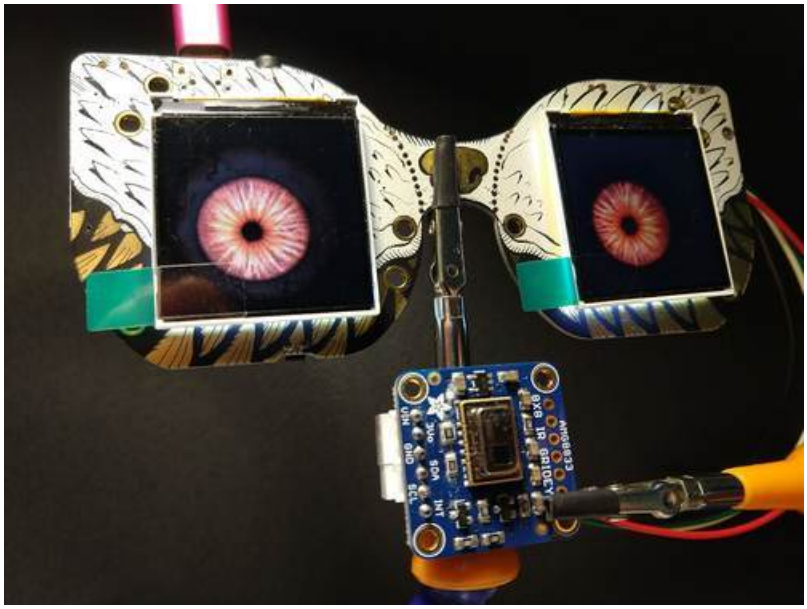
adafruit learning system

Monster M4sk Is Watching You

Created by Timothy Weber



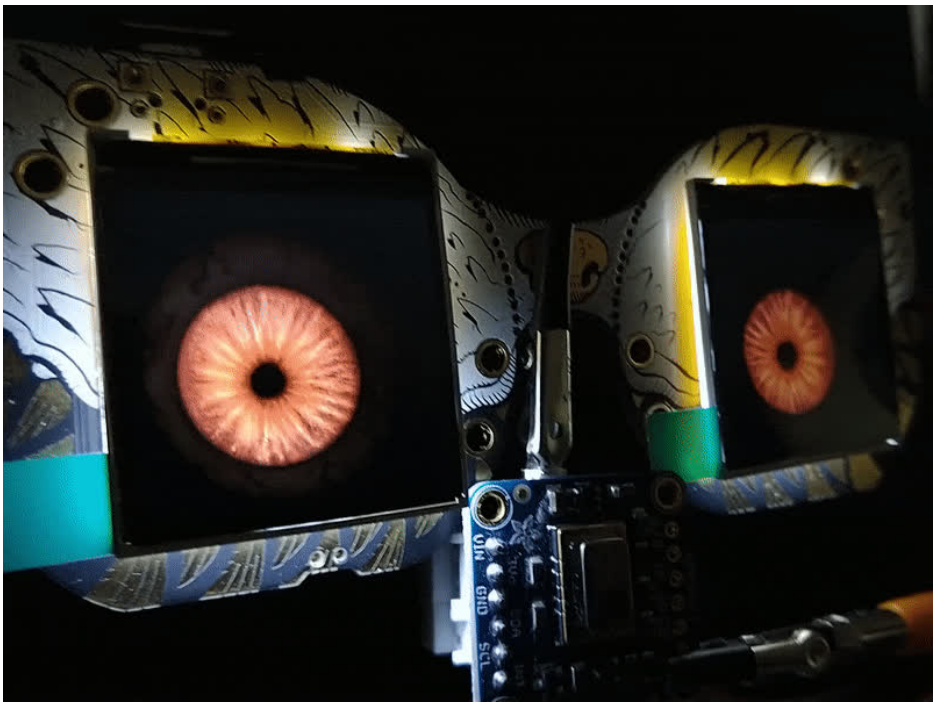
Last updated on 2020-05-19 12:26:02 AM EDT

Overview

I was excited when the Adafruit MONSTER M4SK was announced, because for years I've wanted to make an interactive skull as a Halloween prop - and now there's a project board custom-made to support that!

There are lots of cheesy Halloween decorations out there. That's fine. But I wanted to build something with a bit of subtlety to it, that might attract random Halloween visitors into a memorably creepy interaction rather than just make noise or flash.

This guide is the beginnings of the guts of that project. It modifies the MONSTER M4SK so that its eyes, instead of looking around randomly, **track a person in front of them,** following their face as it moves.



Disclaimers

This design is cheap and cheerful, using an infrared (IR) grid sensor and a simple heuristic to figure out where the human is. There are other approaches using more expensive sensors and processors that might work better.

In particular, this is **designed to work outdoors on a cool night.** That means the infrared noise floor should be low, and "hot spots" should correspond with uncovered parts of humans, like faces. If you take it into a warm environment, with lots of people around, it may glance around rapidly instead of following the nearest face, and if you put it in front of a bonfire, it may not even notice the humans at all (to be fair, that's how I often react in both of those situations too).

