ECE 341 ACCELERATED PROJECT REPORT

AP Report

Authors: Shelby Westerberg, Trevor Horine, Bryson Goto

December 3, 2020

Instructor: Matthew Shuman
Grading TA: Shane Witsell

Contents

1 Introduction

The objective of the project is to create a tone detector. The device must detect notes from middle C to high C, with at least +/-0.5% tolerance for the tuning of the sound. This tone detector must create visual output by lighting exactly one of eight LED's which corresponds to the note being detected, or no LED at all if no note is being played. The device should be able to work with the audio source at least ten feet away from the microphone. Quantitative measures of success in getting a quality reading are as follows: a signal to noise ratio of at least 20 for the sampled audio signal, and at least 20 samples acquired in the detected period. Additionally, the output of the audio amplifier should be greater than 1 volt when an audio source is playing, and less than 0.2 volts when no sound is playing.

2 Background

The range of frequencies to be detected are shown in Table 1 below, along with the lower and upper bounds of the smallest acceptable range (+/-0.5%) that must be detected for each. [1]

Table 1: These are the frequencies that need to be detected and the acceptable range that can be identified as each frequency.

Note	Ideal Frequency (Hz)	Lower bound (Hz)	Upper bound (Hz)
C4	261.6256	260.0190	262.6322
D4	293.6648	292.1965	295.1331
E4	329.6276	327.9795	331.2757
F4	349.2282	347.4821	350.9743
G4	391.9954	390.0354	393.9554
A4	440.0000	437.8000	442.2000
B4	493.8833	491.4139	496.3527
C5	523.2511	520.6348	525.8674

To meet the objective of acquiring at least 20 samples per period, the device must sample with a frequency 20 times higher than the frequency being detected. The highest frequency that needs to be detected is the upper bound of C5. This means that sampling must occur at a rate of at least 10520Hz to meet this objective for all desired notes.

3 Procedure

3.1 Simulation and Calculations

The circuit design was based around Figure 1 from page 1 of the microphone amplifier documentation provided [2]. Hand calculations were done based on the information given in the datasheets for the microphone and the amplifier to be used in the project to find:

- The maximum AC current produced by the microphone.

- The input voltage that this would feed into the amplifier.
- The gain necessary to achieve the goal of having at least 1V AC swing when the microphone was picking up a noise.
- The value of the gain controlling resistor (R4 in Figure 1) necessary to achieve the desired gain.

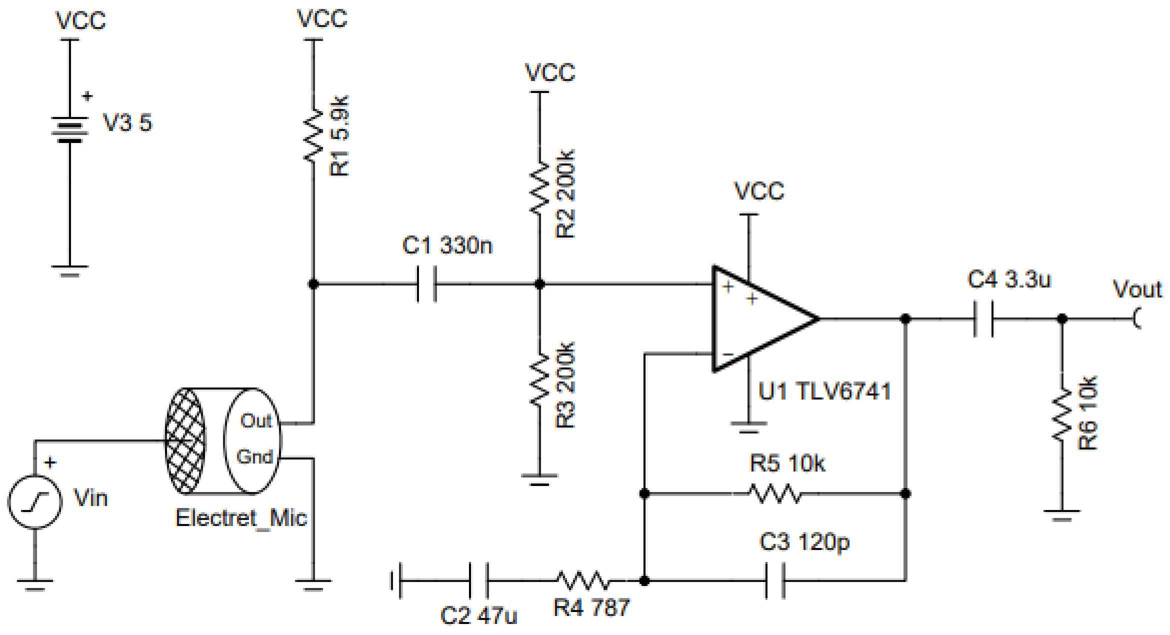


Figure 1: Non-inverting Microphone Amplifier Circuit Example from Texas Instruments

The circuit proposed based on these calculations was simulated in LTSpice using the following model shown in Figure 2. The PSpice model for the amplifier was downloaded from the manufacturer's website and modified to work in LTSpice. [3]

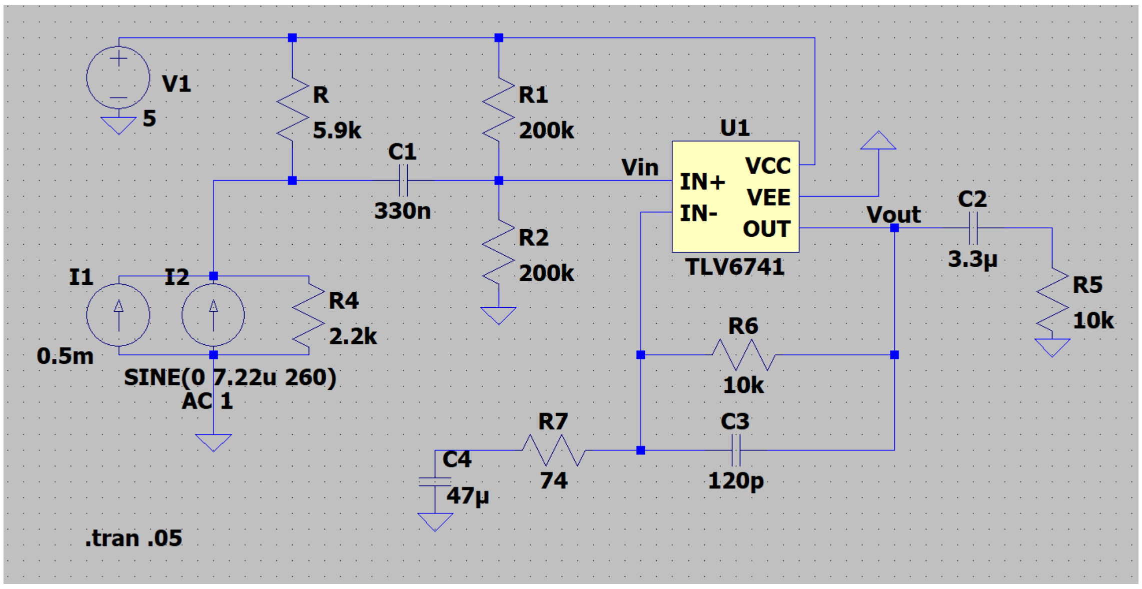


Figure 2: TLV6741 Amplifer LTSpice Simulation

To simulate the circuit, a sweep command was used to define the AC signal coming from the current source, I2, which represented the microphone. The microphone was represented with two current sources and a resistor for the internal resistance of the microphone. One current source, I1, represented the DC signal and the other current source, I2, represented the AC signal. Through these simulations, the gain of the amplifier was graphed to visualize what the amplifier was doing at the various frequencies being tested. The transient option was also used to model the output voltage at different frequencies.

These simulations were used to ensure that the gain of the amplifier would be close to constant in the frequency range relevant to this project (262Hz to 523Hz), and to adjust the resistor used to control the gain in order to both amplify the signal enough such that the output has at least 1V of amplitude per the specifications, however not so much that the peaks of the wave are clamped due to the 0V to 5V range of the power supplied to the amplifier. These simulations started using the value found in the hand calculations, and then used to check resistance values that would be reasonable to create with the resistors available. Through calculations, R7 was calculated to be 74Ω.

3.2 Construction and Verification

The testing of the physical circuit may be disentangled from the testing of the code if access to an oscilloscope is available, however for this project it was not. Additionally, if necessary to debug the code without concern over the predictability of the physical circuit's output, sample data could be generated using MATLAB to ensure the code works before using it to test the physical circuit. This project tested both pieces in tandem, so the following Coding section happened in parallel with the AC verification of the circuit.

