Arduino Uno Oscilloscope

Caleb Powell

November 16, 2022

Contents

Abstract

An analysis of Arduino Uno applications through the construction of an Oscilloscope using the C programming language and Amplifier Circuits.

1 Introduction

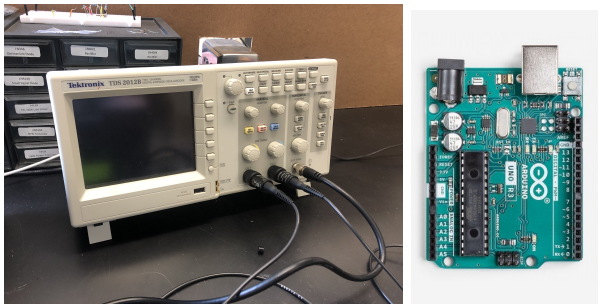

Figure 1: An Image of the Oscilloscope and Arduino Used in this Project.

1.1 Oscilloscope

An oscilloscope is a very important device used for the construction and analysis of electrical circuits. This device allows you to see how voltage changes over time by displaying a waveform of electronic signals. On the display, the voltage is represented on the vertical axis and time on the horizontal axis. These instruments allow you to determine if the behavior of your circuits is working correctly. While these instruments are very useful, they can be very expensive (*around $1,000*) and not everyone can afford to buy one. To solve this problem and get a deeper understanding of the functions of an oscilloscope, we will attempt to build our own using roughly $30 worth of material.

1.2 Arduino Uno

The Arduino Uno is a micro controller board with 14 digital input/output pins, 6 analog inputs, a USB connection, and a power jack. These micro controllers are very easy to program using the C programming language and free Arduino IDE. These boards can be used for a countless number of projects and are relatively cheap costing around $25. For this use case, we will be using one of the six analog inputs to measure varying voltage over time. We will then display this information on an OLED 1.3 inch LCD 128x64 pixel screen which costs around $5.

2 Setup

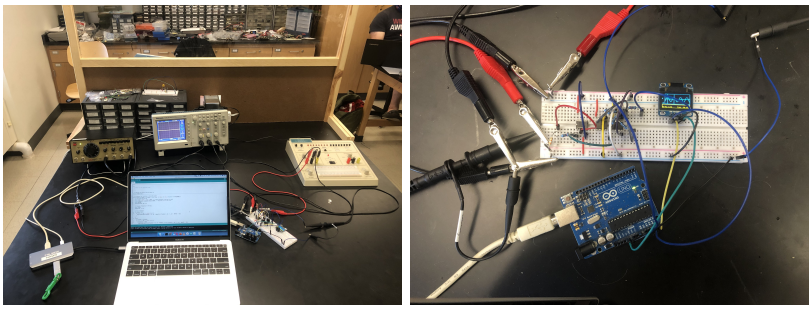

Figure 2: A Picture of the Lab Setup and Circuit.

2.1 Limitations

The idea of this process is very simple. However, because we are using the Arduino, we have limitations we must find solutions to. The analog input on the Arduino Uno will only measure positive voltages between the values 0-5 V, however, in order to create an effective oscilloscope, the input should be able to have values between ±10 V.

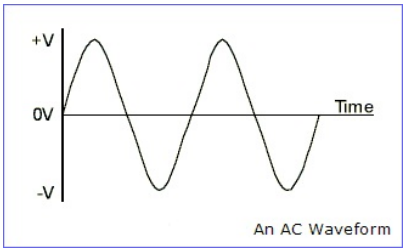

Figure 3: Created by https://www.upsbatterycenter.com/.

For example, the input wave form in Figure 3 has both positive and negative voltage values that might be greater than ±5 V. So how do we get around this?

2.2 Circuit

A very simple answer to this question is to scale the input wave form by a factor of 1/4, and then add 2.5 V to the input. If the input wave is ±10 V, we can scale it by a factor of 1/4 so that it is now ±2.5 V. If we then add 2.5 V to that, we are now reading a ±10 V input wave of as 0-5 V.

