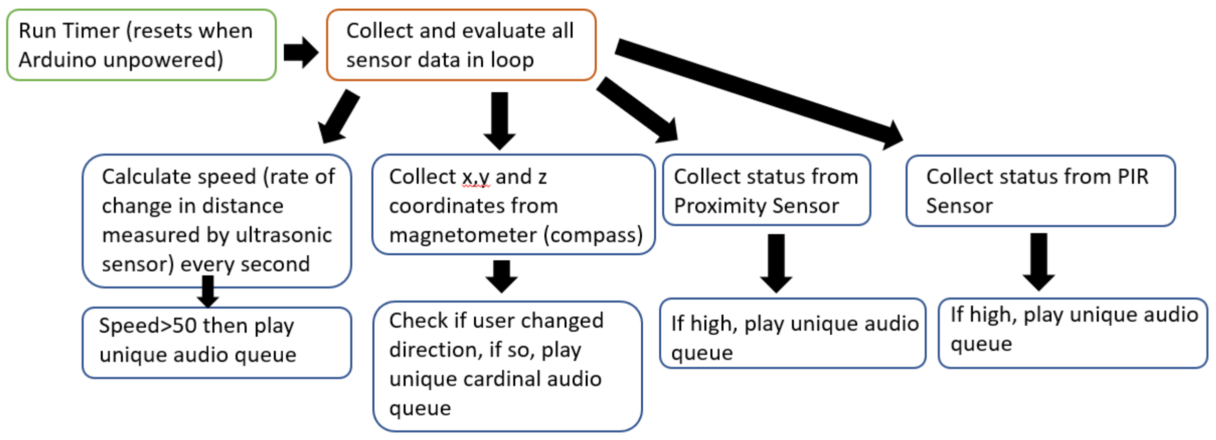
Headband for the Visually Impaired

By George Dobrić 101227367

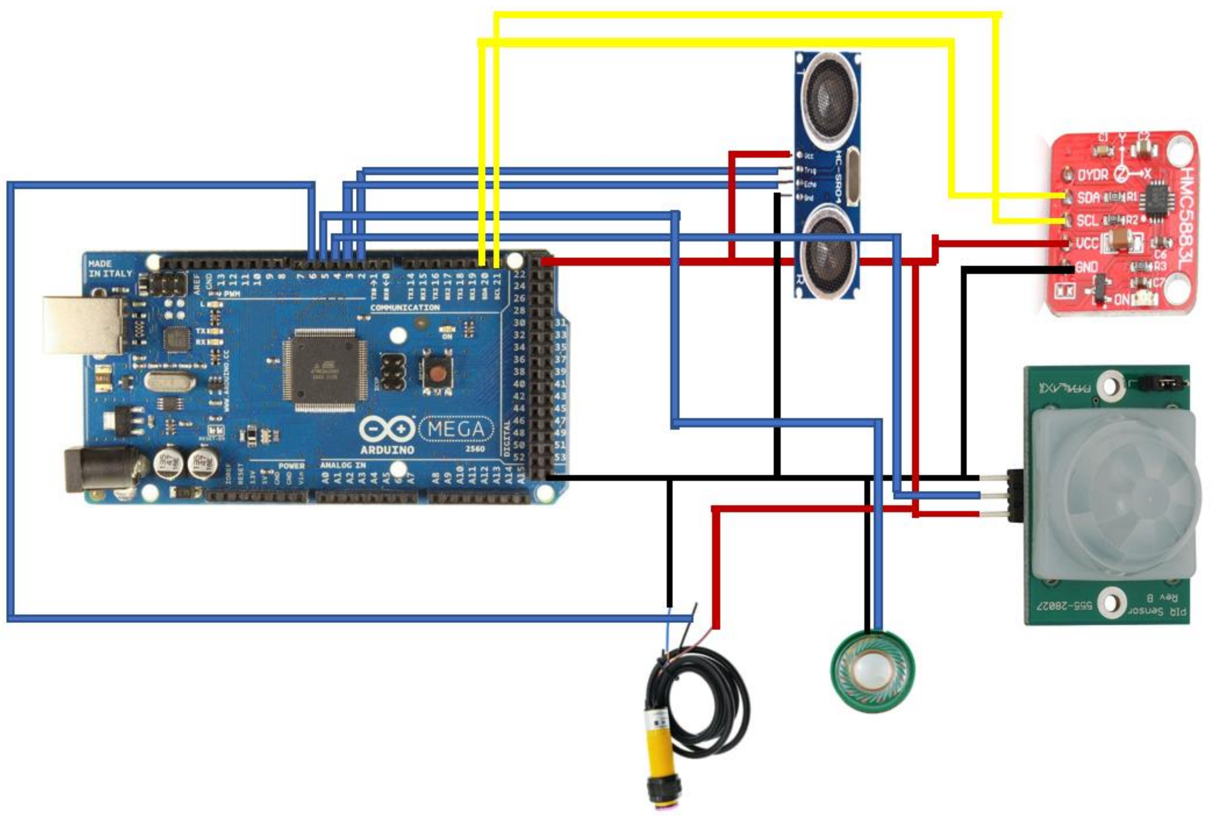
December 10th, 2019

Lab Days: Tuesday 11AM-2PM

Program Flow Chart


Run Timer (resets when Arduino unpowered)
Collect and evaluate all sensor data in loop
Calculate speed (rate of change in distance measured by ultrasonic sensor) every second
Speed>50 then play unique audio queue
Collect x,y and z coordinates from magnetometer (compass)
Check if user changed direction, if so, play unique cardinal audio queue
Collect status from Proximity Sensor
If high, play unique audio queue
Collect status from PIR Sensor
If high, play unique audio queue

