Make a Guggenhat: Bluetooth-connected wearable NeoPixel marquee hat

Created by Phillip Burgess

https://learn.adafruit.com/guggenhat-bluetooth-le-wearable-neopixel-marquee-scrolling-led-hat-display

Last updated on 2024-06-03 01:29:59 PM EDT

Table of Contents

Overview 3

Tools and Supplies 4

- A Recent Smartphone or Tablet!
- A Dapper Top Hat!
- An Arduino!
- A Power Source!
- Other Parts from Adafruit!
- Tools, Craft Supplies and Maker Ingenuity!

Test the NeoPixel Strip 9

Prepare NeoPixel Strip 12

Hat Hacking 14

Coiling NeoPixels 19

Code 23

- Adjustments:

Soldering 27

Dry Run 30

Final Assembly 34

Use It! 39

Variations 41

- Different Arduino Code
- Dynamic Messages
- "Invisible" LEDs
- 8 LED Rows
- Code Changes

Overview

Adafruit's Bluefruit (https://adafru.it/djc) devices are hands-down the easiest way to get Arduino communicating over a Bluetooth wireless connection. Bluetooth LE is a great way to control a project from a smartphone or tablet.

This simplicity could be illustrated with a scrolling LED wall sign that works with your phone or tablet…and it's easy enough to link up a few 8x8 NeoPixel matrices (http://adafru.it/1487)….but we're a little more zany than that. And we've got a thing for wearable electronics. So…

A hat! A dapper top hat! Most exemplary! Rather than flat matrices, we'll use flexible NeoPixel strip (http://adafru.it/1461). To save a lot of cutting and soldering, let's keep the whole strip intact and coil it around the hat. The project takes its name from the Solomon R. Guggenheim museum (https://adafru.it/djd), with its landmark spiral gallery.

To make this project fun and easy to update, we use a Bluefruit LE module to allow text and color updates to be wirelessly controlled from any iOS (all iPhone/iPads) or Android 4.3+ device (check that your Android device has BLE as some older devices do not!)

CC: Photo by Jean-Christophe BENOIST

Tools and Supplies

This project is wide open to interpretation; there's no Single Canonical Parts List™. But here are some guidelines for selecting many of the items you'll need:

A Recent Smartphone or Tablet!

Check for this first; there's no point continuing without it.
You'll need a smartphone or tablet that supports Bluetooth 4.0 LE in the hardware and operating system. Recent versions of the iPhone and iPad both work, as well as the latest round of Android phones and tablets. Some laptops may work too!

If you're not sure, it's easy to test for this: download and try one of the Bluetooth apps mentioned on the "Use It!" page of this guide. Older devices and OS versions (not supporting Bluetooth 4.0 LE) aren't compatible, and the software will simply refuse to work.

A Dapper Top Hat!

"Oh, oh, no more buttered scones for me, mater. I'm off to play the grand piano!"

Authentic top hats can be quite costly, so I searched eBay for an inexpensive costume-grade substitute. Several sizes were available. Though my own head measurement suggested a medium hat, I selected large, anticipating different people will want to try it on. The hat can be padded with craft foam for a smaller head, but you can't squeeze a too-small hat on a larger head.

Most top hats are slightly tapered one way or another. It's unlikely you'll find one with exactly straight sides, but try to find something as close to straight as possible, so the circumference is fairly uniform (we'll tweak this later). The cheap paper Pilgrim hats at the party store are much too tapered, don't use these!

Make sure the sides of the hat are tall enough, 4 inches (10 cm) minimum. You don't need to go all Lincoln stovepipe unless that's the look you're after.

An Arduino!

We'll make our hat using a headerless Arduino Micro (http://adafru.it/1315). It's simplest if you follow suit — we know the board works for this project. An Arduino Leonardo could also work, it's just bulkier. The Micro is small enough to be hidden discreetly inside the hat, providing a little protection from the elements.

The Bluetooth library and all those NeoPixels require a lot of RAM, and most mainstream ATmega-based boards (Uno, Boarduino, etc.) **won't work.** The Arduino Micro's 32U4 processor has an extra 512 bytes of RAM that turn out to be critical to this project!

Cutting-edge boards like the Arduino Due, Netduino or Teensy 3 are unlikely to work here — though powerful, they're based on different processors and only "mostly" Arduino compatible. Trinket and Gemma lack enough RAM, and an Arduino Mega is just overkill.

Arduino Micro or Leonardo for this project. Anything else is a bag of hurt.

A Power Source!

Most anything that can provide 5 Volts (or a little less) at 500 milliamps should suffice. Among the available options are:

- Three AA (http://adafru.it/771) alkaline cells (4.5V total) — cheap and easy, this is what we'll use for ours!
- Four AA (http://adafru.it/830) NiMH rechargeables (4.8V total)
- A slim lithium-polymer battery (https://adafru.it/djk) (3.7V, 1200 mAh or larger)

Don't exceed 5 Volts, or you'll damage the LED strip and Arduino! If using a 4x AA battery holder (http://adafru.it/830), only use NiMH rechargeables; do not install four alkaline cells in series, these have a slightly higher voltage.

Other Parts from Adafruit!

- Adafruit Bluefruit LE nRF8001 breakout (http://adafru.it/1697) (http://adafru.it/1697)
- **4 meters** (1 reel) of 60 NeoPixel LED strip (http://adafru.it/1461)
- Snap-Action 5-Wire Block Connector (http://adafru.it/874) (pack of 3)

NeoPixel strip is sold by the meter. If you add four meters to your cart, you'll receive a contiguous 4-meter reel.

Tools, Craft Supplies and Maker Ingenuity!

The exact construction techniques will depend on your particular skill set and the materials you find most readily available. At the very least, you'll need a **soldering iron, wire and related paraphernalia.** Read through the rest of the guide first for some material ideas, or devise with your own. You'll probably need a couple different types of tape or glue, thin plastic or card stock, perhaps small zip ties, Velcro or magnets.

You're powering FOUR METERS of NeoPixels off a set of AAs? Are you DAFT?

I was surprised by this too! It's possible the brightness is kept low and only a fraction of the pixels are on at any time (our code displays text on a black background). The 4m reel draws about 200 mA with all pixels off (the driver chip inside each NeoPixel draws a tiny bit of current, and there's a of NeoPixels). With brightness set at 1/8 max, various scrolling messages and colors range roughly from 250 mA to 450 mA. One set of AA cells can run it for hours!

Test the NeoPixel Strip

Before assembling everything, let's give the NeoPixel strip a test run. If there's a problem, it's easier to request help or an exchange now than with an assembled project.

If you're not already using the Adafruit_NeoPixel library for Arduino, download and install this first. We have a guide for proper library installation (https://adafru.it/dit).

Download Adafruit_NeoPixel Library

https://adafru.it/cDj

After installing the library and restarting the Arduino software, copy and paste the code below into a new Arduino sketch.

DO NOT use the NeoPixel "strandtest" example for this test! It lights all the LEDs at once — more power than USB can handle. The code below is designed to light only a few LEDs at a time, so we can use the Arduino's 5V or VIN pins.

If you have an Arduino Uno (or similar) around, use that for NeoPixel testing. Not mandatory, but the headers make for easy jumper wire connections.

```
// Simple NeoPixel test.  Lights just a few pixels at a time so a
// long strip can safely be powered from Arduino 5V pin.  Arduino
// may nonetheless hiccup when LEDs are first connected and not
// accept code.  So upload code first, unplug USB, connect pixels
// to GND FIRST, then +5V and digital pin 6, then re-plug USB.
// A working strip will show a few pixels moving down the line,
// cycling between red, green and blue.  If you get no response,
// might be connected to wrong end of strip -- look for the data
// direction arrows printed on the strip.

#include &lt;Adafruit_NeoPixel.h&gt;

#define PIN    6
#define N_LEDS 240 // 4 meter reel

Adafruit_NeoPixel strip = Adafruit_NeoPixel(N_LEDS, PIN, NEO_GRB + NEO_KHZ800);

void setup() {
  strip.begin();
}

void loop() {
  chase(strip.Color(255, 0, 0)); // Red
  chase(strip.Color(0, 255, 0)); // Green
  chase(strip.Color(0, 0, 255)); // Blue
}

static void chase(uint32_t c) {
  for(uint16_t i=0; i&lt;strip.numPixels()+4; i++) {
      strip.setPixelColor(i  , c); // Draw new pixel
      strip.setPixelColor(i-4, 0); // Erase pixel a few steps back
      strip.show();
      delay(25);
  }
}
```

Upload this code to the Arduino board, then unplug USB and make the following three connections:

- **GND** from Arduino to **GND** or **–** on strip, usually a black wire (always connect GND first).
- **VIN** from Arduino to **+5V** or **+** on strip (usually a red wire).

- **Pin 6** from Arduino to **DIN** (or **unmarked** input) on strip (usually a white wire).

As mentioned before, testing is easiest with an Arduino Uno with its row headers. If you only have the Arduino Micro that you got for this project, that's okay…you can make connections using a combination of jumper wires and alligator clips (but be super extra careful that they connect the to right points and have a firm grip…losing the ground connection even for a moment can ruin the first pixel…so don't move parts around while testing). Or you can temporarily solder wires to the board (best if you have some experience with desoldering — it's easy to damage things if done wrong).

Lay the strip out flat so you can see the entire thing, then re-connect the USB cable. After a few seconds, you should see a few LEDs chasing down the length of the strip, cycling between red, green and blue. Watch carefully, noting any skipped or off-color pixels.

Nothing lights up!

1. If the computer reports a USB device is drawing too much power, unplug the Arduino immediately.
2. Make sure the extra wires at either end of the strip are not touching each other or anything conductive.
3. Confirm the three connections between the strip and Arduino: GND, +5V and pin 6.
4. If using an Arduino Leonardo or Mini, it takes about 10 seconds when USB is connected before the sketch actually runs. Be patient.
5. If you soldered any connections, make sure there's no cold joints or solder bridges between adjacent pads.
6. Make sure you're connected to the INPUT end of the strip.
7. Check the USB cable is properly seated between the Arduino and computer or powered USB hub.

If you have a multimeter, check the voltage across +5V and GND at the OUTPUT end of the strip. It should be around 5 Volts.

The lights cut out part way down the strip.

Confirm the value of N_LEDS in the code matches the actual NeoPixel strip length.

One or more pixels won't light up, or show the wrong color.

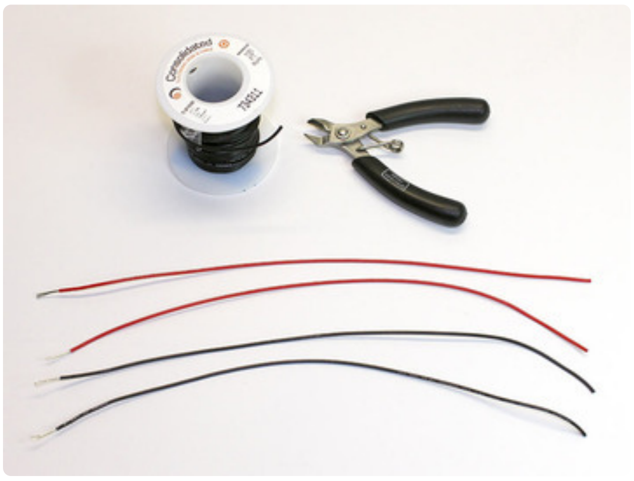
Possibly defective pixel(s). Read on…

If you encounter any of these problems (or others), search the Adafruit Customer Support Forums (https://adafru.it/cer) for similar issues and their resolutions. If your situation is not addressed, make a new post with a description of the problem and (if possible) a clear photo showing the wiring between Arduino and NeoPixels. We'll help troubleshoot the problem or have a replacement sent if needed.

Do not proceed until you have a fully tested and working NeoPixel strip.

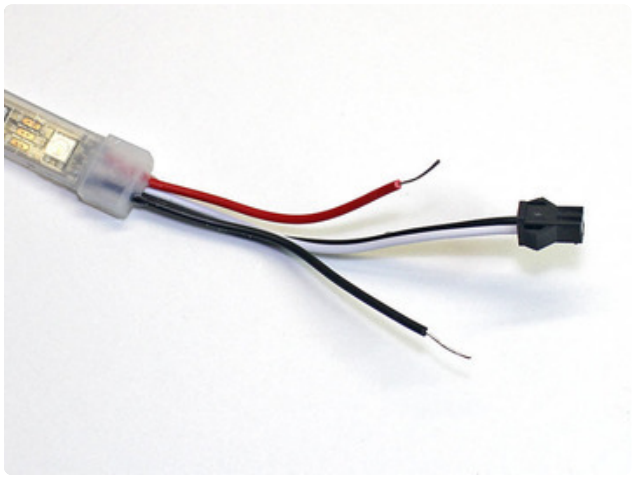
Prepare NeoPixel Strip

Before we get on with the arts & crafts part of the project, let's make a couple modifications to the NeoPixel strip...

