Circuit Playground Express Compass

Created by Dave Astels

https://learn.adafruit.com/circuit-playground-express-compass

Last updated on 2023-09-12 10:31:01 AM EDT
# Table of Contents

## Overview
- Calibration
- Operation
- Parts
- Supplies and Tools

## Hardware
- Adding the LSM303
- Putting it in a Case

## Arduino
- Install Libraries
- Install Board Support Package
- Upload Code
- Compass Calibration
- A Guided Tour

## CircuitPython
- A Guided Tour
- Compass Calibration
Overview

The Circuit Playground Express has a 3-axis LIS3DH accelerometer on board which can be used for all sorts of projects. This chip has the advantage of being very small and fits nicely at the center of the Circuit Playground boards. Sometimes, though, "just" being a 3-axis accelerometer is a disadvantage. Times like those when you need compass-like capabilities. An accelerometer only tells you how the board is moving (the forces acting on it, to be precise), but not how it's oriented relative to the magnetic north pole of whichever planet you're on. For that, a magnetometer is needed.

Magnetometers do come as separate chips (the BBC Micro:bit has one, for example) but it doesn't seem as easy to find on a breakout board. However, breakouts containing a single chip with both an accelerometer and a magnetometer are easy to find. Specificity, there are breakouts containing the LSM303 in both the usual rectangular header-strip format as well as a Flora breakout.

In this guide we'll add the Flora LMS303 breakout to a Circuit Playground Express and use it to build a compass.

Coding in C (Arduino), you can actually use any breakout whose supporting library conforms to the Adafruit Unified Sensor Library and provides magnetometer readings. The project is also programmable in CircuitPython and demonstrates use of the accelerometer.

The NeoPixels on the Circuit Playground Express will be used to indicate North.
Calibration

During calibration (when all the pixels are green) move the box in a figure eight and rotate it around the x, and y axes multiple times.

By sampling a range of readings, we can get a sense of the expected range of values. Using that, we can calculate the center on the range on both the X and Y axes. The distance of that from (0, 0) can be used to correct subsequent readings to be relative to (0, 0).

When calibrating from the original code (i.e. you haven't tweaked the min/max arrays) or if you ask for a recalibration (by pressing button A until the pixels turn green) the result will be printed on the console. You can copy these two lines and replace the similar ones near the top of the code. Rebuild it (if using the Arduino code) and load it onto your compass.

Operation

We can then map each corrected reading to the range -100 to +100, and pass these normalized values to the atan2 function that gives us a heading, the angle as shown in the range -Pi to Pi.

Then we find which 30 degree wedge the heading falls into. We start by adding 180 to the angle from above, giving us something between 0 and 360. We add 15 (because the lowest segment is centered on zero) and divide by 30.

And that's how we know which NeoPixel to light up.

Parts

Circuit Playground Express
Circuit Playground Express is the next step towards a perfect introduction to electronics and programming. We've taken the original Circuit Playground Classic and...
https://www.adafruit.com/product/3333
FLORA Accelerometer/Compass Sensor - LSM303
Add motion and direction sensing to your wearable FLORA project with this high precision 3-axis Accelerometer+Compass sensor. Inside are two sensors, one is a classic 3-axis...
https://www.adafruit.com/product/1247

JST PH 2-Pin Cable – Male Header 200mm
For a really long time we assumed that the JST PH didn't have a free-hanging male header version. But then we found this JST-PH 2-pin Male Cable, and we were...
https://www.adafruit.com/product/3814

JST PH 2-Pin Cable - Female Connector 100mm
Red and black tinned wires with a 2-pin JST PH connector on the end. 4" / 100mm long. Matches up nicely with our Lipoly chargers!
https://www.adafruit.com/product/261

Breadboard-friendly SPDT Slide Switch
These nice switches are perfect for use with breadboard and perfboard projects. They have 0.1" spacing and snap in nicely into a solderless breadboard. They're easy to switch...
https://www.adafruit.com/product/805
Lithium Ion Polymer Battery - 3.7v 150mAh
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This... https://www.adafruit.com/product/1317

Silicone Cover Stranded-Core Wire - 2m 26AWG Blue
Silicone-sheathing wire is super-flexible and soft, and its also strong! Able to handle up to 200°C and up to 600V, it will do when PVC covered wire wimps out. We like this wire... https://www.adafruit.com/product/1878

Supplies and Tools

• Hot Glue & Glue Gun
• Solder and Soldering Iron

For the build described here, you'll probably want a case for the Circuit Playground Express. If you have a 3D printer or have access to 3D printing facilities, this guide shows how to produce a nice one.