Wiring Diagram



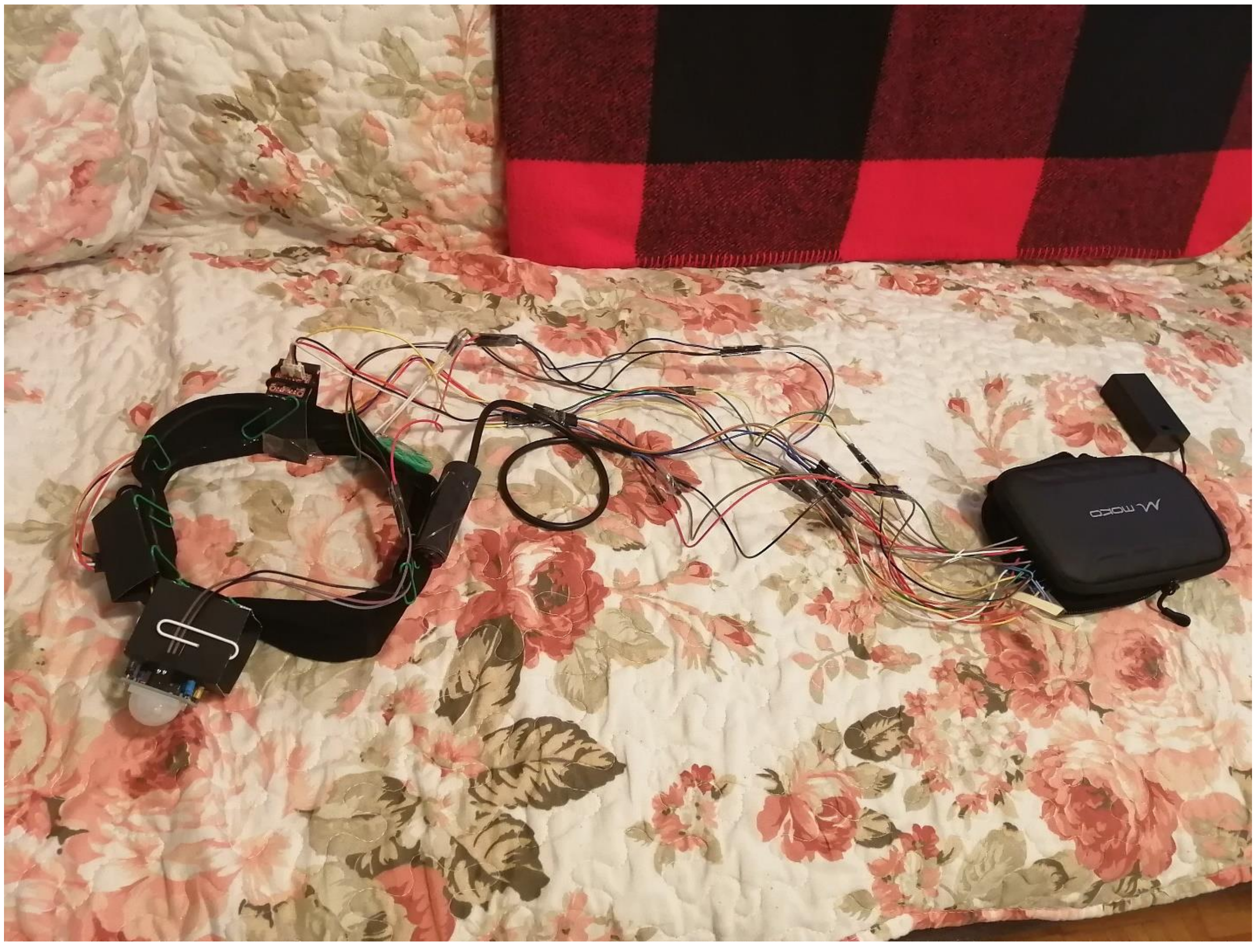
Description

Through the use of a magnetometer, an ultrasonic sensor, PIR, proximity sensor, as well as a speaker- all of which are to be structured on a headband or crown-like structure, this creation is intended to serve as a substitute for the walking stick used by the visually impaired. The user is provided with auditory signals based on the information gathered from the four sensors. The magnetometer will keep the user oriented - as would a compass. The ultrasonic sensor informs the user of low-level objects that stand in their way and serve as a tripping hazard. The PIR motion detector detects if any people are walking in the user's path. The proximity sensor detects any wall that the user is too close to. Each piece of information is signaled with unique audio queues. The North, East, South, and West directions are signaled with one, two, three, and four beeps respectively (of rising tones) through the speaker. The Ultrasonic, Proximity, and PIR sensors each have their unique two-tone beeps played through the speaker.

