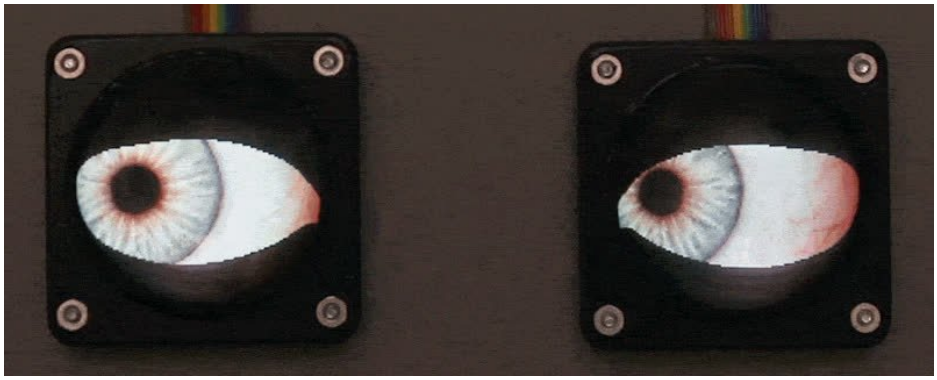
adafruit learning system

Electronic Animated Eyes for ARM Microcontrollers

Created by Phillip Burgess

Last updated on 2019-11-19 04:41:35 AM UTC

Overview



Are you looking into these creepy animated eyes...or are they looking into you?

These peepers were inspired by a concept by David Boccabella (Marcwolf) on the Stan Winston School of Character Arts forums (https://adafru.it/fXO). David is creating servo-driven animatronic eyes with a small OLED screen to simulate a dilating pupil. I'd chimed in with some suggestions and code to boost the refresh rate. Taking it to the next level here, we can render the *entire eye*...this makes the overall project simpler, as animatronics can be very fussy work (and the noise detracts from live performance).

UPDATE: This project was originally designed with the PJRC Teensy 3.1 or 3.2 microcontroller board (https://adafru.it/j7a) in mind. Since then, the code's been expanded to support many Adafruit boards: most Feather (https://adafru.it/G-A) or ItsyBitsy boards with the "M0 (https://adafru.it/BI9)" or "M4 (https://adafru.it/BQC)" designation (*not* AVR or 32u4 boards), plus HalloWing M0 (https://adafru.it/CmY), and Circuit Playground Express (https://adafru.it/wpF) with the optional TFT Gizmo (https://adafru.it/Gff). For the HalloWing M4 (https://adafru.it/G-B) and MONSTER M4SK (https://adafru.it/FLO) boards, we have a more evolved eye project (https://adafru.it/GfQ).

Most of the documentation and diagrams here proceed with the Teensy 3.2 in mind. With a pinout diagram for one of the above boards, it's usually straightforward to adapt.

This is a "choose your own adventure" project. There are many ways to build it...this guide is not really aiming toward any particular finished *thing.* It's the start of a recipe, but where it goes is up to your imagination...

- Spooky eyes in the window for Halloween
- An amazing costume for Dragon*Con
- A single eye worn in a pendant or a bracer, or in the headpiece of a staff
- World's creepiest taxidermy

Read through to see what parts are involved for different configurations.

Parts from Adafruit:

- **PJRC Teensy 3.2 microcontroller (https://adafru.it/j7a)** (one board will control 1 or 2 eyes); Teensy 3.1 will also work. **(As noted above, many Adafruit M0 and M4 boards can also be made to work.)**
- Display(s) - **one per eye**: either **1.5" OLED (https://adafru.it/fXP)** or **1.44" TFT (https://adafru.it/dXl)**. OLED looks *amazing* but costs more. TFT is affordable but colors are less intense. Tradeoffs!

Additional parts and tools:

- Soldering iron and paraphernalia
- 28 gauge ribbon cable
- Optional: 1.5" Acrylic cabochons (half-spheres)
- Optional: 3D printer to make enclosures, held with #2-56 screws and nuts
- Optional: LiPoly Backpack and battery (500mAh for OLED, 150mAh for TFT)

The following components are OPTIONAL. Our software can handle all of these effects autonomously, but you can optionally add any or all of these parts to enable manual control:

- Analog joystick for movement
- Button(s) for eye blinks or winks
- Photocell makes pupils react to light

This project *requires* the Teensy 3.2 (or 3.1) microcontroller (or one of the aforementioned Adafruit boards). *Not* the Teensy 2, 3.0, LC, nor any other Arduino-like board, period. **It relies on features unique to ARM microcontrollers.**

This project involves detailed soldering around costly parts; **read through before deciding if this guide is for you**. Newcomers to electronics might start with the Animating Multiple LED Backpacks (https://adafru.it/iwB) guide — it achieves a similar effect with easier, more affordable components!

3D Printing

This project doesn't necessarily *require* 3D printing. Depending what you're making, it may be sufficient to mount the display breakout boards on something as-is.

These little 3D-printed enclosures are *useful* for holding domed lenses over the displays. And they're absolutely ***essential*** **if creating something wearable.** The ambient humidity in a costume will *kill* exposed circuit boards in no time!

We'll start with the 3D printing because it affords the opportunity to **test-fit** these parts before buying the electronics, to check whether they'll even work in the spot you have planned. They add some bulk behind the eyes and the whole idea may be a bust.

Lenses and Hardware

1.5 inch (38mm) cabochons (domes) magnify the screens slightly and give the eyes a cool 3D shape. I found mine at Tap Plastics (https://adafru.it/iwC), but any good plastics supplier should have these...or there's eBay or Etsy. For good magnification and for the cases to hold them properly, the lenses you use should have a high dome to them...a full half-sphere.

Each enclosure requires *four* (4) #2-56 flat-head machine screws, 3/8" long, plus matching nuts.

This is another "probably easier to find online" part, unless you're blessed with a well-stocked local hardware store.

3D Printables

The enclosure pieces are small and will fit even on compact entry-level printers.

https://adafru.it/iwD

https://adafru.it/iwD

There are separate versions of the enclosure for OLED vs TFT LCD...the mounting holes and cutouts are slightly different. There's a top and bottom piece for each: for example, "LCD Top.stl" and "LCD Bottom.stl."

I found it best to print each part as a separate job (rather than tiling all the parts on the printer bed) — less oozing / strings means less post-print cleanup — but every printer is different and maybe yours fares better in this regard.

Some filaments such as ABS are known to shrink slightly (about 2%). You may need to scale the .STL files very slightly larger before printing. DO NOT force parts into a too-small case...THEY WILL BREAK.

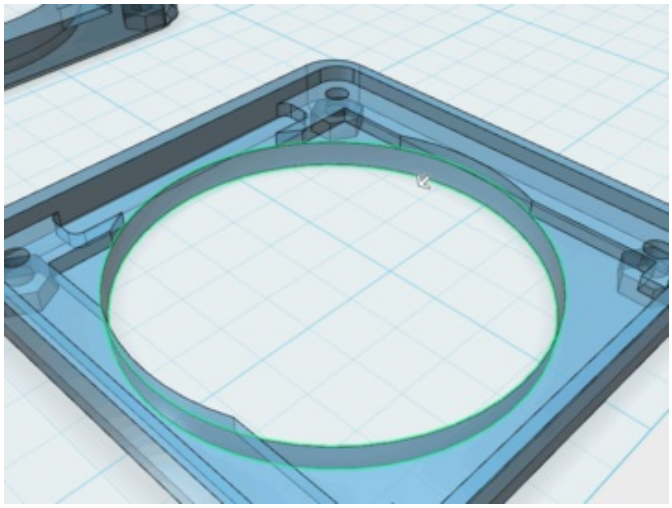
File or sand away any major protruberances. If you rinse off the parts afterward, make sure they're *completely* dry before assembly, maybe leave them on a fan for a couple hours for good measure.

There's a lot of variation among cabochons (lenses) from different sources, and even different batches from the same source. So it's possible they won't fit perfectly on the first try...

The ideal goal is for the lens to *just* fit in the case front, with the back faces flush. Too snug and the lens will press against the screen (possibly cracking it), too loose and it will simply fall through.

If a little too snug: use sandpaper around the opening to make it just a little wider, and try again.

If too loose: if it falls through but is a *close* fit, that may be good enough...we'll glue the lens and case later.



If it's *really* loose or snug, you may need to tweak the geometry to fit your specific cabochons. A CAD model for Autodesk 123D is included with the files. Use the "Press/Pull" feature to tweak this ring...positive values for a tighter fit, negative for looser...try just a fraction of a millimeter at a time. Export as STL and try again.

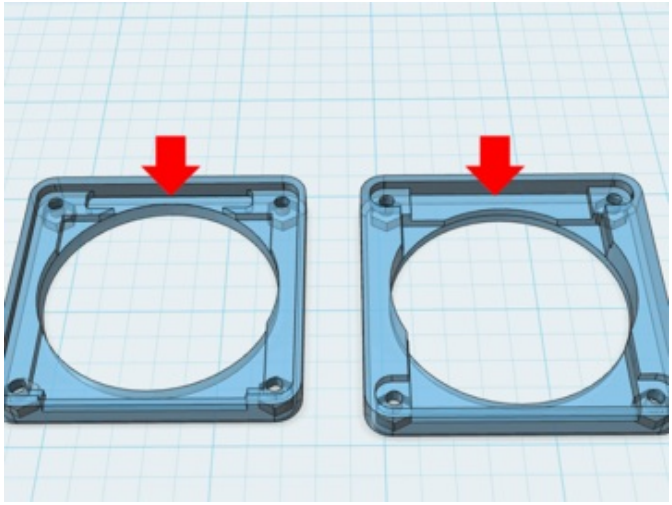
Assembly

Assembling the screens requires that the electronics be completed first. But since the 3D printing doesn't apply to everyone, it's all kept on this page rather than throughout the guide. Therefore, **go ahead and work on the electronic assembly** starting on the next page, and **return here** when you're ready to assemble the enclosures.

Electronics should be tested and working first, then return to this page.

