adafruit learning system

# STEAM-Punk Goggles

Created by Bill Earl

Last updated on 2018-08-22 03:37:44 PM UTC

## Guide Contents

## Overview and Materials

**Science + Technology + Engineering + Art + Math = STEAM** (https://adafru.it/cMl)

Everyone loves funky goggles and the Adafruit Neopixel rings are perfect for building a flashy pair. To kick it up a notch, we STEAMed up these goggles with some high tech sensors and a bit of applied math and physics.

The goggles are controlled by a Flora microcontroller with a LSM303 accelerometer/magnetometer to track the motion of the wearer's head. A simple physics engine implements virtual pendulum display on the LED rings that swings in response to the motion of the wearer. The effect is much like a pair of hyperactive electronic googly eyes.

Everywhere we go with these, people ask us where they can get a pair!

## Materials:

- 1 pair of Goggles - Any pair of goggles with 50mm lenses will be a perfect fit for the neopixel rings. The prototype for this project was built with these German-made safety goggles (https://adafru.it/cMJ) - using the optional tinted lenses.
- 2 Adafruit Neopixel Rings (http://adafru.it/1463)
- 1 Adafruit Flora (http://adafru.it/659) (the code will NOT fit onto a Trinket or Gemma)
- 1 Adafruit Flora LSM303 Magnetometer/Accelerometer (http://adafru.it/1247)
- 1 3xAAA Battery Pack (http://adafru.it/727)
- Scrap of Leather or Upholstry Vinyl for mounting electronics to the temple.

### Optional:

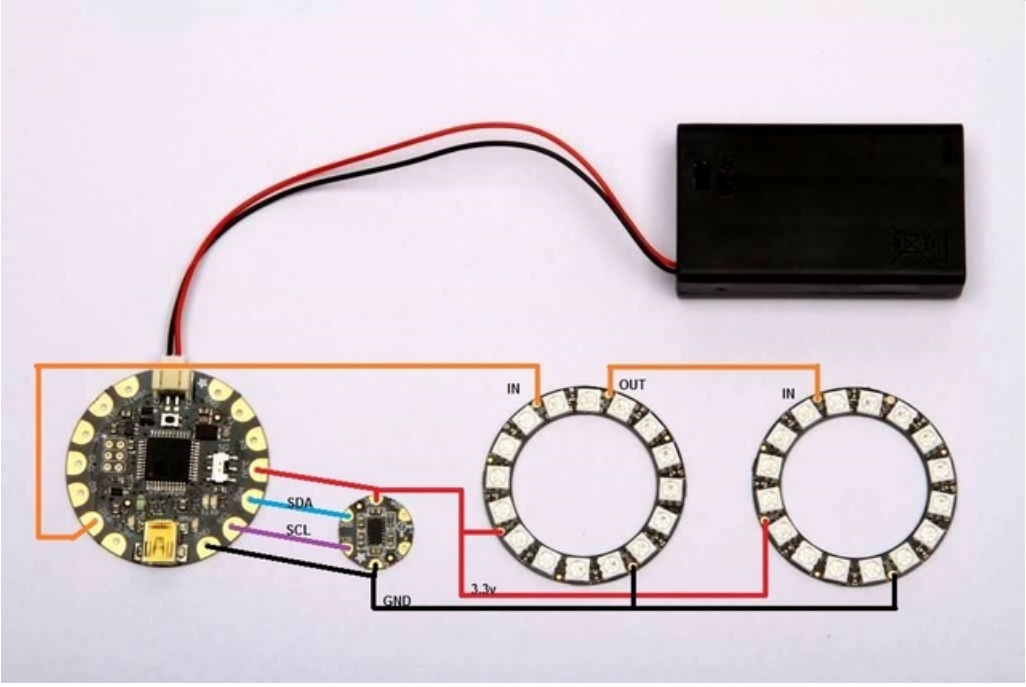
- 1 53mm Watchmaker's Case (https://adafru.it/cMK) to house the Flora & Sensor

### Tools:and Supplies:

- 22 AWG **stranded** hookup wire (solid core wire is not suitable for this project)
- Servo extension cable (http://adafru.it/972) (optional)

