```
// Test code for Adafruit Flora GPS modules
//
// This code shows how to listen to the GPS module in an interrupt
// which allows the program to have more 'freedom' - just parse
// when a new NMEA sentence is available! Then access data when
// desired.
//
// Tested and works great with the Adafruit Flora GPS module
// ------> http://adafruit.com/products/1059
// Pick one up today at the Adafruit electronics shop
// and help support open source hardware & software! -ada

#include <Adafruit_GPS.h>
#include <Adafruit_NeoPixel.h>
#include <SoftwareSerial.h>
#include <Time.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>

Adafruit_GPS GPS(&Serial1);
/* Assign a unique ID to this sensor at the same time */
Adafruit_LSM303_Mag_Unified mag = Adafruit_LSM303_Mag_Unified(12345);

// Set GPSECHO to 'false' to turn off echoing the GPS data to the Serial console
// Set to 'true' if you want to debug and listen to the raw GPS sentences
```

```
#define GPSECHO false

// this keeps track of whether we're using the interrupt
// off by default!
boolean usingInterrupt = false;

//-------------------------------------------------|
//                   WAYPOINT                      |
//-------------------------------------------------|
//Please enter the latitude and longitude of your  |
//desired destination:                             |
#define GEO_LAT              52.5072231             // 48.009551
#define GEO_LON              13.3096588             //-88.771131
//-------------------------------------------------|
//Your NeoPixel ring may not line up with ours.    |
//Enter which NeoPixel led is your top LED (0-15). |
#define TOP_LED               1
//-------------------------------------------------|
//Your compass module may not line up with ours.   |
//Once you run compass mode, compare to a separate |
//compass (like one found on your smartphone).     |
//Point your TOP_LED north, then count clockwise   |
//how many LEDs away from TOP_LED the lit LED is   |
#define LED_OFFSET            0
//-------------------------------------------------|
```

```
// Navigation location
float targetLat = GEO_LAT;
float targetLon = GEO_LON;

// Trip distance
float tripDistance;

Adafruit_NeoPixel strip = Adafruit_NeoPixel(16, 6, NEO_GRB + NEO_KHZ800);

// Offset hours from gps time (UTC)
const int offset = 1;   // Central European Time
//const int offset = -4;  // Eastern Daylight Time (USA)
//const int offset = -5;  // Central Daylight Time (USA)
//const int offset = -8;  // Pacific Standard Time (USA)
//const int offset = -7;  // Pacific Daylight Time (USA)

int topLED = TOP_LED;
int compassOffset = LED_OFFSET;

int lastMin = 16;
int lastHour = 16;
int startLED = 0;
int startLEDlast = 16;
int lastCombined = 0;
int start = 0;
int mode = 0;
```

```
int lastDir = 16;
int dirLED_r = 0;
int dirLED_g = 0;
int dirLED_b = 255;
int compassReading;

// Calibration offsets
float magxOffset = 2.55;
float magyOffset = 27.95;

// Pushbutton setup
int buttonPin = 10;             // the number of the pushbutton pin
int buttonState;               // the current reading from the input pin
int lastButtonState = HIGH;    // the previous reading from the input pin
long buttonHoldTime = 0;         // the last time the output pin was toggled
long buttonHoldDelay = 2500;      // how long to hold the button down

// the following variables are long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long lastDebounceTime = 0;     // the last time the output pin was toggled
long debounceDelay = 50;       // the debounce time; increase if the output flickers
long menuDelay = 2500;
long menuTime;

float fLat = 0.0;
float fLon = 0.0;
```

```
void setup()
{
  // connect at 115200 so we can read the GPS fast enough and echo without dropping chars
  // also spit it out
 Serial.begin(115200);
  Serial.println("Adafruit GPS library basic test!");

  // 9600 NMEA is the default baud rate for Adafruit MTK GPS's- some use 4800
 GPS.begin(9600);
  // uncomment this line to turn on RMC (recommended minimum) and GGA (fix data) including
altitude
 GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  // uncomment this line to turn on only the "minimum recommended" data
 //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
  // For parsing data, we don't suggest using anything but either RMC only or RMC+GGA since
  // the parser doesn't care about other sentences at this time
  // Set the update rate
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ); // 1 Hz update rate
  // For the parsing code to work nicely and have time to sort thru the data, and
  // print it out we don't suggest using anything higher than 1 Hz

  // Request updates on antenna status, comment out to keep quiet
 GPS.sendCommand(PGCMD_ANTENNA);

  /* Initialise the sensor */
```

```
  if(!mag.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println("Ooops, no LSM303 detected ... Check your wiring!");
    while(1);
  }
  // Ask for firmware version
 Serial1.println(PMTK_Q_RELEASE);

  strip.begin();
  strip.show(); // Initialize all pixels to 'off'

  // Make input & enable pull-up resistors on switch pins for pushbutton
  pinMode(buttonPin, INPUT);
  digitalWrite(buttonPin, HIGH);
}

uint32_t gpsTimer = millis();
uint32_t startupTimer = millis();
uint32_t compassTimer = millis();

void loop() // run over and over again
{
  compassCheck();
  // read the state of the switch into a local variable:
  int buttonState = digitalRead(buttonPin);
```

```
 if (buttonState == LOW) {
   buttonCheck();
 }

 lastButtonState = buttonState;

//Serial.println(buttonState);
 // read data from the GPS in the 'main loop'
 char c = GPS.read();
 // if you want to debug, this is a good time to do it!