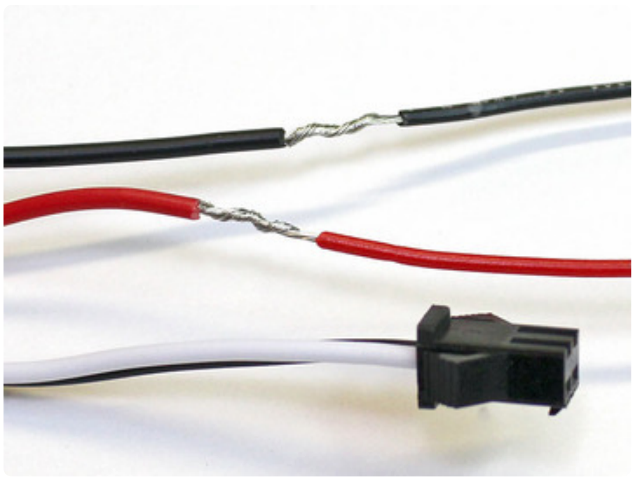


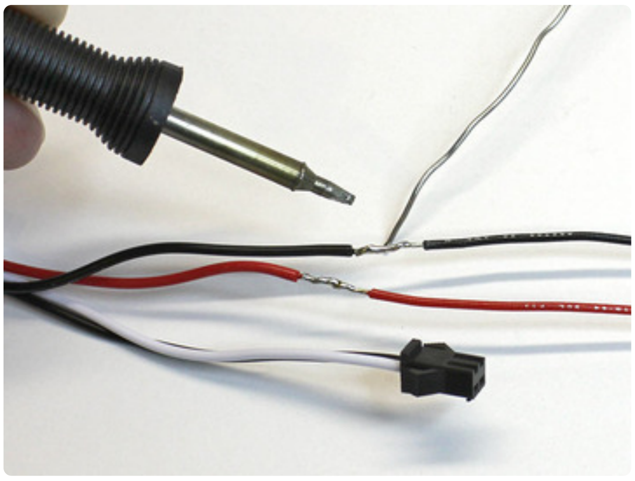
Cut four pieces of wire about 10 inches (25 cm) long. Strip about 1/2" (12 mm) of insulation from one end of each wire.

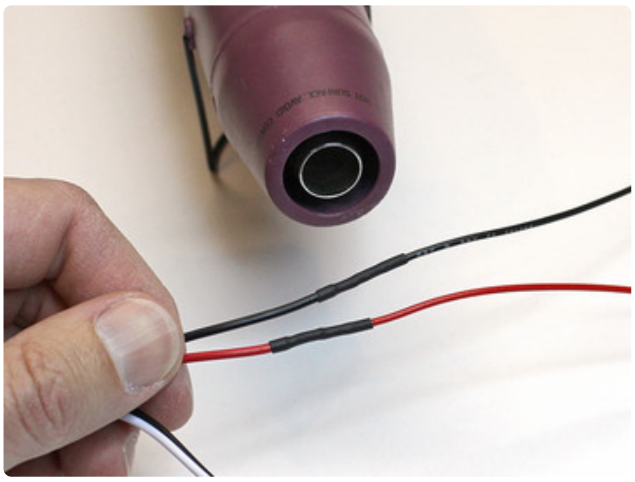
If you have color-coded wire available, make two red and two black. If not, that's okay, just need to be extra careful later when making connections.



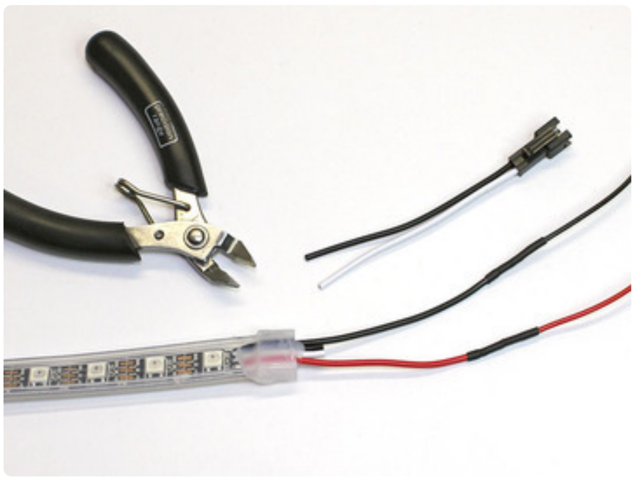
You might need to strip a little extra insulation from the red and black wires on the NeoPixel strip, to match the wires you cut and stripped above.


Twist the stripped ends of the wires together...


...and solder. An inline splice! Bravo.


Apply heat-shrink tube over the connections to prevent electrical shorts and keep out moisture.

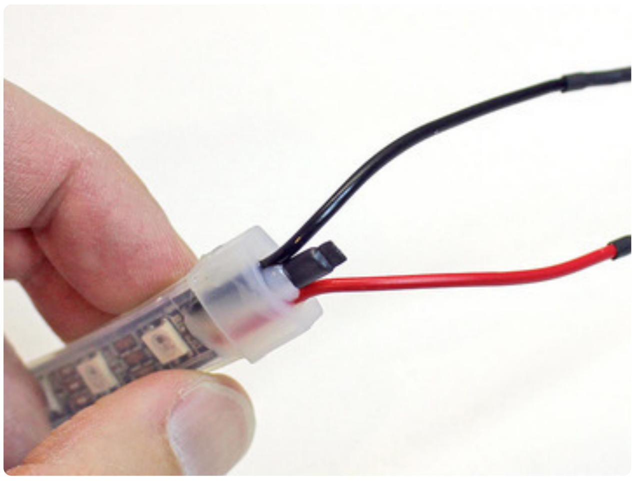
Repeat the above on both ends of the NeoPixel strip. Four connections total.



Next, clip the two-pin plug off the **OUTPUT** end of the NeoPixel strip.

Be super extra special careful that you're indeed cutting the output plug. Look for the tiny arrows printed on the strip.

Later we'll solder this plug to the Arduino, to make a removable connector for the input end.



A small piece of heat-shrink can be used to cap the stubby wires and prevent electrical mishaps. Pinch the end of the heat-shrink tube with pliers while it's still hot to seal off the end.

The photos on the next page show a strip without these wire extensions. That was simply poor planning — your strip should have the long wires added regardless of the images there.

Hat Hacking

So we've got a little problem...

We'd like for — no, we need — the pixels to form a fairly uniform grid, so we can draw legible text.

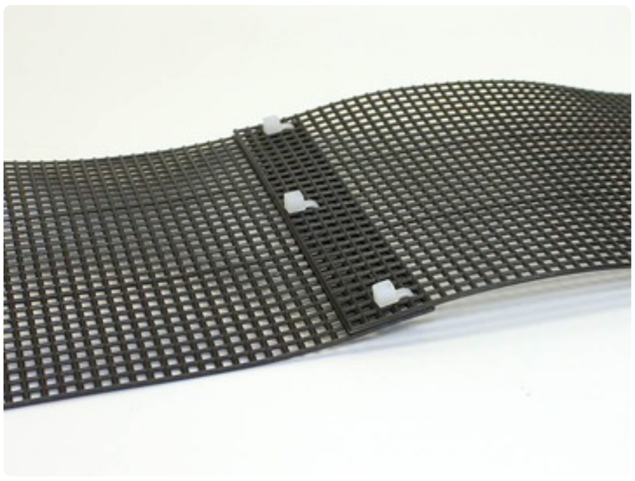
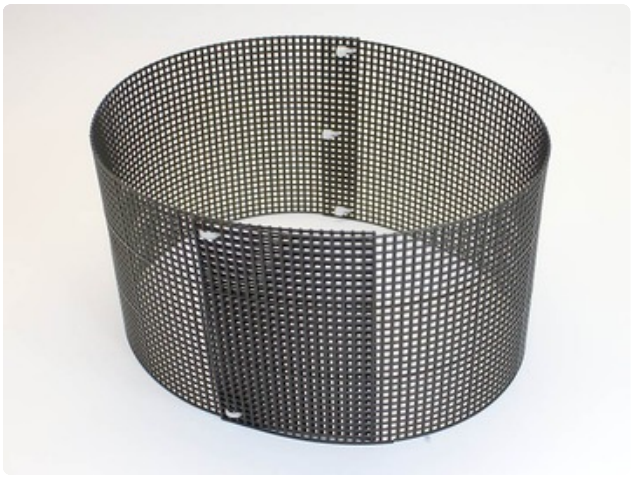
As previously mentioned, the hat is not a perfect cylinder...it's tapered. If we simply coil the NeoPixel strip around it, the circumference of each loop is slightly different, and everything will be a jumble rather than a neat grid. Even if you do find a straight-sided hat, the circumference is very unlikely to be a precise multiple of the pixel spacing along the strip.

To address this, we'll **build a sleeve that slips over the hat.** This will give us straight sides and an ideal circumference.

First order of business was to cut and peel the hat band off; it interfered with the fitting of the sleeve. Doing this may leave hot glue stains behind. Rubbing alcohol and some scraping with an X-Acto knife helped somewhat. But no biggie…most people will be too dazzled by the LEDs to notice.

I’d originally made a sleeve out of a certain type of heavy acetate film, before realizing another problem: my craft stash has stuff dating back to the Byzantine Empire. The well-stocked local art supply shop is an extinct bird, and I don’t want to send anyone on a costly internet goose chase for one simple material. This is where the maker ingenuity ingredient comes in: some possible alternatives are mentioned below, but maybe you have better ideas for cheap materials you can source locally.

Card stock or poster board are reasonable last-ditch substitutes, but something plastic is much preferred. Not expecting anyone to wear this in a downpour…but sometimes one is caught off-guard by a little drizzle or fog, and paper would just go pulpy and fall apart.

Fail: one material I'd experimented with was plastic canvas (aka needlepoint mesh) and small cable ties, which could be found in more mainstream craft and hardware stores.

This almost worked, but the material seemed just a bit too thick, and all those cable ties added a lot of ugly nubblyness.

If you have the patience to do this with actual hand sewing instead of the cable ties, I wouldn't rule it out as a possibility! Read on regardless, there's still the matter of sizing and assembly...

To wrap around the hat, we'd like to find a thin, flexible plastic with just enough "body" to stand up on its own when formed into a cylinder, but not too thick and bulky. It should be a minimum of 4.5 inches (115mm) wide, and 2 feet (600mm) long or more (notice above how the needlepoint mesh had to be joined from two smaller pieces — the local craft store stuff is only 13.5 inches long).

Ideally, the material would be affordable and easily found in a local "big box" store. There were field trips involved.

Craft foam? Too thick. Poly envelope? Too flopsy. Vinyl floor runner? Too thick and too flopsy. The placemat looked promising (and on clearance! And BATMAN!)...

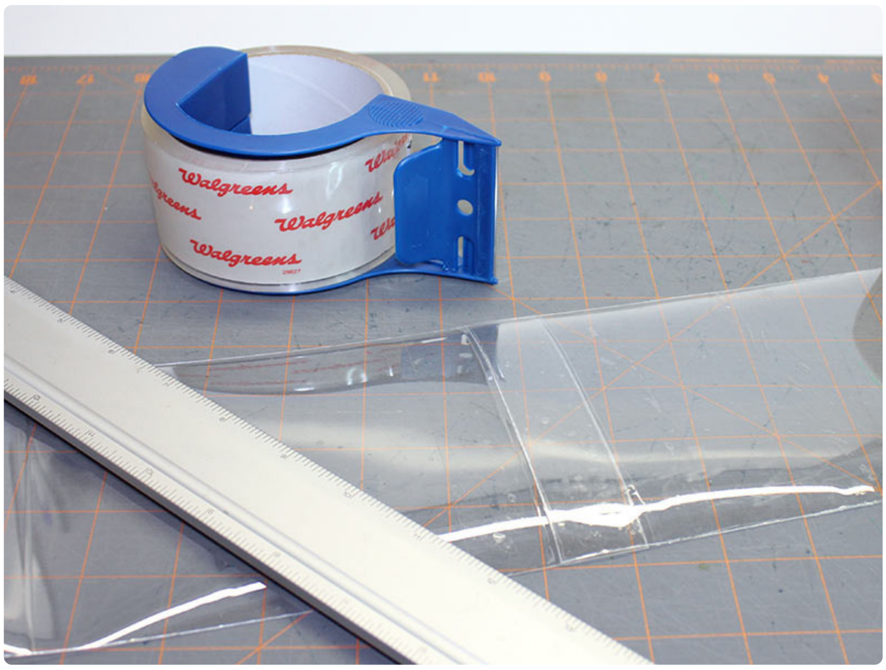
...but ultimately arrived at the thin plastic used for 2-liter soda bottles. Can find these anywhere!