For each eye, you should have **four** major parts: a top and bottom case, a lens, and the display with ribbon cable attached. (Plus the aforementioned screws and nuts, not shown here.)

Peel off the plastic screen protector if you haven't done this already.



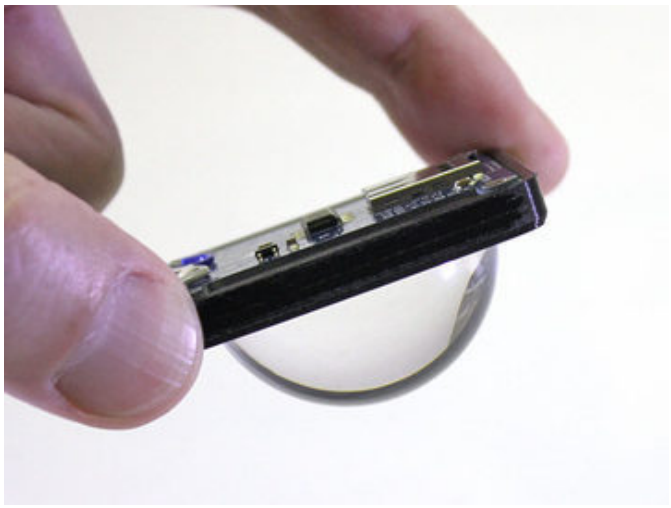
The display and front piece fit together a certain way — it's not symmetrical. A little notch provides some clearance for solder connections that tend to protrude from the front of the board (with the wires on the back).

The solder connections should be at this end of the case. NOT the thin plastic ribbon cable to the screen.

I don't have a 3D-printed case design for the Teensy yet. If this is going inside a costume, you'll need one, to keep out moisture.

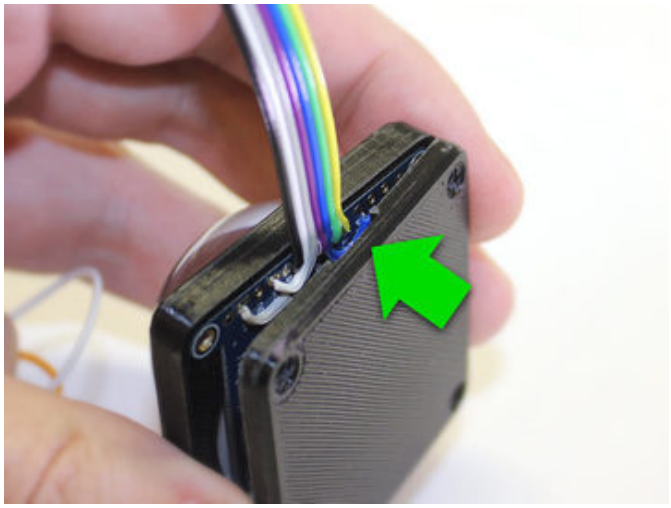
If you're handy with 3D CAD, it shouldn't be too hard...a rectangle with some cutouts for wires. Otherwise, you can just get creative with a small plastic box (like some mints or breath strips come in) and hot glue. No rocket science required.

DO NOT FORCE ANYTHING. The screen glass is thin and incredibly fragile. If there is resistance during installation, stop and check tolerances, make modifications to the case as needed.

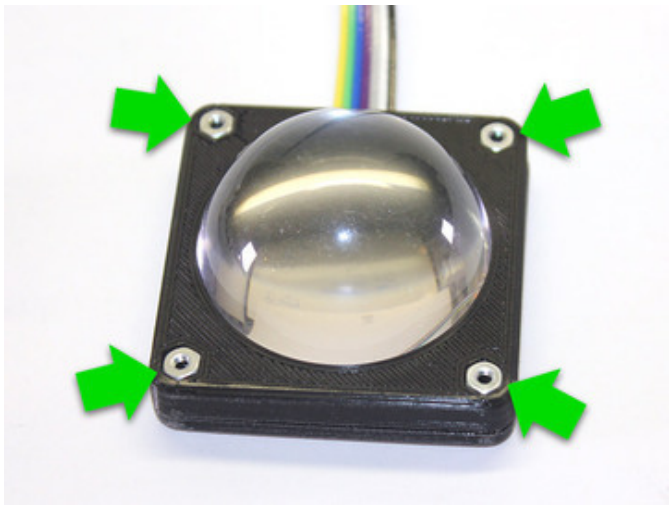


Ideally, this is how the front will go together. Board sits flush in case, dome sits flush against display...doesn't press hard against it and doesn't fall out.

As mentioned earlier, if it's too tight you can sand the opening, if too loose the dome can be glued later.

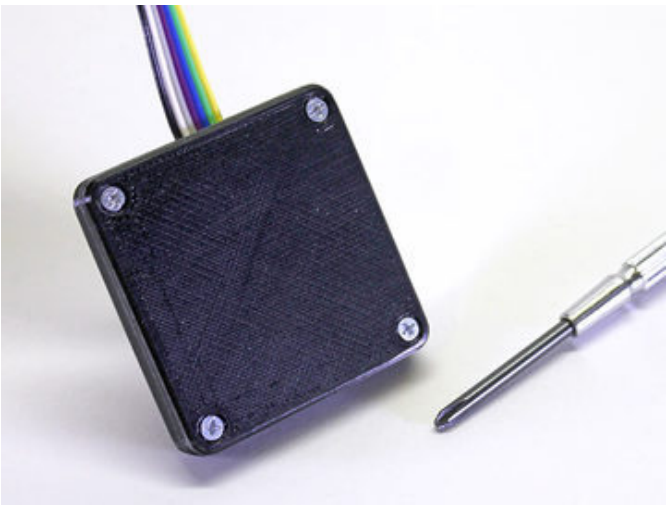


A small notch on the back of the display provides clearance for the ribbon cable. (It's slightly off-center on the OLED case, this is on purpose.) File as necessary if it doesn't quite fit.



Fit four #2-56 nuts to the notches in the front of the case.

The notches may have some detritus from 3D printing... clean these out with tweezers or a file.



Add four #2-56 x 3/8" flat-head machine screws from the back, and tighten carefully.

If the case puts up resistance to closing, **STOP.** Something inside isn't fitting right. Open the case and look for any plastic that needs filing down, or wires not sitting flat.

The display is made of glass and will break if forced.

Gluing the Lens

If the lens is loose, it needs to be glued in place. Even if it's a good fit, gluing is a good idea for added durability! But choose wisely...

Hot glue is too clumsy and random; we need **fine control**. Other glues are too runny...they'll **seep into the gap and ruin the screen**. Some react badly with acrylic and will make the cabochon hazy. (Cyanoacrylate glue is both too runny *and* causes haze...do not use it for this!)

DO NOT USE CYANOACRYLATE ("KRAZY") GLUE.

Don't laugh...**t-shirt puff paint** actually makes a decent adhesive and sealant for this project! The applicator tip lets you draw a fine bead where the case and lens meet. Just like caulking a bathtub! Allow several hours to a full day to dry completely.

You could also use this to draw a gasket between the two case pieces before closing it up, and add a bead around the hole where the wires enter. Obviously, you should test all the electronics 100% first before committing to this step.

For a more *industrial* bond, these same steps can be done (*very carefully*) with a toothpick dipped in **epoxy** or **E6000** craft glue.

DO NOT USE CYANOACRYLATE ("KRAZY") GLUE.

The enclosures can now be installed into something else (e.g. taxidermy dinosaur head) with your adhesive of

preference...hot glue, E6000, etc. and will link up with the Teensy board as described on the “Wiring” page.

Wiring

Before diving in, give some thought to how you'll be using this.

At the very least, this project uses a Teensy microcontroller and a display or two. The animation effects are all capable of running **autonomously** if need be, or there are options to have them **manually controlled**. The **method of assembly** is also open to interpretation.

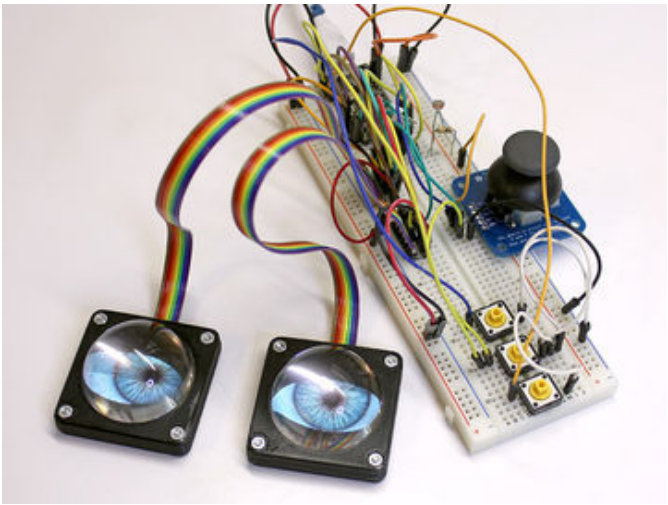
If you're making a spooky prop to sit in the window on Halloween, but plan to **disassemble** it afterward and use the parts in other projects, you can use a **breadboard** and **jumper wires** for quick assembly and re-use.

For something **portable**, like jewelry or a costume piece, **soldering** wires directly between components is vital — both for space savings and for durability.

If creating eyes for a puppet, you probably want *manual controls* for nearly everything, as that's the very *nature* of puppetry.

For a costume, I think *autonomous* works better. Good cosplay is all body language...but when electronics are added, fantastic characters are spoiled when the performer is focused on modes and buttons. But hey, it's up to **you.**

Give it some thought. I'll wait!