 if (GPSECHO)
   if (c) Serial.print(c);
 // if a sentence is received, we can check the checksum, parse it...
 if (GPS.newNMEAreceived()) {
   // a tricky thing here is if we print the NMEA sentence, or data
   // we end up not listening and catching other sentences!
   // so be very wary if using OUTPUT_ALLDATA and trytng to print out data
   Serial.println(GPS.lastNMEA()); // this also sets the newNMEAreceived() flag to false
   if (!GPS.parse(GPS.lastNMEA())) // this also sets the newNMEAreceived() flag to false
     return; // we can fail to parse a sentence in which case we should just wait for another
 }

 // if millis() or timer wraps around, we'll just reset it
 if (gpsTimer > millis()) gpsTimer = millis();
```

```
if (start == 0) {
  if (GPS.fix) {
    // set the Time to the latest GPS reading
   setTime(GPS.hour, GPS.minute, GPS.seconds, GPS.day, GPS.month, GPS.year);
    delay(50);
    adjustTime(offset * SECS_PER_HOUR);
    delay(500);
   tripDistance = (double)calc_dist(fLat, fLon, targetLat, targetLon);
    start = 1;
  }
}
// approximately every 60 seconds or so, update time
if ((millis() - gpsTimer > 60000) && (start == 1)) {
  gpsTimer = millis(); // reset the timer
  if (GPS.fix) {
    // set the Time to the latest GPS reading
   setTime(GPS.hour, GPS.minute, GPS.seconds, GPS.day, GPS.month, GPS.year);
    delay(50);
    adjustTime(offset * SECS_PER_HOUR);
    delay(500);
  }
}

if (GPS.fix) {
  fLat = decimalDegrees(GPS.latitude, GPS.lat);
  fLon = decimalDegrees(GPS.longitude, GPS.lon);
```

```
  }

  if (mode == 0) {
    clockMode();
  }

  if (mode == 1) {
    navMode();
  }

  if (mode == 2) {
    compassMode();
  }
}

// Fill the dots one after the other with a color
void colorWipe(uint32_t c, uint8_t wait) {
  for(uint16_t i=0; i<strip.numPixels(); i++) {
    strip.setPixelColor(i, c);
    strip.show();
    delay(wait);
  }
}

void buttonCheck() {
  menuTime = millis();
```

```
  int buttonState = digitalRead(buttonPin);
  if (buttonState == LOW && lastButtonState == HIGH) {
    buttonHoldTime = millis();
  }

  if (buttonState == LOW && lastButtonState == LOW) {
    if ((millis() - buttonHoldTime) > buttonHoldDelay) {

      if(mode == 2) {
        mode = 0;
        lastMin = 16;
        lastHour = 16;
       colorWipe(strip.Color(0, 0, 0), 20);
       buttonHoldTime = millis();
      }
      else {
        mode = mode + 1;
       colorWipe(strip.Color(0, 0, 0), 20);
       buttonHoldTime = millis();
      }
    }
  }
}

void clockMode() {
  if (start == 1) {
```

```
strip.setPixelColor(startLEDlast, strip.Color(0, 0, 0));
strip.show();

float gpsMin = (minute() + (second()/60.0));
unsigned int ledMin = 0;
int minTemp = 0;
minTemp = topLED - (gpsMin + 1.875)/3.75;

if (minTemp < 0) {
  ledMin = minTemp + 16;
}
else {
  ledMin = minTemp;
}

float gpsHour = (hour() + (minute()/60.0));
if (gpsHour > 12) {
  gpsHour = gpsHour - 12;
}
unsigned int ledHour = 0;
int hourTemp = 0;
hourTemp = topLED - (gpsHour + .375)/.75;

if (hourTemp < 0) {
  ledHour = hourTemp + 16;
}
```

```
else {
  ledHour = hourTemp;
}

if ((ledHour == ledMin) && (lastCombined == 0)) {
 strip.setPixelColor(lastHour, strip.Color(0, 0, 0));
 strip.setPixelColor(lastMin, strip.Color(0, 0, 0));
 strip.setPixelColor(ledHour, strip.Color(255, 0, 255));
  strip.show();
  lastCombined = 1;