- Soldering Iron & soldering supplies (https://adafru.it/aTk)
- Needle & Thread
- Scissors
- Hot glue gun
- Double sided foam tape or GOOP adhesive

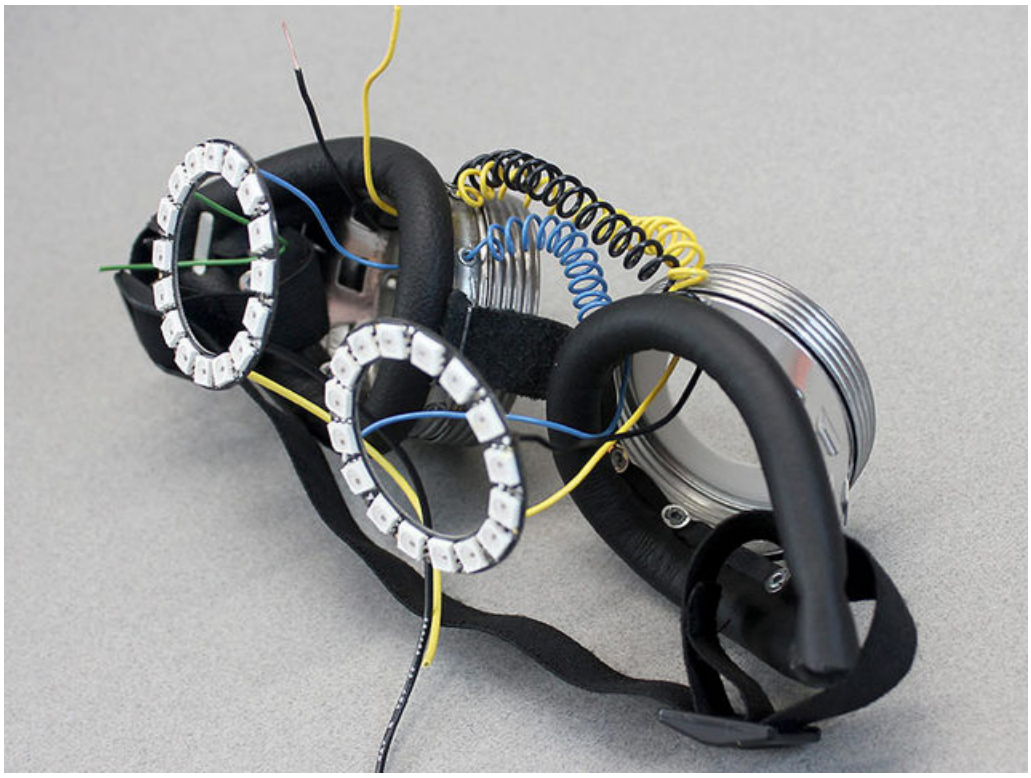
## Assembly and Wiring




The diagram above shows all the connections in the completed goggles. But don't solder it all together just yet! The circuit needs to be built into the goggles, take it step-by-step.

The neopixel wiring is a little tricky since it is routed through the goggle frames. Other than the mounting of the processor and sensor, it the same as for the Kaliedoscope Eyes project (https://adafru.it/cHL). See Phil's excellent wiring instructions here (https://adafru.it/cML),

When installing the rings, leave about 6" of wire attached to 3.3v, GND and D10 to route the right-side of the goggle frame. For the prototype, we used a 3-wire servo-extension cable with the connectors cut off. But 22 awg stranded hookup wire will work as well.



## Mounting the Processor

The code for this project will not fit on a tiny processor like the Trinket, so we use a Flora which we have to mount on the outside of the goggles. But hey! We're geeks and we're proud!

There are many ways to approach the problem. Here's your chance to get creative. We'll show you how we built the prototype below. Feel free to add your own personal touch.

Measure a piece of leather or upholstry vinyl to wrap about 1/4 the way around the outside of one of the lenses. Then cut a roughly triangular piece large enough to mount the Flora and the battery case. (Some overhang on the battery case is OK).

Sew the corners of the short side to the padding on the rgoggles. (If the goggles you are using are not symmetrical, make sure that you sew it to the right-side of the goggles. Sew the back corner of the leather to the strap.