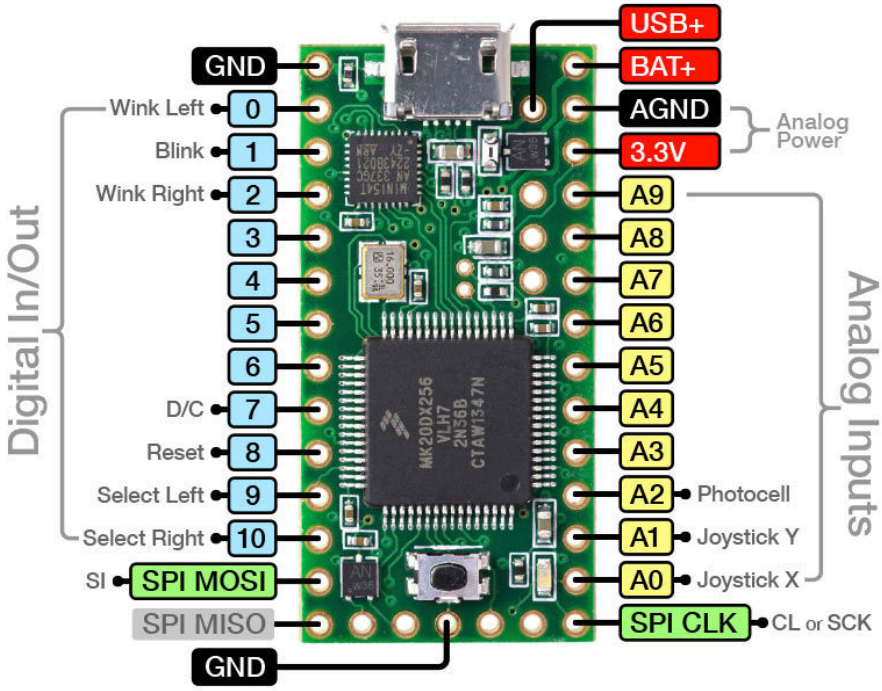
Half and half: these eyes are neatly assembled in 3D-printed enclosures while the rest of the circuit is a messy breadboard. Once everything's tested and working, the breadboard side will be replaced with a more permanent solution and fitted inside a Halloween prop. Whatever works for your needs!

Connections

Many configurations of this project are possible, depending on the features you're after. Rather than a single complex wiring diagram, a few subassemblies are illustrated here...pick and choose to match your needs.

What's shown here are *schematic* diagrams — they indicate *where* to connect wires, but **not necessarily their exact actual layout or lengths** when you build the thing. You'll need to think how everything fits in your own setting. Using color-coded wires helps a *lot* here!

We'll be referring to several pins by name or number, so here's a pin map for the Teensy 3.1 or 3.2 microcontroller:



This is a simplified pinout to clarify things specifically for this guide. If you plan to add your own advanced bells and whistles, a more complete pinout map is available on the PJRC web site (https://adafru.it/iwE).

Ground connections are vital for distributing power throughout the circuit. In addition to the two GND pins labeled here, there are two copper pads on the back of the board — one near the center, and a second large one near the USB port (next to the GND pin).

The **AGND** pin provides a cleaner ground reference specifically for **analog inputs** — this is *not* a current-carrying pin for distributing power, do not connect the displays here.

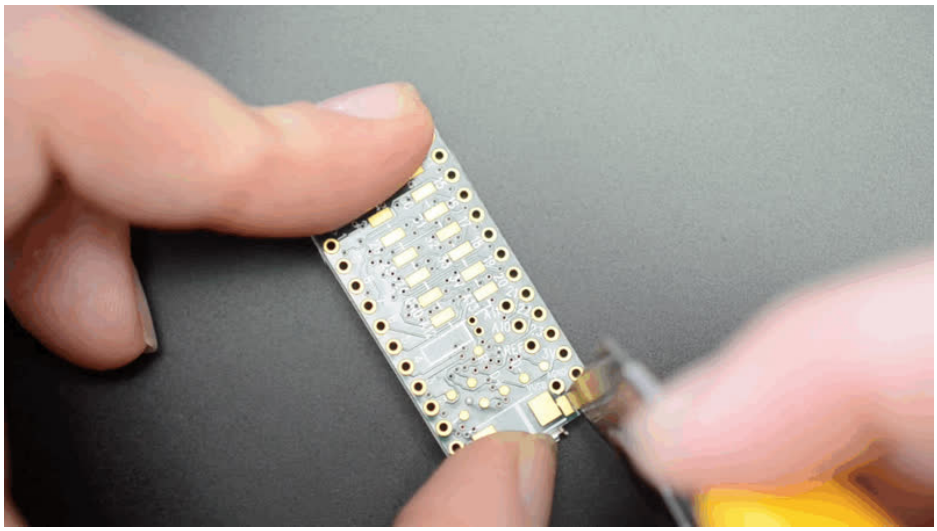
Some of the pin numbers we'll be referencing are negotiable...if you find that a different pin would make routing wires easier, there's usually a setting in the code that can be made for it. **Anything related to power or SPI is not negotiable...those wires *must* go to the pins stated.**

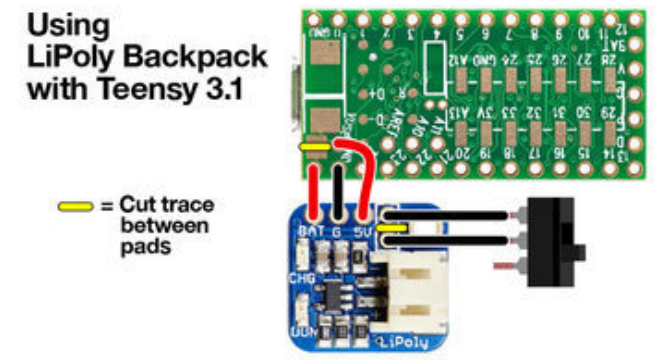
Power

If you have an opportunity to power everything from the Teensy's USB port (running a USB cable to a power bank or wall charger), that's easiest and reduces parts and steps.

The most compact, portable installations may optionally want a built-in Lithium-Polymer (LiPoly) battery. If your project uses **one or two TFT LCDs**, a **150 mAh** battery may suffice (though a larger capacity will provide a longer run time). For

one or two OLEDs, a **500 mAh** battery is the minimum size.



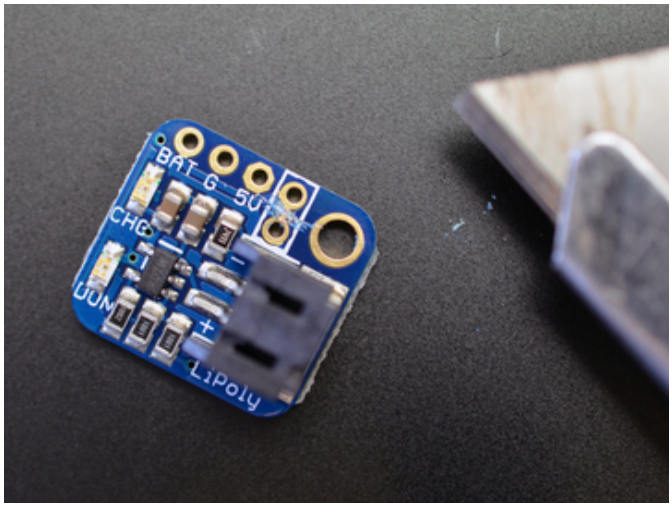


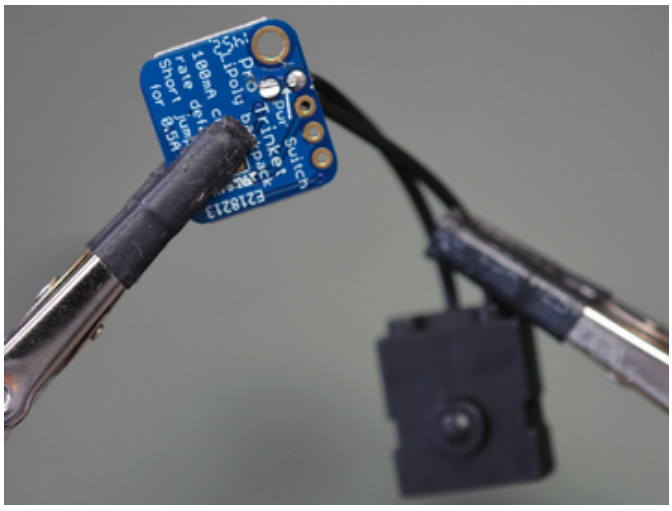
To use the Teensy board with the Adafruit LiPoly Backpack (allowing USB charging), first **two copper traces need to be cut:** between the two pads next to the Teensy's VUSB pin, and between the switch pads on the LiPoly Backpack (marked on back). Then add these **three wires** between the boards:

- LiPoly **BAT** to Teensy **VIN/BAT+** (unmarked pin at corner)
- LiPoly **G** to Teensy **GND**
- LiPoly **5V** to Teensy **USB+** pin

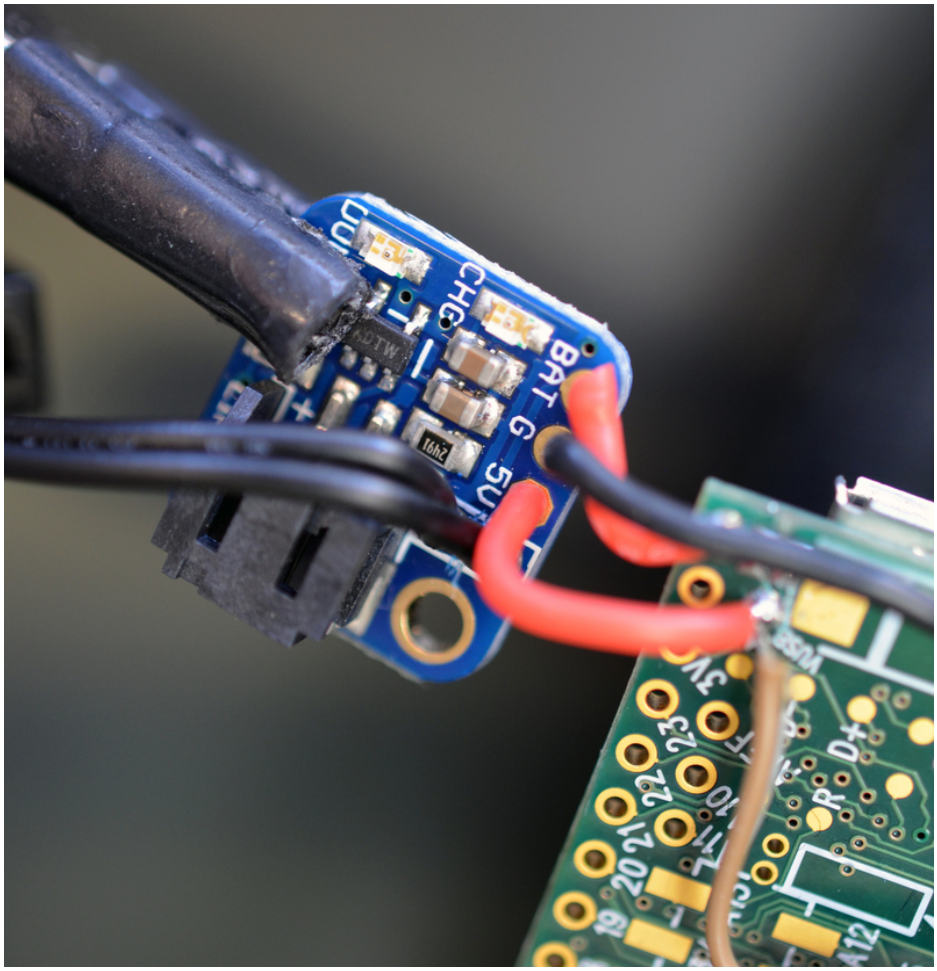
Add **power switch** to pins on LiPoly backpack. These tactile on/off switches (http://adafru.it/1092) are my favorite!