2.2.1 Inverting Amplifier

In order to scale the input wave by a factor of 1/4 we use an Inverting amplifier. An inverting amplifier circuit consists of an operational amplifier and two resistors.

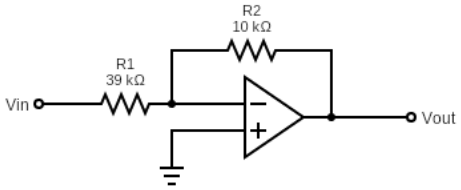

Figure 4: Inverting Amplifier Circuit used in this Project.

Because there is negative feedback, you know the output will attempt to make the difference between V_+ and V_- zero. This means $V_+ = V_- = 0$ since V_+ is grounded. Using Ohm's law, you find I_1 and I_2 to be:

$$I_1 = \frac{V_{in} - V_-}{R_1} = \frac{V_{in}}{R_1} \tag{1}$$

$$I_2 = \frac{V_{out} - V_-}{R_1} = \frac{V_{out}}{R_2} \tag{2}$$

So the ratio between the output and the input is:

$$Gain = \frac{V_{out}}{V_{in}} = -\frac{R_2}{R_1} = -\frac{10k\Omega}{39k\Omega} = -\frac{1}{3.9}. \tag{3}$$

2.2.2 Summing Amplifier

Now we wish to add 2.5V to the input. To do this, we must use what is called a summing amplifier circuit.

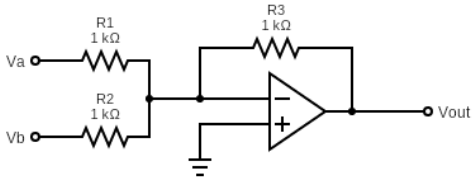

Figure 5: Summing Amplifier Circuit used in this Project.

Again, there is negative feedback, so $V_+ = V_- = 0$. Therefore, the two inputs provide a total current of

$$I = I_1 + I_2 = \frac{V_A}{R_1} + \frac{V_B}{R_2} \tag{4}$$

which means we have an output voltage of:

$$V_{out} = -R_3(\frac{V_A}{R_1} + \frac{V_B}{R_2}) = -(V_A + V_B). \tag{5}$$

Because all of the resistor are of the same value, we now have an output voltage equal to the negative sum of the two inputs.

2.2.3 Final Circuit

We can then combine these two types of circuits together to get the following:

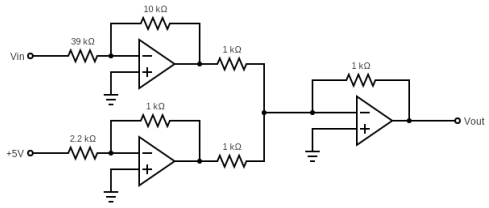

Figure 6: Circuit to Correct the Input Signal.

Notice here we have added a second inverting amplifier on the bottom left with a fixed input voltage of +5 V and a gain of -1/2. Thus, the output voltage of the bottom left inverting amplifier should be (*by equation 3*):

$$V_B = -(5V)\frac{1k\Omega}{2.2k\Omega} = -\frac{5V}{2.2}. \tag{6}$$

Since the output of top left circuit is (*by equation 3*):

$$V_A = -\frac{V_{in}}{3.9}, \tag{7}$$

and the output of the right most circuit is (*by equation 5*):

$$V_{out} = -(V_A + V_B), \tag{8}$$

the overall output for the circuit should be:

$$\boxed{V_{out} = (\frac{V_{in}}{3.9} + \frac{5V}{2.2})} \tag{9}$$

2.3 Procedure

1. Setup the Arduino IDE on your computer.
2. Connect your Arduino Uno to your PC via the USB port.
3. Construct the Circuit in Figure 6.
4. Using a Signal Generator, input a signal into V_{in} of the circuit.
5. Power the JRC4558 Op Amp with ± 10 V from an external power supply.
6. Connect the A0 Analog Input pin on the Arduino to V_{out} on the circuit
7. Connect A4, A5, 5V Power, and Ground pins on the Arduino Uno to SDA, SCL, VCC, and GND on the OLED LCD respectively.
8. Connect the same 5V pin on the Arduino to the bottom left Inverting amplifier in figure 6.
9. Connect V_{in} to Channel one of a true Oscilloscope.
10. Connect V_{out} to Channel two of a true Oscilloscope.