Also, you'll need **some way to mount the parts** for display - in a creature, mask, pumpkin, skull, scarecrow, or whatever. I plan to share my skull model for 3D printing when it's done, but it's not done yet.

Parts List

The parts needed for this project:

Your browser does not support the video tag. Adafruit MONSTER M4SK - DIY Electronic Eyes Mask

OUT OF STOCK

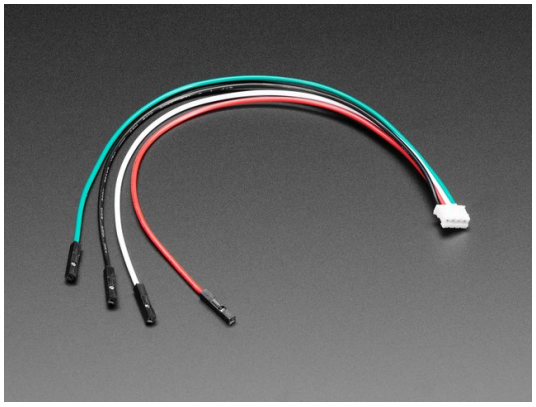
Out Of Stock

Your browser does not support the video tag.

Adafruit AMG8833 IR Thermal Camera Breakout

OUT OF STOCK

Out Of Stock



JST PH 4-Pin to Female Socket Cable - I2C STEMMA Cable - 200mm

$1.50

IN STOCK

Add To Cart

Optional Parts

You'll also need something to mount the components in, and a way to power them. Some quick options:

Lithium Ion Battery - 3.7v 2000mAh

$12.50
IN STOCK

Add To Cart

Hobby Creek Mag Hand Professional Workstation

$64.95
IN STOCK

Add To Cart

Prepare and connect the parts

MONSTER M4SK

Follow the MONSTER M4SK Guide (https://adafru.it/FYV) to ensure your M4SK is working properly, and get the eyes configured the way you want them.

Change of Eyes

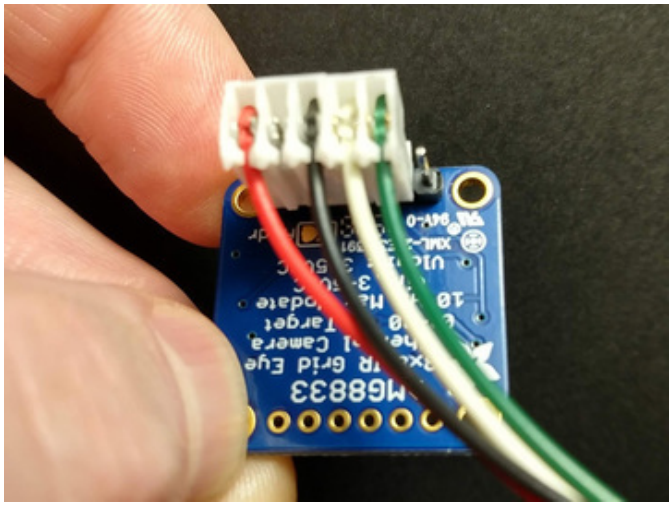
If you'd like to duplicate the glowing red irises from the photos here, choose the "skull" set from the "Ready-Made Graphics" section of that guide.

Stock STEMMA cable to camera board

There's nothing to prep on the stock STEMMA cable.

Follow the AMG8833 Guide (https://adafru.it/G8b) to add a header strip to the camera breakout board.

You only need to solder the headers on the bottom row (**VIN** through **INT**).



Then plug the wires into the camera board as follows:

1. Red to **VIN**
2. **3Vo**, no wire connected
3. Black to **GND**
4. White to **SDA**
5. Green to **SDC**

(The photo shows different connectors - but the colors and placement are correct.)

That's it - you can go to the next step.

Making your own cable to connect to the camera board

If the STEMMA cable is out of stock, you can make your own cable.

To make a cable, you'll need these parts for the M4SK end:

1 x 4 Position Rectangular Housing Connector Receptacle (Digi-Key)
The female socket for a 4-pin JST-PH (aka STEMMA) plug

BUY NOW

4 x Socket Contact Tin 24-28 AWG Crimp (Digi-Key)
Crimp pins to go around the wires and insert into the plug (buy more, you'll mangle at least one)

BUY NOW

1 x Crimping pliers

The easiest tool for crimping many kinds of connectors to wires

Add To Cart

1 x Hook-up Wire Spool Set - 22AWG Stranded-Core - 6 x 25ft

Basic hookup wire

Add To Cart

Crimp the pins onto the wires, and insert the wires into the housing. Be sure to attach the wires in this order at the M4SK end (left-to-right as you look into the end of the plug, with the polarizing fins up):

1. Black, **GND**
2. Red, **V+**
3. White, **SDA**
4. Green, **SDC**

For the other end of the cable, that goes to the camera breakout board, you have options:

- You can solder a polarized header to the camera board, and attach a matching plug to the cable wires. This is the most reliable option.
- For the simplest and quickest build, solder the wires directly to the camera board.