If battery capacity is **500 mAh or larger**, solder between the charge rate jumpers on the back of LiPoly backpack. **Do not do this with small batteries!**





When you cut the trace on the Teensy board, it won't run from USB power until the LiPoly backpack is connected and switched on. This is normal. If a sketch won't upload on a half-built project, this may be the reason why.



Power the display(s) from the BAT+ (corner) pin. If you're not using the LiPoly backpack (powering off a USB cable instead), that's okay — by default this pin also connects to USB+.

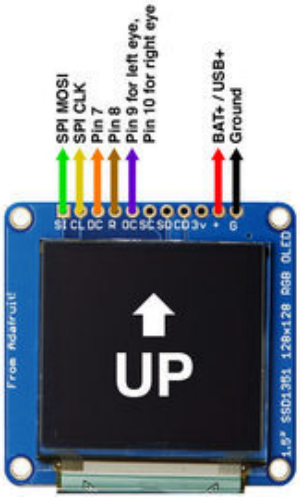
Displays

You have a choice of using one or two displays, either OLED or LCD. But you can't mix one of each — both must be the same type.

OLED displays have brighter colors and contrast. Downside is the price, and also that they flicker a little when captured on video.

TFT LCD displays are more affordable. Not as bright, but still a good effect. And they're rock-steady on video.

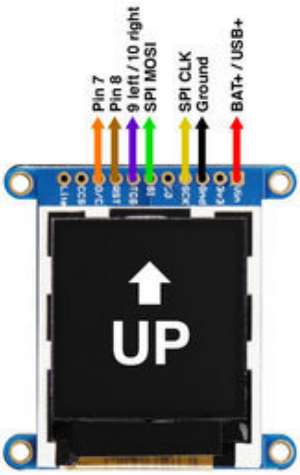
If making two eyes, both displays need to connect to the same SPI MOSI and CLK pins on the Teensy, plus a few other wires. The "OC" or "TCS" pins are unique to each display, left or right.



For **OLED** displays, make the following connections from the display breakout board to the Teensy:

- **SI** to **SPI MOSI**
- **CL** to **SPI CLK**
- **DC** to Digital **Pin 7**
- **R** to Digital **Pin 8**
- **OC** to Digital **Pin 9** (left eye) or **10** (right eye)
- **+** to **BAT+** (if using LiPoly Backpack) or **USB+**
- **G** to **GND**

The remaining four pins are not connected.

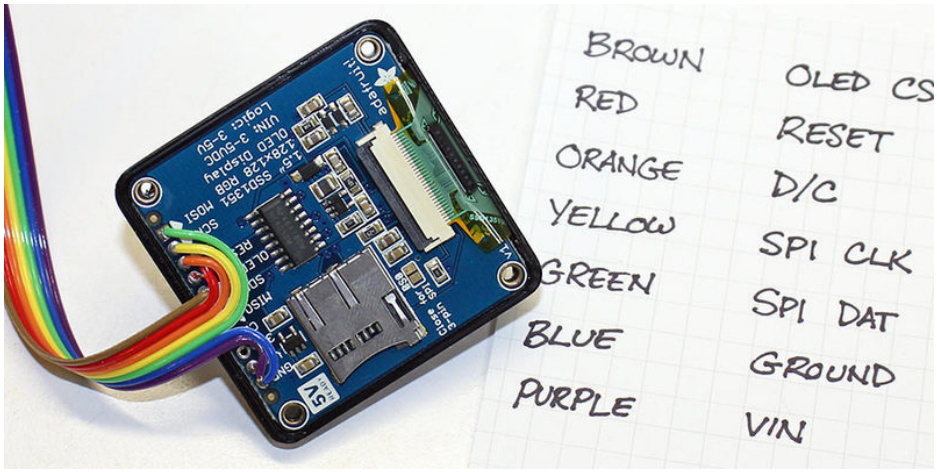


For **TFT LCD** displays, use these connections:

- **Vin** to **BAT+** (if LiPoly) or **USB+**
- **Gnd** to **GND**
- **SCK** to **SPI CLK**
- **SI** to **SPI MOSI**
- **TCS** to Digital **Pin 9** (left eye) or **10** (right)
- **RST** to Digital **Pin 8**
- **D/C** to Digital **Pin 7**

"Left" and "right" eye in this case refer to the positions when looking *at* the eyes, not from their point of view. This nomenclature is used throughout this guide and in the software.

If you're using them, the 3D-printed enclosures have a small notch that's *just* wide enough for a 7-conductor ribbon cable to fit through. Space inside is *really* tight (I wanted them to fit inside a mask), so it's necessary to route these wires to their pins *very* carefully so they lie as flat as possible on the back of the board, not all piled up. It's delicate work that requires tweezers and patience.

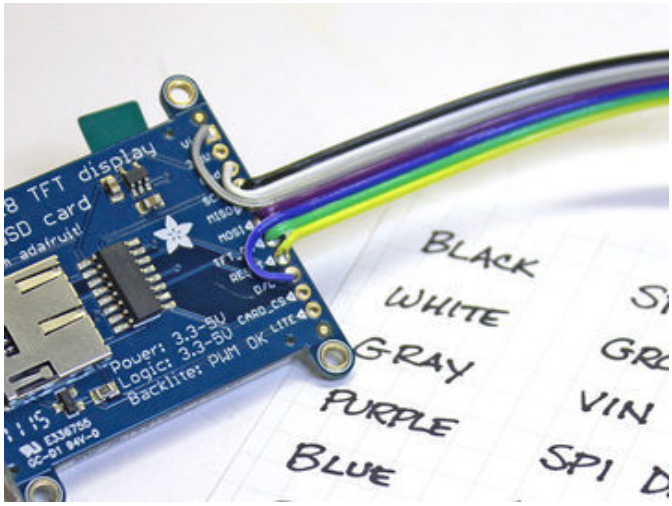


If using a ribbon cable as shown above, **write down your own legend that maps wire colors (or wire *numbers* if single-colored cable) to pin functions.**

Because the cable colors are in a fixed order and there's little space to reroute inside the case, certain wiring conventions used in electronics (such as using red wire for positive voltage and black for ground) no longer apply… you're forced to take what you're given. What's more, with wires doubled back, the conductors along the ribbon don't necessarily match the order along the display breakout header, it's all jumbled now.

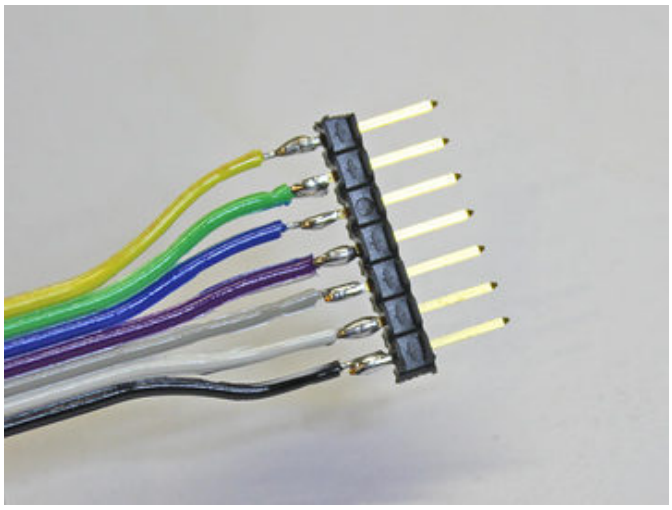
Do not use our photos for reference. *Do not* rely on the colors shown in any diagrams here. **Write down the *exact* sequence for *your* cable and *your* routing, and use *only* that for reference, nothing else.**

Write down your own wiring legend. Do not rely on the colors used in this guide.



The wiring for the TFT was even more "creative," with some wires a straight shot and others doubled back. Again, notice the wiring legend. Do what works for you.

If you're not using the 3D-printed enclosures, everything is much simpler. You can just connect to the pins in-order, there's no space constraint that must be met.



Wanting to test with a breadboard first, I made the ribbon cables extra long (about 8 inches) and soldered row pin headers on the end.

Later, for permanent installation, I'll cut the ribbon cables down to size and solder wires directly to the Teensy board.

This is totally optional...maybe you only want to build it once. Just try to **keep the wire lengths to a minimum**... high-speed SPI can be very finicky about this. Even 8 inches is pushing it.

The displays arrive with a plastic sheet on them. Keep this in place when soldering, but remove it once you're done. This isn't like a phone screen protector, just for shipping and soldering protection.

The display breakout boards include microSD slots, but those are not wired up in this project and the code doesn't reference them at all. Advanced users who want this capability can connect the SPI MISO and card select pins (you'll need to modify the enclosure slightly to accommodate the extra wires) and make the required changes in the code.