Two bottles provided sufficient plastic to fit around the hat and sufficient caffeine to complete the project. Diet soda, though vile, rinses out nicely with no sticky residue.

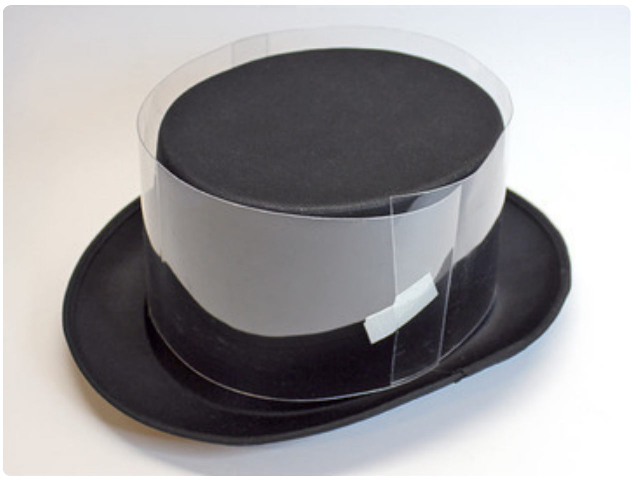
Coke bottles were unsuitable due to their funny shape; had to be something with straight sides.

The tops and bottoms were cut off the bottles, and the resulting tubes were sliced down one side. These were then unrolled, laid down flat and trimmed more carefully to 4.5" wide.

The two pieces were joined into a longer strip (overlapping about 1 inch) using packing tape on both sides.



The resulting plastic strip has a strong proclivity to roll up. That’s okay, it’ll behave better once wrapped around the hat…



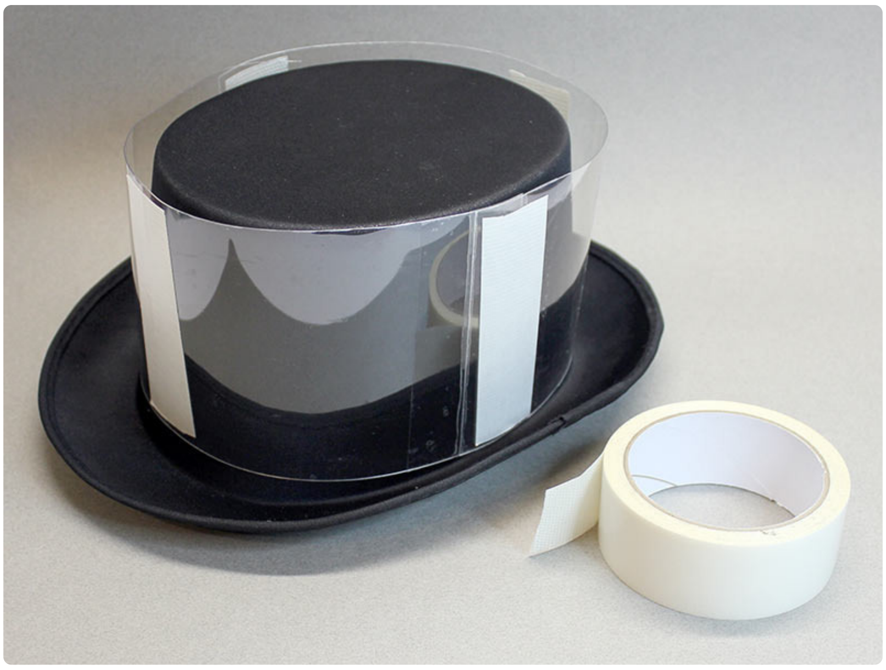
Hold the sleeve in place temporarily with a piece of removable masking tape on the inside. Don’t join this side with packing tape…the circumference is going to need some tweaking.

On the next page, we’ll start coiling the NeoPixel strip around this sleeve.

The silicone coating on the NeoPixel strip is wonderful for weatherproofing, but does present a challenge: almost nothing on this planet sticks to silicone!

Fortunately there’s one thing that sticks just well enough…**carpet tape**. Most hardware stores carry this. It’s a double-sided tape that’s wicked sticky and has a removable waxy paper covering.

Cut and apply four strips of carpet tape to the sleeve, running vertically: one each at the front and back, one on either side. Leave the paper backing in place until the next step.

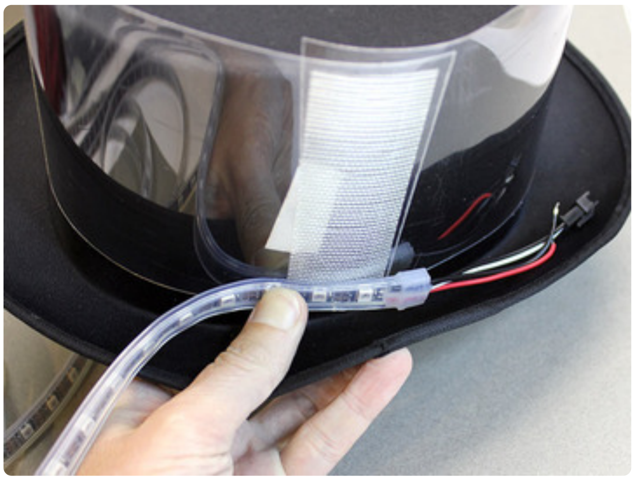


Don’t worry if these aren’t straight, they’ll be covered up. And if you cut a piece too short, just add another small piece to make up the difference.

Coiling NeoPixels

Okay, peel the backing off the four strips of carpet tape. Be careful how you set the hat down at this stage…this stuff sticks to everything!

The stickiness of the tape, combined with the silicone’s resistance to most adhesives, has a side-effect that’s now quite useful: it has a weak hold like a Post-It® Note, making it possible to reposition the strip as we work through this next sequence.

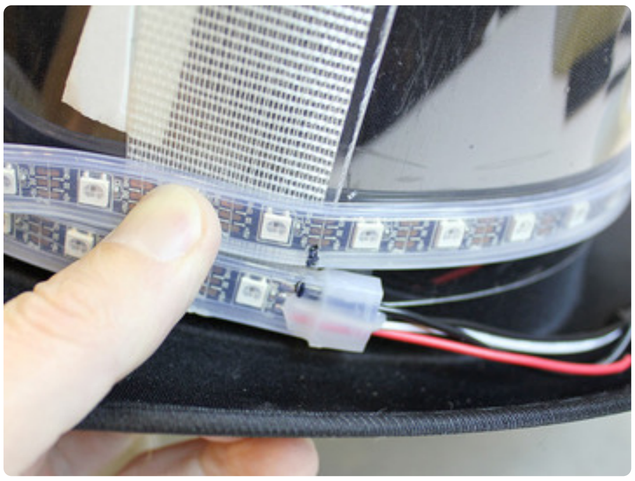


Start with the **OUTPUT** end of the strip.

Identify the **LEFT TEMPLE** of the hat and press the strip against the carpet tape at the **BOTTOM** (closest to the hat’s brim) with the **WIRES** toward the **BACK** of the hat and the **STRIP** toward the **FRONT**.

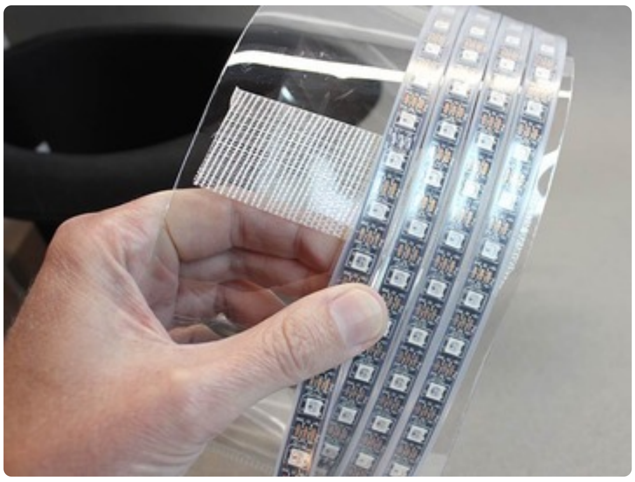
(In this photo, the front of the hat faces left.)

Wrap the NeoPixel strip **ONCE** around the hat, adhering to each of the carpet tape strips as you go. Don’t follow the edge of the sleeve exactly, you want it to rise very slightly as it goes around, so it’s offset by the strip’s width when you come back to the start.



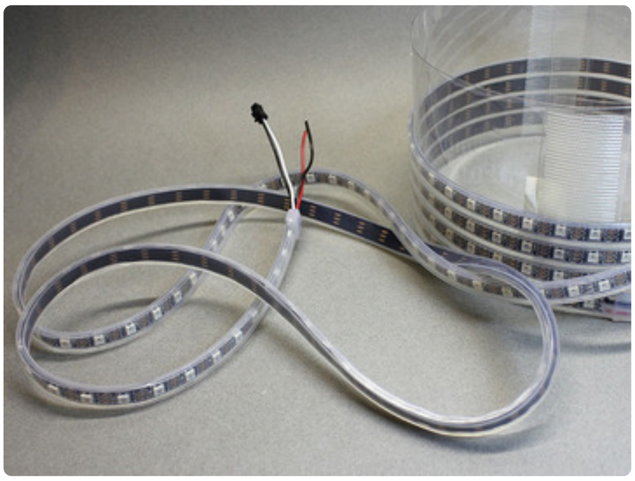
You’ll probably find that the first pixel and the pixel above it **aren’t aligned.** This is normal! If it’s just a few millimeters short, you can cinch the strip a little tighter to make them line up. Otherwise, relax the sleeve around the hat to get these pixels to align: peel the LED strip away from the carpet tape, adjust the masking tape that’s holding the sleeve together, then re-stick the LED strip.

Once they’re reasonably aligned, make a mark with a Sharpie pen and count the number of NeoPixels between the marks. You’ll need this number later when we program the Arduino.



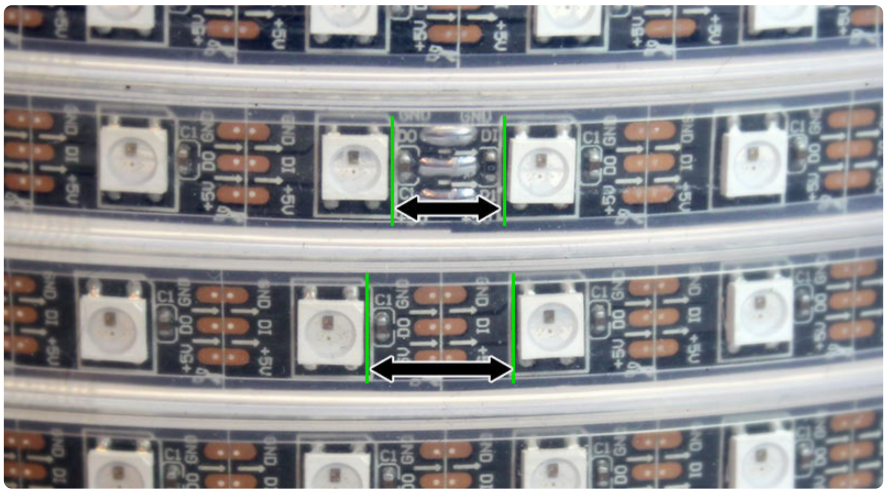
Continue coiling the NeoPixel strip around the sleeve. You can slip it off the hat if it makes it easier.

The rows of strip don’t have to butt right up against each other. I tried to leave a couple millimeters spacing. The Post-It®-like hold of the tape makes it easy to back up and try again repeatedly.



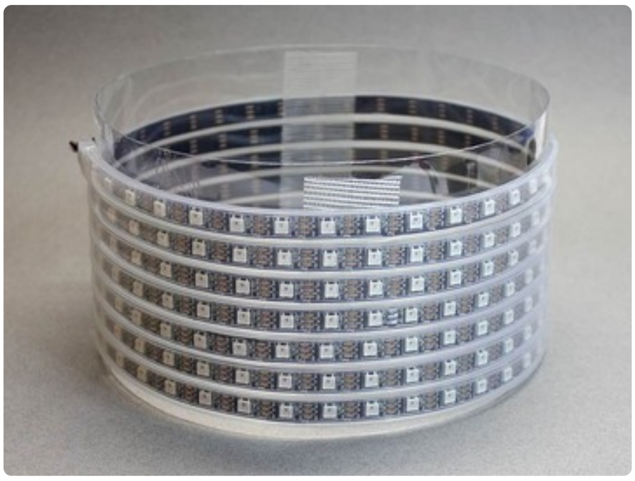
CAREFUL! With 4 meters of NeoPixels on the desk, it’s easy for things to get kinked or knotted. Periodically tidy up the strip so nothing gets damaged.