No matter how you make the connection, the wire order on the camera board side is:

1. Red to **VIN**
2. **3Vo**, no wire connected
3. Black to **GND**
4. White to **SDA**
5. Green to **SDC**

These Molex parts will give you a securely polarized connection:

5 x Non-Gendered Contact Tin 22-30 AWG Crimp (Digi-Key)

Standard Molex KK crimp pins for 22 AWG wire

BUY NOW

1 x 5 Position Rectangular Housing Connector Receptacle White 0.100" (Digi-Key)

Polarized 5-pin female Molex KK housing

BUY NOW

1 x Connector Header Through Hole 5 position 0.100" (Digi-Key)

Polarized 5-pin maile Molex KK header

BUY NOW

Replace the M4SK firmware

Next, you'll need to upload the new firmware that reads the camera and moves the eyes accordingly.

Upload from a firmware image

The simplest way is to use the following firmware image:

https://adafru.it/G8C

https://adafru.it/G8C

Then upload it just as you may have done for the original M4SK firmware: (https://adafru.it/G8d)

1. Connect a USB cable and put the power switch in the “on” position
2. Double-tap the reset button
3. Wait for the MASKM4BOOT drive to appear on your host computer
4. Drag the M4WATCH.UF2 file to the MASKM4BOOT drive and wait for it to copy over
5. The board will automatically reboot.

(Note: this will not disturb your current eye configuration.)

If all goes well, you should be able to see the eyes moving to follow the warmest thing in front of them.

Advanced - compiling the source code

You can also compile the source code from the Arduino IDE. Building from source is more advanced than the rest of this guide, so consider using the UF2 file above unless you are very comfortable with the Arduino IDE and changing code.

Follow **all** the instructions in the M4SK Guide on "Building the eyes from source code." (https://adafru.it/G8e)

You may also need to install more libraries than are listed in the the M4SK Guide. If you see vague error messages when you build with Arduino, make sure you have all of these installed:

- **Adafruit AMG88xx**
- **Adafruit Arcada**
- **Adafruit Circuit Playground**
- **Adafruit GFX**
- **Adafruit BusIO**
- **Adafruit ImageReader**
- **Adafruit seesaw**
- **Adafruit SPIFlash**
- **Adafruit ST7735 and ST7789**
- **Adafruit TinyUSB**
- **Adafruit TouchScreen**
- **Adafruit Unified Sensor**
- **Adafruit Zero DMA**
- **Adafruit ZeroTimer**
- **ArduinoJSON**
- **SdFat - Adafruit Fork**

The M4_Eyes repository on GitHub is here (https://adafru.it/FAD) for reference.

Once you've got the base firmware compiling, **enable the "user_watch" module** as follows:

1. In `user.cpp`, change the first line from `#if 1` to `#if 0`.
2. In `user_watch.cpp`, do the opposite: change the first line from `#if 0` to `#if 1`.

This ensures that exactly one version of the `user_setup()` and `user_loop()` functions is defined.

Compile and upload that code from the Arduino IDE.

Watch the Watcher

You should now be able to see the eyes follow you.

Some tips for a good demo:

- Hold the two boards fixed in relationship to one another. If you don't have a set of Helping Hands as I've pictured here, a quick prototype option could be to tape the two boards to a piece of cardboard temporarily.
- If the eyes don't follow you when you move your head from side to side, try turning the boards instead. Something warm behind you might be distracting the sensor.

For some debugging information, view the output on a connected computer using a serial monitor (**Tools** > **Serial Monitor** in the Arduino IDE). If it's working, you'll see repeating output that looks like this:

```
 26
-0.18 -0.29 3.75
-0.19 -0.27 4.25
-0.16 -0.24 4.00
-0.19 -0.27 4.25
-0.18 -0.30 4.00
-0.21 -0.30 3.75
```

The first line is the current refresh rate, in frames per second.

The following lines are the **X and Y coordinates** of the focus point found from the sensor data, followed by an estimate of the overall **magnitude**, which will increase as a face gets nearer to the sensor. The coordinates are scaled from -1.0 to +1.0, and the magnitude is scaled from 0 to 50.

How it works

The main work of the code is to figure out where to look. I use a simple average of the pixel positions, weighted by temperature:

```
x = 0, y = 0, magnitude = 0;
float minVal = 100, maxVal = 0;
int i = 0;
for (float yPos = 3.5; yPos > -4; yPos -= 1.0) {
  for (float xPos = 3.5; xPos > -4; xPos -= 1.0) {
    float p = pixels[i];
    x += xPos * p;
    y += yPos * p;
    minVal = min(minVal, p);
    maxVal = max(maxVal, p);
    i++;
  }
}
```

I then scale the output to the desired ranges:

```
x = - x / AMG88xx_PIXEL_ARRAY_SIZE / 5.0;
y = y / AMG88xx_PIXEL_ARRAY_SIZE / 5.0;
x = max(-1.0, min(1.0, x));
y = max(-1.0, min(1.0, y));
magnitude = max(0, min(50, maxVal - 20));
```

The rest is glue code to integrate with the rest of the M4_Eyes framework. There's plenty to improve, but this works surprisingly well.

Further updates

I'm working on building this into a 3D-printed skull, and I'll update this as that progresses. For more detailed progress notes, see my project blog on Factr (https://adafru.it/G8B).