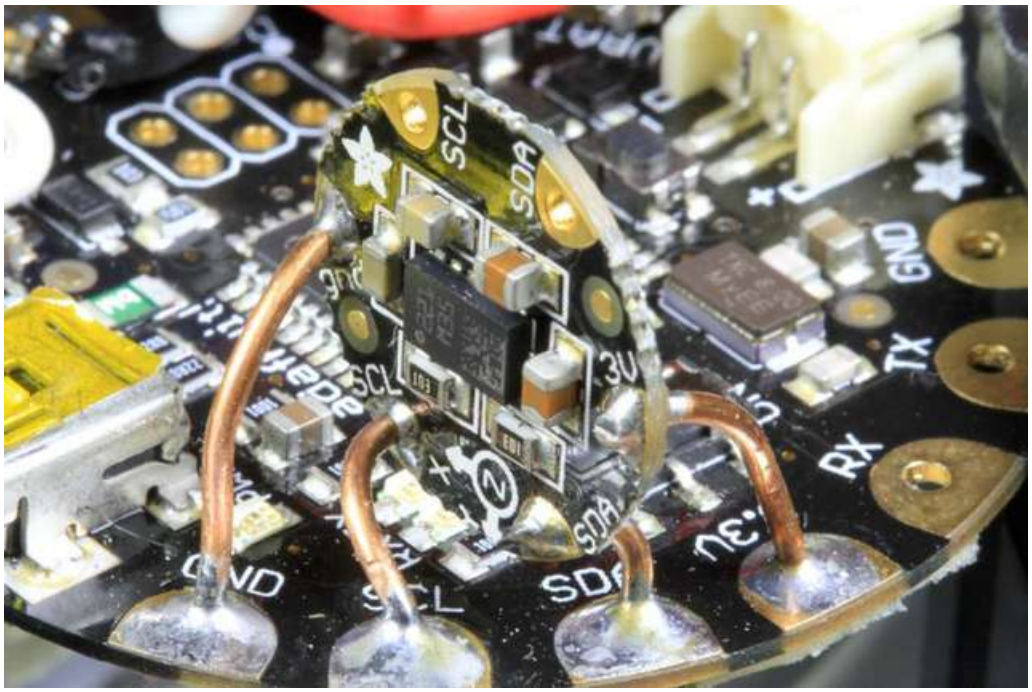
## Attaching the sensor

For the prototype we attached the LSM303 Accelerometer/Magnetometer directly to the Flora as shown below to allow for fine tuning of its position (and because we thought it looked cool). You can mount it anyway you like - but you

will need to modify the code to match the orientation of the axis.

When attached to the goggles, the sensor is near the front edge of the Flora. The code assumes that the X axis is pointing straight back and the Z axis is pointing straight down. There are comments in the code indicating what needs to be changed.

In any case for best results, make sure that you mount the sensor with one axis vertical and one pointing front-to-back when you are wearing the goggles. (The axis are labeled on the sensor silkscreen)




## Attaching the Processor and Battery Pack

You can attach the processor and battery pack directly to the leather temple piece using double-sided foam tape or an adhesive like GOOP.

If you mounted your sensor as we did, make sure that the sensor is at the front edge of the Flora and the USB port is facing down.

**Optional step:**
To add some protection for the electronics, we mounted the Flora and sensor inside a watchmaker's case with holes for the wiring and a cutout for the USB programming cable. Our watchmaker's case is riveted to the temple-piece with aluminum pop-rivets hammered flat on the inside of the case.

