Part A. Writing to the Serial Monitor

a. Based on the readings from the serial monitor, what is the range of the analog values being read?
0 to 1023

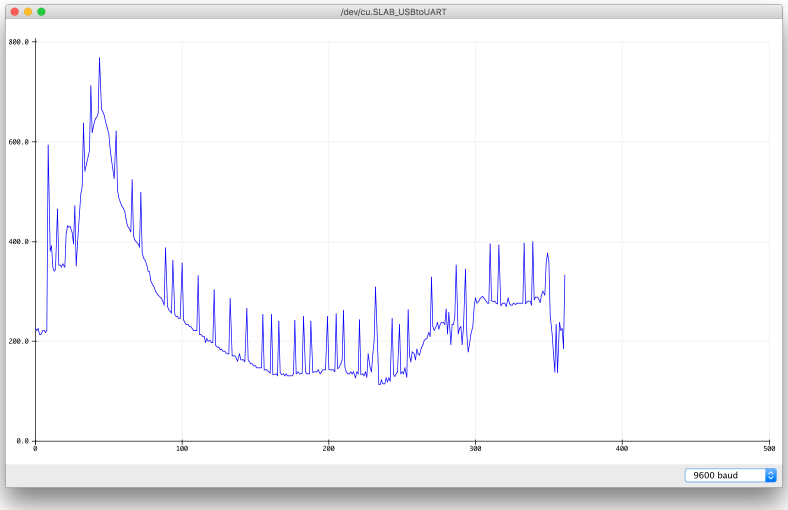
b. How many bits of resolution does the analog to digital converter (ADC) on the Arduino have
The analog to digital converter (ADC) on the Arduino has 10 bits of resolution and 10 bits were used with the range of values I was seeing.

Part B. Voltage Varying Sensors

1. IR Distance Sensor

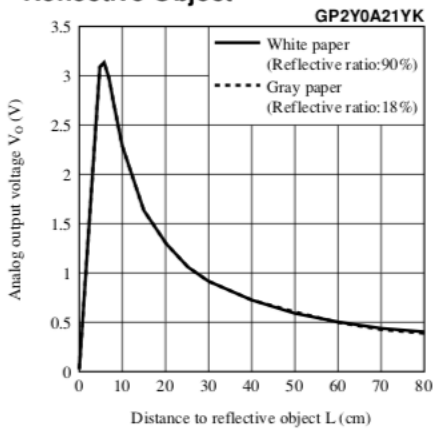
a. Describe the voltage change over the sensing range of the sensor. A sketch of voltage vs. distance would work also. Does it match up with what you expect from the datasheet?

When the distance was almost 0, the reading was around 300, which was equivalent to 1.47V. As the distance approached 10cm, the reading rapidly increased to 650, which was roughly 3.18V. As the distance continued to increase, the reading started to drop and reached 170, which was 0.83V, when the distance was around 50cm. After that, the reading started to increase again and flatted out around 270, which was 1.32V when the field of view is clear.