**code is at the end of this document

Pictures

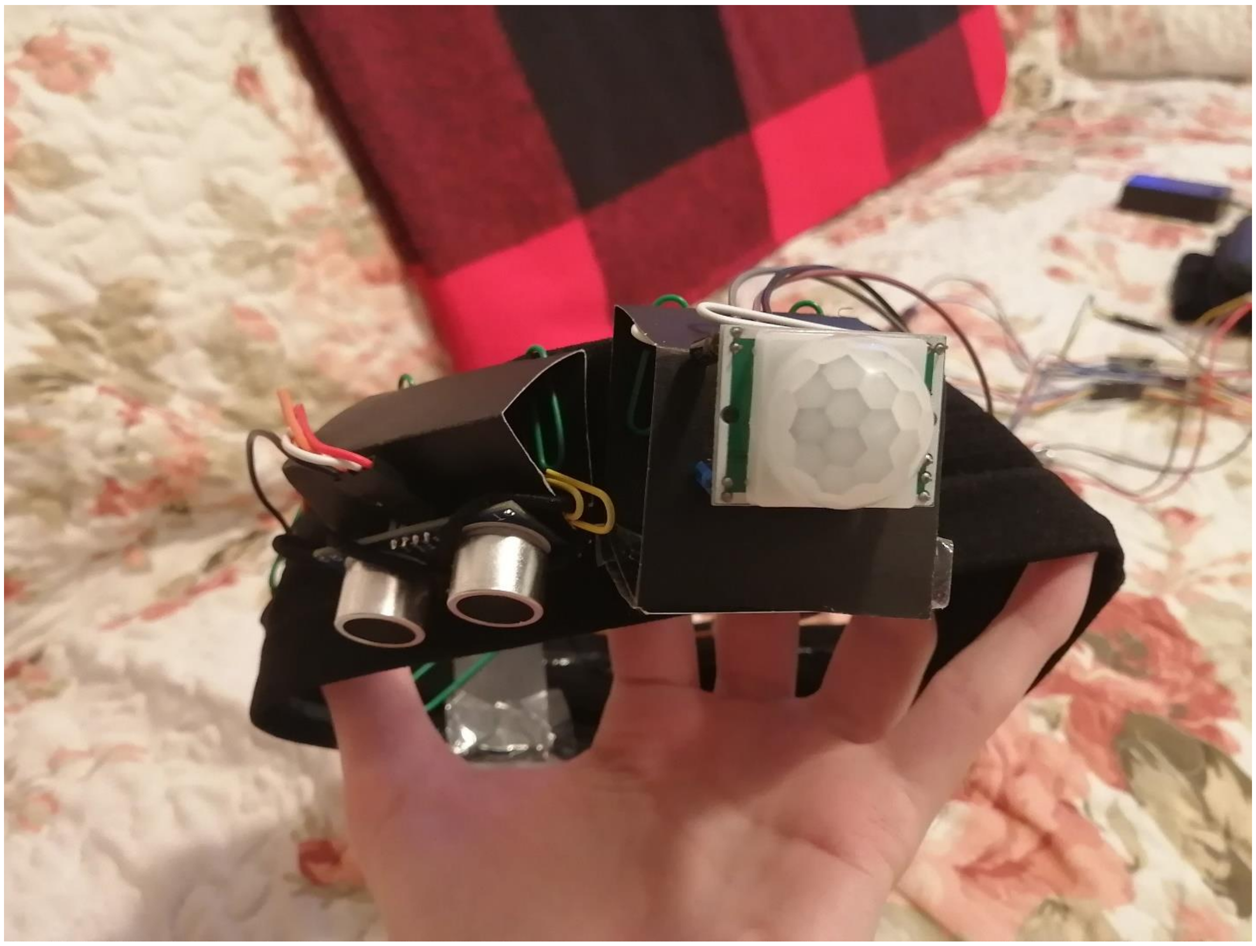
Full View (Armband on the right is to contain the Arduino, battery pack, and mini breadboard. On the left is the headband with aforementioned sensors attached to it as well as a speaker.)