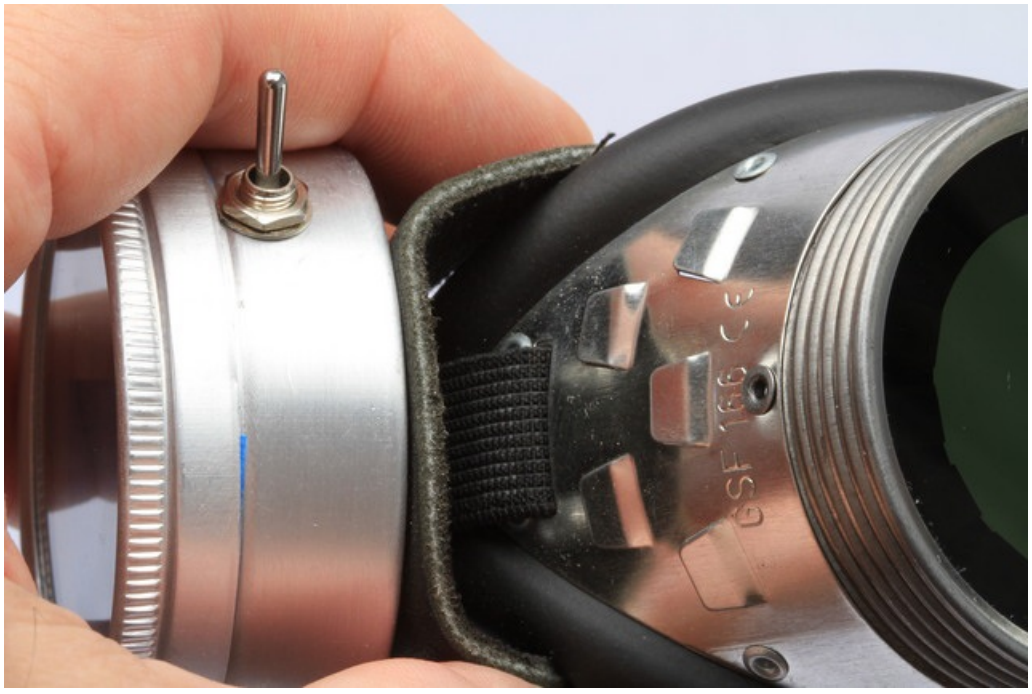
Cut a slit in the leather and route the wires from the led rings. We used a servo extension cable with the connectors cut off to make the wiring a little neater, but regular stranded hookup wire will work.

- Connect the power ground and signal wires to VBATT, GND and D10 on the Flora.
- Plug the JST connector from the battery pack into the battery socket on the Flora.
- Make sure that the Flora battery switch is ON!

*(Note that we added an external battery switch to the watchmaker's case, but you can just use the switch on the battery pack.)*

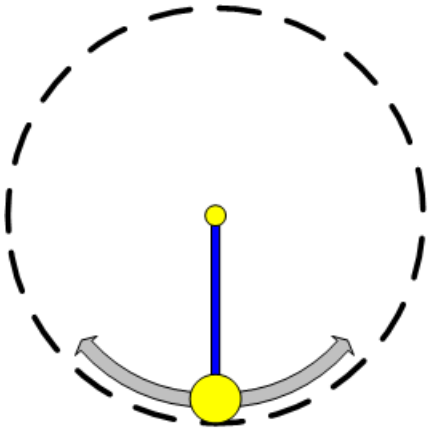
## Final Assembly:

Secure the led rings and internal wiring inside the goggles with hot-glue. For our prototype we used the tinted lenses. For other lens treatment options, see this page (https://adafru.it/cMM).



## Code Design

## The Physics:



The physics engine used in these glasses simulates a pendulum. The horizontal and vertical axis forces measured by the LSM303 accelerometer create torque on the virtual pendulum to make it swing. Variables such as friction and gravity can be tweaked to change the responsiveness of the pendulum. The sample code below uses minimal friction and reduced gravity to make the display more lively.

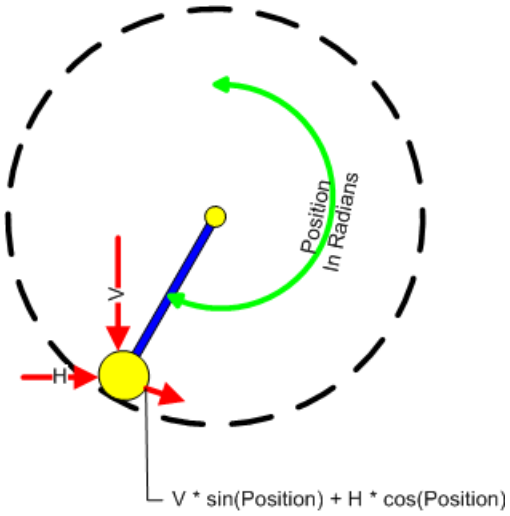
We use the measured acceleration forces to calculate the changes to the momentum of the pendulum. Since the pendulum operates in 2 dimensions, we only look at the horizontal and vertical axis. The 2-dimesional acceleration force vector is converted into a torque based on the rotational position of the pendulum.