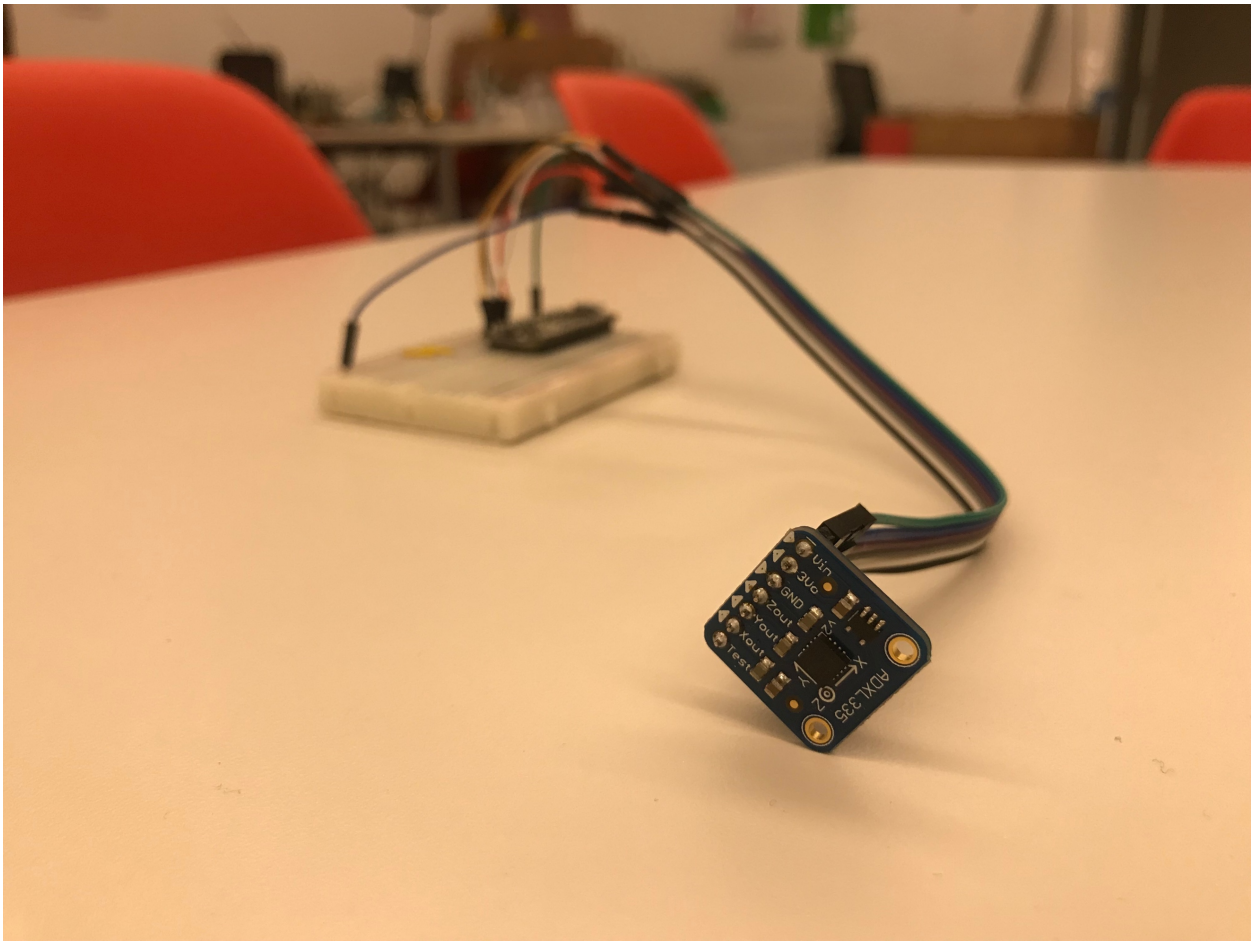


This matches the datasheet pretty well except for the tail part where the voltage starts to increase a little bit instead of decreasing.

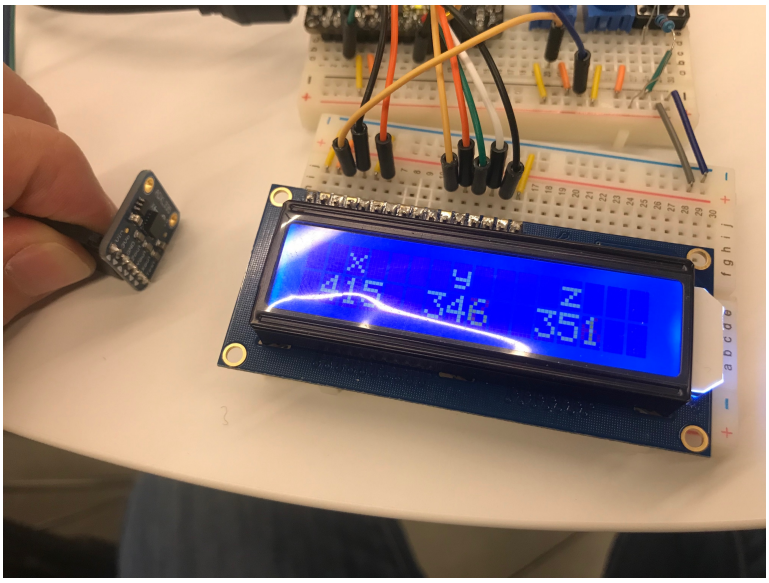
Fig.5 Analog Output Voltage vs. Distance to Reflective Object