Hardware

Adding the LSM303

Wiring the LSM303 is easy - like most sensors, it uses I2C - so you just need to wire up:

• 3.3V to 3.3V
• GND to GND
• SDA to SDA
pretty easy!

Start by hot gluing the LSM303 to the back of the Circuit Playground Express such that the boards are back to back. Be sure to position the LSM303 in the center of the Circuit Playground Express with the X axis running between the USB and power connections.

Once the glue cools, you can wire the breakout. The arrangement of power, ground and I2C matches up with that of the Circuit Playground Express neatly.

While that's really all that's needed to make a working compass, we can go the next step and package it nicely.

Putting it in a Case
Start with a case. This is a good one: it provides space for a small LiPo battery and an on-off switch, as well as access to the USB connector.

To connect the battery you will need male and female JST cables and a slide switch. Cut the wires on the JST connectors to fit neatly in the case, under the Circuit Playground Express. Connect the black wires together and connect the red wires to two adjacent connections of the switch. Using a bit of heat shrink on each of the connections is almost always a good idea: it helps avoid accidental shorts.

Next, secure the switch to the case with a little hot glue. Once it cools, connect the battery and Circuit Playground Express and tuck everything into place.
Secure the board as appropriate for the case you're using.

All that's left is to snap the cover into place and load up the software.

Arduino

We'll use the Arduino IDE for the compass code - download & install it if you haven't already.

If you're new to Arduino, check out the Getting Started with Arduino guide.

Install Libraries

Along with Arduino IDE, we'll need to install the following three libraries.

Open up the Arduino IDE and from the top menu, go to Sketch --> Include Library --> Manage Library, search for Unified Sensor and install the latest version of the Adafruit Unified Sensor library.

Next follow the same steps as above to install the Unified LSM303 library (or the library to match the accel/mag sensor you are using) and the Adafruit NeoPixel libraries.
Install Board Support Package

Additionally, go to Tools --> Board: --> Board Manager and update the Adafruit SAMD boards library if you haven't already. Additional info for installing boards in the Arduino IDE is available [here](#).

Upload Code

Go to Tools --> Board, and choose Adafruit Circuit Playground Express. Then go to Tools --> Port and choose the corresponding port for your board.

Create a new sketch, copy the code you see below, and paste it into that new sketch.

```
// SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
// SPDX-License-Identifier: MIT

/* Circuit Playground Express compass. */

/* Adafruit invests time and resources providing this open source code. */
/* Please support Adafruit and open source hardware by purchasing */
/* products from Adafruit! */

/* Written by Dave Astels for Adafruit Industries */
/* Copyright (c) 2018 Adafruit Industries */
```
// Licensed under the MIT license. */

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

/* Assign a unique ID to this sensor at the same time */
Adafruit_LSM303_Mag_Unified mag = Adafruit_LSM303_Mag_Unified(12345);

// Replace these two lines with the results of calibration
VENTORY

float raw_mins[2] = {1000.0, 1000.0};
float raw_maxes[2] = {-1000.0, -1000.0};

float mins[2];
float maxes[2];
float corrections[2] = {0.0, 0.0};

// Support both classic and express
#ifdef __AVR__
#define NEOPIXEL_PIN 17
#else
#define NEOPIXEL_PIN 8
#endif

// When we setup the NeoPixel library, we tell it how many pixels, and which pin to
// use to send signals.
// Note that for older NeoPixel strips you might need to change the third
// parameter--see the strandtest
// example for more information on possible values.
Adafruit_NeoPixel strip = Adafruit_NeoPixel(10, NEOPIXEL_PIN, NEO_GRB + NEO_KHZ800);

// Map direction pie slices (of 30 deg each) to a neopixel, or two for the missing
// ones at USB & power.
int led_patterns[12][2] = {{4, 5}, {5, -1}, {6, -1}, {7, -1}, {8, -1}, {9, -1}, {9, 0}, {0 -1}, {1, -1}, {2, -1}, {3, -1}, {4, -1}};

#define BUTTON_A 4

void fill(int red, int green, int blue) {
  for (int i = 0; i < 10; i++) {
    strip.setPixelColor(i, red, green, blue);
  }
  strip.show();
}