The circuit planned using the LTSpice simulation and hand calculations was first drafted into the schematic shown in Figure 3. The circuit was initially built on a breadboard for easier troubleshooting and modifications. The DC operating bias of the input and output of the circuit was verified using a DMM. 2.5V is the ideal bias point to see the maximum amount of both the upper and lower parts of the resulting AC output, as it is halfway between 0V and 5V. Due to lack of access to an oscilloscope, the AC operation of the circuit had to be checked using the Arduino Nano and MATLAB code to take and parse through the data. The method of collection and evaluation is discussed further in the corresponding code section.

This circuit was first tested using input from a phone's audio cable to minimize complications from the microphone in the initial testing. A 3.5 mm audio cable was cut in half and the ground placed where the ground of the microphone is in Figure 3 and one of the stereo wires was placed where the positive side of the microphone is in the schematic. The cable was then plugged in to a phone and a tone played to produce the sine wave at the desired frequency as seen in Figure 4. The output from this test was recorded using an Arduino Uno with the code shown in Appendix section 6.1, and analyzed using the MATLAB code shown in Appendix section 6.2.

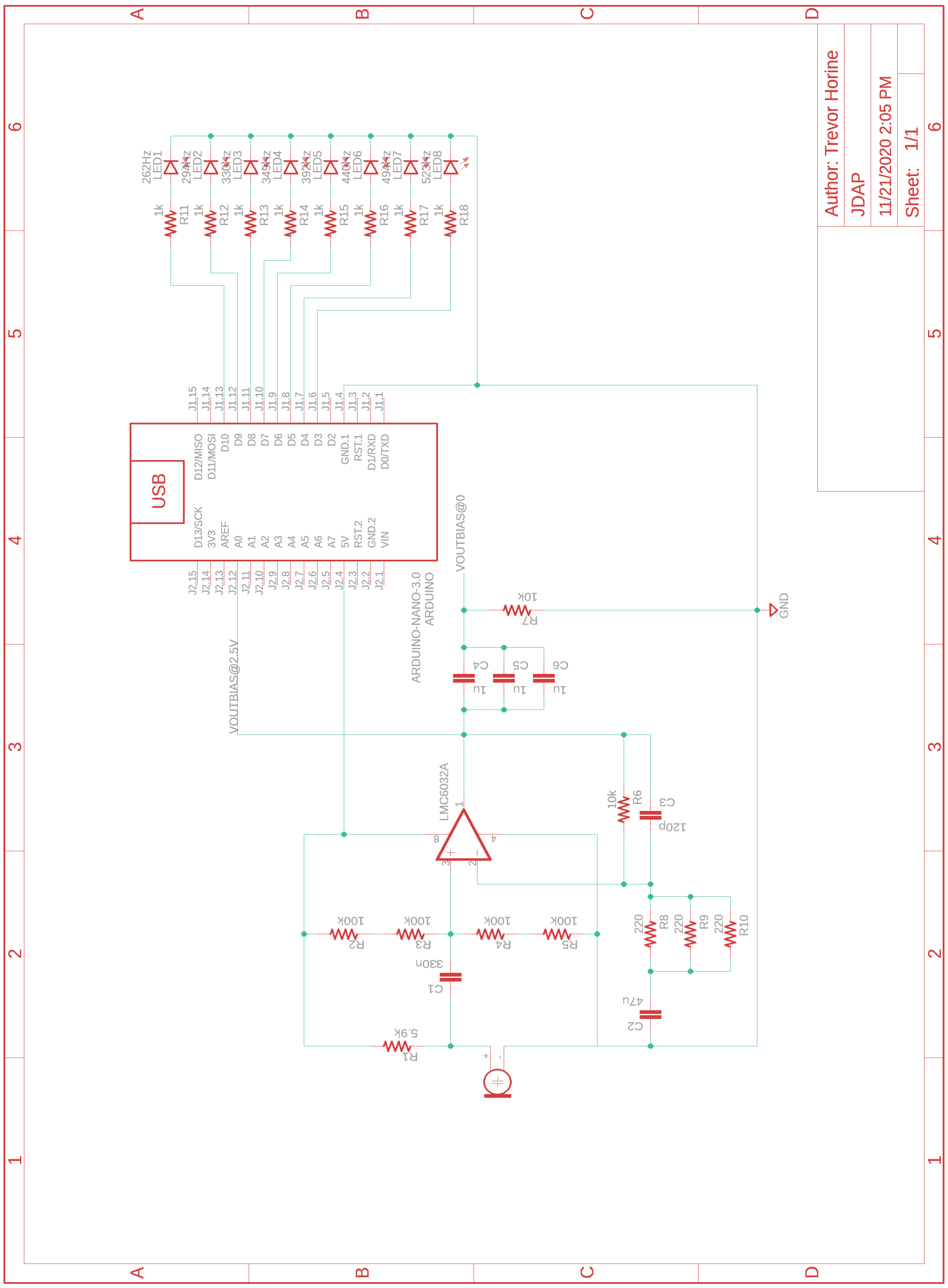


Figure 3: Detailed schematic of audio detection circuit

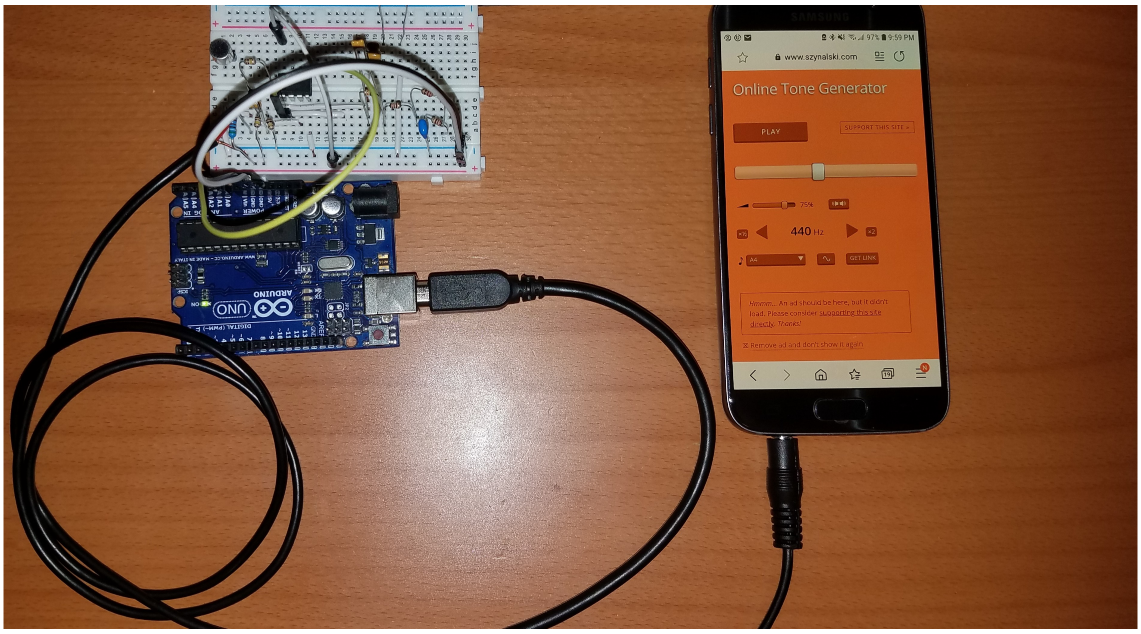


Figure 4: Amplifier and FFT Testing Setup

Once the verification of the circuit was completed, the circuit was then constructed more robustly by soldering components on to test pads such that there would be no concern of bumping components out of place as there is with a breadboard. Verification was then confirmed to match the previous breadboard circuit as MATLAB graphs were produced, to ensure the construction was completed correctly.

3.3 Coding

In order to achieve the desired sampling frequency of at least 10520Hz, the Arduino code needed to sample efficiently. In the pursuit of a very high sampling frequency, the Arduino code used an ISR (interrupt service routine) and the prescaler was set to $\frac{1}{8}$th using register programming in the set up of the Arduino script. This allowed the Arduino to move on to the next sample as soon as each analog to digital conversion was done.

MATLAB code was separated into two separate programs, one of which takes one set of measurements and is used to produce graphs and calculate the SNR, and one of which runs a loop in order to allow the device to update the LED's. The Fourier transform is done in MATLAB using MATLAB's FFT (fast Fourier transform) function, as is determining the specific frequency being played. The if statement which determines which if any of the notes from middle C to high C are detected was written with at least enough tolerance to detect all frequencies inside the +/-0.5% range for each note, as described in Table 1 in the Background section. This code can be found in the Appendix section 6.3.

