Project Report

ECE4081-Medical Instrumentation

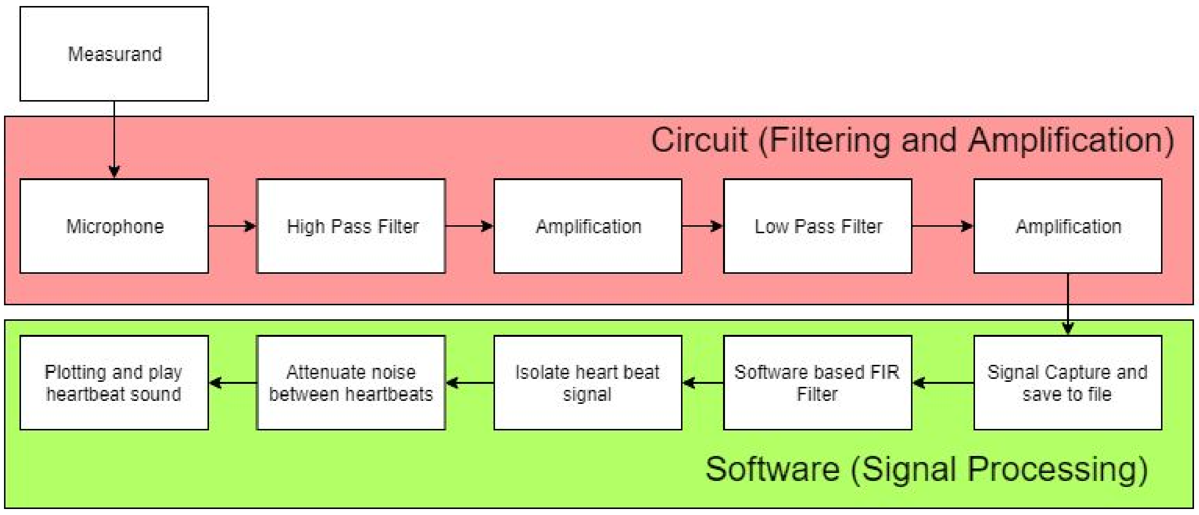
Eric Horng (26935449)

Table of Contents

Aim

The aim of this project was to construct a working microphone system that can pick up the sound of the atrioventricular and semilunar heart valves and determine the heart rate of the patient. The system will run off an Arduino Uno and involve filtering and signal processing in MATLAB.

Block Diagram


Measurand
Circuit (Filtering and Amplification)
Microphone
High Pass Filter
Amplification
Low Pass Filter
Amplification
Signal Capture and save to file
Software based FIR Filter
Isolate heart beat signal
Attenuate noise between heartbeats
Plotting and play heartbeat sound
Software (Signal Processing)

1. Microphone - The sensor we used was an electret microphone connected to a voltage divider. Sound waves are translated into electrical signals through the change of capacitance and hence impedance in the microphone. This is a small AC signal present on the DC operating voltage of the microphone. The amplitude of this AC signal is approximately 1-2 mV for heart sounds.
2. An active high pass filter is used to remove low frequency and DC offset, then amplified.
3. This is then fed into a low pass filter to remove high frequency noise and to further amplify the signal.
4. The output of the low pass filter is sampled by the Arduino ADC, sent over serial to the computer and saved to a text file.
5. The data is fed into a FIR filter to further remove frequencies not in estimated frequency range.
6. Findpeaks is used to isolate the first and second heart valve sounds.
7. Any data points not within the heartbeat segment is attenuated.
8. The heart rate is calculated and the data from the microphone is played over the speaker.

Signal Filtering

The system uses an active low-pass and high-pass filter (effectively a band-pass) to attenuate unwanted frequencies and amplify the signal.