// Do some initial reading to let the magnetometer settle in.
// This was found by experience to be required.
// Indicated to the user by blue LEDs.
void warm_up(void) {
  sensors_event_t event;
  fill(0, 0, 64);
  for (int ignore = 0; ignore < 100; ignore++) {
    mag.getEvent(&event);
    delay(10);
  }
}

// Find the range of X and Y values.
// User needs to rotate the CPX a bunch during this.
// Can be refined by doing more of the same by pressing the A button.
// Indicated to the user by green LEDs.
void calibrate(bool do_the_readings)
{
    sensors_event_t event;
    float values[2];

    fill(0, 64, 0);

    if (do_the_readings) {
        unsigned long start_time = millis();
        while (millis() - start_time < 5000) {
            mag.getEvent(&event);
            values[0] = event.magnetic.x;
            values[1] = event.magnetic.y * -1;
            if (values[0] != 0.0 && values[1] != 0.0) { /* ignore the random zero readings... it's bogus */
                for (int i = 0; i < 2; i++) {
                }
            }
            delay(5);
        }
    }

    for (int i = 0; i < 2; i++) {
        corrections[i] = (raw_maxes[i] + raw_mins[i]) / 2;
        mins[i] = raw_mins[i] - corrections[i];
        maxes[i] = raw_maxes[i] - corrections[i];
    }
    fill(0, 0, 0);
}

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

    pinMode(BUTTON_A, INPUT_PULLDOWN);

    /* Enable auto-gain */
    mag.enableAutoRange(true);

    /* Initialise the sensor */
    if(!mag.begin())
    {
        /* There was a problem detecting the LSM303 ... check your connections */
        fill(255, 0, 0);
    }
}

warm_up();

// If reset with button A pressed or calibration hasn't been done, run calibration and report the results
if (digitalRead(BUTTON_A) || (raw_mins[0] == 1000.0 && raw_mins[1] == 1000.0)) {
    while (!Serial);
    Serial.begin(9600);
    Serial.println("Compass calibration\n");

    raw_mins[0] = 1000.0;
    raw_mins[1] = 1000.0;
    raw_maxes[0] = -1000.0;
    raw_maxes[1] = -1000.0;
    calibrate(true);
Serial.println("Calibration results
");
Serial.println("Update the corresponding lines near the top of the code
");
Serial.print("float raw_mins[2] = { "); Serial.print(raw_mins[0]);
Serial.print(", "); Serial.print(raw_mins[1]); Serial.println("};");
Serial.print("float raw_maxes[2] = { "); Serial.print(raw_maxes[0]);
Serial.print(", "); Serial.print(raw_maxes[1]); Serial.println("};
");

while(1);
}

// Map a value from the input range to the output range
// Used to map MAG values from the calibrated (min/max) range to (-100, 100)
float normalize(float value, float in_min, float in_max) {
  float mapped = (value - in_min) * 200 / (in_max - in_min) + -100;
  float max_clipped = mapped <  100 ? mapped : 100;
  float min_clipped = max_clipped > -100 ? max_clipped : -100;
  return min_clipped;
}

void loop(void)
{
  // Pressing button A does another round of calibration.
  if (digitalRead(BUTTON_A)) {
    calibrate(true);
    
    sensors_event_t event;
mag.getEvent(&event);
      
    float x = event.magnetic.x;
    float y = event.magnetic.y * -1;

    if (x == 0.0 && y == 0.0) {
      return;
    }

    float normalized_x = normalize(x - corrections[0], mins[0], maxes[0]);
    float normalized_y = normalize(y - corrections[1], mins[1], maxes[1]);

    int compass_heading = (int)(atan2(normalized_y, normalized_x) * 180.0 / 3.14159);
    // compass_heading is between -180 and +180 since atan2 returns -pi to +pi
    // this translates it to be between 0 and 360
    compass_heading += 180;
    // We add 15 to account to the zero position being 0 +/- 15 degrees.
    // mod by 360 to keep it within a circle
    // divide by 30 to find which pixel corresponding pie slice it's in
    int direction_index = ((compass_heading + 15) % 360) / 30;
    // light the pixel(s) for the direction the compass is pointing
    // the empty spots where the USB and power connects are use the two leds to
    // either side.
    int *leds;
    leds = led_patterns[direction_index];
    for (int pixel = 0; pixel < 10; pixel++) {
      if (pixel == leds[0] || pixel == leds[1]) {
        strip.setPixelColor(pixel, 4, 0, 0);
      } else {
        strip.setPixelColor(pixel, 0, 0, 0);
      }
    }
    strip.show();
  }
delay(50);

Click the Upload button and wait for the process to complete. Once you see Done Uploading at the bottom of the window, the Circuit Playground Express should warm up and enter calibration (be sure to open the serial monitor first).