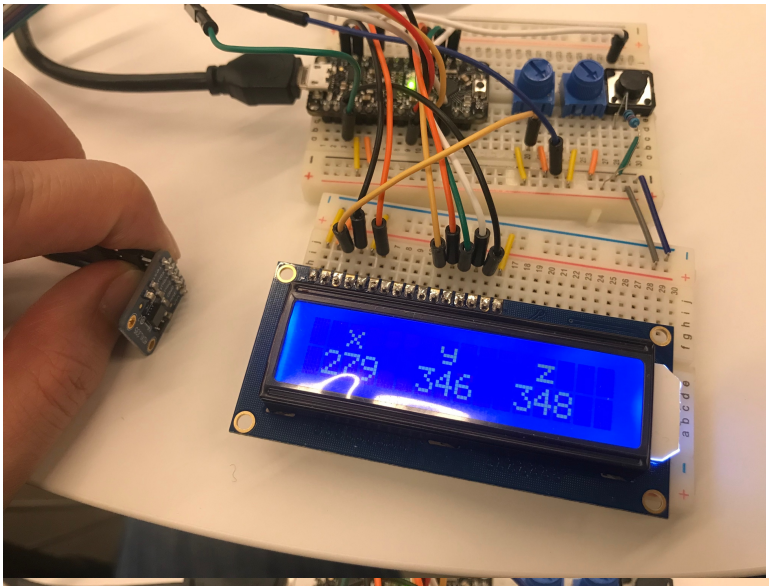


2. Accelerometer

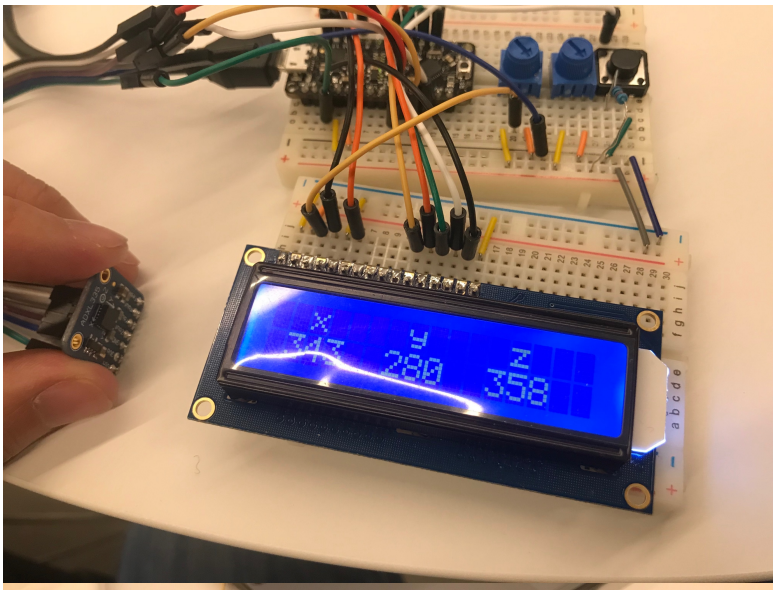


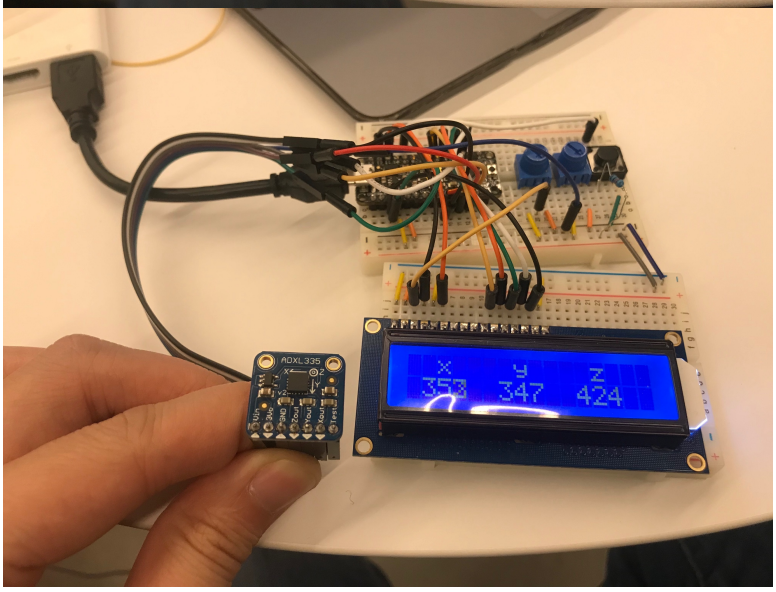
The accelerometer measures the magnitudes and directions of forces applied along the x, y, and z axes, respectively. As can be seen in the following pictures, the readings range from 279 to 424. Details are shown in Table 1.