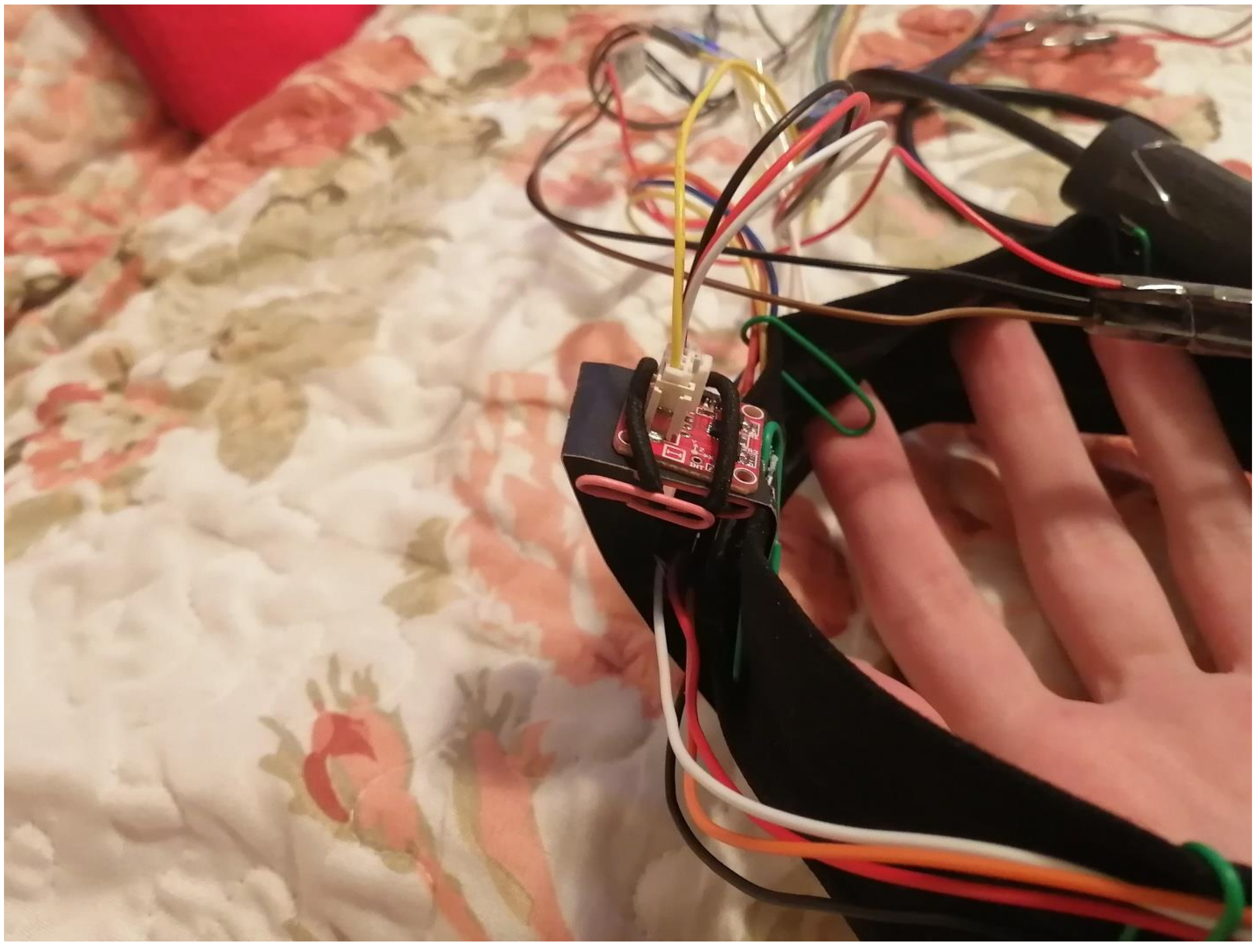
**note, the battery pack is kept inside the armband shown on the right along with the Arduino, only left out for visual clarity of its presence