Compass Calibration

The compass needs to be calibrated before use. This adapts it to the variation in the chip as well as the local magnetic environment. Simply spin the compass when the NeoPixels are green. If the compass acts erratically, pressing the left button (A) will do an additional calibration phase, which will hopefully improve the calibration. Be sure to spin it a couple full rotations for the widest range of values.

To calibrate you need the serial console open in order to get the calibrated values if you wish to set them in the code.

There are a few ways to run a calibration. If the compass has never been calibrated (the `raw_mins` array contains all 1000.0) it will be calibrated. Additionally it will be done if the A/left button is pressed when the power is turned on or reset is pressed. In both these cases it is assumed that you have a USB serial connection. When calibration completes, two lines are output, similar to:

```plaintext
float raw_maxes[2] = {-136.96, 262.61};
```

Copy these into the code, replacing the two that look the same (other than the values), rebuild, and load onto the Circuit Playground Express. Subsequently powering on or resetting will then skip calibration.
A Guided Tour

The `warm_up` function is used when starting up to make multiple reads from the magnetometer. Experience showed that the initial reads were unreliable; the magnetometer needs to warm up it would seem. The NeoPixels are set to blue during this phase.

```c
void warm_up(void) {
    sensors_event_t event;
    fill(0, 0, 64);
    for (int ignore = 0; ignore < 100; ignore++) {
        mag.getEvent(&event);
        delay(10);
    }
}
```

Calibration works by making about 1000 readings, and keeping track of the lowest and highest values along the X and Y axes. At the end, the origin offset is calculated by finding the center point of the readings (averaging the minimum and maximum values). Now we have the offset for readings as well as the expected value ranges. If the SAMD21 had EEPROM this could be stored and subsequent automatic calibration could be skipped, without having to rebuild the code.

```c
void calibrate(bool do_the_readings) {
    sensors_event_t event;
    float values[2];
    fill(0, 64, 0);
    if (do_the_readings) {
        unsigned long start_time = millis();
        while (millis() - start_time &lt; 5000) {
            mag.getEvent(&amp;event);
            values[0] = event.magnetic.x;
            values[1] = event.magnetic.y * -1;
            if (values[0] != 0.0 &amp;&amp; values[1] != 0.0) {/* ignore the random zero readings... it's bogus */
                for (int i = 0; i &lt; 2; i++) {
                    raw_mins[i] = values[i] &lt; raw_mins[i] ? values[i] : raw_mins[i];
                    raw_maxes[i] = values[i] &gt; raw_maxes[i] ? values[i] : raw_maxes[i];
                }
                delay(5);
            }
        }
        for (int i = 0; i &lt; 2; i++) {
            corrections[i] = (raw_maxes[i] + raw_mins[i]) / 2;
            mins[i] = raw_mins[i] - corrections[i];
            maxes[i] = raw_maxes[i] - corrections[i];
        }
        fill(0, 0, 0);
    }
}
```
The main loop runs about 20 times per second. It starts by checking for the left button being pushed. If it is, more calibration is done. Next the magnetometer values are read for X and Y.

```c
if (digitalRead(BUTTON_A)) {
    calibrate(true);
}

sensors_event_t event;
mag.getEvent(&event);
float x = event.magnetic.x;
float y = event.magnetic.y * -1;
```

Notice that the Y value is negated since the breakout is mounted upside down, flipping the Y axis.