The Signal to Noise Ratio (SNR) was done using the MATLAB SNR function for real sinusoids. This allows a number of harmonics of the detected frequency to also be ignored for the purposes of the SNR. Additionally, it means that it is not necessary to know in advance what the input frequency will be, as the function already picks out the most prevalent frequency as the signal. For an accurate measurement using this function, it is necessary that the sound inputted is sinusoidal. Sinusoidal input from an online tone generator was used to test the SNR of the system.

3.4 Final Product

Figure 5 shows the block diagram of the construction of the finished product. Note that the output produced by the output filter is not used, however it was left in the schematic to ensure that should it be necessary to filter out any low sounds not in the desired testing range. Not pictured is the USB connection from the Arduino to a serial port of a PC running the associated MATLAB script.

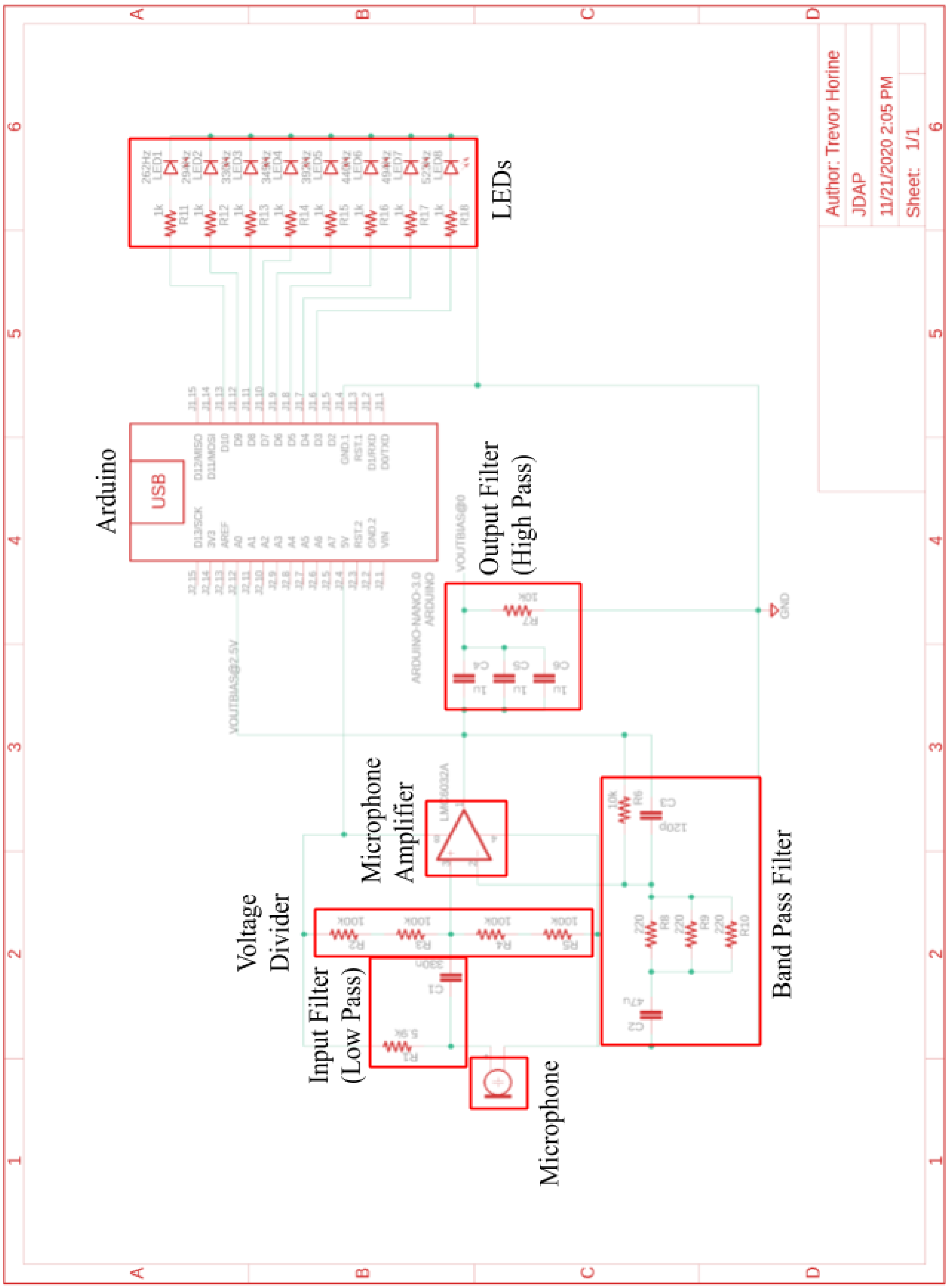


Figure 5: Block Diagram of Audio Detection Circuit

4 Results

4.1 Simulation and Calculations

The following hand calculations were done to find the appropriate resistor values needed to get an appropriate gain using the materials available.

1. Mic Sensitivity in V/Pa
   $10^{-42/20} = 7.94mv/Pa$

2. V/Pa to A/Pa
   $(7.94mV/Pa)/(2.2k\Omega) = 3.61\mu A/Pa$

3. Maximum Possible current
   $3.61\mu A * 2Pa = 7.22\mu A$

4. Desired value of R_1
   $(V_{CC} - V_{mic})/(I_{mic}) = (5-2)/0.5 = 6000\Omega \approx 5.9k\Omega$

5. DC Input Voltage Bias point
   $R_2 = R_3 = 200k\Omega$

6. AC Input Voltage
   $R_{in} = R_1||R_{eq} = R_1||(R_2||R_3) = 5571.29\Omega$
   $V_{in} = I_{max} * R_{in} = 7.22\mu A * 5571.29\Omega = 40.2mV$

7. Desired Gain
   $Gain = V_{outmax}/V_{in} = 5V/40.2mV = 119V/V$

8. Value of R_4
   $R_4 = R_5/(Gain-1) = 10k\Omega/118 = 84\Omega \approx 73\Omega \approx 220||220||220$

The graph in Figure 6 shows the gain expected at various frequencies using the resistor value closest to the hand calculated value that would be possible to create. When zoomed in to the range of frequencies that are being tested for (260Hz to 526Hz), the gain of the circuit is fairly constant with a difference of 0.00113 from the lowest to highest frequency. Having a steady gain ensures that the difference between V_{in} and V_{out} remains stable and does not damage the Arduino, should the amplifier be powered by a source other than the Arduino.