### And some Math:

OK, now it is time for a little trigonometry:

Since the motion of the pendulum is constrained to circular motion, we need to do a little math with the horizontal and vertical accelerations to calculate the effective torque on the pendulum.

Here it goes now, the circular motion... (https://adafru.it/cMN)

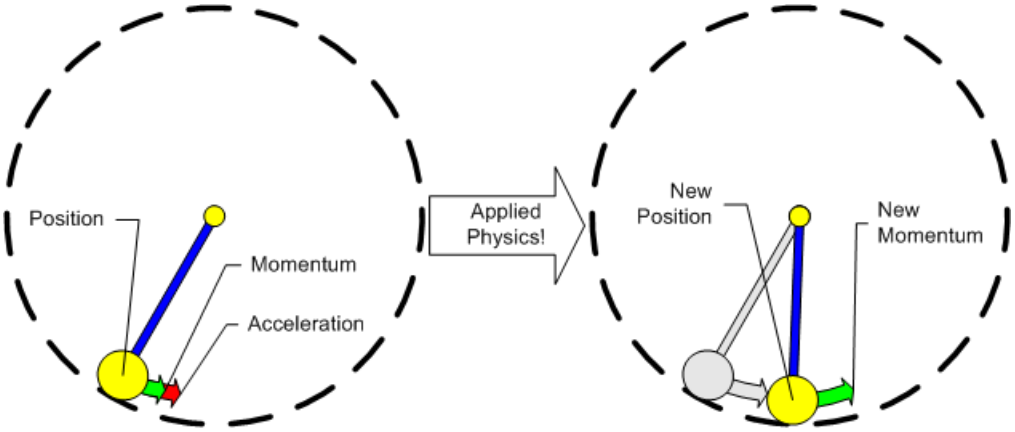


The torque due to horizontal acceleration is proportional to the cosine of the pendulum position in radians.

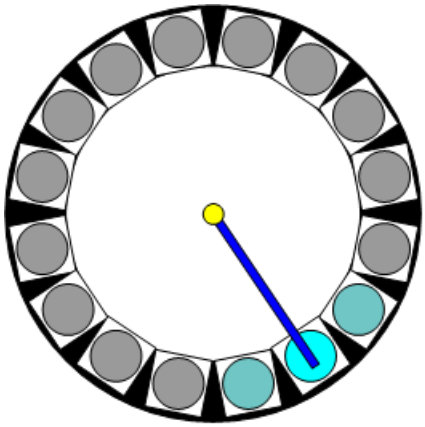
The torque due to vertical acceleration is proportional to the sine of the pendulum position in radians.

The sum of the two values is the total torque due to acceleration.

After accounting for a little friction, we add the calculated torque to the momentum of the pendulum and calculate a new position & momentum.



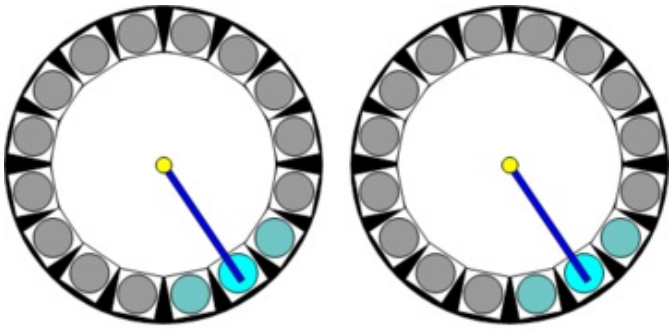

## The Display:



The pendulum is represented on the Neopixel ring by a cluster of 3-4 pixels with the pixel intensity adjusted proportional to the proximity to the center of the virtual pendulum. This results in a smooth transition and avoids the jumpy appearance of a single pixel. To add some variety, the pixel color changes according to the compass heading calculated from the magnetometer readings of the LSM303.