Occasionally there will be a problem and the read values will both be zero. If so the reading is ignored.

```c
if (x == 0.0 && y == 0.0) {
    return;
}
```

Assuming the values are good, they are normalized. This is a simple mapping of the value from the range that resulted from calibration, to the range from -100 to +100.

```c
float normalize(float value, float in_min, float in_max) {
    float mapped = (value - in_min) * 200 / (in_max - in_min) + -100;
    float max_clipped = mapped < 100 ? mapped : 100;
    float min_clipped = max_clipped > -100 ? max_clipped : -100;
    return min_clipped;
}
```

Once we have normalized values the `atan2` function is used to compute the angle of the vector made by those values. The angle will be in radians (ranging from -pi to +pi). That gets converted to degrees by multiplying by 180/pi. Now we have a value in degrees that ranges from -180 to +180. That gets 180 added to it to give a value from 0 to 360. This is then divided by 30 to give the number of a wedge which corresponds to 1 of 10 NeoPixel positions. Because the wedge at the top is defined by the angles at +/-15 degrees, 15 is added (and the result clipped to 360 by using the modulus operator %) before the division.

```c
float normalized_x = normalize(x - corrections[0], mins[0], maxes[0]);
float normalized_y = normalize(y - corrections[1], mins[1], maxes[1]);
int compass_heading = (int)(atan2(normalized_y, normalized_x) * 180.0 / 3.14159) + 15;
```
compass_heading is between -180 and +180 since atan2 returns -pi to +pi
// this translates it to be between 0 and 360
compass_heading += 180;

// We add 15 to account to the zero position being 0 +/- 15 degrees.
// mod by 360 to keep it within a circle
// divide by 30 to find which pixel corresponding pie slice it's in
int direction_index = ((compass_heading + 15) % 360) / 30;

The final step is to update the NeoPixels. The *led_patterns* array that is defined at
the top of the code is used to specify which pixels correspond to each wedge. Notice
that two of the wedges specify two pixels, while the other 10 use only one. This is
because the Circuit Playgrounds only have 10 NeoPixels; the top and bottom positions
are used for the USB and battery connectors. To work around this, the two adjacent
NeoPixels are used.

```
int *leds;
leds = led_patterns[direction_index];
for (int pixel = 0; pixel < 10; pixel++) {
  if (pixel == leds[0] || pixel == leds[1]) {
    strip.setPixelColor(pixel, 4, 0, 0);
  } else {
    strip.setPixelColor(pixel, 0, 0, 0);
  }
}
strip.show();
```

The complete code is below:

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

// Assign a unique ID to this sensor at the same time */
Adafruit_LSM303_Mag_Unified mag = Adafruit_LSM303_Mag_Unified(12345);

// Replace these two lines with the results of calibration
//---------------------------------------------------------------------------------------------
float raw_mins[2] = {1000.0, 1000.0};
float raw_maxes[2] = {-1000.0, -1000.0};
//---------------------------------------------------------------------------------------------
```
float mins[2];
float maxes[2];
float corrections[2] = {0.0, 0.0};

// Support both classic and express
#ifdef __AVR__
define NEOPIXEL_PIN 17
#else
define NEOPIXEL_PIN 8
#endif

// When we setup the NeoPixel library, we tell it how many pixels, and which pin to use to send signals.
// Note that for older NeoPixel strips you might need to change the third parameter—see the strandtest
// example for more information on possible values.
Adafruit_NeoPixel strip = Adafruit_NeoPixel(10, NEOPIXEL_PIN, NEO_GRB + NEO_KHZ800);

// Map direction pie slices (of 30 deg each) to a neopixel, or two for the missing ones at USB & power.
int led_patterns[12][2] = {{4, 5}, {5, -1}, {6, -1}, {7, -1}, {8, -1}, {9, -1}, {9, 0}, {0, -1}, {1, -1}, {2, -1}, {3, -1}, {4, -1}};
#define BUTTON_A 4

void fill(int red, int green, int blue) {
    for (int i = 0; i < 10; i++) {
        strip.setPixelColor(i, red, green, blue);
    }
    strip.show();
}

// Do some initial reading to let the magnetometer settle in.
// This was found by experience to be required.
// Indicated to the user by blue LEDs.
void warm_up(void) {
    sensors_event_t event;
    fill(0, 0, 64);
    for (int ignore = 0; ignore < 100; ignore++) {
        mag.getEvent(&event);
        delay(10);
    }
}

// Find the range of X and Y values.
// User needs to rotate the CPX a bunch during this.
// Can be refined by doing more of the same by pressing the A button.
// Indicated to the user by green LEDs.
void calibrate(bool do_the_readings) {
    sensors_event_t event;
    float values[2];
    fill(0, 64, 0);
    if (do_the_readings) {
        unsigned long start_time = millis();
        while (millis() - start_time < 5000) {
            mag.getEvent(&event);
            values[0] = event.magnetic.x;
            values[1] = event.magnetic.y * -1;
            if (values[0] != 0.0 && values[1] != 0.0) { /* ignore the random zero readings... it's bogus */
                for (int i = 0; i < 2; i++) {
                    }
```c
}
}
delay(5);
}

for (int i = 0; i < 2; i++) {
corrections[i] = (raw_maxes[i] + raw_mins[i]) / 2;
mins[i] = raw_mins[i] - corrections[i];
maxes[i] = raw_maxes[i] - corrections[i];
}
fill(0, 0, 0);
}