Note: *Step 9 and 10 are only needed to analyze the result. You do not necessarily need an oscilloscope to construct this circuit, although it does help.*

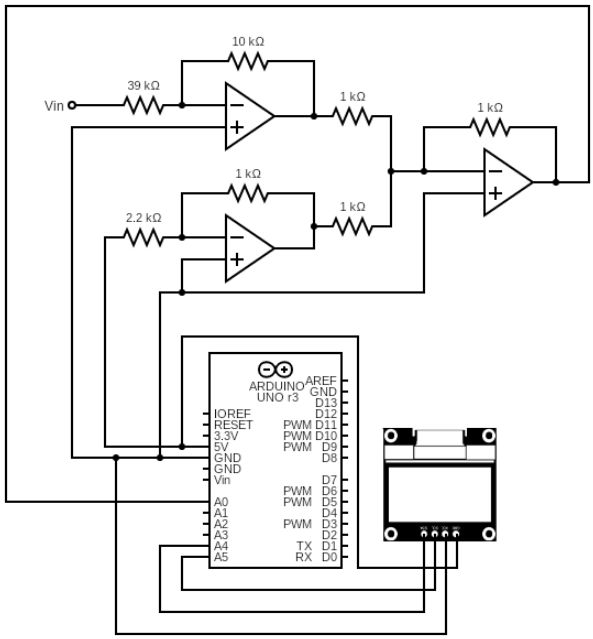

Figure 7: Full Circuit Diagram for this Project.

3 Programming

3.1 Collecting The Data

```
17  int sensorPin = A0;    // select the input pin for the potentiometer
```

The "sensorPin" variable defines the analog input pin on the arduion Uno that reads voltage values exiting the circuit.

```
212   // collect the value from the sensor:
213   unsigned int data[size];
214   float time[size];
215   float start = millis();
216
217   for (int i = 0; i < size; i++) {
218     data[i] = analogRead(sensorPin);
219     time[i] = millis() - start;
220   }
```

The voltage value is then collected in an array named "data" and the time the data is collected is placed into an array named "time". Once we have the time measured for each data point, we can very easily find the frequency of the wave.

3.2 Measuring Frequency and Amplitude

```
150 void peaks(unsigned int data[500], int* peak1, int* peak2)
151 {
152   int max = data[0];
153   int min = data[0];
154
155   for (int i = 1; i < size; i++) {
156     // Check if we are increasing
157     if (data[i] > max) {
158       max = data[i];
159       *peak1 = i;
160     }
161     // if we are not increasing
162     else if (*peak1 > 0) {
163       break;
164     }
165   }
166
167   // Find second peak
```

```
  for (int i = *peak1; i < size; i++) {
    // Check if we are increasing
    if (data[i] < min) {
      min = data[i];
      *peak2 = i;
    }
    // if we are not increasing
    else if (*peak2 > 0) {
      break;
    }
  }
  *peak2 = (*peak2 - *peak1) + *peak2;
}
```

The function "peaks" finds the index of the data array in which the first crest occurs and then the index in which the first trough occurs. In a perfect uniform wave, the distance in time between a consecutive crest and trough should be exactly half the distance in time between two consecutive crests. So, the peak function then takes the difference between the trough and crest index and multiplies it by 2 to estimate the index of the second peak. What we have left is two pointers to values named "peak1" and "peak2" respectively that define indices in the arrays for two consecutive troughs. Since we also have a time in which these values were collected (from the "time" array), we should then be able to calculate the frequency of the wave.

```
  //Find the Frequency
  int peak1 = 0;int peak2 = 0;
  peaks(data, &peak1, &peak2);
  float dt = (time[peak2] - time[peak1])/1000;
  float freq = 1 /dt;
```