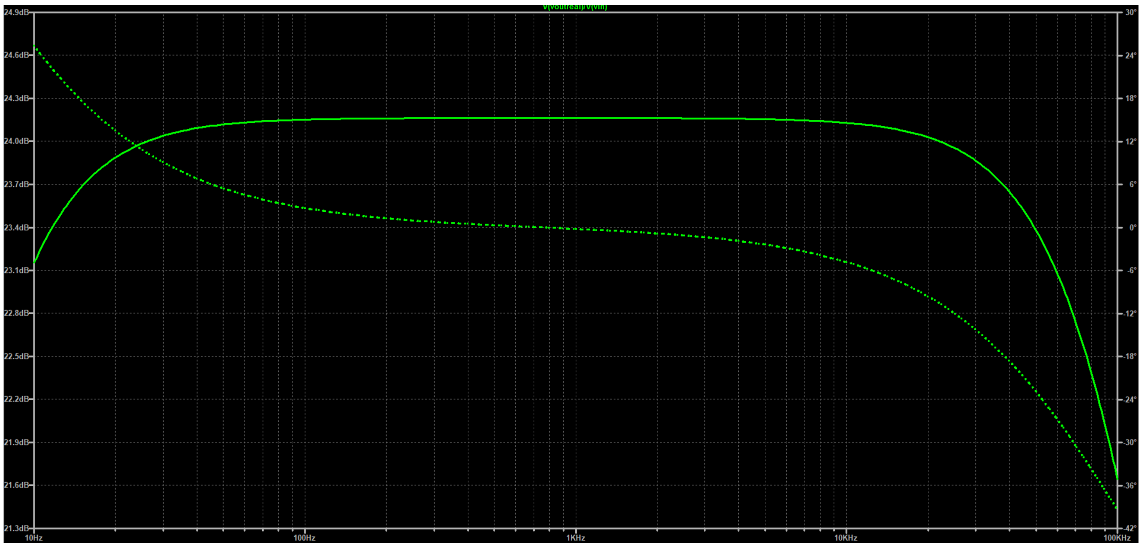


Figure 6: AC Simulation Gain Frequency Sweep

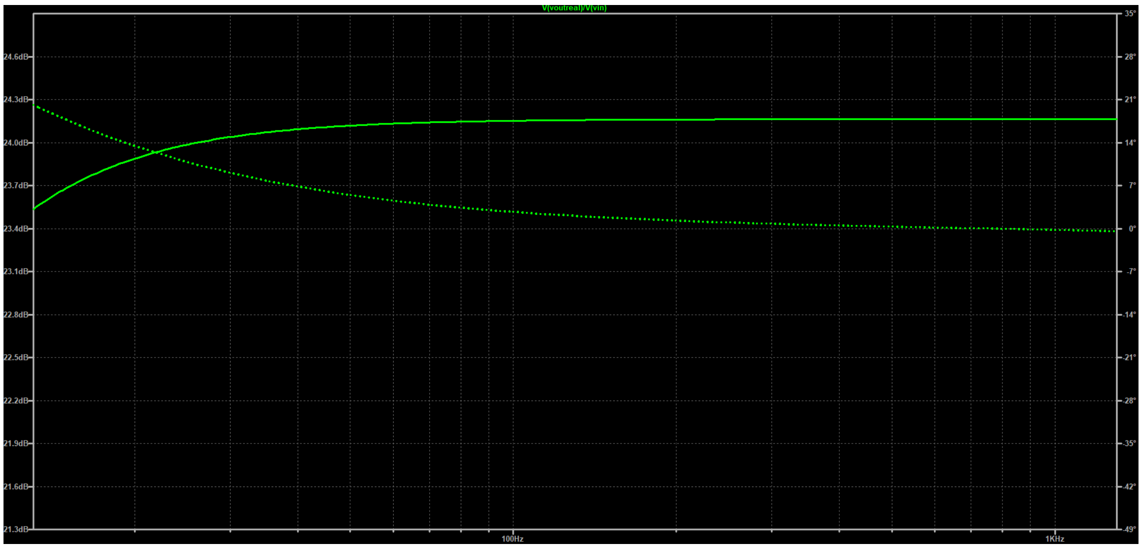


Figure 7: AC Simulation Gain Frequency Sweep (100Hz-1000Hz)

4.2 Construction and Verification

The DC bias points of the input and output of the amplifier were verified using a DMM. Small variance is possible due to the tolerances of resistors R2, R3, R4, and R5, shown in Figure 5 as the voltage divider creating the input DC bias point. The results of this test are reported in Table 2.

Table 2: Ideal and Measured DC Voltage bias at the input and output of the amplifier.

Node	Ideal DC Voltage (V)	Measured DC Voltage (V)
Amplifier Pin 3 (V_{in})	2.5	2.41
Amplifier Pin 1 (V_{out})	2.5	2.20

These values are within the expected tolerance given component tolerances and the variation in power supplied by the Arduino's 5V supply. These are also far enough from the limits of the supplied power (0V and 5V) to allow a large waveform which is not clipped on either side.

The amplifier circuit and MATLAB signal processing were tested using a clean signal produced by an audio cable and a phone at frequencies between 261.626Hz (middle C) and 523.251Hz (high C). A sample graph from this test are shown in Figure 8. The amplitude spectrum produced when run with an input of 261.626Hz shows a tall spike at 260.7Hz, which is only 0.926Hz away from the expected value. This suggests that the amplifier circuit and MATLAB code are successful in determining the frequency of a relatively clean signal. Note that this graph was produced before the Arduino code had incorporated the ISR, so the sampling time is substantially higher.

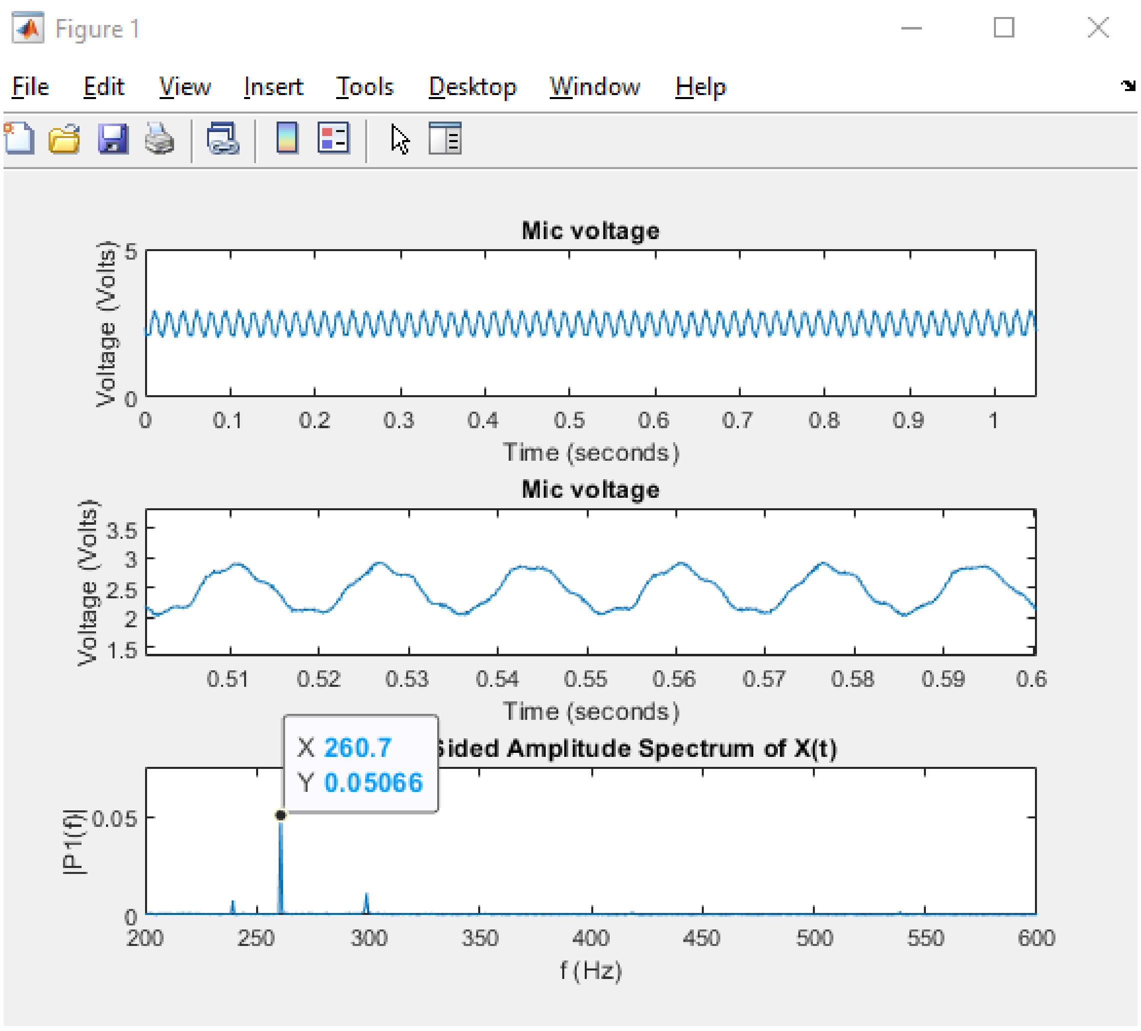


Figure 8: MATLAB Output from audio cable clean signal test at middle C

The magnitude of the output when using the microphone, with and without sound stimulation, was then tested using the graphs produced in MATLAB. The tones used to generate the sound from a cell phone to test the device were found from an online source.[4] Figure 9 shows the output of a test with no sound being played.

While some larger spikes are visible, the peaks are typically below the 0.2V. Figure 10 shows the output of a test with middle C being played very near the microphone on the left and high C on the right. Figure 11 shows the output of a test with sound being played from a source playing G at a distance of 10ft from the microphone. Recall that the objective is for the amplitude of the AC voltage output to be less than 0.2V of amplitude when not stimulated with an audio source, and to be at least 1V of amplitude when stimulated with an audio source that is 10ft from the microphone or closer.