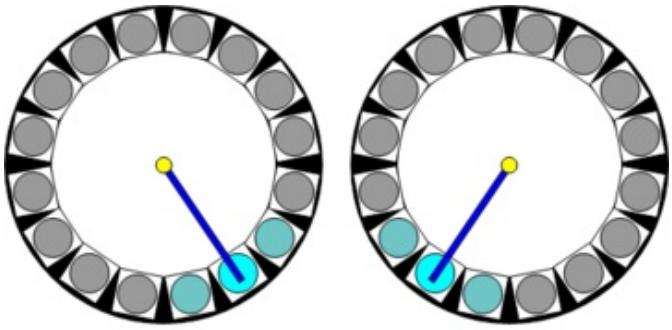
### Alternate Modes of Operation:

The display can switch between normal and “anti-gravity” modes. And between synchronized and mirrored movement of the two eyes. Spin-up and Spin-down effects add visual interest and signal the changes between operating modes.



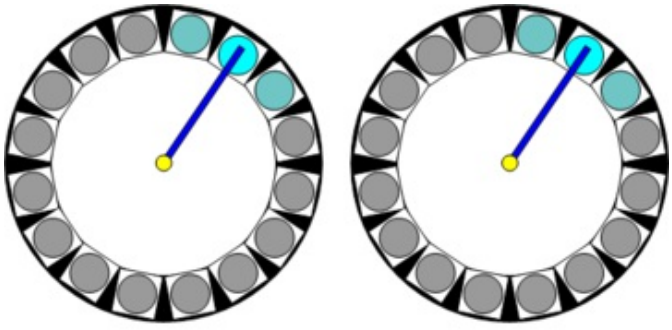
## Normal Mode

In Normal Mode, the pendulums swing in synchronization with each other.



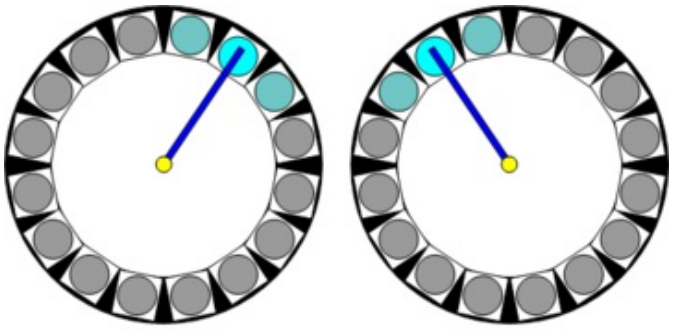
## Mirrored Mode

In Mirrored Mode, the pendulums swing in opposite directions.



## Anti-Gravity Mode

In Anti-Gravity Mode, the pendulums are inverted.



## Mirrored Anti-Gravity Mode

This one speaks for itself.

## Code Implementation

To compile this code you will need the following libraries.

https://adafru.it/cMO

https://adafru.it/cMO

https://adafru.it/cMP

https://adafru.it/cMP

https://adafru.it/cDj

https://adafru.it/cDj

This guide will show you how to install the libraries:

https://adafru.it/c9X

https://adafru.it/c9X

```
// Googly Eye Goggles
// By Bill Earl
// For Adafruit Industries
//
// The googly eye effect is based on a physical model of a pendulum.
// The pendulum motion is driven by accelerations in 2 axis.
// Eye color varies with orientation of the magnetometer

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>
#include <Adafruit_NeoPixel.h>

#define neoPixelPin 10

// We could do this as 2 16-pixel rings wired in parallel.
// But keeping them separate lets us do the right and left
// eyes separately if we want.
Adafruit_NeoPixel strip = Adafruit_NeoPixel(32, neoPixelPin, NEO_GRB + NEO_KHZ800);

Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(54321);
Adafruit_LSM303_Mag_Unified mag = Adafruit_LSM303_Mag_Unified(12345);

float pos = 8;  // Starting center position of pupil
float increment = 2 * 3.14159 / 16; // distance between pixels in radians
float MomentumH = 0; // horizontal component of pupil rotational inertia
float MomentumV = 0; // vertical component of pupil rotational inertia

// Tuning constants. (a.k.a. "Fudge Factors)
// These can be tweaked to adjust the liveliness and sensitivity of the eyes.
const float friction = 0.995; // frictional damping constant.  1.0 is no friction.
const float swing = 60;  // arbitrary divisor for gravitational force
const float gravity = 200;  // arbitrary divisor for lateral acceleration
const float nod = 7.5; // accelerometer threshold for toggling modes
```

```
long nodStart = 0;
long nodTime = 2000;

bool antiGravity = false;  // The pendulum will anti-gravitate to the top.
bool mirroredEyes = false; // The left eye will mirror the right.

const float halfWidth = 1.25; // half-width of pupil (in pixels)

// Pi for calculations - not the raspberry type
const float Pi = 3.14159;

void setup(void)
{
   strip.begin();
   strip.show(); // Initialize all pixels to 'off'  sensor_t sensor;

   // Initialize the sensors
   accel.begin();
   mag.begin();

   resetModes();
}

// main processing loop
void loop(void)
{
   // Read the magnetometer and determine the compass heading:
   sensors_event_t event;
   mag.getEvent(&event);

   // Calculate the angle of the vector y,x from magnetic North
   float heading = (atan2(event.magnetic.y,event.magnetic.x) * 180) / Pi;

   // Normalize to 0-360 for a compass heading
   if (heading < 0)
   {
      heading = 360 + heading;
   }

   // Now read the accelerometer to control the motion.
   accel.getEvent(&event);

   // Check for mode change commands
   CheckForNods(event);

   // apply a little frictional damping to keep things in control and prevent perpetual motion
   MomentumH *= friction;
   MomentumV *= friction;

   // Calculate the horizontal and vertical effect on the virtual pendulum
   // 'pos' is a pixel address, so we multiply by 'increment' to get radians.
   float TorqueH = cos(pos * increment);  // peaks at top and bottom of the swing
   float TorqueV = sin(pos * increment);    // peaks when the pendulum is horizontal

   // Add the incremental acceleration to the existing momentum
   // This code assumes that the accelerometer is mounted upside-down, level
   // and with the X-axis pointed forward.  So the Y axis reads the horizontal
   // acceleration and the inverse of the Z axis is gravity.
   // For other orientations of the sensor, just change the axis to match
```

```
// For other orientations of the sensor, just change the axis to match!
MomentumH += TorqueH * event.acceleration.y / swing;
if (antiGravity)
{
  MomentumV += TorqueV * event.acceleration.z / gravity;
}
else
{
  MomentumV -= TorqueV * event.acceleration.z / gravity;
}

// Calculate the new position
pos += MomentumH + MomentumV;

// handle the wrap-arounds at the top
while (round(pos) < 0) pos += 16.0;
while (round(pos) > 15) pos -= 16.0;

// Now re-compute the display
for (int i = 0; i < 16; i++)
{
   // Compute the distance bewteen the pixel and the center
   // point of the virtual pendulum.
   float diff = i - pos;

   // Light up nearby pixels proportional to their proximity to 'pos'
   if (fabs(diff) <= halfWidth)
   {
      uint32_t color;
      float proximity = halfWidth - fabs(diff) * 200;

      // pick a color based on heading & proximity to 'pos'
      color = selectColor(heading, proximity);

      // do both eyes
      strip.setPixelColor(i, color);
      if (mirroredEyes)
      {
        strip.setPixelColor(31 - i, color);
      }
      else
      {
        strip.setPixelColor(i + 16, color);
      }
   }
   else // all others are off
   {
      strip.setPixelColor(i, 0);
      if (mirroredEyes)
      {
        strip.setPixelColor(31 - i, 0);
      }
      else
      {
        strip.setPixelColor(i + 16, 0);
      }
   }
}
// Now show it!
strip.show();
```

```
}

// choose a color based on the compass heading and proximity to "pos".
uint32_t selectColor(float heading, float proximity)
{
     uint32_t color;

     // Choose eye color based on the compass heading
     if (heading < 60)
     {
        color = strip.Color(0, 0, proximity);
     }
     else if (heading < 120)
     {
        color = strip.Color(0, proximity, proximity);
     }
     else if (heading < 180)
     {
        color = strip.Color(0, proximity, 0);
     }
     else if (heading < 240)
     {
        color = strip.Color(proximity, proximity, 0);
     }
     else if (heading < 300)
     {
        color = strip.Color(proximity, 0, 0);
     }
     else // 300-360
     {
        color = strip.Color(proximity, 0, proximity);
     }
}

// monitor orientation for mode-change 'gestures'
void CheckForNods(sensors_event_t event)
{
   if (event.acceleration.x > nod)
   {
     if (millis() - nodStart > nodTime)
     {
       antiGravity = false;
       nodStart = millis(); // reset timer
       spinDown();
     }
   }
   else if (event.acceleration.x < -(nod + 1))
   {
     if (millis() - nodStart > nodTime)
     {
       antiGravity = true;
       spinUp();
       nodStart = millis(); // reset timer
     }
   }
   else if (event.acceleration.y > nod)
   {
     if (millis() - nodStart > nodTime)
     {
       mirroredEyes = false;
```

```
      mirroredEyes = false;
      spinDown();
      nodStart = millis(); // reset timer
    }
  }
  else if (event.acceleration.y < -nod)
  {
    if (millis() - nodStart > nodTime)
    {
      mirroredEyes = true;
      spinUp();
      nodStart = millis(); // reset timer
     }
  }
  else // no nods in progress
  {
    nodStart = millis(); // reset timer
  }
}

// Reset to default
void resetModes()
{
  antiGravity = false;
  mirroredEyes = false;

  /// spin-up
  spin(strip.Color(255,0,0), 1, 500);
  spin(strip.Color(0,255,0), 1, 500);
  spin(strip.Color(0,0,255), 1, 500);
  spinUp();
}

// gradual spin up
void spinUp()
{
  for (int i = 300; i > 0;  i -= 20)
  {
    spin(strip.Color(255,255,255), 1, i);
  }
  pos = 0;
  // leave it with some momentum and let it 'coast' to a stop
  MomentumH = 3;
}

// Gradual spin down
void spinDown()
{
  for (int i = 1; i < 300; i++)
  {
    spin(strip.Color(255,255,255), 1, i += 20);
  }
  // Stop it dead at the top and let it swing to the bottom on its own
  pos = 0;
  MomentumH = MomentumV = 0;
}


// utility function for feedback on mode changes.
void spin(uint32_t color, int count, int time)
```

```
{
  for (int j = 0; j < count; j++)
  {
    for (int i = 0; i < 16; i++)
    {
      strip.setPixelColor(i, color);
      strip.setPixelColor(31 - i, color);
      strip.show();
      delay(max(time / 16, 1));
      strip.setPixelColor(i, 0);
      strip.setPixelColor(31 - i, 0);
      strip.show();
    }
  }
}
```