**note, this is the only featured picture with the green speaker visible

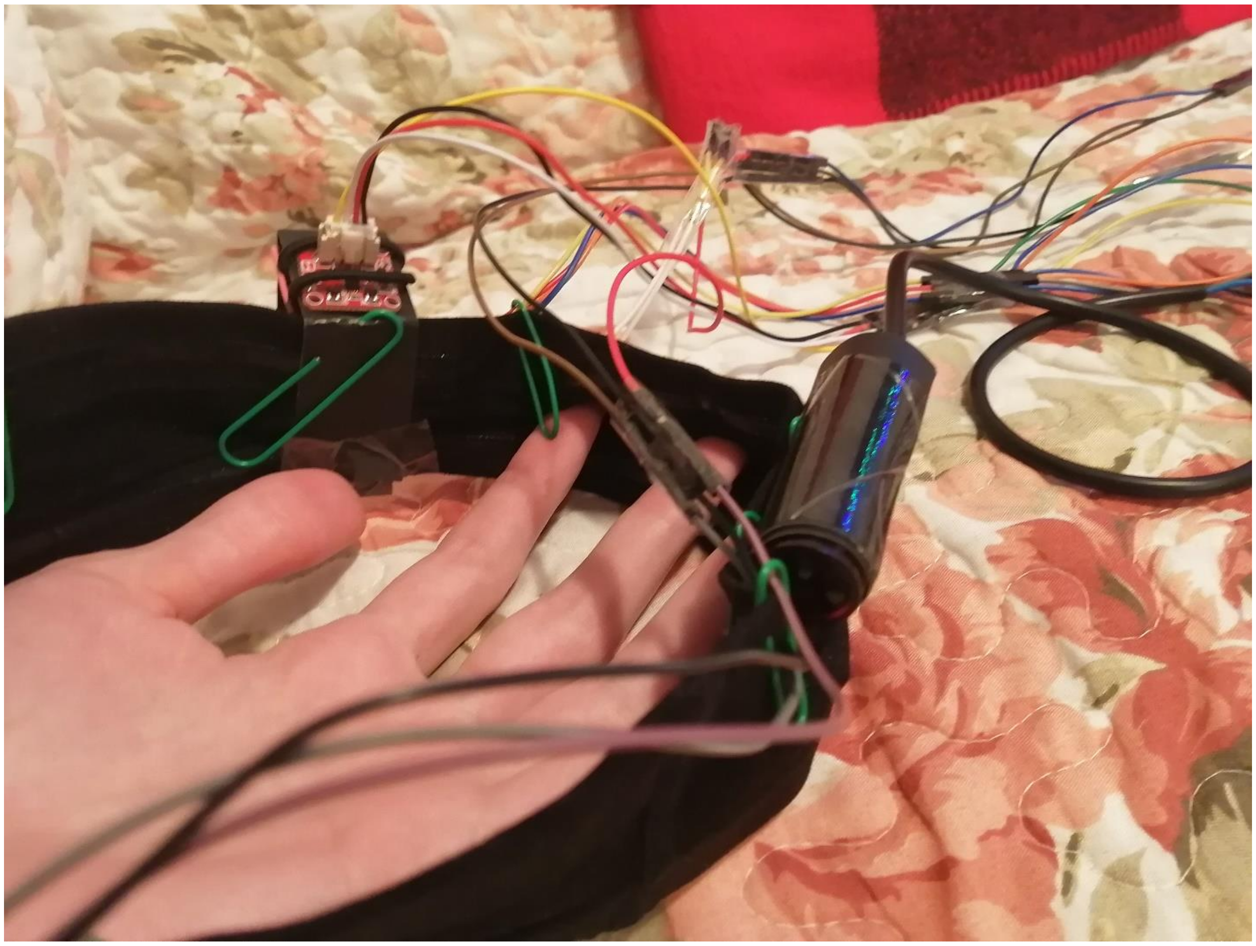
Front View (featuring Ultrasonic Sensor angled to check for obstacles afoot, as well as PIR motion sensor to check for people in the way)



Right Side (Featuring Magnetometer compass held flat to ensure functionality, right behind is where the green speaker is located (not visible in photo))



Left Side (Featuring proximity sensor used to detect when the user is close to a wall)