These lines of code find the difference in time (in milliseconds) between when the two peaks occurred. It then divides this value by 1000 to convert the measurement of 1 ms into 1s. We then find the reciprocal of this value to get units of 1 Hz (or 1 s^{-1}). We have now calculated the frequency of the input wave and store this frequency in a float named "freq". Now we would like to measure the "Peak-to-Peak" value of the wave.

```
void amplitude(unsigned int data[500], int* max, int* min)
{
  *max = data[0];
  *min = data[0];
  // Find Real Max-Min
  for (int i = 0; i < size; i++) {
    // Check if we are increasing
```

```
189       if (data[i] > *max) {
190         *max = data[i];
191       }
192       if (data[i] < *min) {
193         *min = data[i];
194       }
195     }
196   }
```

The "amplitude" function finds the maximum and minimum voltage in the data array by comparing the value at each index. We then have pointers to the "max" and "min" values of the array. We can subtract these two from each other to find the "peak-to-peak" value.

3.3 Centering The Data

Now that we are able to find the peak-to-peak and frequency of the wave, we have to address the problem that the data is not centered.

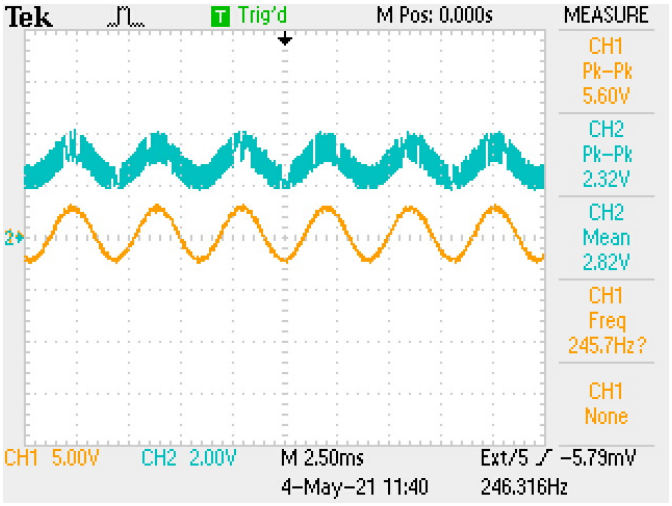

Figure 8: Oscilloscope Screenshot of Circuit Input (Orange) and Output (Blue).

The figure above shows the output from the circuit as measured by a true oscilloscope. The desired outcome was for the wave to have the troughs at 0 V. As you can see, this is not the case. However, now that we have the max and min voltage values, its very easy to correct this.

```
143   //Center The Data
144   for (int i = 0; i < size; i++) {
```

```
145     data[i] = ((data[i] - min) / 10);
146   }
147   return data;
148 }
```

The "center" function in the code works by subtracting the minimum value in the data array from each element in the array. It then divides this difference by 10 to convert the Arduino's decivolts measurement into volts. It then returns the new centered array.

3.4 Plotting The Data

Now that the data is centered appropriately we can then display some information onto the LCD screen.

```
5  // define varaiables for the plot
6  int x1 = 12;
7  int x2 = 128;
8  int y1 = 0;
9  int y2 = 48;
10 int width = x2 - x1;
11 int height = y2 - y1;
12
13 double scale = 1; // define the zoom scale
14
15 int size = width/scale;
```

The variables above are placed at the very beginning of the code and define the size of the plot in pixels. The scale value is discussed in the next section of this report, however, for now we will say that it determines the amount of data that we collect and how we plot the data.