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

    pinMode(BUTTON_A, INPUT_PULLDOWN);

    /* Enable auto-gain */
    mag.enableAutoRange(true);

    /* Initialise the sensor */
    if (!mag.begin())
    {
        /* There was a problem detecting the LSM303 ... check your connections */
        fill(255, 0, 0);    
        while(1);
    }
    warm_up();

    // If reset with button A pressed or calibration hasn't been done, run
    // calibration and report the results
    if (digitalRead(BUTTON_A) || (raw_mins[0] == 1000.0 && raw_mins[1] == 1000.0)) {
        while (!Serial);
        Serial.begin(9600);
        Serial.println("Compass calibration\n");

        raw_mins[0] = 1000.0;
        raw_mins[1] = 1000.0;
        raw_maxes[0] = -1000.0;
        raw_maxes[1] = -1000.0;
        calibrate(true);
        Serial.println("Calibration results\n");
        Serial.println("Update the corresponding lines near the top of the code\n");
        Serial.println("float raw_mins[2] = "); Serial.println(raw_mins[0]);
        Serial.print("", "); Serial.println(raw_mins[1]); Serial.println(";");
        Serial.println("float raw_maxes[2] = "); Serial.println(raw_maxes[0]);
        Serial.print("", "); Serial.println(raw_maxes[1]); Serial.println(";\n");
        while(1);
    } else {
        calibrate(false);
    }
}

// Map a value from the input range to the output range
// Used to map MAG values from the calibrated (min/max) range to (-100, 100)
float normalize(float value, float in_min, float in_max) {
    float mapped = (value - in_min) * 200 / (in_max - in_min) + -100;
    float max_clipped = mapped < 100 ? mapped : 100;
}
```
float min_clipped = max_clipped > -100 ? max_clipped : -100;
return min_clipped;
}

void loop(void)
{
  // Pressing button A does another round of calibration.
  if (digitalRead(BUTTON_A)) {
    calibrate(true);
  }
  sensors_event_t event;
  mag.getEvent(&event);
  float x = event.magnetic.x;
  float y = event.magnetic.y * -1;
  if (x == 0.0 && y == 0.0) {
    return;
  }
  float normalized_x = normalize(x - corrections[0], mins[0], maxes[0]);
  float normalized_y = normalize(y - corrections[1], mins[1], maxes[1]);
  int compass_heading = (int)(atan2(normalized_y, normalized_x) * 180.0 / 3.14159);
  // compass_heading is between -180 and +180 since atan2 returns -pi to +pi
  // this translates it to be between 0 and 360
  compass_heading += 180;
  // We add 15 to account to the zero position being 0 +/- 15 degrees.
  // mod by 360 to keep it within a circle
  // divide by 30 to find which pixel corresponding pie slice it's in
  int direction_index = ((compass_heading + 15) % 360) / 30;
  // light the pixel(s) for the direction the compass is pointing
  // the empty spots where the USB and power connects are use the two leds to
  // either side.
  int *leds;
  leds = led_patterns[direction_index];
  for (int pixel = 0; pixel < 10; pixel++) {
    if (pixel == leds[0] || pixel == leds[1]) {
      strip.setPixelColor(pixel, 4, 0, 0);
    } else {
      strip.setPixelColor(pixel, 0, 0, 0);
    }
  }
  strip.show();
  delay(50);
}
We'll be using CircuitPython for this project. Are you new to using CircuitPython? No worries, there is a full getting started guide here.

Adafruit suggests using the Mu editor to edit your code and have an interactive REPL in CircuitPython. You can learn about Mu and its installation in this tutorial.

In addition, the LSM303 module is required. We have a great guide on that. Make sure you have the latest version of this module.

A Guided Tour

The `warm_up` function is used when starting up to make multiple reads from the magnetometer. Experience showed that the initial reads were unreliable; the magnetometer needs to warm up it would seem. The NeoPixels are set to blue during this phase.