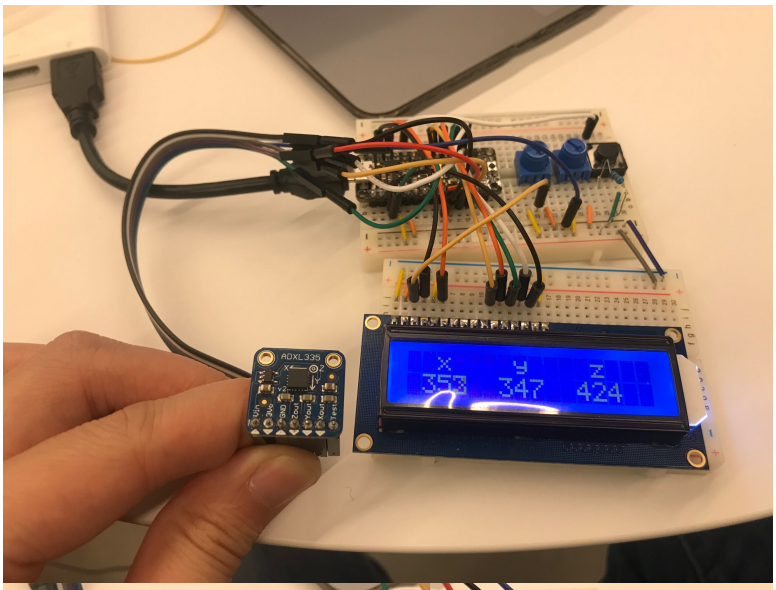


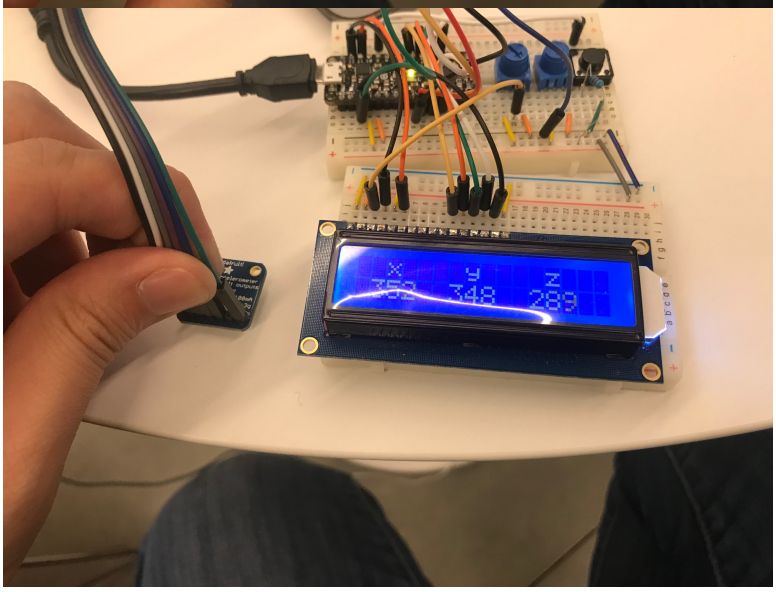

x
279
y
346
z
348


y
416
z
356


x
y
z
280
358

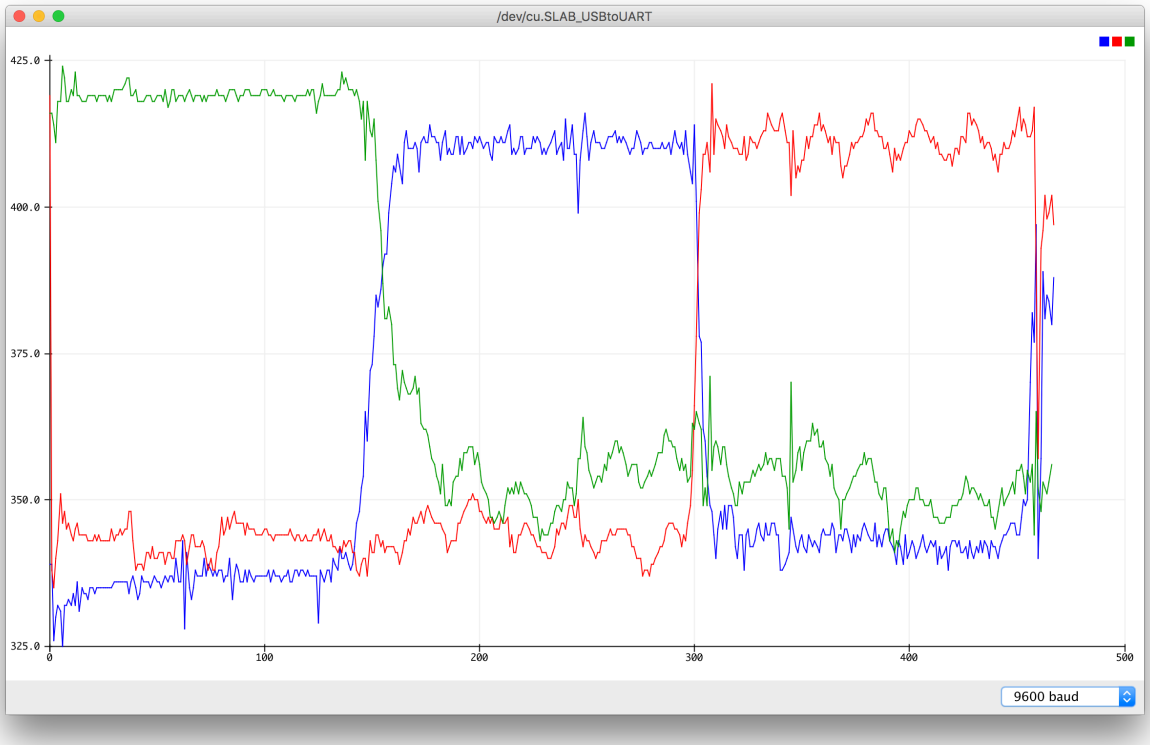

x
y
z
353
347
424
ADXL335





	X	Y	z
Up	415	416	424
Horizontal	~350	~347	~350
Down	279	280	289

Table 1. Readings of accelerometer.



Green: Z, Red: Y, Blue: X

a. Include your accelerometer read-out code in your write-up.

```
// these constants describe the pins. They won't change:
//const int groundpin = 18;             // analog input pin 4 -- ground
//const int powerpin = 19;              // analog input pin 5 -- voltage
const int xpin = A3;                  // x-axis of the accelerometer
const int ypin = A2;                  // y-axis
const int zpin = A1;                  // z-axis (only on 3-axis models)

int x = 0;
int y = 0;
int z = 0;


// include the library code:
#include <LiquidCrystal.h>

// initialize the library by associating any needed LCD interface pin
// with the arduino pin number it is connected to
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);


void setup() {
  // initialize the serial communications:
  Serial.begin(9600);
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);

  // Provide ground and power by using the analog inputs as normal digital pins.
  // This makes it possible to directly connect the breakout board to the
  // Arduino. If you use the normal 5V and GND pins on the Arduino,
  // you can remove these lines.
  //pinMode(groundpin, OUTPUT);
```