Code

```
/******************************************
 * Name: George Dobric                     *
 * Student# 101227367                      *
 * Date: December 10th, 2019               *
 * Lab Days: Tuesday 11AM-2PM              *
 * Description: Crown with Magnetometer,   *
 *  Ultrasonic, PIR, Proximity sensors and*
 *  speaker used to guide the Blind        *
 ******************************************/


// notes in the melody:
int melody[] = { //frequencies to be played for PIR audio signal
  100, 150
};
int melody2[] = { //frequencies to be played for cardinal audio signals
  40, 50, 60, 70
};
int melody3[] = { //frequencies to be played for PIR audio signal
  200, 150
};
int melody4[] = { //frequencies to be played for PIR audio signal
  200, 250
};
// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {
  4, 4
};
int noteDurations2[] = {
  4, 4, 4, 4
};
#include <Wire.h>
#define address 0x1E
unsigned long time;
int trigger_pin = 2;
int echo_pin = 3;
int pirPin = 4;
int speaker = 5;
int proximity = 6;
int time2;
int distance;
boolean north, south, east, west;
float speed;
long t1;
long t2;
int d1;
int d2;
int state = 0;
long *tt;
int *dd;
int pirr = 0;
int proxx = 0;
int spkr = 0;
int spkr2 = 0;
int spkr3 = 0;
String direction = "neither";
boolean firstRun = true;
boolean change = false;
int changeNoteCount;
void setup() {
  Serial.begin(9600);
  Wire.begin();
  //Init_HMC5883L();
  Wire.beginTransmission(address); //open communication with HMC5883
  Wire.write(0x02); //select mode register
  Wire.write(0x00); //continuous measurement mode
  Wire.endTransmission();
  pinMode (trigger_pin, OUTPUT);
  pinMode (echo_pin, INPUT);
  pinMode (pirPin, INPUT);
  pinMode (speaker, OUTPUT);
  pinMode (proximity, INPUT);

}
void loop() {

  //Serial.print("Time: ");
  time = millis();

  //All code bellow is for Reading and Processing the PIR input signal
  if (digitalRead(pirPin) == HIGH) {
    pirr += 1; //variable used to confirm that the PIR is high, used for PIR
audio signal
  }
  if (digitalRead(pirPin) == LOW) {
    pirr = 0;  //variable used to confirm PIR is low, once again for PIR
audio signal
    spkr = 0;  //variable used to control the audio signal loops
  }
  if (pirr == 1 && spkr == 0) { //if PIR is high and spkr is zero, play the
PIR audio signal
    for (int thisNote = 0; thisNote < 2; thisNote++) {

      // to calculate the note duration, take one second divided by the note
type.
      //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
      int noteDuration = 1000 / noteDurations[thisNote];
      tone(5, melody[thisNote], noteDuration);

      // to distinguish the notes, set a minimum time between them.
      // the note's duration + 30% seems to work well:
      int pauseBetweenNotes = noteDuration; //* 1.30;
      delay(pauseBetweenNotes);
      // stop the tone playing:
      noTone(5);
    }
    spkr += 1;
  }
  //PIR signal processing ends
  //All code bellow is for Reading and Processing the Proximity input signal
  if (digitalRead(proximity) == HIGH) {
    proxx += 1; //variable used to confirm that the PIR is high, used for PIR
audio signal
  }
  if (digitalRead(proximity) == LOW) {
    proxx = 0;  //variable used to confirm PIR is low, once again for PIR
audio signal
    spkr3 = 0;  //variable used to control the audio signal loops
  }
  if (proxx == 1 && spkr3 == 0) { //if PIR is high and spkr is zero, play the
PIR audio signal
    for (int thisNote = 0; thisNote < 2; thisNote++) {

      // to calculate the note duration, take one second divided by the note
type.
      //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
      int noteDuration = 1000 / noteDurations[thisNote];
      tone(5, melody4[thisNote], noteDuration);

      // to distinguish the notes, set a minimum time between them.
      // the note's duration + 30% seems to work well:
      int pauseBetweenNotes = noteDuration; //* 1.30;
      delay(pauseBetweenNotes);
      // stop the tone playing:
      noTone(5);
    }
    spkr3 += 1;
  }
  //Proximity signal processing ends

  //Ultrasonic signal processing through "speed" variable - detects if object
is afoot - plays unique audiotory signal
  if (abs(speed)<=50){
    spkr2=0;
  }
  if (abs(speed)>50 && spkr2 == 0) { //if PIR is high and there is no audio
signal conflict, play the PIR audio signal
    for (int thisNote = 0; thisNote < 2; thisNote++) {

      // to calculate the note duration, take one second divided by the note
type.
      //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
      int noteDuration = 1000 / noteDurations[thisNote];