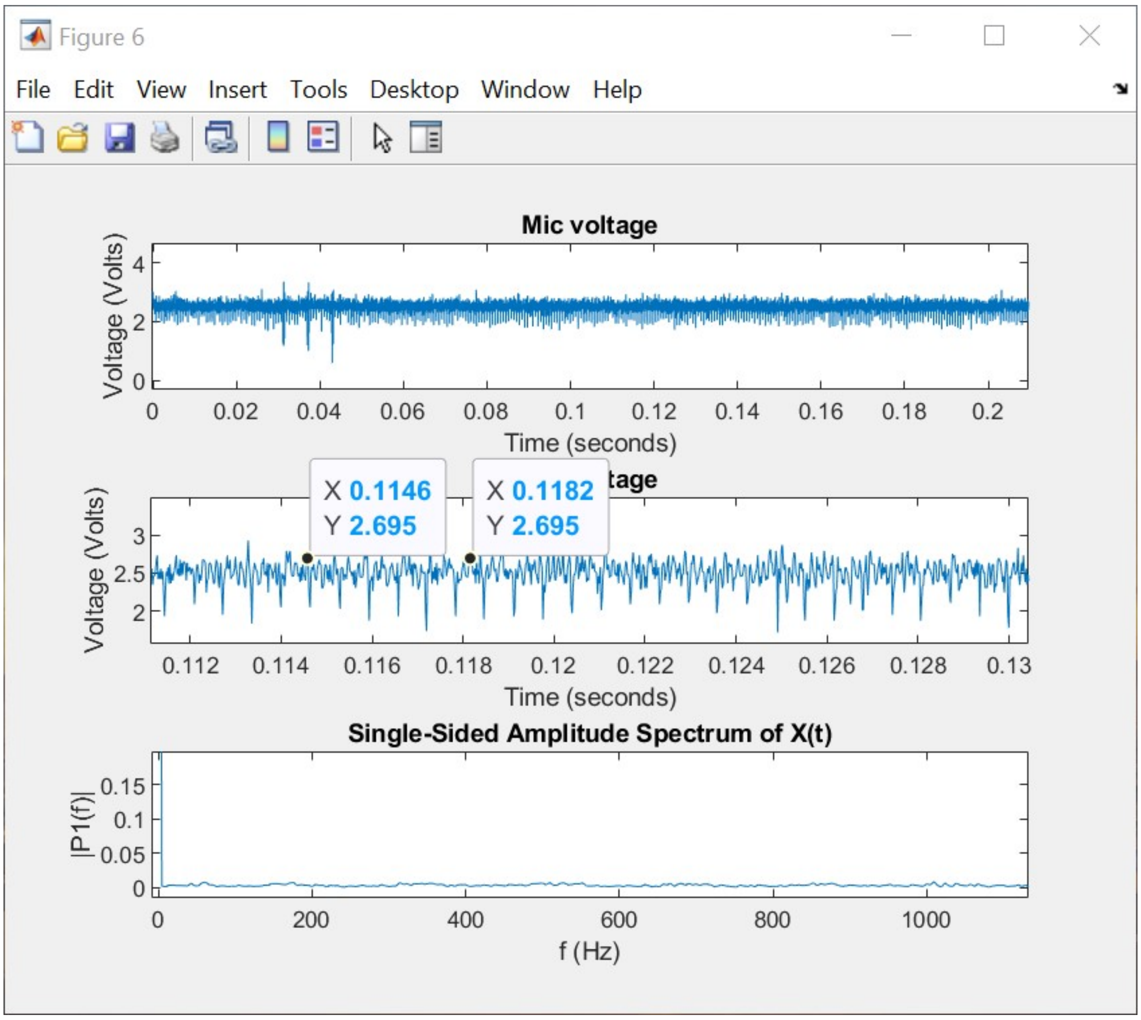


Figure 9: MATLAB Output with no signal being played.

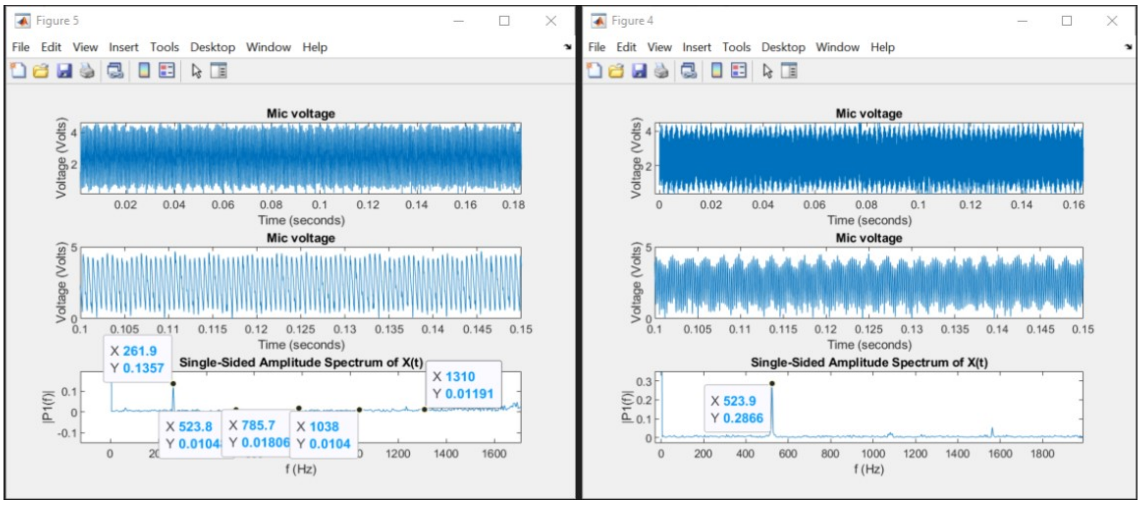


Figure 10: MATLAB Output of middle C on the left and high C on the right.

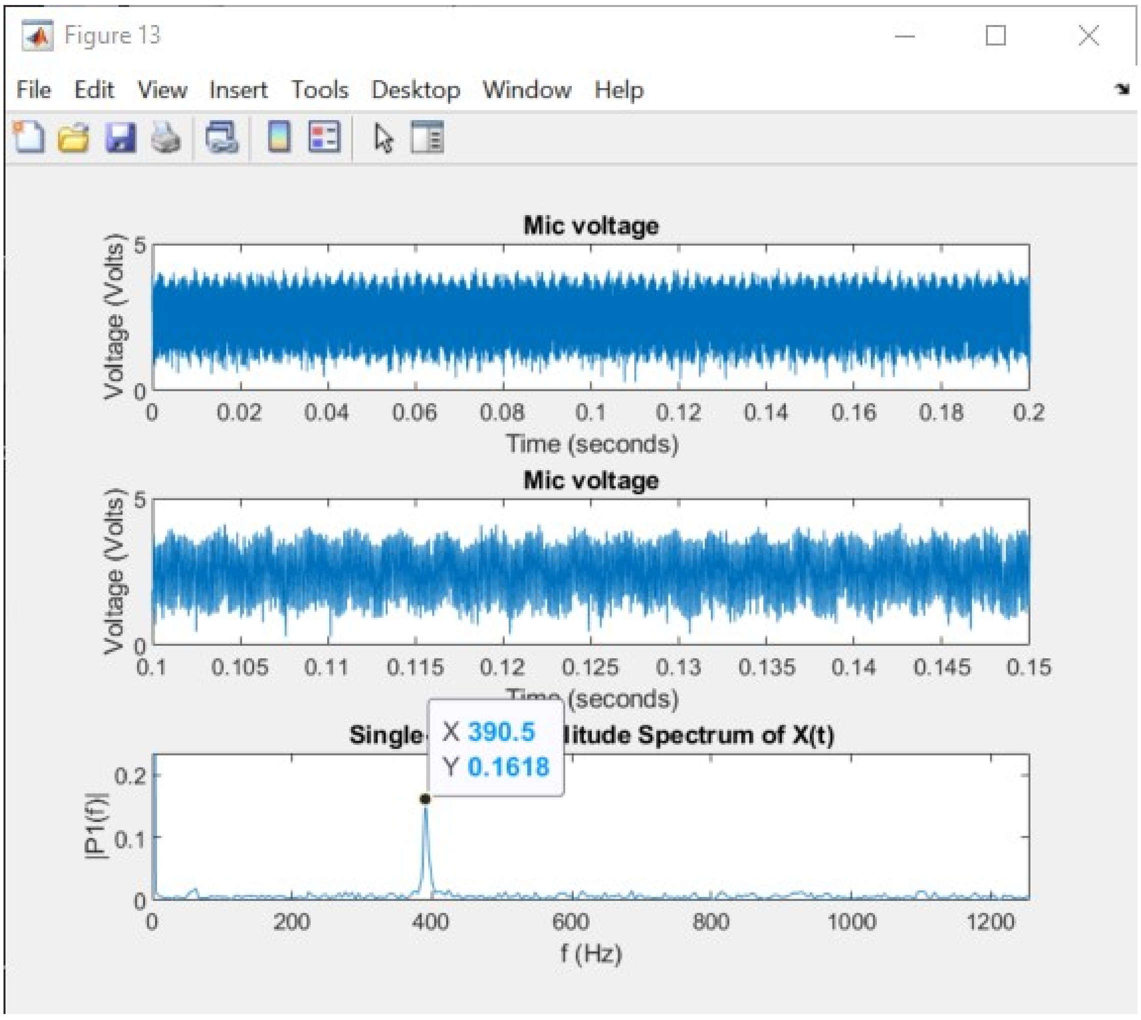


Figure 13
File Edit View Insert Tools Desktop Window Help
Mic voltage
Voltage (Volts)
Time (seconds)
Single-Sided Amplitude Spectrum of X(t)
|P1(f)|
f (Hz)
X 390.5
Y 0.1618

Figure 11: MATLAB Output G played from 10 feet away from microphone.