# Use Them!

## Safety First!

These goggles are not suitable for general use as eyewear, and certainly not safe to use as protective lenses. The flashing lights are very visible inside the goggles. They will impair your vision and may cause dizziness headaches or even nausea with prolonged use. The LED rings themselves will severely limit your peripheral vision, making it dangerous to walk-about, much less drive a car, juggle chainsaws or pilot a starship.

Please use them safely! When moving around, wear them on your forehead or around your neck.

## Color Control

The LED color is controlled by the compass heading computed from the LSM303 magnetometer readings. If you are ever lost in the woods, you will find these goggles to be slightly more accurate for direction finding than moss on the side of a tree.

- North-NorthEast – Blue
- East – Cyan
- South-SouthEast – Green
- South SouthWest – Yellow
- West – Red
- North NorthWest – Purple

## Operating Modes:

The STEAM-Punk Goggle Operating System recognizes several "gesture" commands for changing operating modes. All it takes is a nod of the head to engage the anti-gravity circuits:

### Anti-Gravity Mode

To engage the anti-gravity circuit, look straight up for 2 seconds. You will see the spin-up display while the anti-gravity field is building. When the anti-gravity field is active, the pendulums will be upside-down.

To disengage the anti-gravity field, look straight down for 2 seconds. The spin-down display will indicate that the anti-gravity field is winding down.

## Mirror Mode:

To engage mirror mode operation, lean to the left for 2 seconds. The spin up display will indicate successful activation of the mirroring circuits.

To disengage the mirroring circuits, lean to the right for 2 seconds. The spin-down display will indicate the return to normal synchronized operation.