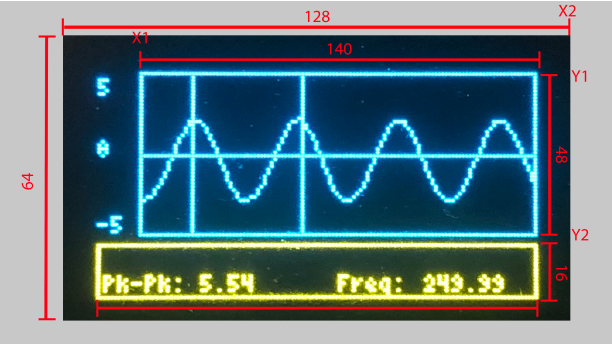

Figure 9: The Dimensions (in pixels) of the LCD Display.

```
// defines the boxes the plot goes in
void draw(void) {
  // graphic commands to redraw the complete screen should be placed here
  u8g.drawStr(0, 7, "5");
  u8g.drawStr(0, (u8g.getHeight() - 14) / 2, "0");
  u8g.drawStr(0, u8g.getHeight() - 17, "-5");

  // Top Frame Box
  u8g.drawFrame(x1, y1, width, y2);
  // Horizontal Line
  u8g.drawLine(x1, height / 2, x2, height / 2);
  // Bottom Frame Box
  u8g.drawFrame(0, u8g.getHeight() - 16, u8g.getWidth(), 16);
}

// displays the function on the LCD screen
void plot(int peak1, int peak2, float freq, int max, int min, unsigned int data[500]) {
  // Calculate the Pk-Pk
  float pkpk = (max - min) / 50.0; // finding distance from peak to peak
  float adjusted = (pkpk+0.5353)/0.9696; // adjusted to fit the error
  char pp[20];
  ftoa(adjusted, pp, 2);

  // Convert frequency to String
  char f[20];
  ftoa(freq, f, 2);

  // Plot The Wave
  if (scale >1){
    for (int i = 0; i < size; i++) {
      u8g.drawLine((i*scale) + x1, y2 - data[i], (i*scale) + x1 + 1, y2 - data[i + 1]);
    }
  }
  else{
    int factor = 1/scale;
    for (int i = 0; i < size; i++) {
      if(i%factor == 0){
        u8g.drawLine((int)(i*scale) + x1, y2 - data[i], (int) (i*scale) + x1 + 1, y2 - data[i + 1]);
      }
    }
  }
  s
  //Draw lines for period
  u8g.drawLine(x1 + (peak1*scale), 0, x1 + (peak1*scale), height);
  u8g.drawLine(x1 + (peak2*scale), 0, x1 + (peak2*scale), height);

  // Write the data
  u8g.drawStr(70, u8g.getHeight() - 2, "Freq: ");
  if (freq != INFINITY) {
    u8g.drawStr(95, u8g.getHeight() - 2, f);
  }
  u8g.drawStr(2, u8g.getHeight() - 2, "Pk-Pk: ");
  u8g.drawStr(30, u8g.getHeight() - 2, pp);
}
```

The "draw" function creates the boxes that define the area in which we will plot the graph. The "plot function" first calculates the peak-to-peak (*we will discuss the math behind this in the data analysis section of this report*). It must then convert the peak-to-peak value and the frequency into a string in order to be properly displayed onto the LCD. The function used to make this conversion and the source it was taken from can be seen in the full code. Once these have been converted into strings, we can now plot the wave and display some information about it.

The plot function works by starting at the $(x1, y2)$ pixel on the display. It then moves in the y direction according to the value placed in the first index of the data array. What we now have is a point $(x1, y2 - data[i])$ that defines a

pixel on the display representative of the voltage value collected at that time. We can then repeats this process for the next value in the array but starting at the point $(x1 + 1, y2 - data[i])$. If we repeat this process for every index in the array and then draw a line connecting the current pixel to the previous, we can very easily see a plot of the wave. Once the wave is plotted, vertical lines are drawn on the plot to show which peaks the frequency function is getting its value from. The frequency and peak-to-peak value are then displayed on the LCD beneath the graph.

3.5 Scale Function

Because the plot space on the LCD is 140 pixels wide, we can only see 140 different points in the data array. This becomes an issue when the frequency is too low or too high.