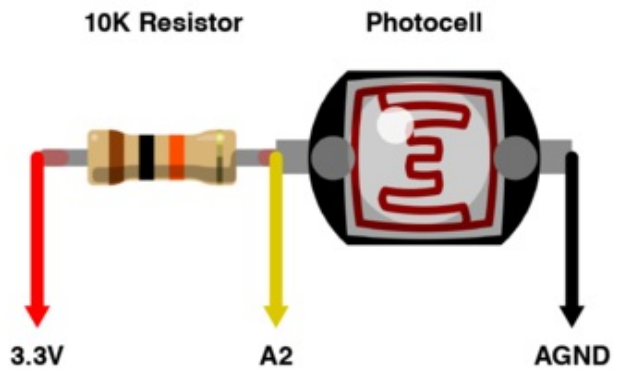
Analog Controls

Any **analog** controls that are used should include connections to the **3.3V** and **AGND** pins. Don't use the other power pins or there will be...trouble.

XOUT and **YOUT** from a joystick can connect to Analog Pins **A0** and **A1**.

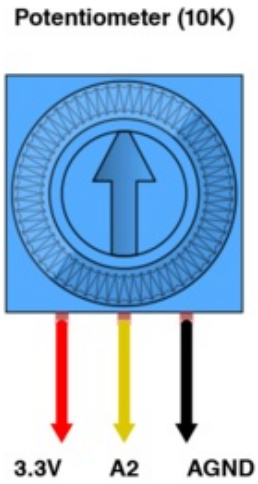
The eyes move autonomously by default — settings in the code enable the joystick instead.

If you need to mount the joystick in a different orientation, there are also settings to invert each axis. Swap the X and Y pins in the code to use the joystick sideways.



To have the pupils contract or expand in response to light, connect a photocell and 10K resistor in series. The midpoint connects to Analog Pin **A2**.

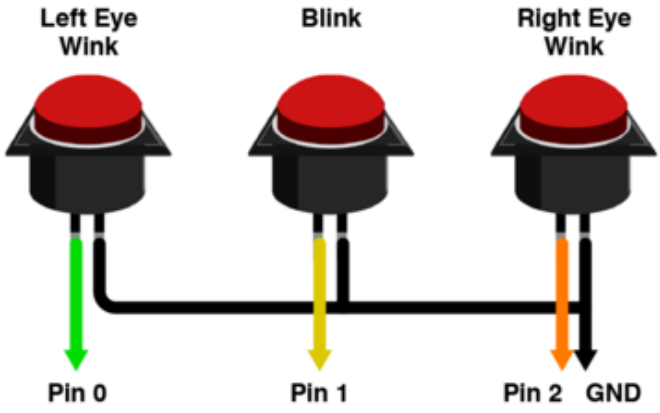
Analog input for the pupils (either photocell or the dial below) are enabled in the code by default. You can comment out IRIS_PIN in the code to have this move autonomously.



For manual control of pupil dilation (instead of responding to light) a 10K potentiometer can be used. The center leg connects to Analog Pin **A2** (same input as the photocell, just substituting a different analog control).

Buttons

The eyes normally blink autonomously, but you can also add one or more buttons to make them blink (or even wink individually) on command.



For all buttons, connect one leg of each to **GND**, and the opposite leg to a digital pin:

- Digital **Pin 0** is the **left** eye wink.
- Digital **Pin 1** blinks **both** eyes.
- Digital **Pin 2** winks the **right** eye.

If using our analog joystick breakout board, that stick includes a clicky button when you press down on it (on the SEL pin). This can optionally be used for manual blink control, or you can use a separate button for this (I find the joystick button a bit hamfisted).

Pro tip: a fully-equipped version of the circuit requires several ground connections, but there are only a few GND pins on the Teensy board. Two- or three-way splices are one option, but another way to provide extra ground connections is to repurpose unused I/O pins. Let's suppose you want to make Digital Pin 4 a spare ground pin. Add the following code in the setup() function:

```
pinMode(4, OUTPUT);
digitalWrite(4, LOW);
```

OUTPUT LOW makes the pin function as an ersatz ground connection. This works perfectly for button connections like the blink controls, but don't use it for heavy loads (like powering the displays) — it won't work and might even damage the Teensy.

Next Steps...

After wiring everything up, proceed to the next page and test out the software.

Once it's all tested and working...if using the 3D-printed enclosures, return to that page to complete the assembly. Otherwise, you're on your own now, to install the electronics in your own finished design.

Software

Before diving too deep into the software, there are **some gotchas** to be aware of...

- **Do not** install the Adafruit_GFX, Adafruit_SSD1351 or Adafruit_ST7735 libraries offered by the Teensyduino installer! **Use the Arduino Library Manager** or **install these manually** from Github code. The Teensyduino-installed libraries sometimes diverge from the latest Adafruit code and might prevent this project from compiling.
- When first building this project, please test initially with the canonical "Uncanny Eyes" sketch linked later in this guide, not anyone's derivative code. This will help with any troubleshooting/support. Once the default code works, *then* you can try out variants that may be out there.

Teensy uses the Arduino environment for programming, so it's pretty familiar and simple to work with, but it does require a little extra setup first...

If you're not using a recent version of the Arduino IDE (1.6.5 or newer), this would be a good time to upgrade (https://adafru.it/fvm). Once you have that software installed and working, download and run the Teensyduino installer (https://adafru.it/iwF), which adds support for the full line of Teensy microcontroller boards in the Arduino IDE (but remember, *don't* install the display/graphics libraries there...use the Arduino Library Manager instead).

From the **Tools** menu, select **Board→Teensy 3.2** and **CPU Speed→72 MHz** (and whatever optimization settings you'd like)**.** Confirm that you can compile and upload the classic "blink" sketch to the Teensy board. Don't use the 96 MHz setting; the code actually performs a bit better at 72 MHz (due to SPI settings).

Do not continue until you have the Blink sketch working on the Teensy board.

Using the Arduino Library Manager (**Sketch→Include Library→Library Manager...**) install **Adafruit_GFX** and **Adafruit_ZeroDMA**, plus the library compatible with your display: **Adafruit_SSD1351** for the **OLED** display, **Adafruit_ST7735** for **TFT LCD**.

(If you're still using an oldschool version of the Arduino IDE, these libraries can be fetched from Github: Adafruit_GFX (https://adafru.it/aJa), Adafruit_SSD1351 (https://adafru.it/cBz), Adafruit_ST7735 (https://adafru.it/aHm)and installed manually.)

Finally, there's the sketch code itself:

https://adafru.it/BU1

https://adafru.it/BU1

The sketch is utterly *ginormous.* In addition to several hundred lines of code in the main sketch, arrays containing graphics take up most of the space in the Teensy's prodigious 256K flash program space.

Before uploading to the board, check lines 56 and 57 in the file **config.h**:

```
#include <Adafruit_SSD1351.h>  // OLED display library -OR-
//#include <Adafruit_ST7735.h> // TFT display library (enable one only)
```

One line is enabled, the other is commented out. By default, OLED is used. Comment out the opposite line if using TFTs.

If using all the same wiring as the previous page, it should be possible to compile and upload to the board and see some results…you should at least see an eye doing *something.* (If you changed the wiring, skip ahead to the next section below to make the code match, then return here.)

The sketch doesn't compile!

Either Teensy support has not been correctly installed with the Teensyduino installer, or one or more of the libraries is not installed (Adafruit_GFX, Adafruit_SSD1351 or Adafruit_ST7735). **Please see the notes at the top of this page regarding IDE & library compatibility.**

The code compiles and uploads but nothing happens!

Check the connections between the display and Teensy board.

- Did you enable the correct #include line for the display type (OLED vs TFT)?
- Are you following the correct order-of-wires for the display type (OLED vs TFT)?
- Are any wires off-by-one on the Teensy?
- Any cold solder joints, or solder bridges between pads?

It kinda works, but the display is glitchy!

- Keep your wires as short and as tidy as possible, check solder connections for good form. High speed SPI is *really* persnickety about connections.
- Is the right CPU speed selected?

Still having trouble? Start a new thread in the Adafruit forums (https://adafru.it/cer) describing the symptoms. It's extremely helpful if you can provide in-focus and well-lit photos that clearly show all the connections between the display and Teensy.

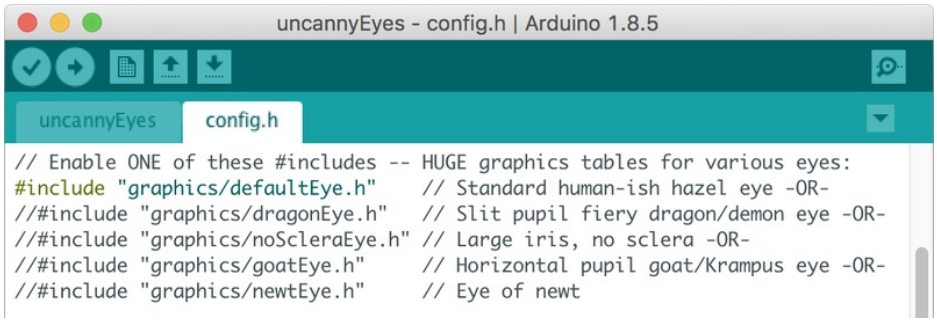
Do not continue until you see an eye. If it's not doing exactly what you want, that's okay, just need an eye to start.