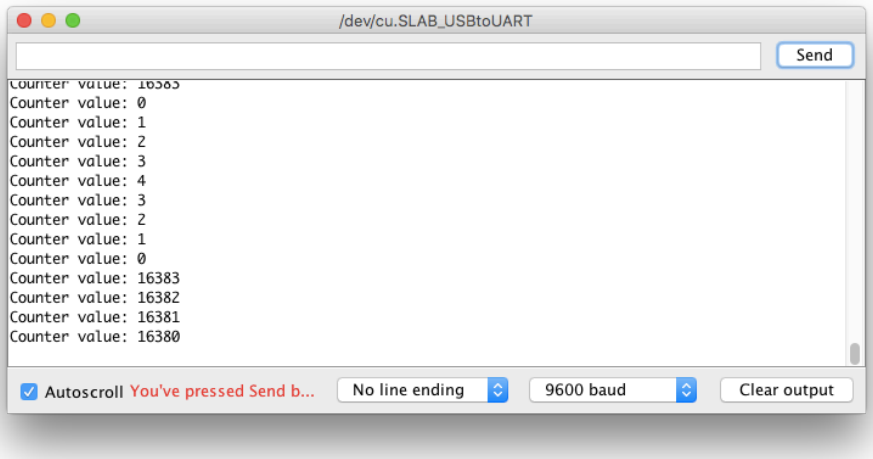
```
  //pinMode(powerpin, OUTPUT);
  //digitalWrite(groundpin, LOW);
  //digitalWrite(powerpin, HIGH);
}

void loop() {
  // print the sensor values:
  x = analogRead(xpin);
  Serial.print(x);
  // print a tab between values:
  y = analogRead(ypin);
  Serial.print("\t");
  Serial.print(y);
  // print a tab between values:
  z = analogRead(zpin);
  Serial.print("\t");
  Serial.print(z);
  Serial.println();
  lcd.clear();
  lcd.setCursor(2,0);
  lcd.print('x');
  lcd.setCursor(7,0);
  lcd.print('y');
  lcd.setCursor(12,0);
  lcd.print('z');
  lcd.setCursor(1,1);
  lcd.print(x);
  lcd.setCursor(6,1);
  lcd.print(y);
  lcd.setCursor(11,1);
  lcd.print(z);

  // delay before next reading:
  delay(100);
}
```