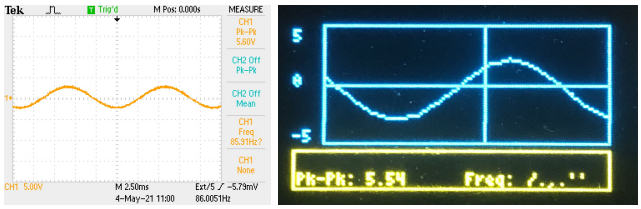

Figure 10: Low Frequency Wave Measured on a Real Oscilloscope (Left) and the Arduino (Right).

As we can see in this example, the frequency is too low to get two peaks in 140 data points. So we need to increase the number of data points we are collecting. If we change the scale from 1 to 0.5, it doubles the amount of data we are collecting (*see line 15*) but only plots every other point in the array (*see line 122*).

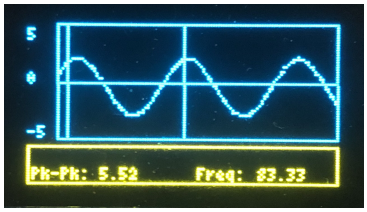

Figure 11: Same Wave Using a Scale Value of 0.5.

This fixes the problem and we now have enough data to get an accurate measurement for the frequency.

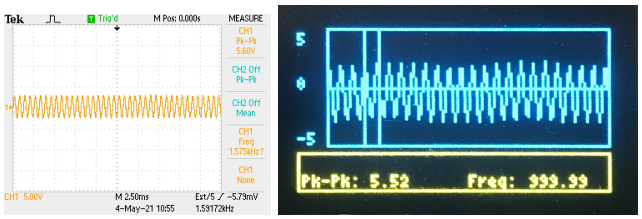

Figure 12: High Frequency Wave Measured on a Real Oscilloscope (Left) and the Arduino (Right).

In this example, the frequency is getting so high that it is hard to see the wave form. However, if we change the scale from 1 to 4, we will be collecting one fourth the amount of data (*see line 15*) and spacing each point out 4 times the amount (*see line 116*).

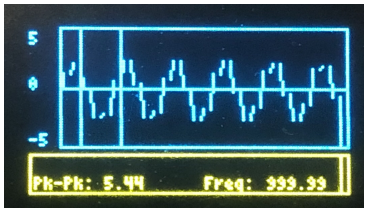

Figure 13: Same Wave Using a Scale Value of 4.

This allows us to see the wave form a little better but it is very jagged and does not make the frequency reading any more accurate. However, it does lessen the amount of work the Arduino has to do to display the wave since the array is a fourth the size it was before.

4 Data Analysis

Using 16 different frequencies in the range of 50-1000 Hz, we recorded the value of the frequency as read by an actual oscilloscope and compared it the the frequency measured by the Arduino oscilloscope. We then held the frequency at a constant 250 Hz and varied the voltage for 20 different points recording the peak-to-peak from both oscilloscopes.

	Data Set				
	Frequency (Hz)	Frequency 2 (Hz)	Pk-PK (V)	Pk-PK2 (V)	Pk-Pk3 (V)
1	50.47	47.61	1	0.64	1.12
2	61.27	58.82	1.6	0.98	1.68
3	73.78	71.46	2	1.38	2.01
4	81.11	76.92	2.6	1.94	2.57
5	91.79	90.9	3	2.35	2.9
6			3.6	2.9	3.56
7	95.73	90.9	4	3.37	3.95
8	121.17	111.11	4.6	3.9	4.57
9	178.31	166.6	5	4.23	5
10	248.24	250	5.6	4.78	5.6
11	323.35	333.33	6	5.3	5.95
12	391.56	333.33	6.6	5.88	6.55
13			7	6.4	6.98
14	443.23	500	7.6	6.73	7.6
15	563	500	8	7.21	7.99
16	766.36	1000	8.6	7.8	8.59
17	947.67	1000	9	8.26	9.02
18			9.6	8.88	9.58
19	1309.3	1000	10	9.1	10

Figure 14: Data Table.