All LED strips are actually made in **1/2 meter segments** which are soldered together to produce a full reel. The pixel spacing across these joins is slightly different from the rest of the strip:



In order to maintain a somewhat regular grid, it's therefore necessary to cinch the strip a little tighter in some areas, and relax it slightly in others. If you notice the pixels getting out of alignment, back up (un-stick the strip 1/2 to 1 full coil) and adjust the tension until things line up better.

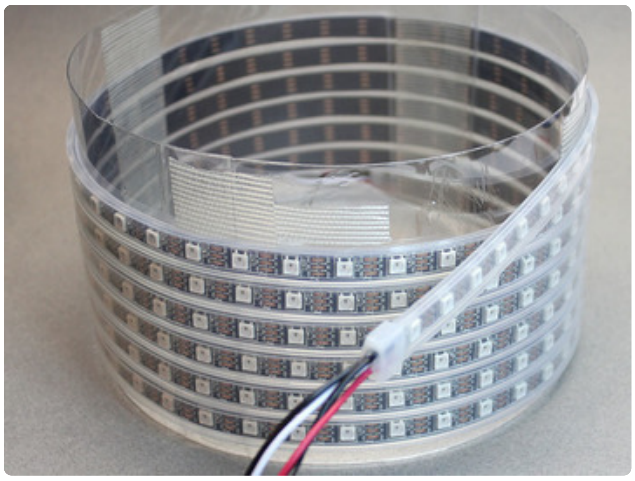
It's bothersome up close, but at any reasonable distance nobody will notice. Sometimes referred to as the "ten foot rule" in cosplay.



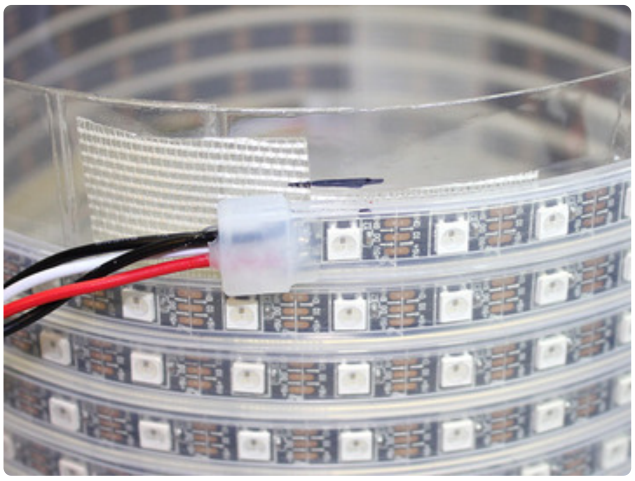
Keep wrapping until you run out of strip. This probably won't go all the way to the top of the sleeve, that's okay.

Also, it's unlikely that the start and end of the strip will be aligned. This too is okay, we'll simply **turn the sleeve to put the most pixels at the front.**

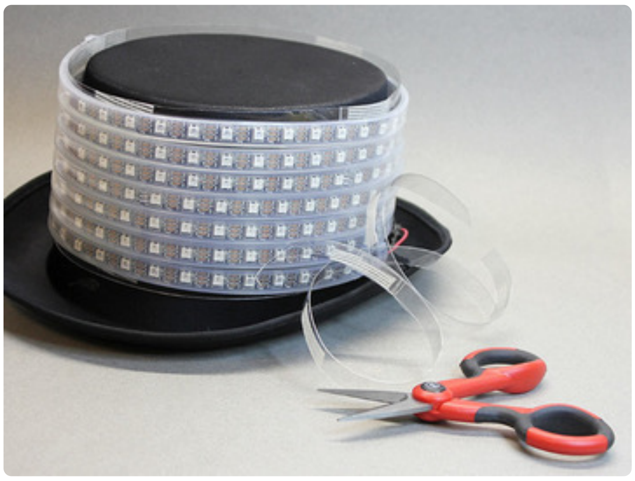
My hat has seven rows of NeoPixels in the front, six in the back. Part of the text will be cut off in back, but don't worry about this…it still gets peoples' attention, and they'll come around the front to ask questions about your hat!



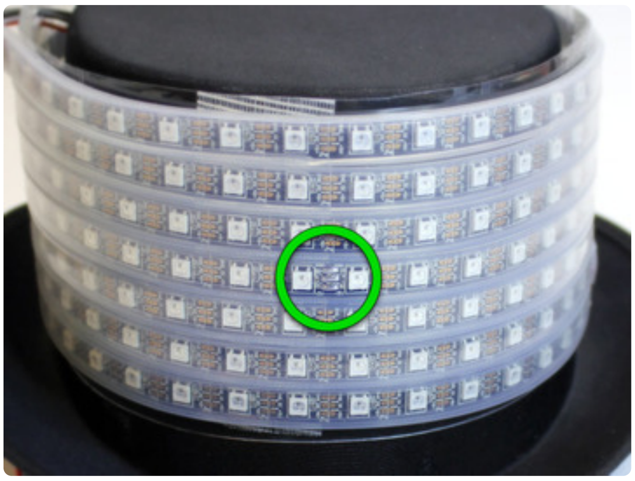
If the strip doesn't end on one of the existing carpet tape pieces, stick down a little extra piece to catch this end.



On the sleeve, mark the highest point where the NeoPixel strip reaches…



…then trim all the way around the sleeve at this level (don't follow the edge of the LED strip, since it's slightly askew).



Those bothersome little solder joints mentioned earlier? One of them might be very helpful now!

If your hat has seven rows in front, one of these joints (at the 2 meter mark along the strip) will now be in the very center. You can use this to align the sleeve with the hat, so it's facing directly forward.

Code

Before continuing with the build, let's get the software onto the Arduino. It's much easier to reach right now when it's not built into a hat.

Copy-and-paste the code below into a new Arduino sketch. Select your board type (e.g. Arduino Micro) from the Tools→Board menu, then click the Verify icon (the checkmark at the top left). If all goes well, you can connect the board and upload the code.

See the notes following the code for adjustments you may need to make.

If the code fails to compile, this is usually due to a missing library. This sketch depends on four Arduino libraries, and all of them must be correctly named and installed. We have a guide for proper library installation (https://adafru.it/dit). You may already have some of these installed, but here are all the links for posterity:

Download Adafruit_NeoPixel Library

https://adafru.it/cDj

Download Adafruit_NeoMatrix Library

https://adafru.it/cDt

Download Adafruit_GFX Library

https://adafru.it/cBB

Download Adafruit_BLE_UART Library

https://adafru.it/dd9

If using an earlier version of the Arduino IDE (prior to 1.8.10), also locate and install Adafruit_BusIO (https://adafru.it/Ldl) (newer versions will install this dependency automatically if using the Arduino Library Manager).

The other reason this might fail to compile is if you're using a "cutting edge" Arduino-like board; these frequently use a different processor than the plain vanilla Arduino, and the software (expecting specific hardware and registers) is not compatible. It also won't work on tiny boards like the Trinket and Gemma.

This sketch uses a ton of RAM and won't work on the common Arduino Uno. The Arduino Micro (and Leonardo) have an extra 512 bytes that are crucial to making this work.

```
/*--------------------------------------------------------------------------
  GUGGENHAT: a Bluefruit LE-enabled wearable NeoPixel marquee.

  Requires:
  - Arduino Micro or Leonardo microcontroller board.  An Arduino Uno will
    NOT work -- Bluetooth plus the large NeoPixel array requires the extra
    512 bytes available on the Micro/Leonardo boards.
  - Adafruit Bluefruit LE nRF8001 breakout: www.adafruit.com/products/1697
  - 4 Meters 60 NeoPixel LED strip: www.adafruit.com/product/1461
  - 3xAA alkaline cells, 4xAA NiMH or a beefy (e.g. 1200 mAh) LiPo battery.
  - Late-model Android or iOS phone or tablet running nRF UART or
    Bluefruit LE Connect app.
  - BLE_UART, NeoPixel, NeoMatrix and GFX libraries for Arduino.

  Written by Phil Burgess / Paint Your Dragon for Adafruit Industries.
  MIT license.  All text above must be included in any redistribution.
  --------------------------------------------------------------------------*/

#include &lt;SPI.h&gt;
#include &lt;Adafruit_BLE_UART.h&gt;
#include &lt;Adafruit_NeoPixel.h&gt;
#include &lt;Adafruit_NeoMatrix.h&gt;
#include &lt;Adafruit_GFX.h&gt;

// NEOPIXEL STUFF ----------------------------------------------------------

// 4 meters of NeoPixel strip is coiled around a top hat; the result is
// not a perfect grid.  My large-ish 61cm circumference hat accommodates
// 37 pixels around...a 240 pixel reel isn't quite enough for 7 rows all
// around, so there's 7 rows at the front, 6 at the back; a smaller hat
// will fare better.
#define NEO_PIN     6 // Arduino pin to NeoPixel data input
#define NEO_WIDTH  37 // Hat circumference in pixels
#define NEO_HEIGHT  7 // Number of pixel rows (round up if not equal)
#define NEO_OFFSET  (((NEO_WIDTH * NEO_HEIGHT) - 240) / 2)

// Pixel strip must be coiled counterclockwise, top to bottom, due to
// custom remap function (not a regular grid).
Adafruit_NeoMatrix matrix(NEO_WIDTH, NEO_HEIGHT, NEO_PIN,
  NEO_MATRIX_TOP  + NEO_MATRIX_LEFT +
  NEO_MATRIX_ROWS + NEO_MATRIX_PROGRESSIVE,
  NEO_GRB         + NEO_KHZ800);

char          msg[21]       = {0};            // BLE 20 char limit + NUL
uint8_t       msgLen        = 0;              // Empty message
int           msgX          = matrix.width(); // Start off right edge
unsigned long prevFrameTime = 0L;             // For animation timing
#define FPS 20                                // Scrolling speed

// BLUEFRUIT LE STUFF-------------------------------------------------------

// CLK, MISO, MOSI connect to hardware SPI.  Other pins are configrable:
#define ADAFRUITBLE_REQ 10
#define ADAFRUITBLE_RST  9
#define ADAFRUITBLE_RDY  2 // Must be an interrupt pin

Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(
  ADAFRUITBLE_REQ, ADAFRUITBLE_RDY, ADAFRUITBLE_RST);
aci_evt_opcode_t  prevState  = ACI_EVT_DISCONNECTED;

// STATUS LED STUFF --------------------------------------------------------

// The Arduino's onboard LED indicates BTLE status.  Fast flash = waiting
// for connection, slow flash = connected, off = disconnected.
#define LED 13                   // Onboard LED (not NeoPixel) pin
int           LEDperiod   = 0;   // Time (milliseconds) between LED toggles
boolean       LEDstate    = LOW; // LED flashing state HIGH/LOW
unsigned long prevLEDtime = 0L;  // For LED timing

// UTILITY FUNCTIONS -------------------------------------------------------

// Because the NeoPixel strip is coiled and not a uniform grid, a special
// remapping function is used for the NeoMatrix library.  Given an X and Y
// grid position, this returns the corresponding strip pixel number.
// Any off-strip pixels are automatically clipped by the NeoPixel library.
uint16_t remapXY(uint16_t x, uint16_t y) {
  return y * NEO_WIDTH + x - NEO_OFFSET;
}

// Given hexadecimal character [0-9,a-f], return decimal value (0 if invalid)
uint8_t unhex(char c) {
  return ((c &gt;= '0') &amp;&amp; (c &lt;= '9')) ?      c - '0' :
         ((c &gt;= 'a') &amp;&amp; (c &lt;= 'f')) ? 10 + c - 'a' :
         ((c &gt;= 'A') &amp;&amp; (c &lt;= 'F')) ? 10 + c - 'A' : 0;
}

// Read from BTLE into buffer, up to maxlen chars (remainder discarded).
// Does NOT append trailing NUL.  Returns number of bytes stored.
uint8_t readStr(char dest[], uint8_t maxlen) {
  int     c;
  uint8_t len = 0;
  while((c = BTLEserial.read()) &gt;= 0) {
    if(len &lt; maxlen) dest[len++] = c;
  }
  return len;
}

// MEAT, POTATOES ----------------------------------------------------------

void setup() {
  matrix.begin();
  matrix.setRemapFunction(remapXY);
  matrix.setTextWrap(false);   // Allow scrolling off left
  matrix.setTextColor(0xF800); // Red by default
  matrix.setBrightness(31);    // Batteries have limited sauce

  BTLEserial.begin();

  pinMode(LED, OUTPUT);
  digitalWrite(LED, LOW);
```

```
}

void loop() {
  unsigned long t = millis(); // Current elapsed time, milliseconds.
  // millis() comparisons are used rather than delay() so that animation
  // speed is consistent regardless of message length &amp; other factors.

  BTLEserial.pollACI(); // Handle BTLE operations
  aci_evt_opcode_t state = BTLEserial.getState();

  if(state != prevState) { // BTLE state change?
    switch(state) {        // Change LED flashing to show state
     case ACI_EVT_DEVICE_STARTED: LEDperiod = 1000L / 10; break;
     case ACI_EVT_CONNECTED:      LEDperiod = 1000L / 2;  break;
     case ACI_EVT_DISCONNECTED:   LEDperiod = 0L;         break;
    }
    prevState   = state;
    prevLEDtime = t;
    LEDstate    = LOW; // Any state change resets LED
    digitalWrite(LED, LEDstate);
  }

  if(LEDperiod &amp;&amp; ((t - prevLEDtime) &gt;= LEDperiod)) { // Handle LED flash
    prevLEDtime = t;
    LEDstate    = !LEDstate;
    digitalWrite(LED, LEDstate);
  }

  // If connected, check for input from BTLE...
  if((state == ACI_EVT_CONNECTED) &amp;&amp; BTLEserial.available()) {
    if(BTLEserial.peek() == '#') { // Color commands start with '#'
      char color[7];
      switch(readStr(color, sizeof(color))) {
       case 4:                  // #RGB    4/4/4 RGB
        matrix.setTextColor(matrix.Color(
          unhex(color[1]) * 17, // Expand to 8/8/8
          unhex(color[2]) * 17,
          unhex(color[3]) * 17));
        break;
       case 5:                  // #XXXX   5/6/5 RGB
        matrix.setTextColor(
          (unhex(color[1]) &lt;&lt; 12) +
          (unhex(color[2]) &lt;&lt;  8) +
          (unhex(color[3]) &lt;&lt;  4) +
           unhex(color[4]));
        break;
       case 7:                  // #RRGGBB 8/8/8 RGB
        matrix.setTextColor(matrix.Color(
          (unhex(color[1]) &lt;&lt; 4) + unhex(color[2]),
          (unhex(color[3]) &lt;&lt; 4) + unhex(color[4]),
          (unhex(color[5]) &lt;&lt; 4) + unhex(color[6])));
        break;
      }
    } else { // Not color, must be message string
      msgLen      = readStr(msg, sizeof(msg)-1);
      msg[msgLen] = 0;
      msgX        = matrix.width(); // Reset scrolling
    }
  }

  if((t - prevFrameTime) &gt;= (1000L / FPS)) { // Handle scrolling
    matrix.fillScreen(0);
    matrix.setCursor(msgX, 0);
    matrix.print(msg);
    if(--msgX &lt; (msgLen * -6)) msgX = matrix.width(); // We must repeat!
    matrix.show();
    prevFrameTime = t;
    }
  }
```