  lastHour = ledHour;
  lastMin = ledMin;
}
else {
  if (lastHour != ledHour) {
   strip.setPixelColor(lastHour, strip.Color(0, 0, 0));
   strip.setPixelColor(ledHour, strip.Color(255, 50, 0));
    strip.show();
    lastHour = ledHour;
  }
  if (lastMin != ledMin) {
   strip.setPixelColor(lastMin, strip.Color(0, 0, 0));
   strip.setPixelColor(ledMin, strip.Color(200, 200, 0));
    if (lastCombined == 1) {
     strip.setPixelColor(ledHour, strip.Color(255, 0, 0));
      lastCombined = 0;
```

```
        }
        strip.show();
        lastMin = ledMin;
      }
    }
  }
  else {
    // if millis() or timer wraps around, we'll just reset it
    if (startupTimer > millis()) startupTimer = millis();

    // approximately every 10 seconds or so, update time
    if (millis() - startupTimer > 200) {
      startupTimer = millis(); // reset the timer
      if (startLED == 16) {
        startLED = 0;
      }
     strip.setPixelColor(startLEDlast, strip.Color(0, 0, 0));
     strip.setPixelColor(startLED, strip.Color(0, 255, 0));
      strip.show();
      startLEDlast = startLED;
      startLED++;
      //delay(200);
    }
  }
}
```

```
void navMode() {
  if (start == 1) {

    compassCheck();

   headingDistance((double)calc_dist(fLat, fLon, targetLat, targetLon));

    if ((calc_bearing(fLat, fLon, targetLat, targetLon) - compassReading) > 0) {
     compassDirection(calc_bearing(fLat, fLon, targetLat, targetLon)-compassReading);
    }
    else {
     compassDirection(calc_bearing(fLat, fLon, targetLat, targetLon)-compassReading+360);
    }

  }
  else {
    // if millis() or timer wraps around, we'll just reset it
    if (startupTimer > millis()) startupTimer = millis();

    // approximately every 10 seconds or so, update time
    if (millis() - startupTimer > 200) {
      startupTimer = millis(); // reset the timer
      if (startLED == 16) {
        startLED = 0;
       }
      strip.setPixelColor(startLEDlast, strip.Color(0, 0, 0));
```

```
     strip.setPixelColor(startLED, strip.Color(0, 0, 255));
      strip.show();
      startLEDlast = startLED;
      startLED++;
    }
  }
}

int calc_bearing(float flat1, float flon1, float flat2, float flon2)
{
  float calc;
  float bear_calc;

  float x = 69.1 * (flat2 - flat1);
  float y = 69.1 * (flon2 - flon1) * cos(flat1/57.3);

  calc=atan2(y,x);

  bear_calc= degrees(calc);

  if(bear_calc<=1){
    bear_calc=360+bear_calc;
   }
  return bear_calc;
}
void headingDistance(int fDist)
```

```
{
  //Use this part of the code to determine how far you are away from the destination.
  //The total trip distance (from where you started) is divided into five trip segments.
  float tripSegment = tripDistance/5;

  if (fDist >= (tripSegment*4)) {
    dirLED_r = 255;
    dirLED_g = 0;
    dirLED_b = 0;
  }

  if ((fDist >= (tripSegment*3))&&(fDist < (tripSegment*4))) {
    dirLED_r = 255;
    dirLED_g = 0;
    dirLED_b = 0;
  }

  if ((fDist >= (tripSegment*2))&&(fDist < (tripSegment*3))) {
    dirLED_r = 255;
    dirLED_g = 255;
    dirLED_b = 0;
  }

  if ((fDist >= tripSegment)&&(fDist < (tripSegment*2))) {
    dirLED_r = 255;
    dirLED_g = 255;
```

```
    dirLED_b = 0;
  }

  if ((fDist >= 5)&&(fDist < tripSegment)) {
    dirLED_r = 255;
    dirLED_g = 255;
    dirLED_b = 0;
  }

  if ((fDist < 5)) { // You are now within 5 meters of your destination.