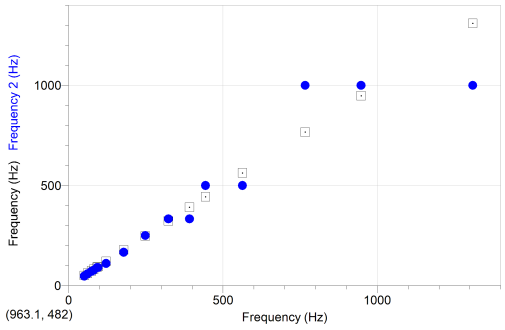

Figure 15: Comparison of Arduino (blue) and Oscilloscope Frequency Measurements (black).

the Arduino is only accurate for frequencies less than 500 Hz. For frequencies above this, it remains relatively the same until breaking 1 kHz. At this point, it will continue to display 999.99 Hz.

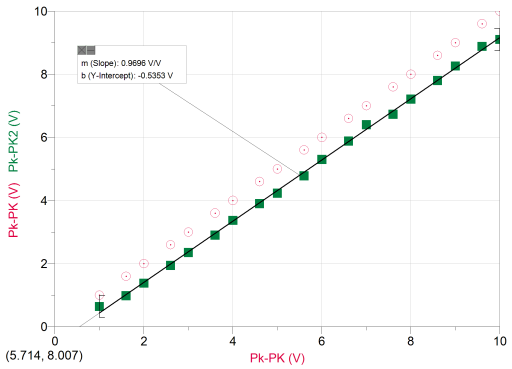

Figure 16: Comparison of Unaltered Arduino (Green) and Oscilloscope Pk-Pk Measurements (Red).

For the peak-to-peak values we got an interesting result. When doing a linear fit we see the Arduino pk-pk has a slope of 0.9696 and a y intercept of -0.535. In order to get as precise of a measurement as the oscilloscope, we should alter the code slightly.

```
// displays the function on the LCD screen
void plot(int peak1, int peak2, float freq, int max, int min, unsigned int data[500]) {
    // Calculate the Pk-Pk
    float pkpk = (max - min) / 50.0; // finding distance from peak to peak
    char pp[20];
    ftoa(pkpk, pp, 2);
    // The rest of the function below
}
```

Originally, the "plot" function in the code was written (as shown above) to calculate the pk-pk from the min and max values. We very simply found the distance between the min and max value of the array and divided by 10 to convert the Arduino's decivolts measurement into volts. However, for some reason we were off by a factor of 5 when testing this. So we divided this by 5 as well. In order for the data to fit the expected value, it needs to be shifted upward by the intercept value and divided by its current slope in order to achieve a slope of 1. Doing this modifies the code in the following way:

```
// displays the function on the LCD screen
void plot(int peak1, int peak2, float freq, int max, int min, unsigned int data[500]) {
  // Calculate the Pk-Pk
  float pkpk = (max - min) / 50.0; // finding distance from peak to peak
  float adjusted = (pkpk+0.5353)/0.9696; // adjusted to fit the error
  char pp[20];
  ftoa(adjusted, pp, 2);
```

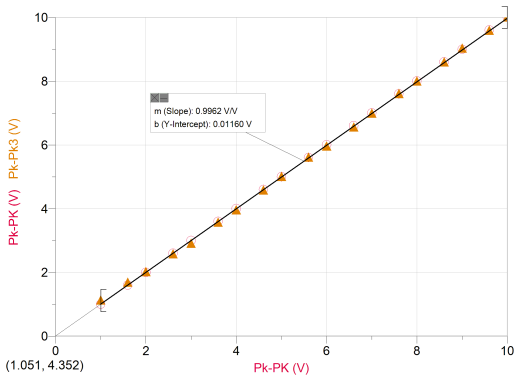

Figure 17: Comparison of Altered Arduino (Orange) and Oscilloscope Pk-Pk Measurements (Red).

We then measured the pk-pk values on the Arduino oscilloscope again in the Pk-Pk3 column of the data table in Figure 14. The new pk-pk value shows a very accurate readings for the pk-pk value on the Arduino oscilloscope.