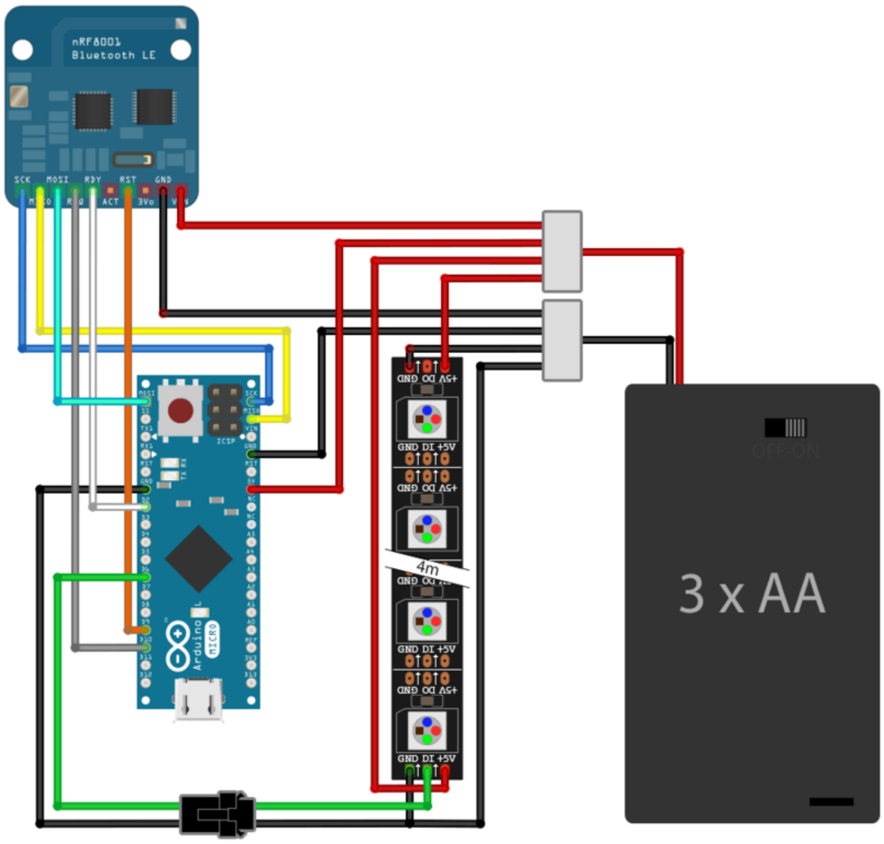
Adjustments:

- You’ll almost certainly need to change NEO_WIDTH to match the actual circumference of your hat, in pixels. NEO_HEIGHT will likely stay at 7, unless you have an exceptionally small or large hat.
- Some of the pin numbers might need changing to reflect your particular wiring, if you’ve changed things around.
- If you’ve coiled your strip clockwise rather than counterclockwise, or the strip input starts at the bottom rather than the top, you might be able to tweak the remapXY() function to use the strip as-is and not have to re-coil it.
- If you’re using a longer or shorter LED strip, the number 240 should be changed in the NEO_OFFSET definition.

After making these adjustments, upload the modified sketch to the Arduino board.

Soldering

This is what we’re aiming for, in schematic form:



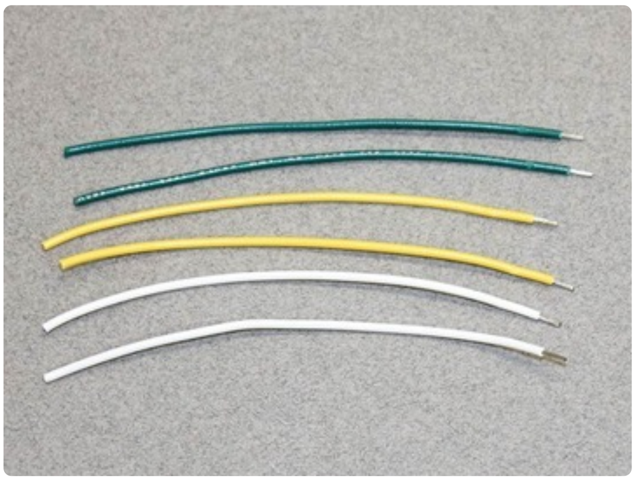
Power from the battery is split four ways: to the Arduino, the Bluetooth module, and to both ends of the NeoPixel strip.

The Bluetooth LE module is connected to the Arduino's SPI bus and a few control pins. This is the same wiring scheme as in the Bluefruit LE tutorial (https://adafru.it/dna), but adapted for the Arduino Micro's SPI pin arrangement.

SCK, MISO and MOSI connect to the same pins on the Arduino (the latter two are labeled MI and MO on the Micro). REQ goes to pin 10, RDY to pin 2, and RST to pin 9 (ACT is skipped).

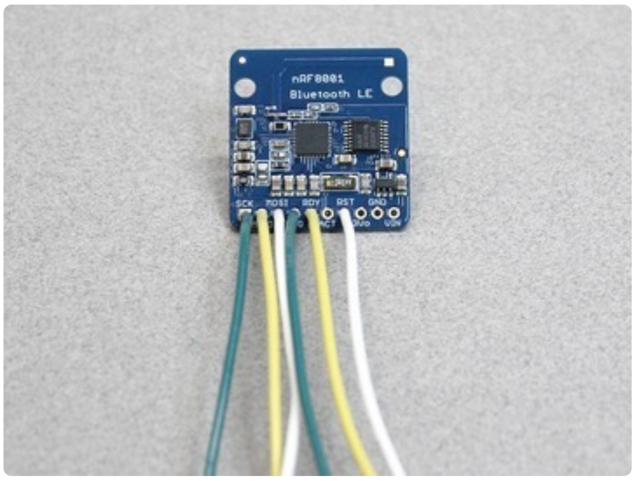
The labels on the Arduino Micro pins are very tiny! It may be easier just to count the pin positions in the diagram above.

The NeoPixel data input is connected to pin 6 on the Arduino. That 2-pin plug we salvaged earlier is now used to make the electronics removable for troubleshooting or modifications.

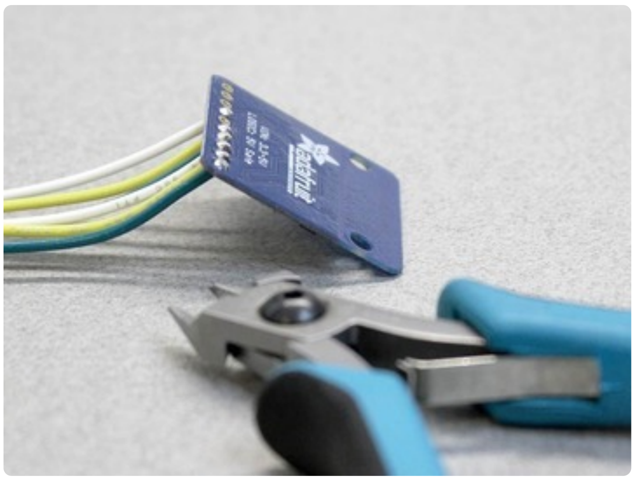


Start by cutting 6 wires about 2 inches (5 cm) long. Strip about 1/4" (6 mm) insulation from one end of each.

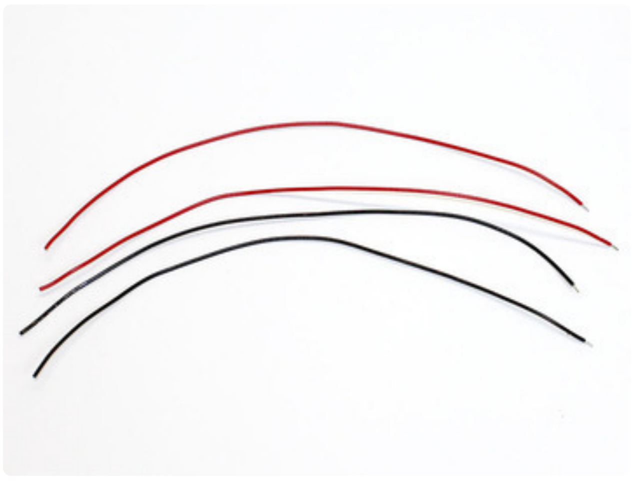
If you have a few different colors of wire, it makes things a little easier to follow, but it's not required. Just follow very methodically to be certain you're making the right connections.



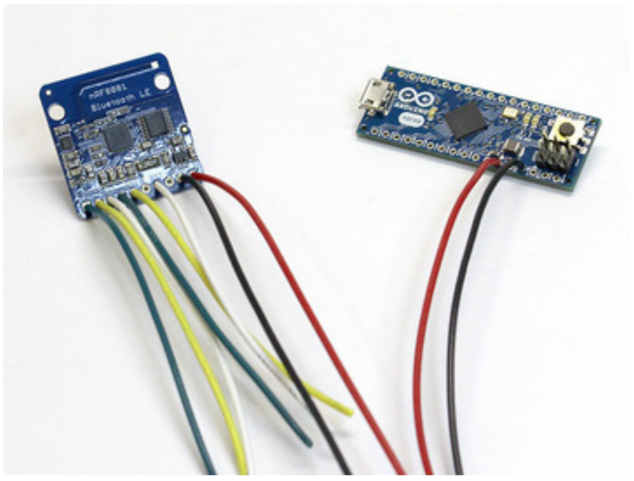
Solder these six wires to the Bluefruit board, starting from the left: SCK, MISO, MOSI, REQ, RDY and RST, skipping over the ACT pin.



After soldering, trim the wires on the back of the board so they don't stick out too much.