4.3 Coding

The finished Arduino code used in the project can be found in Appendix section 6.1. The Arduino code's internal timer reported the time taken to collect all 10500 samples in each run to MATLAB. Over 10 tests, whose results are shown in Table 3 the average time reported was 0.209985s, the maximum time reported was 0.209996s, and the minimum time reported was 0.209972s. Recall that in order to meet the objective of sampling at a rate of at least 20 times the highest frequency in the desired range the sample rate needed to be at minimum 10520Hz. The lowest sampling frequency measured in the 10 tests recorded for this purpose was $10500/0.209996 = 50001Hz$, which is nearly 5x the 10520Hz minimum.

Table 3: Sampling Time and Frequency Reported in MATLAB

Test	Sample Time (s/10500 samples)	Sampling Frequency (Hz)
1	0.209972	50007
2	0.209984	50004
3	0.209984	50004
4	0.209996	50001
5	0.209980	50005
6	0.209988	50003
7	0.209980	50005
8	0.209992	50002
9	0.209984	50004
10	0.209988	50003
Average	0.209985	50004

The MATLAB code used for testing and creating graphs for the project can be found in Appendix section 6.2. To ensure that the Fourier Transform used in MATLAB was correctly converting the time domain signal into a frequency spectrum, the MATLAB code was set up to output both the generated frequency spectrum and the time domain voltage as graphs. The major frequencies detected according to the frequency spectrum were visually checked to match the component frequencies seen in the time domain graph. Figure 8 shows an example of the graphs used to check this, with the frequency of the largest sine wave estimated from the middle graph being approximately equal to 261Hz.

Once the calculations in MATLAB were determined to be accurately detecting the desired frequency, a modified copy of the testing code, found in Appendix section 6.3 was created which allowed MATLAB to run the calculations repeatedly and communicate the results back to the Arduino. This was used to run and test the operation of the LED indicators as a method of communicating the results.

4.4 Final Product

Figure 12 shows the completed circuit built on a protoboard to minimize potential for the circuit to have loose connections. All of the components were transferred and soldered from the breadboard to the protoboard.

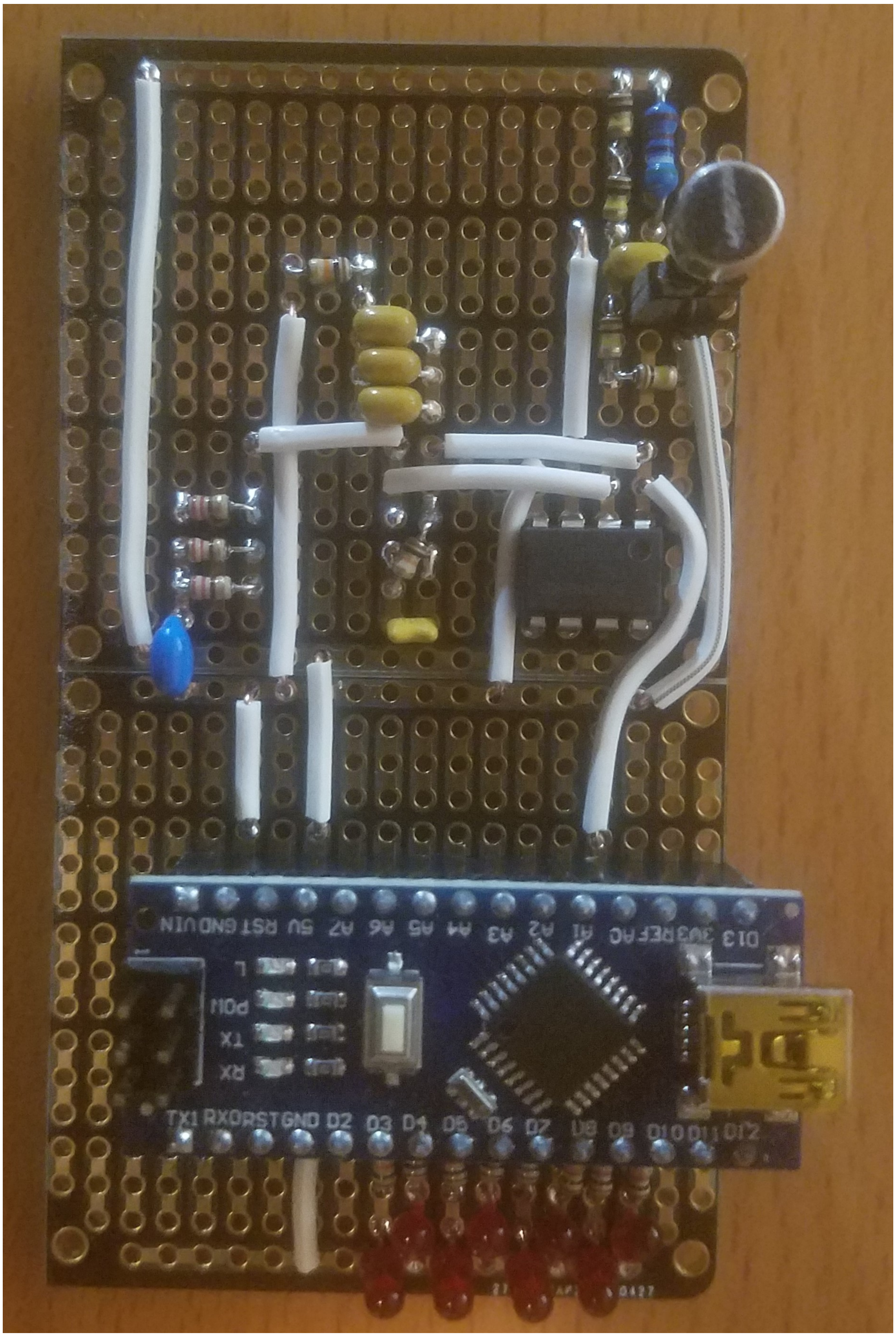
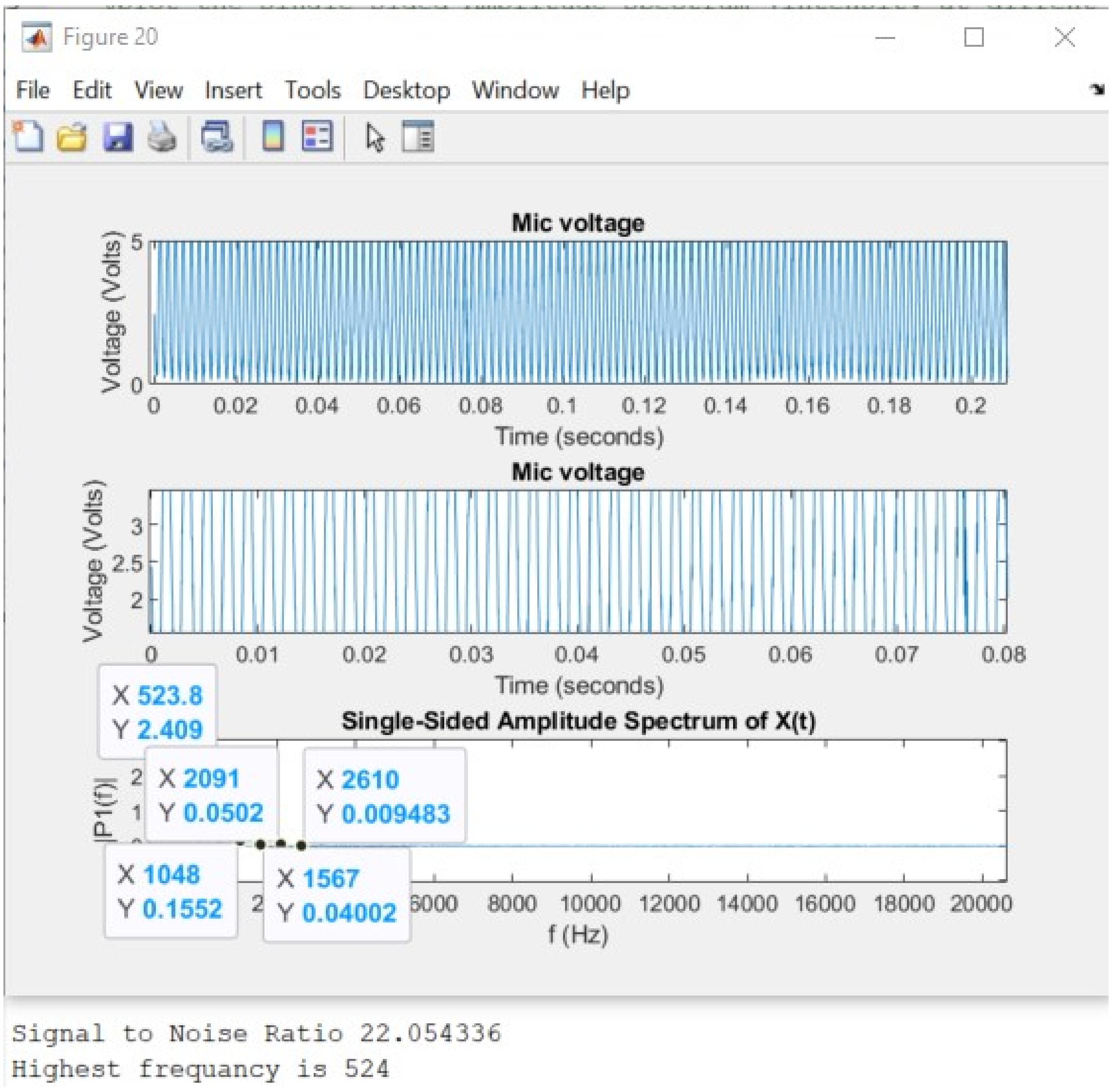