      tone(5, melody3[thisNote], noteDuration);

      // to distinguish the notes, set a minimum time between them.
      // the note's duration + 30% seems to work well:
      int pauseBetweenNotes = noteDuration; //* 1.30;
      delay(pauseBetweenNotes);
      // stop the tone playing:
      noTone(5);
    }
    spkr2 += 1;
  }
  //Ultrasonic signal processing ends
  digitalWrite (trigger_pin, HIGH);
  delayMicroseconds (10);
  digitalWrite (trigger_pin, LOW);
  time2 = pulseIn (echo_pin, HIGH);
  distance = (time2 * 0.034) / 2;
  Serial.print (" \t\t\tDistance= ");
  Serial.println (distance);
  if (state == 0) { //state is used to be alternated after every loop cycle
in order to calculate distance
    tt = &time;
    dd = &distance;
    d1 = *dd;
    t1 = millis();//*tt;
  }
  else if (state == 1) { //state is used to be alternated after every loop
cycle in order to calculate distance
    tt = &time;
    dd = &distance;
    d2 = *dd;
    t2 = millis();//*tt;
    speed = ((d2 - d1) * 1.0 / (t2 - t1)) * 1000;
    Serial.print (" SPEED= ");
    Serial.println(speed);
  }

  if (state == 0) { //state is used to be alternated after every loop cycle
in order to calculate distance
    state = 1;
  }
  else if (state == 1) { //state is used to be alternated after every loop
cycle in order to calculate distance
    state = 0;
  }
  int x, y, z; //triple axis data

  //Tell the HMC5883L where to begin reading data
  Wire.beginTransmission(address);
  Wire.write(0x03); //select register 3, X MSB register
  Wire.endTransmission();


  //Read data from each axis, 2 registers per axis
  Wire.requestFrom(address, 6);
  if (6 <= Wire.available()) {
    x = Wire.read() << 8; //X msb
    x |= Wire.read(); //X lsb
    z = Wire.read() << 8; //Z msb
    z |= Wire.read(); //Z lsb
    y = Wire.read() << 8; //Y msb
    y |= Wire.read(); //Y lsb
  }

  if (80 < x && 75 < y && y < 235 ) { //check coordinates, if true set north
to true
    north = true;
    south = false;
    east = false;
    west = false;
    changeNoteCount = 1; //ensure 1 beep is signaled in later code
  }
  else if (-180 < x && x < 30 && y < 100) { //check coordinates, if true set
west to true
    north = false;
    south = false;
    east = false;
    west = true;
    changeNoteCount = 4; //ensure 4 beeps are signaled in later code
  }
  else if (-100 < x && x < 100 && y > 110) { //check coordinates, if true set
east to true
    north = false;
    south = false;
    east = true;
    west = false;
    changeNoteCount = 2; //ensure 2 beeps are signaled in later code
  }
  else if (x < -180 && x < -120 && y > 135) { //check coordinates, if true
set south to true
    north = false;
    south = true;
    east = false;
    west = false;
    changeNoteCount = 3; //ensure 3 beeps are signaled in later code
  }
  if (firstRun == false) { //below if checks if direction was changed and set
boolean to true if it was changed
    if ((north == true && direction == "north") || (south == true &&
direction == "south") || (east == true && direction == "east") || (west ==
true && direction == "west"))
      change = false;
    else
      change = true;
  }

  //DIRECTIONAL AUDIO SIGNALS BELOW
  if (change == true) {
    for (int thisNote = 0; thisNote < changeNoteCount; thisNote++) {

      // to calculate the note duration, take one second divided by the note
type.
      //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
      int noteDuration2 = 1000 / noteDurations2[thisNote];
      tone(5, melody2[thisNote], noteDuration2);

      // to distinguish the notes, set a minimum time between them.
      // the note's duration + 30% seems to work well:
      int pauseBetweenNotes2 = noteDuration2; //* 1.30;
      delay(pauseBetweenNotes2);
      // stop the tone playing:
      noTone(5);
    }
    change = false;
  }
  //DIRECTIONAL AUDIO SIGNALS END


  if (north == true) //series of ifs is part of process which checks whether
or not direction was changed
    direction = "north";
  else if (south == true)
    direction = "south";
  else if (east == true)
    direction = "east";
  else if (west == true)
    direction = "west";

  //Print out values of each axis
  Serial.print("N: ");
  Serial.println(north);
  Serial.print("S: ");
  Serial.println(south);
  Serial.print("E: ");
  Serial.println(east);
  Serial.print("W: ");
  Serial.println(west);
  Serial.print("x: ");
  Serial.print(x);
  Serial.print("  y: ");
  Serial.print(y);
  Serial.print("  z: ");
  Serial.println(z);

  Serial.print(" \t\t\t\t\t time: ");
  Serial.println(time); //prints time since program started
  delay(1000);
  firstRun = false;
}
```