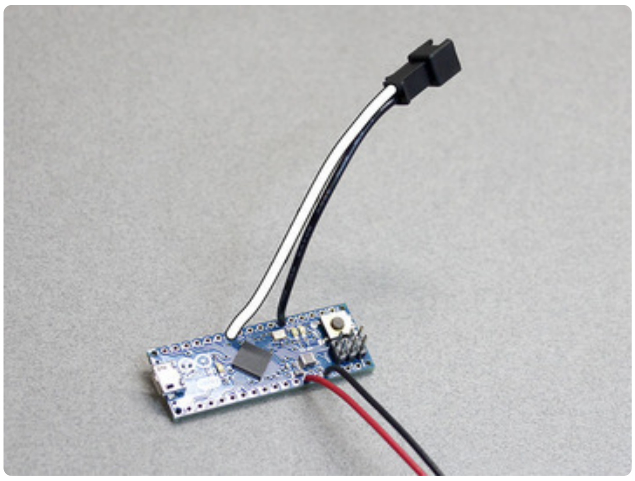


Now cut 4 wires about 10 inches (25 cm) long, and strip them at one end.



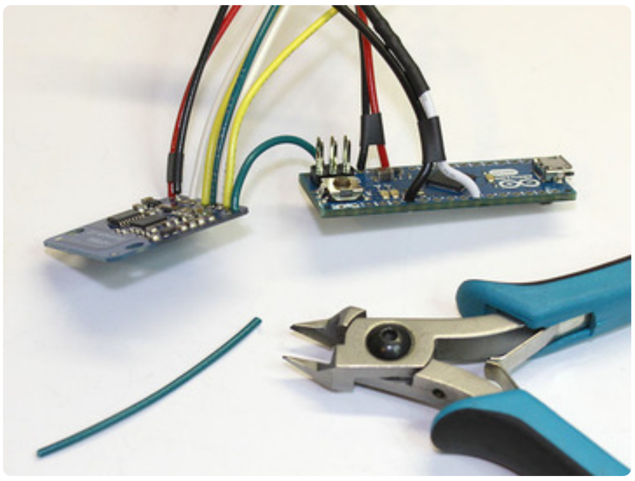
Solder these wires to the **VIN** and **GND** points on the Bluetooth board and the **5V** and **GND** points on the Arduino.

After soldering, trim these wires on the back as you did before.



Now solder the 2-pin JST connector to the Arduino.

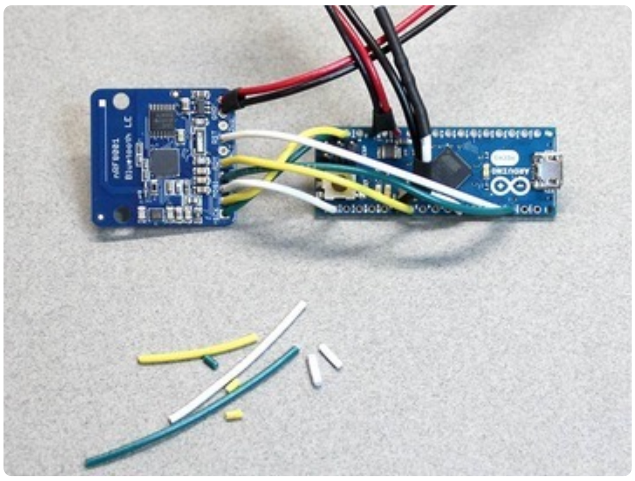
The **WHITE** wire connects to Arduino **PIN 6**. The **BLACK** wire connects to **GND** (there's a second GND connection on this edge of the board).



Arrange the two boards roughly as you expect to have them on or in the hat.

Cut each of the 6 data wires down to its required length, strip the end and solder to the Arduino. Then trim any excess wire on the back.

Here we've started with SCK on the Bluefruit board to SCK on the Arduino Micro.



Repeat with the remaining 5 wires. Each will be a slightly different length; trim as necessary.
SCK to SCK
MISO to MI
MOSI to MO
REQ to 10
RDY to 2
RST to 9

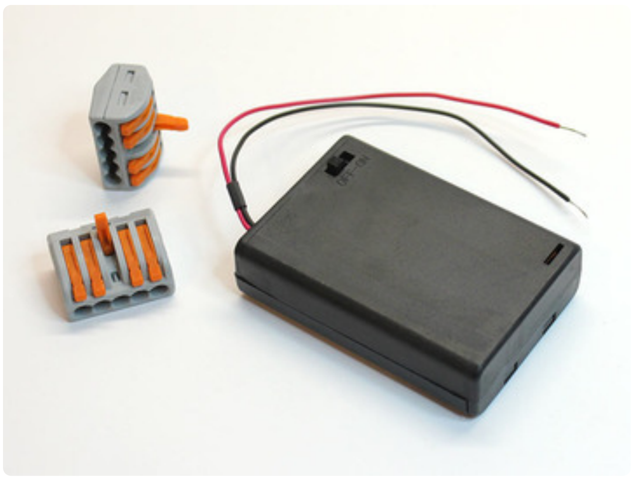
That should be it for the soldering! Keep the iron ready for a few tinned wires though...

Dry Run

Your battery box may look a little different than the one in these photos. If necessary, strip some insulation and twist & tin the end of the wires (melt a little solder into the strands to keep them from fraying).

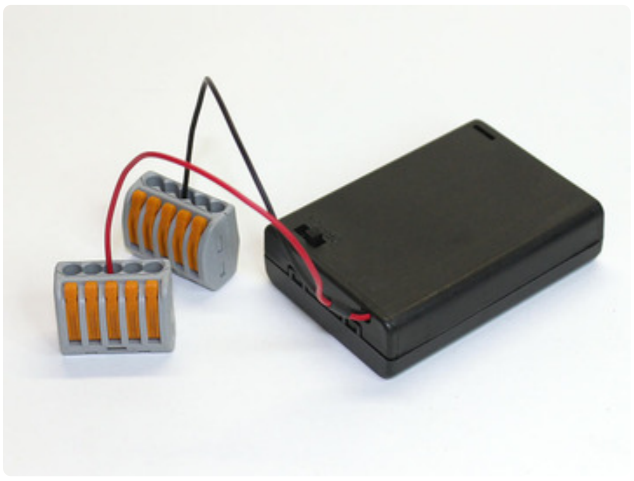
Insert 3 AA alkaline batteries (or 4 NiMH if you're using that type). Make sure it's switched OFF for the time being.

Now take two of the 5-wire block connectors and flip open one lever on each…

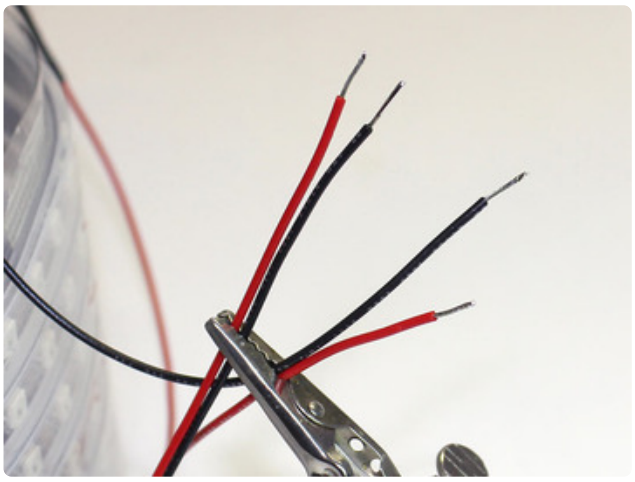


These levers require a LOT of force to open! Young makers might need a parent's assistance with this part.

They also snap shut quite forcefully, so be careful.



Insert the + and – leads from the battery box into separate block connectors and snap the lever shut. Look down the hole to make sure it's clamped down on bare wire, not the insulation.



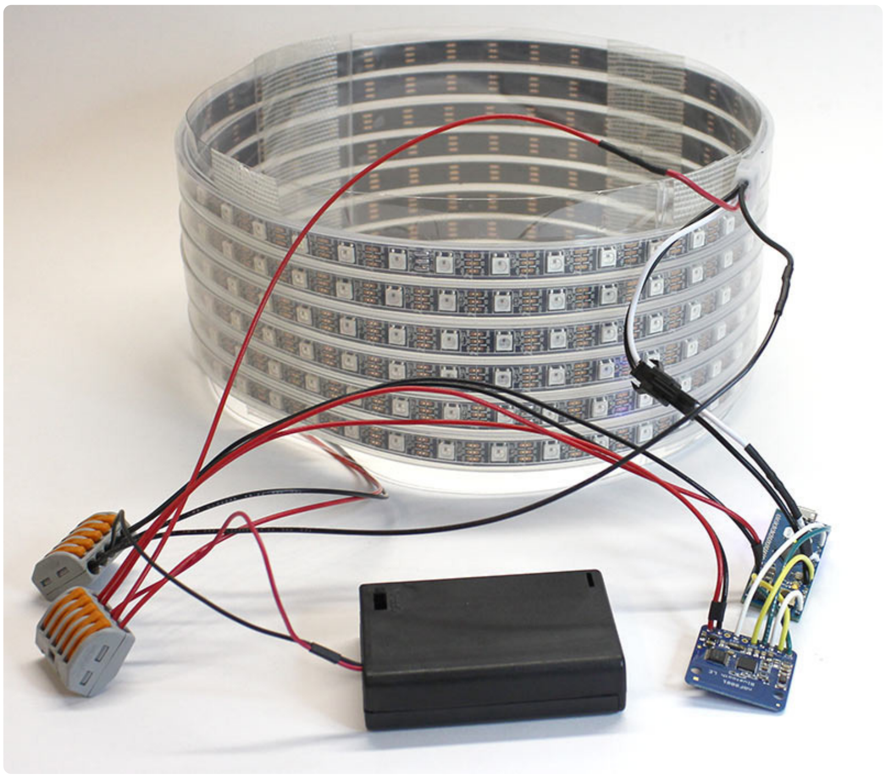
Strip about 1 cm of insulation from the end of each long wire (2 each from the Arduino, Bluetooth board and both ends of the NeoPixel strip — 8 wires total).

If stranded wire, give the ends a slight twist and then "tin" the wires with a little solder.

Open the remaining levers on the block connectors. Insert all of the power wires into the corresponding connectors (all + in one connector, all – in the other) and snap the levers shut. Give each a gentle tug to make sure it's got a proper hold.

Connect the JST plug to join the Arduino and NeoPixels.

Then flick the power switch…



If all goes well, you should get a solid blue light from the underside of the Arduino, and a pulsing green light on top. After about 10 seconds, the pulsing should change to a fast blinking. Good news! That means the code's running, the wiring's right, and the Bluetooth interface is awaiting a connection.

I'm not getting a blue light or a green pulse

- Did you install fresh batteries in the case? Are they each oriented the correct way?
- Look for electrical shorts; + and – may have inadvertently been crossed somewhere.
- Examine the block connectors closely. Are they biting down on wire, or on insulation?

Switch off the battery and try connecting a USB cable. If the computer complains that a device is drawing too much power, that's almost certainly an electrical short.

Blue light's fine, but the green light goes out after 10 seconds; no fast blink

Something's probably gone amiss when uploading the code to the Arduino. Go back to the "Code" page and try those steps again.

Are you using an Arduino Micro or Leonardo board? Other boards like the Uno don't have quite enough RAM, and the code will just hang.

Everything is awesome!

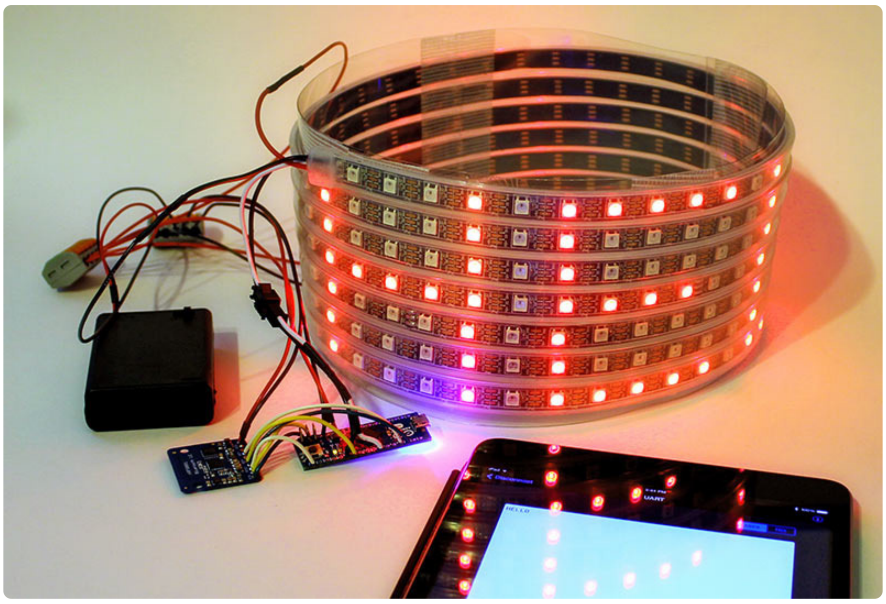
Whip out your phone or tablet…

To communicate with the Arduino, you'll need a late-model Android or iOS phone or tablet (must support Bluetooth 4.0 / Bluetooth Low Energy both in hardware and the OS). Install either the nRF UART app (available for both Android (https://adafru.it/dd6) and iOS (https://adafru.it/dd7)) or the Adafruit Bluetooth LE Connect (https://adafru.it/dd2) app (iOS only).

