable time window for another step...
            TIME_SINCE_LAST_STEP = TIME - LAST_STEP_TIME
            if STEP_INTERVAL_MIN <= TIME_SINCE_LAST_STEP <= STEP_INTERVAL_MAX:
                # It's a step!
                AUDIO.play(STOMP_WAV)
            LAST_STEP_TIME = TIME

    # Dillydally so the accelerometer isn't polled faster than desired rate
    ELAPSED = time.monotonic() - TIME
    if ELAPSED < SAMPLE_INTERVAL:
        time.sleep(SAMPLE_INTERVAL - ELAPSED)
```

Sound and graphics are easily customized, starting around line 29. **16-bit mono PCM WAV** files (**22,050** sample rate **or less**) are ideal. Graphics should be a **128x128 pixel 24-bit color BMP**:

```
STOMP_WAV = load_wav('stomp') # WAV file to play with each step
ROAR_WAV = load_wav('roar')   # WAV when jumping
IMAGEFILE = 'reptar.bmp'      # BMP image to display
```

The WAV files don't require a complete filename; the ".wav" is implied. (e.g. 'stomp' will load the file stomp.wav). The BMP needs a whole filename.

If you find it too sensitive to steps, or not sensitive enough, change the value of PRECISION around line 48:

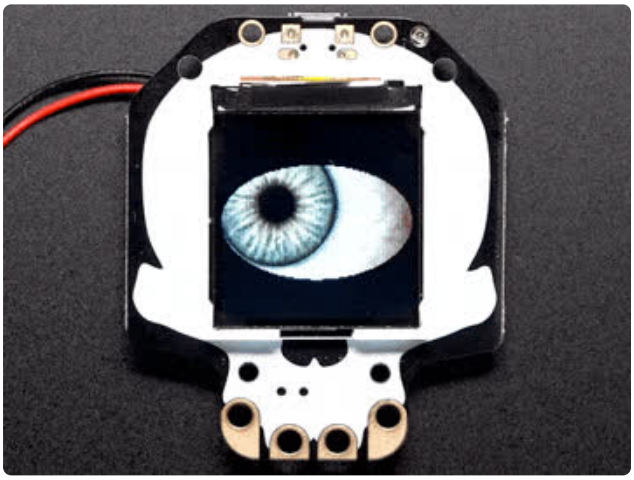
```
PRECISION = 2.0         # Lower numbers = more sensitive to steps
```

Other ways to improve results include **more secure mounting** of the HalloWing board. Worn on a lanyard, HalloWing bobs around a bit and may register false steps. Buttoned in a pocket or attached to a hat with an elastic band is usually more reliable. The nice thing here is that the board doesn't need to be in any particular orientation, it will still detect movements, so use whatever is easiest or most comfortable!

Advanced users could get into fine-tuning the step-detection algorithm, or using only one or two accelerometer axes (requiring the board in a particular orientation for use).

Other Easy Projects

The following can also be done quickly with few parts:



The **eyeball** code that comes **pre-installed** on Hallowing boards can easily be **reloaded** by double-clicking the reset button and then drag-and-drop a file to the **HALLOWBOOT** drive, as explained in the main Hallowing tutorial **(https://adafru.it/CEs)**. There's even a few variants to suit your individual tastes. Quick and easy!

Strap it on a hat or headband, or wear it on a lanyard. Root through your closet for the gothiest garb that work decorum allows. The HalloWing eye can be your little friend.

This spirit board idea (https://adafru.it/CBi) is similarly creepy and quick to set up.

From Cylons to Knight Rider, the **Larson scanner** LED effect (a dot moving back & forth) is a sci-fi staple. It's super-easy to do with HalloWing and this half-meter JST-equipped NeoPixel strip (http://adafru.it/3919) that plugs right into the board.

Light-up effects can add an extra element of safety to trick-or-treat costumes (use the **USB battery** idea below to ensure plenty of run time). Or maybe you've seen those phenomenal Magic Wheelchair (https://adafru.it/CEt) projects but aren't at that level of workmanship...a bit of lighting can make even a modest craft project stand out!

This Larson Scanner tutorial (https://adafru.it/tgA) was originally written with the Trinket microcontroller in mind, but we've since updated it with changes to run on HalloWing as well. It's a little more work than the eyeball idea above...you'll need to edit some code and mount the NeoPixel strip somehow. Attach it to a hat or a quick cardboard box robot head and you're well on your way.

Here's a guide that was written with jack-o'-lanterns in mind (https://adafru.it/CEu), but really it could give anything a simulated **fire-like effect**. It shows a Circuit Playground board being used...but only minor changes are needed (explained in the guide) to work with HalloWing and NeoPixel strip (http://adafru.it/3919).

Maybe it's for the **briefcase** from Pulp Fiction. Maybe just something cool to add to one's **candy bucket**. Or light up a prop **lantern** for Paul Revere. Whatever the idea, it's quick and effective!

Image Slideshow

See this guide on how to make a slideshow of images for your HalloWing or other CircuitPython compatible boards: Creating Slideshows in CircuitPython (https://adafru.it/Fx8).

Use your imagination, stay safe, and have fun!