Figure 12: Final Circuit on Protoboard

The LED (looping) version of the created code was used to test the range of the circuit, which was confirmed to successfully detect all 8 desired notes, as well as silence, from up to 10ft away. The demo video at https://media.oregonstate.edu/media/t/1_n8bh8kyv shows the circuit running with the audio source 10ft away in the background. All 8 notes are shown being detected, as well as silence before and after moving through the notes.

The signal to noise ratio was computed using a sinusoidal tone produced by a phone as input. Figure 13 shows a sample SNR result in MATLAB for high C input. The SNR value is listed below the voltage and power graphs with the corresponding frequency read below. This was considered a success, as the target SNR was 20dB or more, and we achieved an SNR of 22dB.



```
Signal to Noise Ratio 22.054336
Highest frequancy is 524
```

Figure 13: SNR MATLAB Value

5 Conclusion

When working with limited resources for testing, using alternative strategies for troubleshooting is essential. The strategies of creating multiple physical circuits in order to check whether unexpected results in the code would be matching (suggesting a coding error) or separate (suggesting a construction error) was essential in troubleshooting the circuit without access to an oscilloscope. Additionally, careful modeling using LTSpice became even more essential to ensure that the design of the circuit was good before constructing and testing the physical circuit. Knowing what results to expect in a correctly assembled circuit helped to troubleshoot the circuit.

The scope of this project ranged from hand calculating component values, simulating designs, writing code for reading values, building the circuit, and troubleshooting and documenting. A lot of factors that seemed simple on paper turned out to be more complicated than expected. The simulation and actual results were different due to many factors such as tolerance and imperfect simulation models. As such, there were modifications that needed to be made from the real circuit from the simulated one.

Another issue that came up was that in an attempt to reduce problems from the microphone, more problems arose. The cable that was used to transfer the signal directly to the circuit from the phone caused a constant 60Hz noise– disrupting the signal that was trying to be read. It is likely that this was electromagnetic interference caused by the circuit being surrounded by other devices and circuits in the house that were all on 60Hz signals. This was able to be resolved by shortening wires wherever possible, and moving the circuit to an area with less active electronics. Overall, each part of the process was necessary to ensuring the final product worked well, as well as reducing the scope of troubleshooting necessary.

6 Appendix

6.1 Arduino Sample Collection Code

```
/*
This program waits for an input in the serial monotor then prints the next 10500 values captred by the arduino to the serial port. It also passes the time it took to collect all the samples the time it took to gather thoes samples to the serial port.
Trevor Horine
11/6/2020
*/

//Pin assinments and global varables
const int analogInPin = A0;  // Analog input pin that is used as input form amplifier
int sensorValue = 0;         // value read from the output of amplifier
const int MAX_SAMPLES = 10500; //number of samples collected before sending
int last;

void setup() {
  Serial.begin(230400); // initialize serial communications at 230400 baud rate
  pinMode(LED_BUILTIN, OUTPUT); //built in led
  digitalWrite(LED_BUILTIN, HIGH);
}

void loop() {
  int values = MAX_SAMPLES+100;
  digitalWrite(LED_BUILTIN, HIGH);
  while(!(Serial.available() > 0)){ //while nothing in serial port (waiting for matlab to reach out)
    Serial.read();
    values = 0;
    last = true;
  }
  digitalWrite(LED_BUILTIN, LOW);//when found something in serial port (matlab has reached out)
  delay(10);
  long before = micros(); //current time in microseconds before collecting data
  while(values < MAX_SAMPLES) { //while less then desired number of samples has been printed to the serial port
    // read the analog in value form amplifier
    sensorValue = analogRead(analogInPin);
    byte buf[2]; //split in to bytes
    buf[0] = sensorValue & 255;
    buf[1] = (sensorValue >> 8)  & 255;
    Serial.write(buf, sizeof(buf)); // write to serial port
    values++;
  }
  if (last) {
    long diff = micros()-before;//calculate and write time it took to get samples to serial port
    byte buf[4];
    buf[0] = diff & 255;
    buf[1] = (diff >> 8)  & 255;
    buf[2] = (diff >> 16)  & 255;
    buf[3] = (diff >> 24)  & 255;
    Serial.write(buf, sizeof(buf));
    last = false;
  }
}
```

6.2 Matlab FFT and Graphing Single Set of Data

```matlab
%This program will open a serial port, then send a signal to an
    Arduino. It
%will then read in a set number of samples and how long it took to
    collect
%them. The arduino ADC values will then be converted to voltages.
    The
%captured signal will be graphed and run through the MATLAB FFT to
%determine the frequancy of the signal. The Single-Sided Amplitude
    Spectrum
%is graphed and the highest frequancy in the range between 250 and
    540 is
%reported to the command window.
%Trevor Horine
%11/6/2020

%start with a clean workspace
clear

%Number of samples
numSamples = 10500;


%open serial port set this to what port the Arduino is conected to
device = serialport("COM6",500000);
configureTerminator(device,"CR/LF")
%create empty vector to add values of the signal to
y = 0:0:0;
%read in line specified number (50) times from serial port
pause(5);
write(device,"getval","string");
%read in samples from Arduino
tic;
num = read(device,numSamples,"uint8")
toc
%read in time to collect samples from Arduino
timeIn = read(device,1,"uint32");
%convert time to seconds from micro seconds
captureSeconds = timeIn/1000000;
%print time to capture samples
fprintf("Capture Time: %f\n",captureSeconds);
%time per sample in seconds
sampleTime = captureSeconds/numSamples;
%create time vector to graph signal against
%x is the vector of the x-axis and has increments of time that
x = 0:sampleTime:captureSeconds-sampleTime;
%for loop to covert from ADC value and place it in the vector
for i = 0:numSamples-1
    %num is the string that is read in from the serial port
    curnum = num(i+1);
    %add it to the end of the y vector
    y(end +1) = (curnum*5)/256;
end
%release the serial port so it can be used by something else
%clear device
%create figure with a 1x3 array of graphs
figure
%plot the graph using the x and y vectors
```

```
%plot the whole signal
subplot(3,1,1)
plot(x,y)
%title graph and axes
title('Mic voltage')
xlabel('Time (seconds)')
ylabel('Voltage (Volts)')
%range of axis
xlim([0 1.05])
ylim([0 5])

%plot small section of signal so can see the period (might not line
    up with
%the window perficly, if not pan up or down)
subplot(3,1,2)
plot(x,y)
%title graph and axes
title('Mic voltage')
xlabel('Time (seconds)')
ylabel('Voltage (Volts)')
%range of axis
xlim([.1 .2])
ylim([2 4.5])

%MATLAB FFT
Fs = numSamples/captureSeconds;      % Sampling frequency
T = 1/Fs;          % Sampling period
L = numSamples;              % Length of signal (in milliseconds)
t = (0:L-1)*T;               % Time vector
Y = fft(y);
P2 = abs(Y/L);
P1 = P2(1:floor(L/2+1));
P1(2:end-1) = 2*P1(2:end-1);
%0.74074074074074 is weird factor that we found in testing, dont
    know where
%it came from but the FFT is off by this factor every time.
f = Fs*(0:(L/2))/L;
%f = (1/captureSeconds)*Fs*(0:(L/2))/L;

%SNR computation:
r = snr(y,Fs,3); %outputs snr in decibles, ignoring the first 3
    harmonics
fprintf("Signal to Noise Ratio %f\n", r)


%plot the Single-Sided Amplitude Spectrum (intensity at difrent
    frequancies)
subplot(3,1,3);
plot(f,P1)
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')
xlim([200 600])
ylim([0 .075])

%find maximum value in the intensity vector from the FFT and it's
    index
%ignore anything that is not between 250ish to 530ish
```

```
 b = P1(53:120);
 [maximum, maxindex] = max(b);
%if maximum is really small, liklely no signal
feq = round(f(maxindex+52));
if maximum < .001
        fprintf("No Signal\n")
        write(device, "N", "string");
    %print the freqancy of signal
else
    fprintf("Highest frequancy is %i\n", feq)
    if feq >= 260 && feq <= 263
        write(device, "MC","string");
    end
    if feq >= 292 && feq <= 295
       write(device, "D","string");
    end
    if feq >= 327 && feq <= 331
       write(device, "E","string");
    end
    if feq >= 347 && feq <= 351
       write(device, "F","string");
    end
    if feq >= 390 && feq <= 394
        write(device, "G","string");
    end
    if feq >= 437 && feq <= 442
        write(device, "A","string");
    end
    if feq >= 491 && feq <= 496
        write(device, "B","string");
    end
    if feq >= 520 && feq <= 526
        write(device, "HC","string");
    end
end
%clear workspace when done
clear device
```