The status LED on the Arduino board will flash quickly while it awaits a Bluetooth connection. Running one of the above apps on your phone or tablet, select "Connect" or "UART Monitor." You may need to select the Bluetooth device from a list. Once a connection has been established, the LED on the Arduino will flash slowly to indicate that it's working.

If you're having trouble establishing a connection, try working though the Getting Started with the nRF8001 (https://adafru.it/dje) guide first. Once that's working, re-upload the Guggenhat code to the Arduino.

Type "HELLO" and press "Send." This should scroll the word around the hat, illuminated in red LEDs. Try other messages…anything up to 20 characters max. Later we'll explain how to change colors.



It kinda works, but the text is all scrambled gibberish

Count the number of pixels around the circumference of the hat. Don’t include the same column twice.

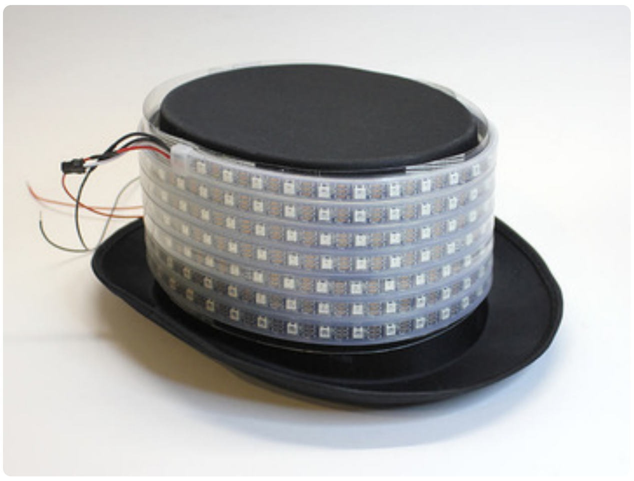
Edit the sketch and change the WIDTH value to this number. Re-upload to the board.

Once you’re satisfied that everything works, switch the power off and we’ll do the final assembly.

Final Assembly

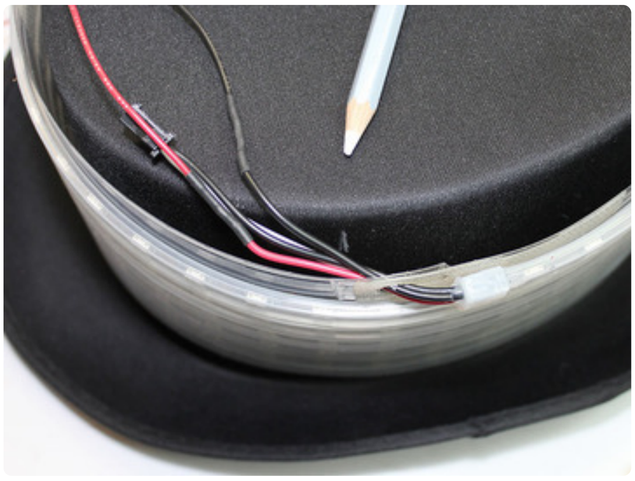
Unplug the JST connector between the Arduino and NeoPixels, and open all of the levers on the block connectors. Stuff will be moved around!

I decided to stick the battery and Arduino inside the hat, so there’s no protruding bits. Except for weather, there’s nothing wrong with keeping everything on the outside! This makes it easier to show others how it works…and you can further decorate it into a steampunk or cyberpunk aesthetic if you like.

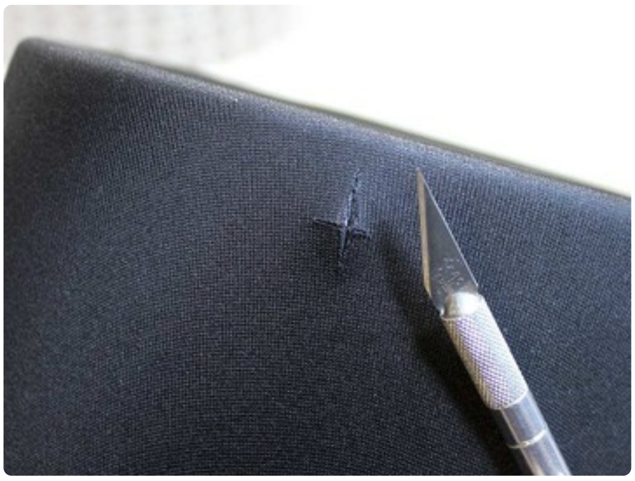


Slide the sleeve onto the hat, making sure it’s centered and facing forward.

Use the solder joint trick on the “Coiling” page to get it straight!

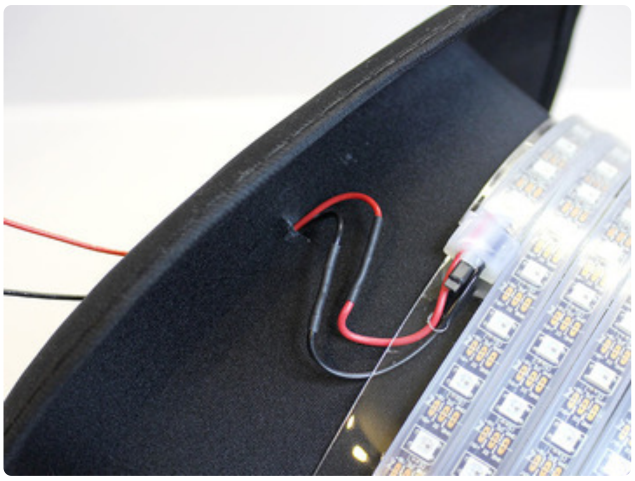


Using chalk or a colored pencil, make a mark where the plug and power wires from the NeoPixel strip should penetrate the hat.

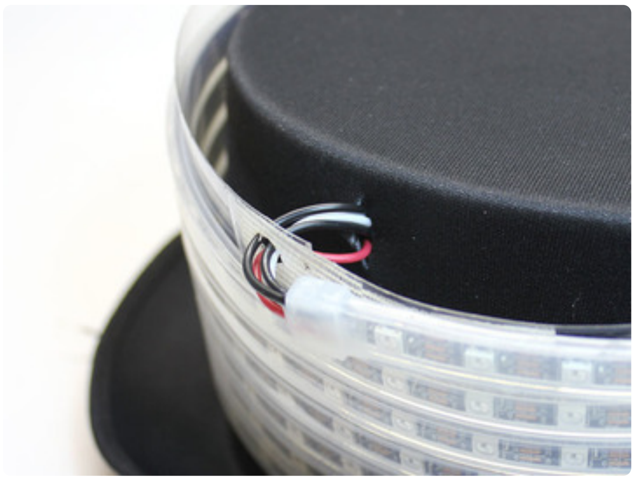


Remove the NeoPixel sleeve for a moment.

Using a hobby knife, cut a + shape in the hat, just large enough to push the JST connector through.

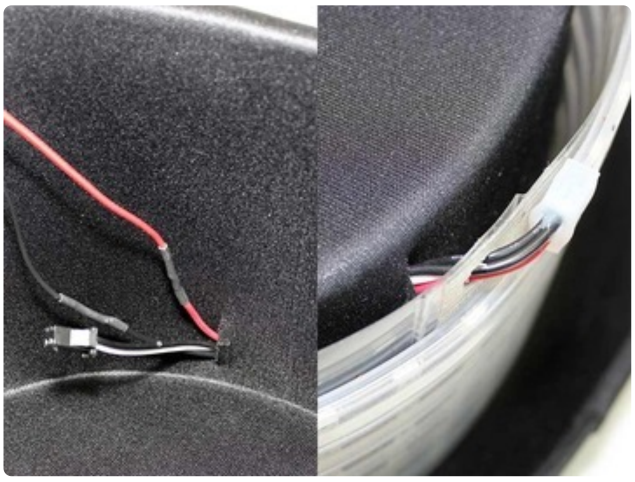


Repeat the marking and cutting operations for the wires at the other end of the strip, down near the brim of the hat. No JST plug here, so the hole can be smaller.



Put the sleeve back in place and push the JST connector through, then the power wires.

I'd previously punched holes through the sleeve for the wires to fit through. Not necessary, just me being OCD-retentive. You can cut a notch or simply pass the wires over the top.

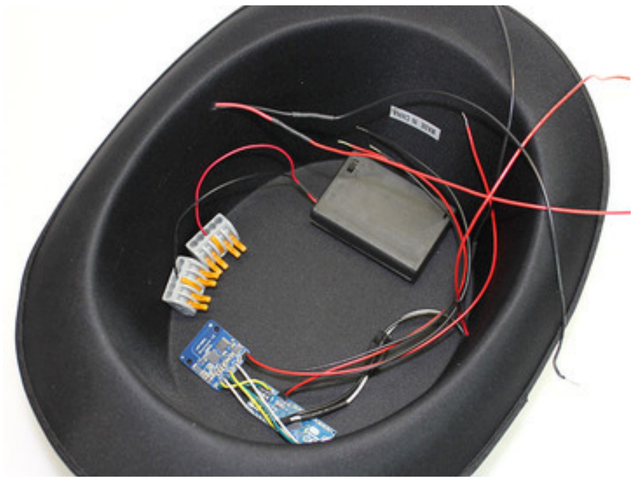


Boop! Pull the wires from the inside until everything's looking sufficiently tidy outside.

OPTIONAL:

Humans are humid. We all release some amount of water vapor through our skin, even if not damp from exertion. If you're particularly gifted in this department — nothing wrong with that — you might choose to seal the electronics in a conformal spray coating, rubber spray or even a thick acrylic.

Insert a USB cable and cover the wire ends with tape before doing this. Allow a few hours (or overnight) for the sealant to dry before continuing.



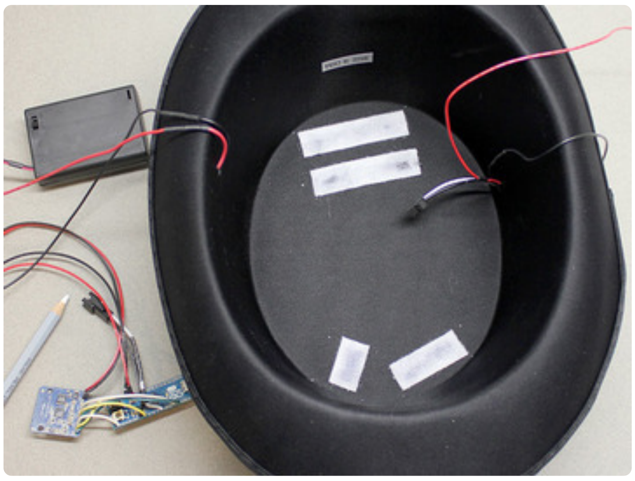
Plan where parts will go inside (or atop) your completed hat. Make sure the JST plugs can meet. Mark the part outlines or corners with chalk or a colored pencil; we'll need these footprints later.

The 3xAA holder now in stock features a push button rather than a switch, in a different corner. You'll probably want this rotated so the button is out near the perimeter of the hat…less likely to be accidentally toggled by the crown of one's head.

We'd like for the parts to be removable, so we can change batteries, update the code on the Arduino, etc. There's a few ways this could be done, whatever works for you: sew a pocket for the electronics, or glue some strong magnets…but the easiest approach is probably Velcro®…

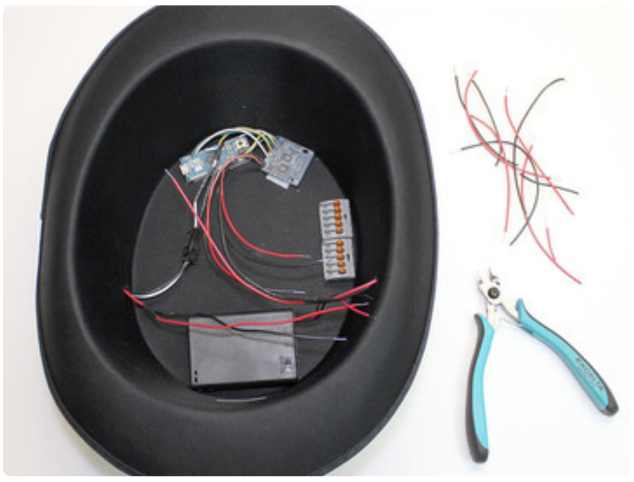
Self-adhesive Velcro® isn't the greatest…the stickum tends to come unglued when hot. I prefer to use plain Velcro® with a monster adhesive like E6000.