```python
def warm_up():
    fill(BLUE)
    for _ in range(100):
        _, _, _ = compass.magnetic
        time.sleep(0.010)
```

Compass Calibration

The compass needs to be calibrated before use. This adapts it to the variation in the chip as well as the local magnetic environment. Simply spin the compass when the NeoPixels are green. If the compass acts erratically, pressing the left button (A) will do...
an additional calibration phase, which will hopefully improve the calibration. Be sure to spin it a couple full rotations for the widest range of values.

There are a few ways to run a calibration. If the compass has never been calibrated (the \texttt{raw_mins} array contains all 1000.0), it will be calibrated. Additionally, it will be done if the A/left button is pressed when the power is turned on or reset is pressed. In both these cases, it is assumed that you have a USB serial connection. When calibration completes, two lines are output, similar to:

\begin{verbatim}
raw_mins = [-2.45455, 2.81818]
raw_maxes = [-1.72727, 3.54545]
\end{verbatim}

Copy these into the code, replacing the two that look the same (other than the values), and load onto the Circuit Playground Express. Subsequently powering on, resetting, or restarting will then skip calibration.

Calibration works by spending 10 seconds making readings, and keeping track of the lowest and highest values along the X and Y axes. At the end, the origin offset is calculated by finding the center point of the readings (averaging the minimum and maximum values). Now we have the offset for readings as well as the expected value ranges.

```python
def calibrate(do_the_readings):
    values = [0.0, 0.0]
    start_time = time.monotonic()
    fill(GREEN)

    # Update the high and low extremes
    if do_the_readings:
        while time.monotonic() - start_time < 10.0:
            values[0], values[1], _ = compass.magnetic
            values[1] *= -1                           # accel is upside down, so y is reversed
        if values[0] != 0.0 and values[1] != 0.0: # ignore the random 0 values
            for i in range(2):
                if values[i] < raw_mins[i]:
                    raw_mins[i] = values[i]
                if values[i] > raw_maxes[i]:
                    raw_maxes[i] = values[i]

        # Recompute the correction and the correct mins/maxes
        for i in range(2):
            corrections[i] = (raw_maxes[i] + raw_mins[i]) / 2
            mins[i] = raw_mins[i] - corrections[i]
            maxes[i] = raw_maxes[i] - corrections[i]
```

©Adafruit Industries

Page 22 of 26
The main loop runs about 20 times per second. It starts by checking for the left button being pushed. If it is, more calibration is done. Next the magnetometer values are read for X and Y.

```python
if not calibrate_button.value:
    calibrate(True)

x, y, _ = compass.magnetic
y = y * -1
```

Notice that the Y value is negated since the breakout is mounted upside down, flipping the Y axis.

Occasionally there will be a problem and the read values will both be zero. If so the reading is ignored.

```python
if x != 0.0 and y != 0.0:
```

Assuming the values are good, they are normalized. This is a simple mapping of the value from the range that resulted from calibration, to the range from -100 to +100.

```python
def normalize(value, in_min, in_max):
    mapped = (value - in_min) * 200 / (in_max - in_min) + -100
    return max(min(mapped, 100), -100)
```

Once we have normalized values the `atan2` function is used to compute the angle of the vector made by those values. The angle will be in radians (ranging from -pi to +pi). That gets converted to degrees by multiplying by 180/pi. Now we have a value in degrees that ranges from -180 to +180. That gets 180 added to it to give a value from 0 to 360. This is then divided by 30 to give the number of a wedge which corresponds to 1 of 10 NeoPixel positions. Because the wedge at the top is defined by the angles at +/-15 degrees, 15 is added (and the result clipped to 360 by using the modulus operator %) before the division.

```python
normalized_x = normalize(x - corrections[0], mins[0], maxes[0])
normalized_y = normalize(y - corrections[1], mins[1], maxes[1])

compass_heading = int(math.atan2(normalized_y, normalized_x) * 180.0 / math.pi)
# compass_heading is between -180 and +180 since atan2 returns -pi to +pi
# this translates it to be between 0 and 360
compass_heading += 180

direction_index = ((compass_heading + 15) % 360) // 30
```
The final step is to update the NeoPixels. The `led_patterns` array that is defined at the top of the code is used to specify which pixels correspond to each wedge. Notice that two of the wedges specify two pixels, while the others use only one. This is because the Circuit Playgrounds only have 10 NeoPixels; the top and bottom positions are used for the USB and battery connectors. To work around this, the two adjacent NeoPixels are used.