   //Serial.println("Arrived at destination!");
    dirLED_r = 0;
    dirLED_g = 255;
    dirLED_b = 0;
  }
}


unsigned long calc_dist(float flat1, float flon1, float flat2, float flon2)
{
  float dist_calc=0;
  float dist_calc2=0;
  float diflat=0;
  float diflon=0;

 diflat=radians(flat2-flat1);
```

```
 flat1=radians(flat1);
 flat2=radians(flat2);
 diflon=radians((flon2)-(flon1));

 dist_calc = (sin(diflat/2.0)*sin(diflat/2.0));
 dist_calc2= cos(flat1);
 dist_calc2*=cos(flat2);
 dist_calc2*=sin(diflon/2.0);
 dist_calc2*=sin(diflon/2.0);
 dist_calc +=dist_calc2;

dist_calc=(2*atan2(sqrt(dist_calc),sqrt(1.0-dist_calc)));

 dist_calc*=6371000.0; //Converting to meters
 return dist_calc;
}

// Convert NMEA coordinate to decimal degrees
float decimalDegrees(float nmeaCoord, char dir) {
 uint16_t wholeDegrees = 0.01*nmeaCoord;
 int modifier = 1;

 if (dir == 'W' || dir == 'S') {
   modifier = -1;
 }
```

```
  return (wholeDegrees + (nmeaCoord - 100.0*wholeDegrees)/60.0) * modifier;
}

void compassMode() {
  dirLED_r = 0;
  dirLED_g = 0;
  dirLED_b = 255;
 compassDirection(compassReading);
}

void compassCheck() {
  // if millis() or timer wraps around, we'll just reset it
  if (compassTimer > millis()) compassTimer = millis();

  // approximately every 10 seconds or so, update time
  if (millis() - compassTimer > 50) {
    /* Get a new sensor event */
    sensors_event_t event;
    mag.getEvent(&event);

    float Pi = 3.14159;

    compassTimer = millis(); // reset the timer

    // Calculate the angle of the vector y,x
    float heading = (atan2(event.magnetic.y + magyOffset,event.magnetic.x + magxOffset) * 180)
```

```
/ Pi;

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

void compassDirection(int compassHeading)
{
  //Serial.print("Compass Direction: ");
 //Serial.println(compassHeading);

  unsigned int ledDir = 2;
  int tempDir = 0;
  //Use this part of the code to determine which way you need to go.
  //Remember: this is not the direction you are heading, it is the direction to the destination
(north = forward).

  if ((compassHeading > 348.75)||(compassHeading < 11.25)) {
      tempDir = topLED;
  }
  for(int i = 1; i < 16; i++){
```

```
  float pieSliceCenter = 45/2*i;
  float pieSliceMin = pieSliceCenter - 11.25;
  float pieSliceMax = pieSliceCenter + 11.25;
  if ((compassHeading >= pieSliceMin)&&(compassHeading < pieSliceMax)) {
    if (mode == 2 ) {
      tempDir = topLED - i;
    }
    else {
      tempDir = topLED + i;
    }
  }
}

if (tempDir > 15) {
  ledDir = tempDir - 16;
}

else if (tempDir < 0) {
  ledDir = tempDir + 16;
}
else {
  ledDir = tempDir;
}

if (mode == 1) {
  ledDir = ledDir + compassOffset;
```

```
    if (ledDir > 15) {
      ledDir = ledDir - 16;
    }
  }
  else {
    ledDir = ledDir + compassOffset;
    if (ledDir > 15) {
      ledDir = ledDir - 16;
    }
  }

  if (lastDir != ledDir) {
   strip.setPixelColor(lastDir, strip.Color(0, 0, 0));
   strip.setPixelColor(ledDir, strip.Color(dirLED_r, dirLED_g, dirLED_b));
    strip.show();
    lastDir = ledDir;
  }
}
```