Part C. Count/Time-Based Sensors

1. Rotary Encoder

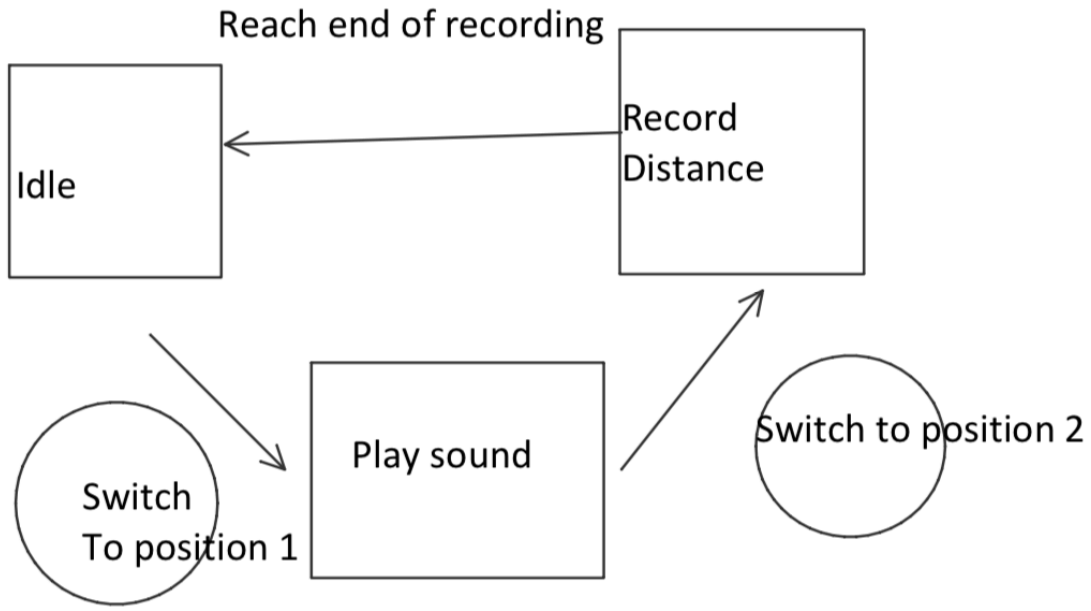


During the experiment, we observed that by turning the knob one step at a time, we can increase or decrease the counter by one. In addition, the counter can count from 0 to 16383, which has 14

bits. The difference between rotary encoder and a potentiometer is that while potentiometer registers the position of the know, meaning any position of the know corresponds to a particular resistance, one knob position of a rotary encoder doesn't differentiate from another. The rotary encoder only registers the direction and number of turnings.

Part D. Logging values to the EEPROM and reading them back

1. Design your logger



2. Reading and writing values to the Arduino EEPROM

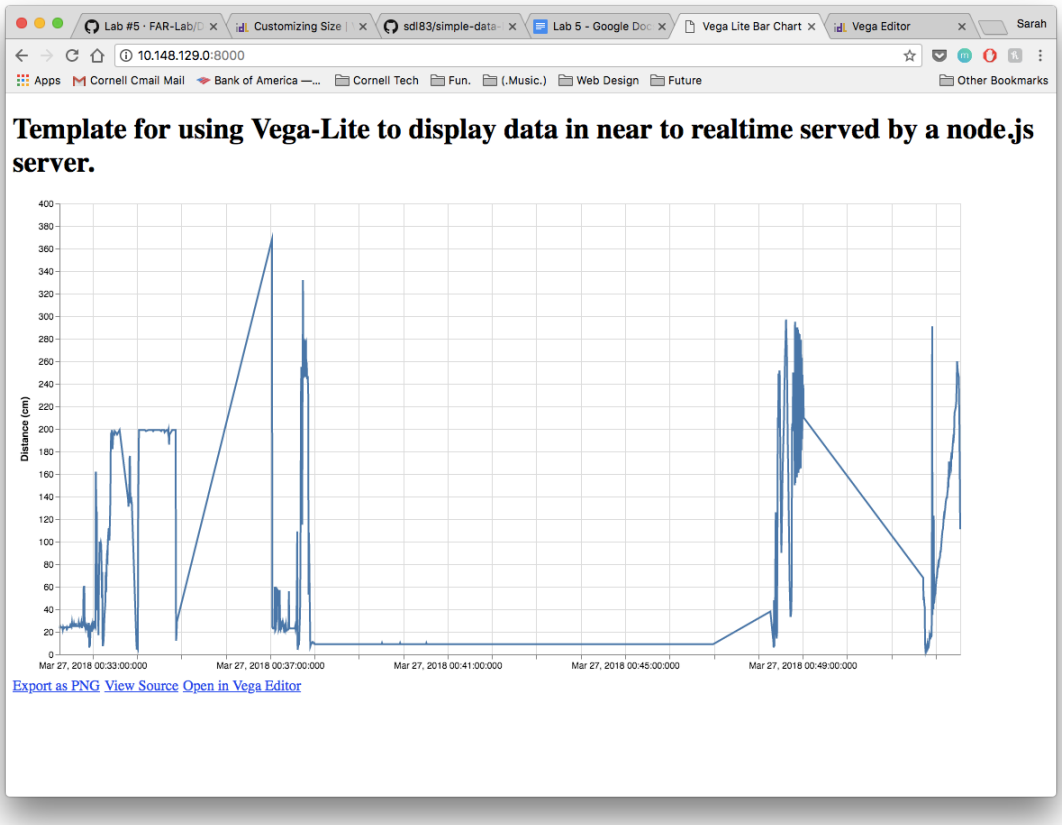
a. How many byte-sized data samples can you store on the Atmega328?
1024 bytes

b. How would you get your analog data from the ADC to be byte-sized?
Dividing analog data from the ADC by 4 using integer division since data from the ADC ranges from 0 to 1023 and one byte contains 256. Alternatively, we can use four bytes to represent data from the ADC so that no precision is lost.