If our mathematical model for the circuit output in equation 9 of the "Final Circuit" section of this report is correct, we should be able to model the output of Figure 8 in Mathematica. To do this, we will first define the input wave as perfect cosine wave with an amplitude 5.6/2. We choose this amplitude since the pk-pk of the input wave in figure 8 is 5.6V. We then multiply equation 9 by 5/2 since the output in Figure 2 is 2V/div where as the input is 5V/div.

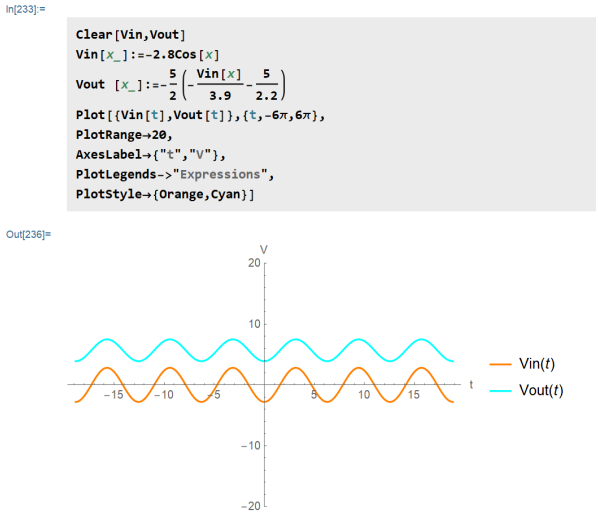

Figure 18: Mathematical Model of Circuit Output.

We can then put this plot on top of Figure 8 to see how they compare.

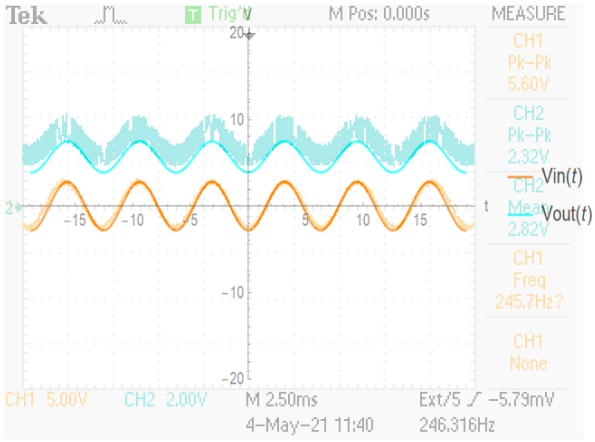

Figure 19: Model Compared with Oscilloscope Screenshot.

5 Conclusion

The wave output from the circuit as measured by the oscilloscope was almost exactly what we expected it to be. However, there was a lot of noise on the output. This could likely be from powering the second inverting amplifier from the Arduino power source which is known to be relatively noisy. The amplitude measurement was very acceptable after corrections. The frequency measurement was not great, however, the result that we got matches perfectly with what we would expect according to "Nyquist Theorem". The Nyquist Theorem states that "in order to adequately reproduce a signal, the signal should be periodically sampled at a rate that is 2X the highest frequency you wish to record." Considering the Arduino records at 1 ms, we can only collect samples at a rate of 1000 Hz. This means we can only accurately reproduce signals at a frequency up to 500 Hz. This aligns perfectly with the results that we achieved.

For next steps in this project, one could add a knob to adjust the value of the scale. We could also 3D print an enclosure for this and power it using a 9V battery. Doing this would also eliminate the noisy power input on the second inverting amplifier as well as make it portable.

Ultimately I feel as though we achieved the best possible result that we could given the limited power of the Arduino Uno. Improvements could still be made by using a more powerful micro controller, using a less noisy power input, or using a higher resolution OLED LCD display. Improvements could also be made to the implementation of the scale value as it does not work for any values less than 1 other than 0.5. If this were to be corrected, we should be able to read frequencies below 50 Hz much easier. However, we can very accurately display and measure alternating voltages with frequencies between 50-500 Hz with an amplitude between $\pm$10V. This can be very useful in low voltage projects or audio related projects since this frequency range covers most of the low and mid range audio we hear. However, I would suggest saving up for a true oscilloscope if you plan to do anything outside this range and wish to have very accurate measurements.