Here I've glued pieces of the "hook" side to the back of the battery cover, the Arduino and Bluetooth module.



Those part footprints outlined earlier? Corresponding "loop" Velcro bits are glued there.

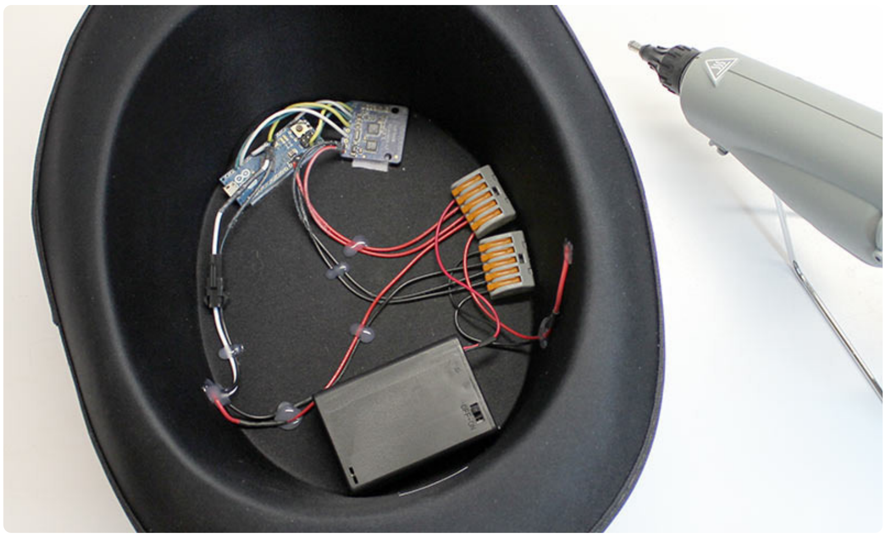
Nasty stuff, E6000. Put everything out in the garage and do something else for a couple hours while it dries. Play video games. Go for a walk. Practice your posh accent.



With the parts now in position, let's clean up the wiring. Determine an optimal path for each power wire, trim it to length, strip some insulation (and tin the end with a soldering iron, if using stranded wire).

Once the wires are all trimmed and routed, they can be plugged back into the block connectors.

I tacked the wires in a few spots with hot glue, and also used this to hold the block connectors in place. Hot glue usually isn't favored, but I'm using it here because it's weak…a dab of rubbing alcohol breaks the bond if needed and the parts and wires can be rearranged and re-glued.



A piece of craft foam covers the electronics. This provides some necessary extra protection from bodily humidity, and also makes bumpy parts (like the battery box) a little more comfortable against one's head.

This piece is not glued in place, it's simply cut a bit larger than the hat and stays in place via friction. It's easy to peel this back and access the power switch and battery box. And you can replace this if it gets a little funky from sweat, hairspray, etc.

If needed, you can cut and glue strips of craft foam around the sides for a better fit. The hat's a little heaver than most and you don't want it tipping off when you look down to button your spats.

Use It!

If you've gone through the "Dry Run" page, you already know how the basics work.

To reiterate: install either the nRF UART app (available for both Android (https://adafru.it/dd6) and iOS (https://adafru.it/dd7)) or the Adafruit Bluetooth LE Connect (https://adafru.it/dd2) app (iOS only) on a late-model phone or tablet that supports Bluetooth LE.

The status LED on the Arduino board will flash quickly while it awaits a Bluetooth connection. Running one of the above apps on your phone or tablet, select "Connect" or "UART Monitor." Once a connection has been established, the LED on the Arduino will flash slowly.

Type some text (up to **20 characters maximum**) and press “Send” to update the scrolling message.

With seven rows of LEDs, certain lowercase letters like ‘j’ or ‘g’ get clipped. Messages tend to look better typed in ALL CAPS.

To change the text color, type a number sign (#) followed by a hexadecimal color value; each digit is in the range 0-F (where A-F equal 10-15). If you’re familiar with HTML color values, it’s essentially the same format, e.g.:

```
#00ffff
```

...will set the text color to cyan.

You can also use three-digit hexadecimal color values. This has a smaller available palette, but the color fidelity of this project is limited anyway (more on that below).

Some common colors include:

```
#f00    Red
#0f0    Green
#00f    Blue
#ff0    Yellow (Red + Green)
#0ff    Cyan (Green + Blue)
#f0f    Magenta (Red + Blue)
#fff    White (Red + Green + Blue)
```

Sometimes I enter a slightly different color value and it looks the same. Other times, the text disappears.

Because we’re trying to run a ton of NeoPixels off a fairly small battery supply, the Guggenhat sketch limits the maximum LED brightness (32 levels rather than 256), reducing the available palette. This is compounded by the NeoMatrix library, which uses a gamma-corrected 16-bit colorspace. Between these two, many color values end up falling into the same bins...and values below a certain threshold simply round down to zero. You’ll do best sticking to bright primary and secondary colors, not nuanced tones.

How long will the batteries last?

This depends on many factors...the choice of colors and messages, and the type of battery.

Primary colors (red, green, blue) will use less power, since only a single color within each RGB pixel is lit. White uses the most power; red, green and blue must all combine for this color.

I've successfully run the hat for over four hours using incredibly crappy dollar-store AA cells.

The message will freeze (not turn off) as the batteries approach depletion. This is the Arduino locking up as the voltage dips too low. It's not harmful, just time to change the batteries.

Variations

Different Arduino Code

We're controlling the LED strip using the Adafruit_NeoMatrix library, so it's easy to display different graphics. All the drawing functions from the Adafruit_GFX library (https://adafru.it/doL) are available. For example, the blinking eyes from this LED matrix tutorial (https://adafru.it/doM) could be adapted.

There are a couple of limitations to keep in mind: first, you can't light up all (or even most) of the pixels at once, nor at full brightness…this would drain the battery within minutes! Second, 240 NeoPixels use a lot of RAM…and the Bluetooth library needs some as well, if you continue to use that. So, for example, adding a microphone and displaying an audio VU meter using an FFT algorithm is probably asking too much (both in terms of RAM and power consumption), but some variant on the Ampli-Tie project (https://adafru.it/doN) could probably be created, provided you don't light too many LEDs at once.

Dynamic Messages

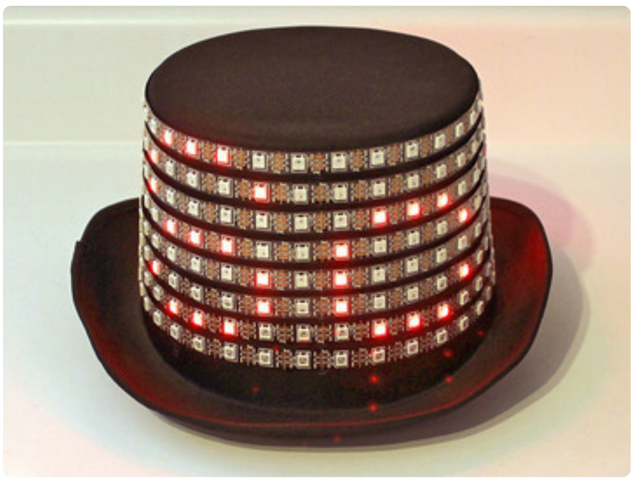
If you're skilled with Android or iOS programming, it should be feasible to send new text to the hat…for example, weather conditions or live stock quotes pulled from the internet. The Bluetooth connection appears as a serial port. The only "gotcha" is that messages must be 20 characters or less, a limitation of Bluetooth LE.

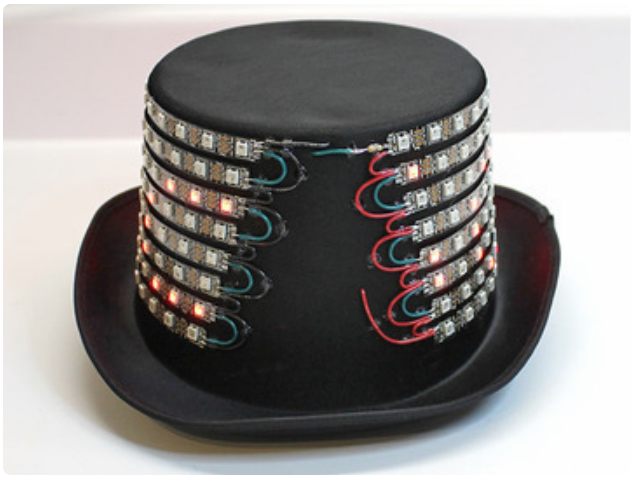
“Invisible” LEDs

A trick learned from this electronic demon costume (https://adafru.it/doO): you can hide the LEDs (so it looks like a plain hat when they’re off) by pulling a black nylon stocking over it. This blocks about half the light, so it’s best seen in dark settings, or compensate by increasing the LED brightness (with a corresponding decrease in battery life).

8 LED Rows

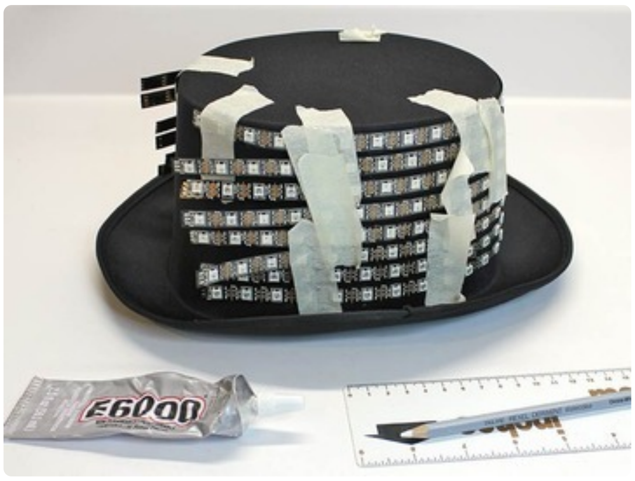
With only 7 rows of LEDs, the “coiled” hat cuts off descenders (lowercase letters like ‘g’ and ‘j’)…it’s even more pronounced at the back, where there’s only six rows.

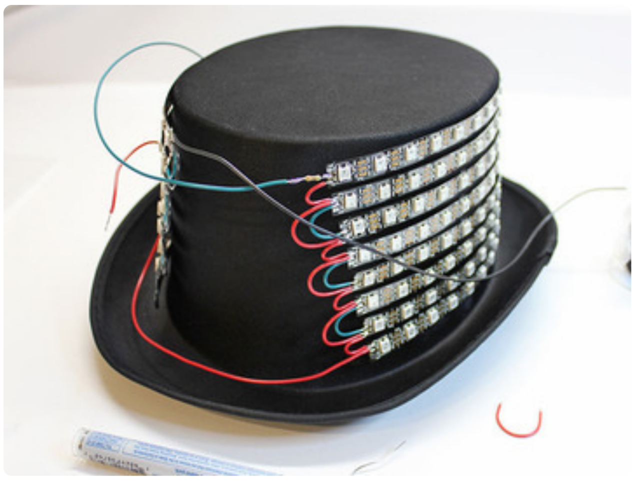




By cutting apart the strip and arranging the LEDs in a grid (rather than a coil), we can give up a few columns at the back to gain an extra row. This sacrifices weather resistance, but the result is sleeker and lower in profile.

This is a challenging and time-consuming modification, recommended for advanced makers with excellent soldering skills and lots of patience.





After cutting off the silicone sleeve and separating the strip at the half-meter solder joints, the sections were glued to the hat with 1/2" spacing all around (they don’t quite sit flush against the hat due to the taper). Alternate rows were flipped so the data direction is right-to-left instead of left-to-right (a zig-zag arrangement). The last 3 inches or so were not glued down until after the soldering was complete.

Code Changes

A few modifications to the sketch are needed to use this “grid” hat. First, near the top of the code, change these lines:

```
#define NEO_WIDTH  30 // Hat circumference in pixels
#define NEO_HEIGHT  8 // Number of pixel rows
```

Change the matrix declaration to use a "zig zag" order:

```
Adafruit_NeoMatrix matrix(NEO_WIDTH, NEO_HEIGHT, NEO_PIN,
  NEO_MATRIX_TOP  + NEO_MATRIX_LEFT +
  NEO_MATRIX_ROWS + NEO_MATRIX_ZIGZAG,
  NEO_GRB         + NEO_KHZ800);
```

And comment out or delete this line in the setup() function:

```
  matrix.setRemapFunction(remapXY);
```