Changing Appearance, Wiring and Inputs

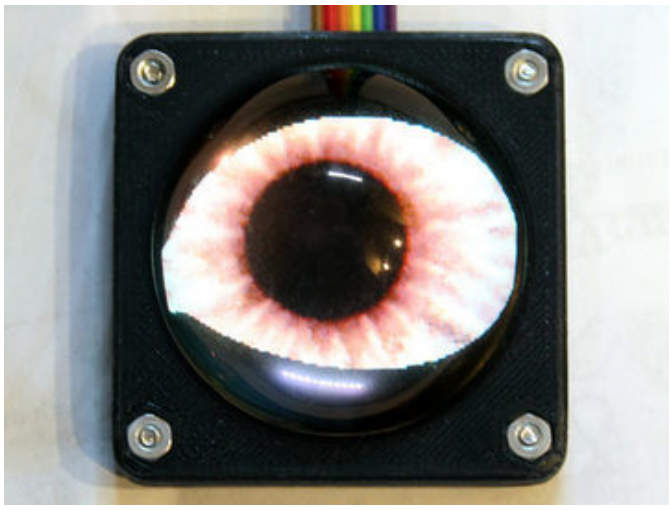
Basic customization — enabling or disabling certain features, or changing pins assigned to functions — is done by editing the file **config.h** (it's the second tab when the code is open in the Arduino IDE).

First section of this file controls the appearance of things.

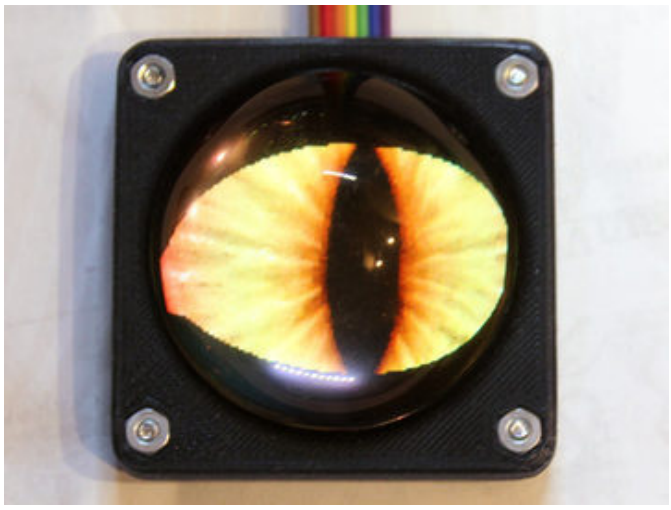
The software includes a few ready-made eyes that can be used by simply enabling the corresponding #include line — remove the comment (//) at the start of that line, and **make sure all the others are commented out** The stock sketch has "defaultEye" enabled:



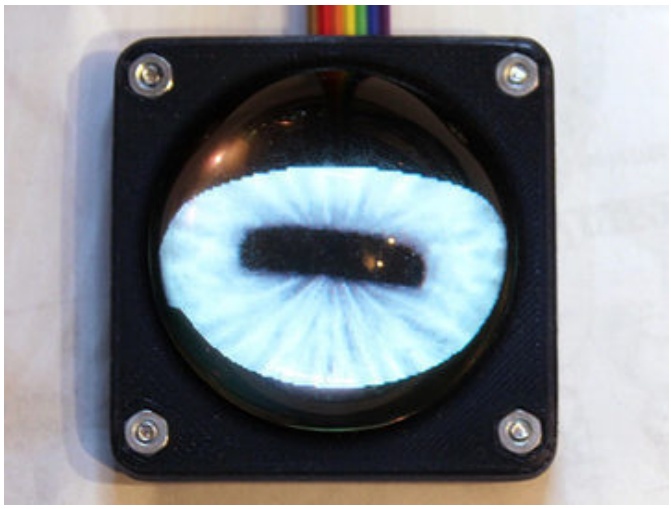
defaultEye.h is human-ish in design. Okay, so the iris is anime-sized, but I'm so proud of that iris-scaling code I *had* to show it off as much as possible.



Some animals have such huge irises you rarely see the sclera (the white part of the eye). **noSclera.h** is an example eye for these situations.



dragonEye.h because *dragons.* It's a *moral imperative.*



Goats (or is it Krampus (https://adafru.it/ixb)?) have the *weirdest* pupils. **goatEye.h** is an attempt at simulating this. I designed this one to *not* move around, just to demonstrate how its done (the sclera image is the same size as the screen...no room to move...normally the sclera image is a bit larger).

Just above the eye selection is this line, disabled by default:

```
//#define SYMMETRICAL_EYELID
```

Normally the software renders the two eyes with distinct left and right shapes (there's a *caruncle* — that corner area near the tear duct). Some projects only use a single eye and this shape just seems odd. Enabling SYMMETRICAL_EYELID uses a simpler "football shape" that's neither left nor right...it's a bit cartoonish but may look better on single-eyed creatures.

A little further down is a list indicating the pins used for each eye. The software can handle any number of eyes (it splits up time between them), but will usually be two (or, if compiling this for the Hallowing (https://adafru.it/CmY) board, there's just one):

```
eyeInfo_t eyeInfo[] = {
#ifdef ADAFRUIT_HALLOWING
  { 39, -1, 2 }, // SINGLE EYE display-select and wink pins, rotate 180
#else
  {  9, 0, 0 }, // LEFT EYE display-select and wink pins, no rotation
  { 10, 2, 0 }, // RIGHT EYE display-select and wink pins, no rotation
#endif
};
```

Each line contains **three items**: the pin that's connected to the corresponding display's OC (OLED chip select) or TCS (TFT chip select) pin, and another pin where a button (connected to ground) makes that eye independently wink. If you don't want or need this wink control, the second pin value can be set to -1. Third item on each line is a **rotation** setting (0–3) for that screen:

- 0 is the default orientation, with the display's connector at the top.
- 1 rotates the graphics 90 degrees clockwise (compensating for a display that's installed 90° counterclockwise, with the connector at the left).
- 2 rotates the display 180° (as on the Hallowing board).
- 3 rotates 90 degrees counterclockwise (compensating for a display with the connector at right).

The stock code, as shown above, has two eyes. To work with just a single eye, comment out or delete one of the two lines. In the Hallowing case (if ADAFRUIT_HALLOWING is defined), this is already done (notice also the rotation value of "2," because the Hallowing screen is oriented upside-down).

As previously explained, the next section decides whether to use the library for OLED or TFT displays, by #including one or the other:

```
#include <Adafruit_SSD1351.h>  // OLED display library -OR-
//#include <Adafruit_ST7735.h> // TFT display library (enable one only)
```

After this, two lines indicate the pins used for the displays' DC (data/command) and RESET signals:

```
#define DISPLAY_DC     7       // Data/command pin for ALL displays
#define DISPLAY_RESET  8       // Reset pin for ALL displays
```

If the project involves multiple eyes, these lines should fan out from the microcontroller to ALL displays; unlike the select pins above, which are unique to each display.

The Hallowing board has a different set of pins already defined, plus some extra items for backlight control.

The last section configures various controls:

```
//#define JOYSTICK_X_PIN A0 // Analog pin for eye horiz pos (else auto)
//#define JOYSTICK_Y_PIN A1 // Analog pin for eye vert position (")
//#define JOYSTICK_X_FLIP   // If defined, reverse stick X axis
//#define JOYSTICK_Y_FLIP   // If defined, reverse stick Y axis
#define TRACKING            // If defined, eyelid tracks pupil
#define BLINK_PIN         1 // Pin for manual blink button (BOTH eyes)
#define AUTOBLINK           // If defined, eyes also blink autonomously
#ifdef ADAFRUIT_HALLOWING
  #define LIGHT_PIN      A1 // Hallowing light sensor pin
  #define LIGHT_CURVE  0.33 // Light sensor adjustment curve
  #define LIGHT_MIN      30 // Minimum useful reading from light sensor
  #define LIGHT_MAX     980 // Maximum useful reading from sensor
#else
  #define LIGHT_PIN      A2 // Photocell or potentiometer (else auto iris)
//#define LIGHT_PIN_FLIP    // If defined, reverse reading from dial/photocell
  #define LIGHT_MIN       0 // Lower reading from sensor
  #define LIGHT_MAX    1023 // Upper reading from sensor
#endif
#define IRIS_SMOOTH         // If enabled, filter input from IRIS_PIN
#if !defined(IRIS_MIN)      // Each eye might have its own MIN/MAX
  #define IRIS_MIN      120 // Iris size (0-1023) in brightest light
#endif
#if !defined(IRIS_MAX)
  #define IRIS_MAX      720 // Iris size (0-1023) in darkest light
#endif
```

JOYSTICK_X_PIN and JOYSTICK_Y_PIN (here set to A0 and A1, respectively) state where the joystick inputs are connected. By default, these lines are commented out — the eye moves autonomously, without user input. If you have a joystick connected, enable these two lines. (X_FLIP and Y_FLIP reverse the input direction if needed)

TRACKING sets whether the upper eyelid follows the pupil (as actual eyes do, it's a neat thing). You can turn this off by commenting out this line.

LIGHT_PIN states where the photocell or dial is connected for adjusting the size of the pupil/iris. LIGHT_PIN_FLIP reverses the direction, like the joystick settings (lower values should be darker). LIGHT_MIN and LIGHT_MAX are the minimum and maximum reliable readings from the sensor (it might occasionally go below or above these, but it's rare, probably noise that can be disregarded), while IRIS_MIN and IRIS_MAX control the dilation of the pupil in response to the light reading (smaller numbers = smaller pupil, range is 0 to 1023).

IRIS_SMOOTH filters the input from LIGHT_PIN so it's not twitchy. This slows the reaction time, but the movement is similar to real eyes, pretty nifty.

BLINK_PIN specifies a pin where a button is connected for blinking both eyes simultaneously (distinct from the individual left/right wink pins previously discussed). If you don't have a button connected for this, the line can be commented out or the value set to -1.

AUTOBLINK (enabled by default) makes the eyes automatically blink randomly every few seconds. You can comment this out to make the eyes only blink with the buttons...or you can use both in combination.