```python
pixels.fill(BLACK)
for l in led_patterns[direction_index]:
    pixels[l] = RED
pixels.show()
```

The complete code is below:

```python
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

""
Circuit Playground Express Compass

Adafruit invests time and resources providing this open source code. Please support Adafruit and open source hardware by purchasing products from Adafruit!

Written by Dave Astels for Adafruit Industries
Copyright (c) 2018 Adafruit Industries
Licensed under the MIT license.

All text above must be included in any redistribution.
""

import time
import math
import sys
import board
import busio
import adafruit_lsm303
import neopixel
import digitalio

BLACK = 0x000000
RED = 0xFF0000
GREEN = 0x00FF00
BLUE = 0x0000FF

i2c = busio.I2C(board.SCL, board.SDA)
compass = adafruit_lsm303.LSM303(i2c)
compass.mag_rate = adafruit_lsm303.MAGRATE_30
compass.mag_gain = adafruit_lsm303.MAGGAIN_8_1

calibrate_button = digitalio.DigitalInOut(board.BUTTON_A)
calibrate_button.direction = digitalio.Direction.INPUT
calibrate_button.pull = digitalio.Pull.UP

# Replace these two lines with the results of calibration
raw_mins = [1000.0, 1000.0]
raw_maxes = [-1000.0, -1000.0]
```
mins = [0.0, 0.0]
maxes = [0.0, 0.0]
corrections = [0.0, 0.0]
pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, brightness=.2, auto_write=False)
led_patterns = [[4, 5], [5], [6], [7], [8], [9], [9, 0], [0], [1], [2], [3], [4]]

def fill(colour):
    pixels.fill(colour)
pixels.show()

def warm_up():
    fill(BLUE)
    for _ in range(100):
        _, _, _ = compass.magnetic
        time.sleep(0.010)

def calibrate(do_the_readings):
    values = [0.0, 0.0]
    start_time = time.monotonic()
    fill(GREEN)
    # Update the high and low extremes
    if do_the_readings:
        while time.monotonic() - start_time < 10.0:
            values[0], values[1], _ = compass.magnetic
            values[1] *= -1  # accel is upside down, so y is reversed
            if values[0] != 0.0 and values[1] != 0.0:  # ignore the random 0 values
                for i in range(2):
                    if values[i] < raw_mins[i]:
                        raw_mins[i] = values[i]
                    if values[i] > raw_maxes[i]:
                        raw_maxes[i] = values[i]
        # Recompute the correction and the correct mins/maxes
        for i in range(2):
            corrections[i] = (raw_maxes[i] + raw_mins[i]) / 2
            mins[i] = raw_mins[i] - corrections[i]
            maxes[i] = raw_maxes[i] - corrections[i]

    fill(BLACK)

def normalize(value, in_min, in_max):
    mapped = (value - in_min) * 200 / (in_max - in_min) + -100
    return max(min(mapped, 100), -100)

# Setup
warm_up()
if not calibrate_button.value or (raw_mins[0] == 1000.0 and raw_mins[1] == 1000.0):
    print("Compass calibration")
    raw_mins[0] = 1000.0
    raw_mins[1] = 1000.0
    raw_maxes[0] = -1000.0
    raw_maxes[1] = -1000.0
    calibrate(True)
print("Calibration results")
print("Update the corresponding lines near the top of the code\n")
print("raw_mins = [{0}, {1}]".format(raw_mins[0], raw_mins[1]))
print("raw_maxes = [{0}, {1}]".format(raw_maxes[0], raw_maxes[1]))
sys.exit()
else:
    calibrate(False)

while True:
    if not calibrate_button.value:
        calibrate(True)

    x, y, _ = compass.magnetic
    y = y * -1

    if x != 0.0 and y != 0.0:
        normalized_x = normalize(x - corrections[0], mins[0], maxes[0])
        normalized_y = normalize(y - corrections[1], mins[1], maxes[1])

        compass_heading = int(math.atan2(normalized_y, normalized_x) * 180.0 / math.pi)
        # compass_heading is between -180 and +180 since atan2 returns -pi to +pi
        # this translates it to be between 0 and 360
        compass_heading += 180

        direction_index = ((compass_heading + 15) % 360) // 30

        pixels.fill(BLACK)
        for l in led_patterns[direction_index]:
            pixels[l] = RED
        pixels.show()
        time.sleep(0.050)