3. Reading and writing values to the Raspberry Pi

```
var timeout=null;  //timeroutfunction
var vlSpec ={
 "$schema": "https://vega.github.io/schema/vega-lite/v2.json",
 "data": { "name": "table","format": {
      "parse": {
        "date": "utc:'%Q'"
      }
    }
  },
 "width": 1000,
  "height":500,
  "mark": "line",
  "encoding": {
    "x": {
      "timeUnit":"utcyearmonthdatehoursminutessecondsmilliseconds",
      "field": "date",
      "type": "temporal",
      "axis": {"title": "Time"}
    },
    "y": {
      "field": "value",
      "type": "quantitative",
      "axis": {"title": "Distance (cm)"}
    }
  }
  };


  function reRender(res,newEntries){
     console.log('Starting to (re)render the visualization');
     var changeSet = vega.changeset().insert(newEntries); // generate change-set
     newEntries.length=0; //empty array
     res.view.change('table', changeSet).run(); // re-render graphic
     timeout=null; // set pointer to time out function to 0
  }
  vegaEmbed("#vis", vlSpec).then(function(res) {// create graphic
    var socket = io(); //connect to websocket server
    var newEntries=[];
    socket.on('new-line', function(line) { // on new data lines
      if(timeout!=null){  //halt current render process
       clearTimeout(timeout);
     }
       var newEntry= {'date':line.split(',')[0],'value':line.split(',')[1]};
       newEntries.push(newEntry); // add new data entry
       timeout = setTimeout(reRender, 100,res,newEntries); // launch new data render
    });
  });
```



4. Create your data logger!

We created a data logger that utilized an ultrasonic distance sensor to record the distances and stored the data in the Arduino EEPROM in the recording mode. When it was in the playback mode, it read the data from EEPROM and map them into audible frequencies and played with a speaker. (We found how to use the ultrasonic sensor at http://www.instructables.com/id/Simple-Arduino-and-HC-SR04-Example)
One issue we kept running into was that very often, the logger entered into playback mode prematurely. It took us a long time before we realized that we needed to actually ground the button pin instead of letting it float when we wanted the button pin in LOW state.

```
#define trigPin 12
#define echoPin 13

#include <EEPROM.h>

long duration, distance;
int distanceLow = 0;
int distanceHigh = 400;

const int buttonPin = 9;
int buttonState = 0;

unsigned long targetTime=0;
const unsigned long intervall=200;

int addr = 0;
int first_adr = 0;
int address;
int pitch;

void setup() {

  // put your setup code here, to run once:
  Serial.begin (9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buttonPin, INPUT);

}
```

```
void loop() {
  // put your main code here, to run repeatedly:

 if (digitalRead(buttonPin) == LOW) {
  // record distances
  distance = measure();

  if (distance >= 0 && distance <= 200) {
    pitch = map(distance, distanceLow, distanceHigh, 200, 4000);
    tone(8, pitch, 200);

  // put it in the mem.
    EEPROM.write(addr, distance);

    addr = addr + 1;
    if (addr == EEPROM.length()) {
      addr = 0;
      Serial.println("Reset");
    }
  }

 } else {
  // play sound
  Serial.println("replay");
  address = first_adr;
  while (address != addr) {
    distance = EEPROM.read(address);
    pitch = map(distance, distanceLow, distanceHigh, 200, 4000);
    tone(8, pitch, 200);
    delay(201);
    address = address + 1;
    if (address == EEPROM.length()) {
      Serial.println("Reset in replay");
      address = 0;
    }
  }

  // clear mem.
  first_adr = address;
 }

  // wait for a moment
  delay(200);
}

long measure() {
  digitalWrite(trigPin, LOW);  // Added this line
  delayMicroseconds(2); // Added this line
  digitalWrite(trigPin, HIGH);
  //  delayMicroseconds(1000); - Removed this line
  delayMicroseconds(10); // Added this line
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  //Serial.println(duration);
  return (duration/2)/29.1;
}
```