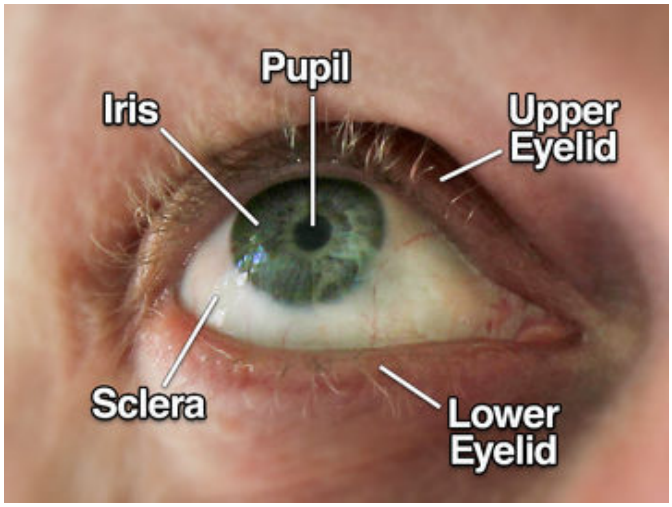
If the example eyes don't deliver quite what you need, it's possible to generate new header files with custom graphics...

Customizing

As we've seen, the Arduino sketch uses massive tables of precomputed *stuff.* These tables are generated with a script written in Python, using image files as inputs.

To generate new tables, you need some familiarity with Python and command-line script usage. Your computer must have Python installed, plus the Python Imaging Library. That's going to vary on different systems and is beyond the scope of this guide, but Googling will turn up some getting-started resources.

So we're singing from the same page, let's lay out some eye terminology...



The "white" of the eye is known as the *sclera.*

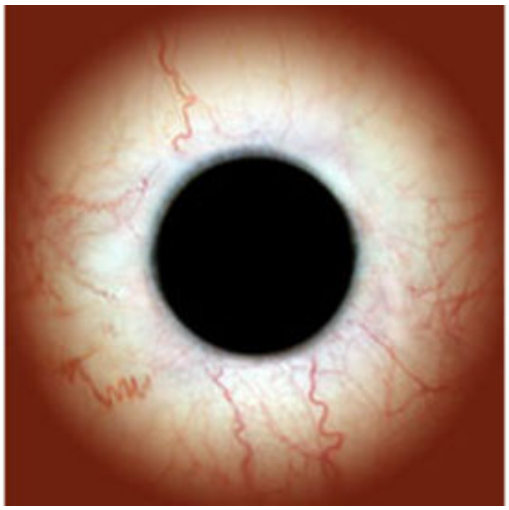
The *iris* is the muscle that contracts to adjust the size of the *pupil* in response to light.

The *upper* and *lower eyelids* are involved in blinking.

Eyes are weirder than you might think. They're *not* simply football-shaped, they've got *two* lids, and the pupil and eyelids *interact.* The Arduino sketch takes a few shortcuts, but makes an effort at simulating these effects.

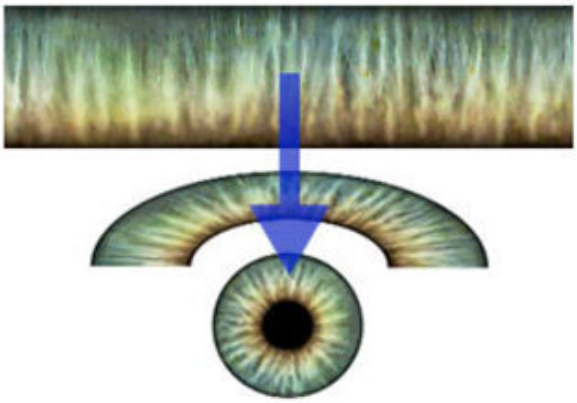
The Python script (located in the "convert" folder from the Github repository) — **tablegen.py** — takes six or seven images (corresponding to eye parts above) as inputs and generates a table for each. There's also an array that it generates on its own — it precomputes and stores a bunch of trigonometry. The output can be redirected as a .h file and used with the Arduino sketch.

Let's look at the defaultEye images (included in the convert/defaultEye directory):



The sclera image is 200x200 pixels, allowing the eye lots of room to pivot (the screens are 128x128 pixels). Aside from icky veins and stuff, this also determines the color of the pupil. The area in the center should match the desired iris size. In the default eye case, that's 80 pixels across.

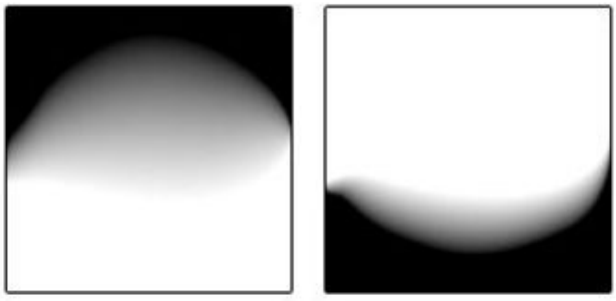
Since the other eye designs have no visible sclera, their sclera images are blank (but still need to be present). Dragon and noSclera are 160x160 pixels (they can pivot a little), while the goat eye is 128x128, same size as the screen, thus it's prevented automatically from moving.



This is interesting...the iris image is “unrolled,” kind of like a map of the earth. The pupil is at the bottom, outer edge of the iris at the top. The Arduino sketch morphs this back into the round shape...this is what makes the cool scaling effect possible when reacting to light.

The optimal iris image width can be computed by multiplying the round iris diameter (in pixels) by *pi* — 3.14 — while the image height is the iris radius in pixels. Doesn’t have to be exact, just “ish.” The default iris is 80 pixels across, so the map image is 256x64 pixels.

Dragon and noSclera irises are 160 pixels in diameter, so their iris map images are 512x80. The goat, 128 pixels across, has a 402x64 map.



Upper and lower lids are grayscale images the same dimensions as the screen (128x128 pixels). For each frame of animation, a *threshold* filter is applied — the eye is rendered only where the eyelid pixel values exceed some limit (which changes to create the blinking effect). Additionally, a point sample taken slightly above the pupil makes the eyelid tracking effect possible.

There are *two* sets of eyelid images — one set is left/right symmetrical, the other is not.

To recreate the defaultEye.h file, you’d go the “convert” directory and enter this command (as a single line):

```
python tablegen.py defaultEye/sclera.png defaultEye/iris.png defaultEye/lid-upper-symmetrical.png
defaultEye/lid-lower-symmetrical.png defaultEye/lid-upper.png defaultEye/lid-lower.png 80 >
defaultEye.h
```

The eye images must be specified in the order: sclera, iris, symmetrical upper lid, symmetrical lower lid, asymmetrical upper lid, asymmetrical lower lid. The iris diameter (80 pixels in this example) is specified next. Then the output is redirected to the file “defaultEye.h” (you’ll want to give your own eyes different names). Check the contents of this file after running the script…any error messages will be at the end.

The resulting file can then be moved to the “graphics” folder within the “uncannyEyes” directory.

Add an #include line for whatever name you’ve assigned it. Compile and upload to the Teensy board and see what you get!

A couple of eye designs (dragon and goat) have strange pupil shapes. In this case, one more argument can be added, the filename of a “pupil map” image that assists this code in generating certain tables. Images for these are located in the corresponding directories. For example:

```
python tablegen.py dragonEye/sclera.png dragonEye/iris.png dragonEye/lid-upper-symmetrical.png
dragonEye/lid-lower-symmetrical.png dragonEye/lid-upper.png dragonEye/lid-lower.png 160
dragonEye/pupilMap.png > dragonEye.h
```

There are size limits to all these images. Sclera and iris images require **two bytes per pixel.** Eyelid images are one **byte per pixel.** Additionally, a lookup table equal to the iris size (80x80 pixels in the defaultEye case) requires **two bytes per element.**

160x160x2 + 256x64x2 + 128x128 + 128x128 + 80x80x2 = 129,536 bytes.

The code requires a little over 50K, so the resulting total compiled sketch size is about 182K, well within the **256K** flash space of the Teensy. The dragon eye, with its large iris, pushes much closer to the limit. Do a little math before investing a lot of time in a new set of images, to make sure it’ll fit in the available space.

Bowler Hat Project

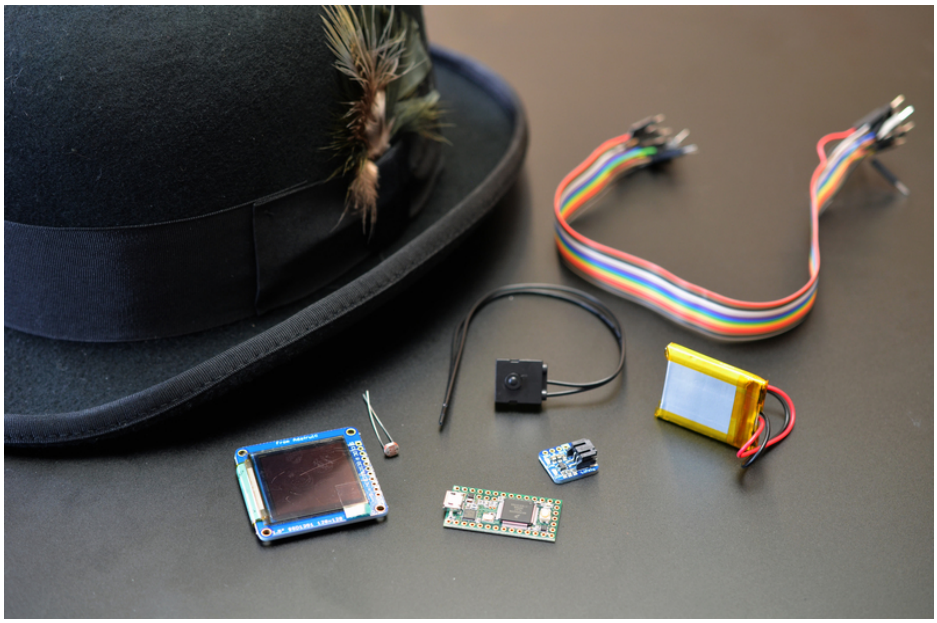
Wear an eyeball on your hat! This is an intermediate-level wearables project built using the one Electronic Animated Eye built using Teensy 3.1/3.2 and an OLED or TFT display. The result is a compact bowler with electronics built into the ribbon band. This hat could be a spooky upgrade to a Clockwork Orange costume and much more!