6.3 Matlab Repeated Calculations and LED Communication

```matlab
%This program will open a serial port, then send a signal to an Arduino. It
%will then read in a set number of samples and how long it took to collect
%them. The arduino ADC values will then be converted to voltages. The
%captured signal will be graphed and run through the MATLAB FFT to
%determine the frequancy of the signal. The Single-Sided Amplitude Spectrum
%is graphed and the highest frequancy in the range between 250 and 540 is
%reported to the command window.
%Trevor Horine
%11/6/2020

%start with a clean workspace
clear

%Number of samples
numSamples = 10500;
%open serial port set this to what port the Arduino is conected to
device = serialport("COM6",500000);
configureTerminator(device,"CR/LF")

while 1
    pause(2);
    %create empty vector to add values of the signal to
    y = 0:0:0;
    write(device,"getval","string");
    %read in samples from Arduino
    tic;
    num = read(device,numSamples,"uint8");
    toc
    %read in time to collect samples from Arduino
    timeIn = read(device,1,"uint32");
    %convert time to seconds from micro seconds
    captureSeconds = timeIn/1000000;
    %print time to capture samples
    fprintf("Capture Time: %f\n",captureSeconds);
    %time per sample in seconds
    sampleTime = captureSeconds/numSamples;
    %create time vector to graph signal against
    %x is the vector of the x-axis and has increments of time that
    x = 0:sampleTime:captureSeconds-sampleTime;
    %for loop to covert from ADC value and place it in the vector
    for i = 0:numSamples-1
        %num is the string that is read in from the serial port
        curnum = num(i+1);
        %add it to the end of the y vector
        y(end +1) = (curnum*5)/256;
    end
    %release the serial port so it can be used by something else
    %clear device
    %create figure with a 1x3 array of graphs
    % figure
    % %plot the graph using the x and y vectors
    % %plot the whole signal
    % subplot(3,1,1)
    % plot(x,y)
    % %title graph and axes
    % title('Mic voltage')
    % xlabel('Time (seconds)')
    % ylabel('Voltage (Volts)')
    % %range of axis
    % xlim([0 1.05])
    % ylim([0 5])
    %
    % %plot small section of signal so can see the period (might not line up with
    % %the window perficly, if not pan up or down)
    % subplot(3,1,2)
    % plot(x,y)
    % %title graph and axes
    % title('Mic voltage')
    % xlabel('Time (seconds)')
    % ylabel('Voltage (Volts)')
    % %range of axis
    % xlim([.1 .2])
    % ylim([2 4.5])

    %MATLAB FFT
    Fs = numSamples/captureSeconds;      % Sampling frequency
    T = 1/Fs;          % Sampling period
    L = numSamples;              % Length of signal (in milliseconds)
    t = (0:L-1)*T;               % Time vector
    Y = fft(y);
    P2 = abs(Y/L);
    P1 = P2(1:floor(L/2+1));
    P1(2:end-1) = 2*P1(2:end-1);
    %0.74074074074074 is weird factor that we found in testing, dont know where
    %it came from but the FFT is off by this factor every time.
    f = Fs*(0:(L/2))/L;
    %f = (1/captureSeconds)*Fs*(0:(L/2))/L;

    %plot the Single-Sided Amplitude Spectrum (intensity at difrent frequancies)
    % subplot(3,1,3);
    % plot(f,P1)
    % title('Single-Sided Amplitude Spectrum of X(t)')
    % xlabel('f (Hz)')
    % ylabel('|P1(f)|')
    % xlim([200 600])
    % ylim([0 .075])

    %find maximum value in the intensity vector from the FFT and it's index
    %ignore anything that is not between 250ish to 530ish
     b = P1(50:120);
     [maximum, maxindex] = max(b);
     feq = round(f(maxindex+49));
    %if maximum is really small, liklely no signal
    if maximum < .001
        fprintf("No Signal\n")
        write(device, "N", "string");
    %print the freqancy of signal
    else
        fprintf("Highest frequancy is %i\n", feq)
        if feq >= 260 && feq <= 263
            write(device, "MC","string");
        end
        if feq >= 292 && feq <= 295
            write(device, "D","string");
        end
        if feq >= 327 && feq <= 331
            write(device, "E","string");
        end
        if feq >= 347 && feq <= 351
            write(device, "F","string");
        end
        if feq >= 390 && feq <= 394
            write(device, "G","string");
        end
        if feq >= 437 && feq <= 442
            write(device, "A","string");
        end
        if feq >= 491 && feq <= 496
            write(device, "B","string");
        end
        if feq >= 520 && feq <= 526
            write(device, "HC","string");
        end
    end
end
%clear device when done
clear device
```

6.4 Bill of Materials

Part	Value	Package	Description
ARDUINO	ARDUINO-NANO-3.0	ARDUINO-NANO-3.0	Arduino Nano 3.0
C1	330n	Through Hole	CAPACITOR
C2	47u	Through Hole	CAPACITOR
C3	120p	Through Hole	CAPACITOR
C4	1u	Through Hole	CAPACITOR
C5	1u	Through Hole	CAPACITOR
C6	1u	Through Hole	CAPACITOR
LED1		LED3MM	LED
LED2		LED3MM	LED
LED3		LED3MM	LED
LED4		LED3MM	LED
LED5		LED3MM	LED
LED6		LED3MM	LED
LED7		LED3MM	LED
LED8		LED3MM	LED
LMC6032		DIL08	OP AMP
M1	ELECTRET_MICROPHONE	CMC-5042PF-AC	Electret Condenser Microphone
R1	5.9k	Through Hole	RESISTOR
R2	100k	Through Hole	RESISTOR
R3	100k	Through Hole	RESISTOR
R4	100k	Through Hole	RESISTOR
R5	100k	Through Hole	RESISTOR
R6	10k	Through Hole	RESISTOR
R7	10k	Through Hole	RESISTOR
R8	220	Through Hole	RESISTOR
R9	220	Through Hole	RESISTOR
R10	220	Through Hole	RESISTOR
R11	1k	Through Hole	RESISTOR
R12	1k	Through Hole	RESISTOR
R13	1k	Through Hole	RESISTOR
R14	1k	Through Hole	RESISTOR
R15	1k	Through Hole	RESISTOR
R16	1k	Through Hole	RESISTOR
R17	1k	Through Hole	RESISTOR
R18	1k	Through Hole	RESISTOR

References

[1] N.A. *Piano key frequencies*. URL: https://en.wikipedia.org/wiki/Piano_key_frequencies. (accessed 11.03.2020).

[2] N.A. *Non-inverting microphone pre-amplifier circuit*. URL: https://www.ti.com/lit/pdf/sboa290. (accessed 10.22.1010).

[3] N.A. *SIMULATION MODEL TLV6741DCKT TINA-TI Spice Model (Rev. A)*. URL: https://www.ti.com/product/TLV6741. (accessed 10.29.2020).

[4] N.A. *Online Tone Generator*. URL: https://www.szynalski.com/tone-generator/.