This iteration does not use the 3D printing files provided earlier in this guide, but rather a more gasket-like shape 3D printed in SemiFlex flexible filament and sewn onto the hat (the cabs I found at the plastics store were 1.25" rather than the recommended 1.5"):

https://adafru.it/iEL

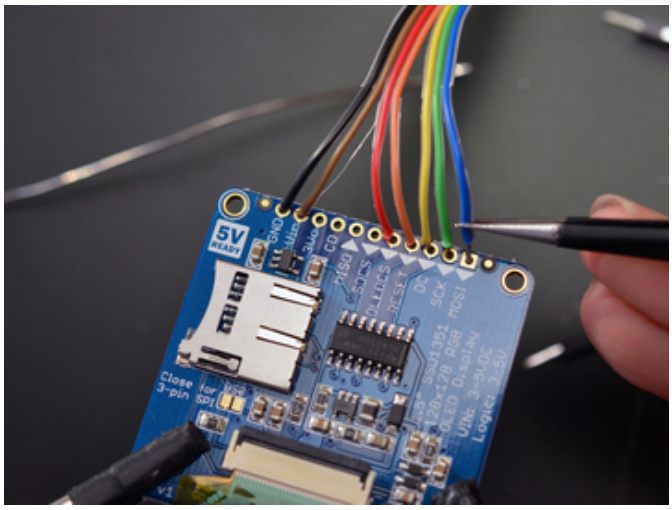
https://adafru.it/iEL

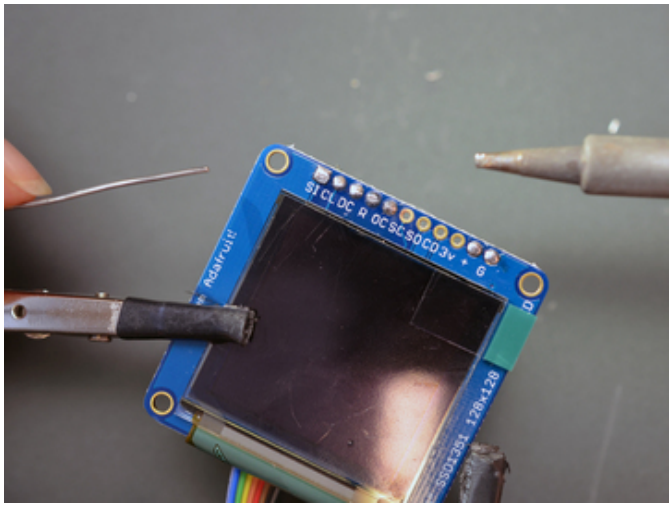
As an alternative to 3D printing, you can cut a piece of rubber or leather to hold the cabochon in place.



In addition to the items listed in the overview, you will need:

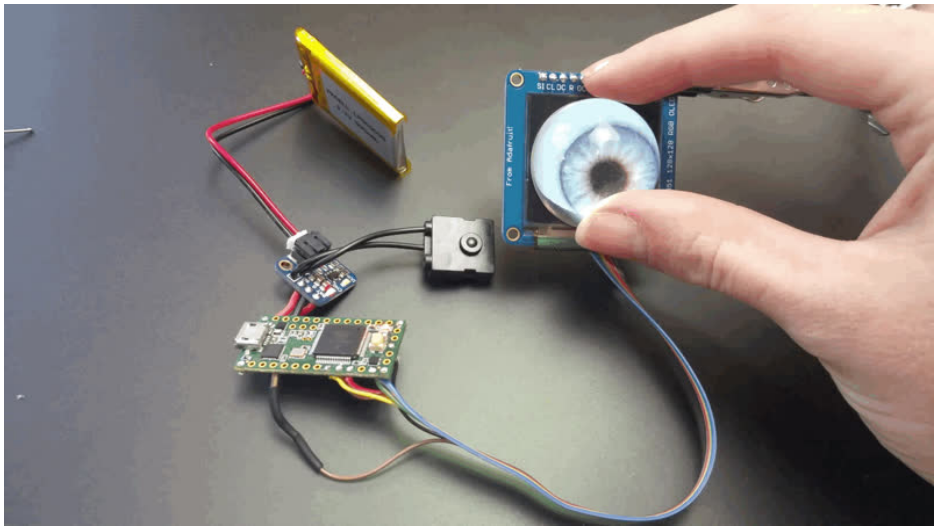
- bowler hat
- scissors
- needles and thread
- tailor's chalk or marking pen





Solder a ribbon cable to your display (OLED shown) as directed on the wiring page, however you can be a bit more relaxed about the wire slack lengths and positioning than the hard plastic 3d enclosure calls for since these wires will be hidden and protected in the band of the hat.

Fold the ribbon cable at a right angle to send it out one side toward the hat's ribbon bow. Cut the ribbon cable to length to reach the approximate center of the bow.



Solder up the connections according to the wiring instructions and test out your circuit. Shown here is the flickering effect you'll get with an OLED **on video only** -- it looks fabulous to the naked eye. The flickering can be mitigated slightly by adjusting your camera shutter speed, but go with a TFT if your primary goal is to capture crystal video!

An optional small change in the code yields better animation. With only a single eye to render, screen updates can occur twice as fast and the animation is super buttery smooth. In the file config.h, look for these two lines of code and delete or comment out one, leaving the other enabled (depending on which 'select' pin you've wired up):

```
{  9, 0, 0 }, // LEFT EYE display-select and wink pins, no rotation
{ 10, 2, 0 }, // RIGHT EYE display-select and wink pins, no rotation
```

Also, with no photocell in the circuit, you can comment out this line a bit further down in the file; the pupil will change size on its own rather than in response to light:

```
#define LIGHT_PIN      A2 // Photocell or potentiometer (else auto iris)
```

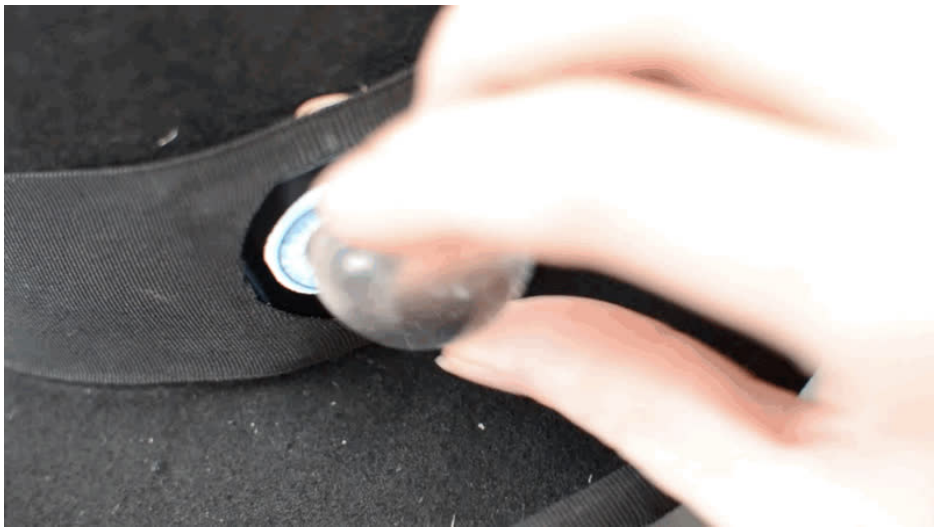
Mark the center front of the hat with tailor's chalk or a pen, then trace your cabochon or gasket to transfer the circle outline to the ribbon.

Carefully cut out the cab-sized circle from the ribbon with sharp scissors.

Slide the display into the ribbon band behind the round cutout, routing the ribbon cable off to the side.

Stow the Teensy 3.1/3.2 and lipoly backpack into the ribbon bow and tuck in the ribbon cable slack behind the band.

The battery can be tucked into the ribbon band near the back of the head, with its wires headed into the bow area for plugging in.



Position the cabochon on the display and sandwich it in place with the 3D printed (or leather or rubber) gasket.

Stitch the gasket's four holes to the hat through the mounting holes of the display (OLED shown). It can be handy to use two needles pierced through the front with pliers, then pull them through to the inside of the hat and tie a knot.

The feather accent serves to obscure the circuit from view-- feel free to glue it in place if you like!

This hat is not weatherproof!If it rains, power down and stow your hat. Use the original 3D enclosure and stitch it on for a more protected display, and consider shrinking large heatshrink tubing over the PCBs. Want to ruggedize this design? There are some tips in this video:

Ready-Made UF2 Files

One of the appealing features of Adafruit's ARM boards is the use of the *UF2 bootloader,* which makes thse boards show up as a small USB flash drive on your computer. Just double-tap the reset button and copy a pre-compiled binary file over to this drive...no need to install or compile anything special with the Arduino IDE! We have a few ready-made eyes for some of the Adafruit boards that have screens attached...

HalloWing M0

Several ready-made designs for **HalloWing M0** are available as UF2 files **in the main HalloWing guide (https://adafru.it/CEs)**.

Circuit Playground Express with TFT Gizmo

These are compiled for **Circuit Playground Express** with the **TFT Gizmo**; they **will not work on other boards**. The light sensor is *not* used here, since it's on the opposite side from the display...instead the pupil dilates on its own. This can all be changed if you compile from the source code.

https://adafru.it/H0a

https://adafru.it/H0a

https://adafru.it/H0b

https://adafru.it/H0b

https://adafru.it/H0c

https://adafru.it/H0c

https://adafru.it/H0d

https://adafru.it/H0d

https://adafru.it/H0e

https://adafru.it/H0e

Circuit Playground Bluefruit with TFT Gizmo

These are similar to the above, but specifically for the **Circuit Playground Bluefruit** board with **TFT Gizmo**.

Differences are that the splash screen is disabled, and the light sensor is enabled, both by necessity...the code doesn't work otherwise. This is a new board and we're still sorting out some of the more obscure details. These will be updated as necessary.

https://adafru.it/H0f

https://adafru.it/H0f

https://adafru.it/H0A

https://adafru.it/H0A

https://adafru.it/H0B

https://adafru.it/H0B

https://adafru.it/H0C

https://adafru.it/H0C

https://adafru.it/H0D

https://adafru.it/H0D