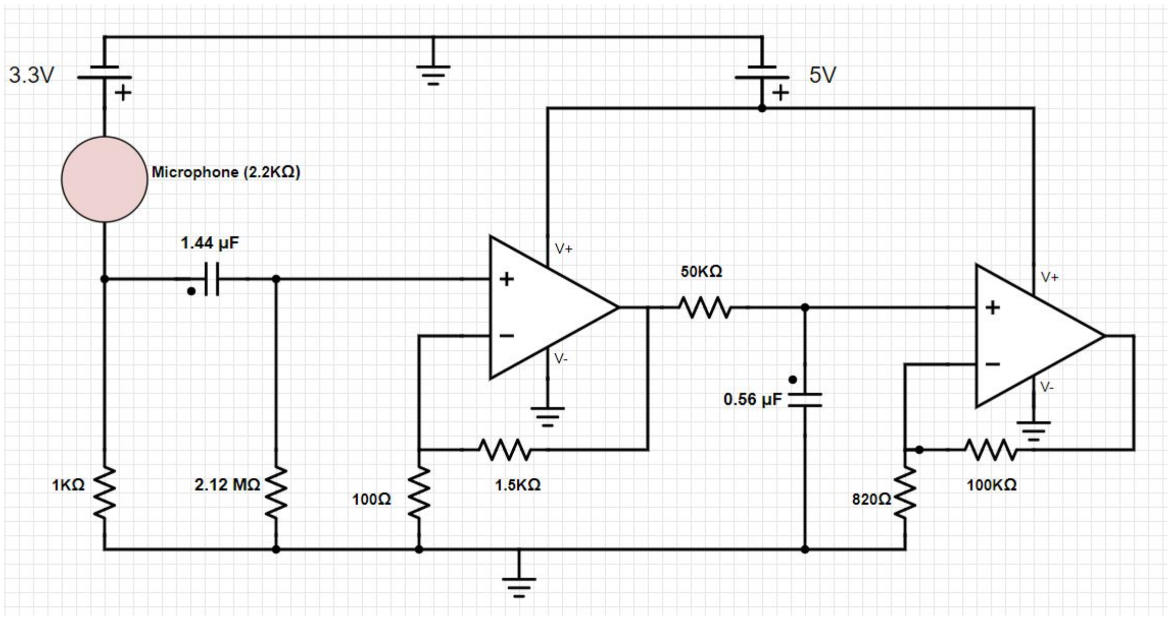


Figure 1. Circuit schematic of device

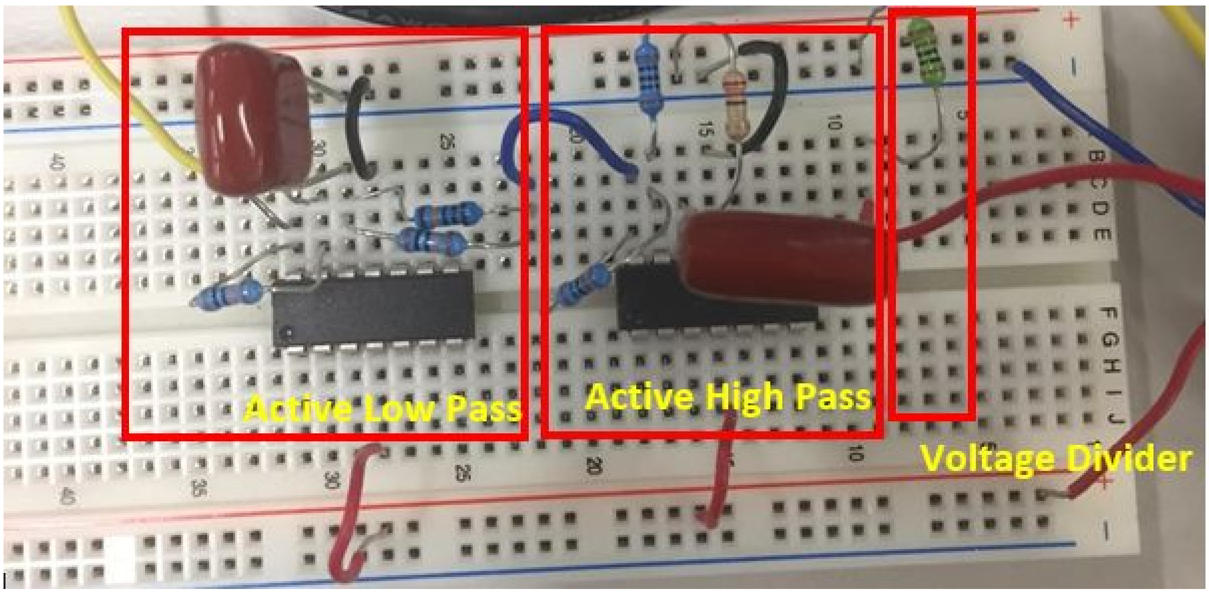


Figure 2. Photograph of constructed circuit

Filtering Calculations

$$Input\ Voltage\ Offset = 3.3\text{V} * \frac{1000}{2200 + 1000} = 1.03V$$

$$High\ Pass\ Desired\ Corner\ Frequency = 0.05\ Hz$$

$$R_{High\ Pass} = \frac{1}{2\pi(0.05)(1.44 * 10^{-6})} = 2{,}210{,}495\ Ohms$$

$$High\ Pass\ Amplification = 1 + \frac{1500}{100} = 16$$

$$Signal\ Range = 1 - 2mV$$

$$High\ Pass\ Signal\ Output = 16 * 2mV = 32mV$$

$$Low\ Pass\ Desired\ Corner\ Frequency = 5\ Hz$$

$$R_{Low\ Pass} = \frac{1}{2\pi(5)(0.56 * 10^{-6})} = 56{,}841\ Ohms$$

$$Freq_{Low\ Pass\ (Practical)} = \frac{1}{2\pi(50{,}000)(0.56 * 10^{-6})} = 5.684\ Hz$$

$$Arduino\ Needed\ Range = 0 - 5V$$

$$Gain\ Needed\ (Low\ End) = \frac{3}{16 * 10^{-3}} = 188$$

$$Gain\ Needed\ (High\ End) = \frac{3}{32 * 10^{-3}} = 94$$

$$Low\ Pass\ Amplification = 1 + \frac{100000}{820} = 123$$

Filtering Considerations

The range of a heartbeat for a normal human is 60-100 bpm, hence the operating frequency is approximately 0.6-1 Hz. 0.05 Hz and 5 Hz were chosen as the corner frequencies to not attenuate around these frequencies. As high frequency noise is more likely to be prevalent in the signal, the low-pass filter stage was placed just before sampling to reduce noise coupled within the circuit. Since the amplitude of the desired signal is incredibly small, amplification is needed. Getting the raw signal of 1mV to the required 0-5V (4.9mV resolution) for the Arduino required a gain of 3000 to get a peak of 3V. The total gain for the constructed circuit is approximately 2000.

Filtering Simulation

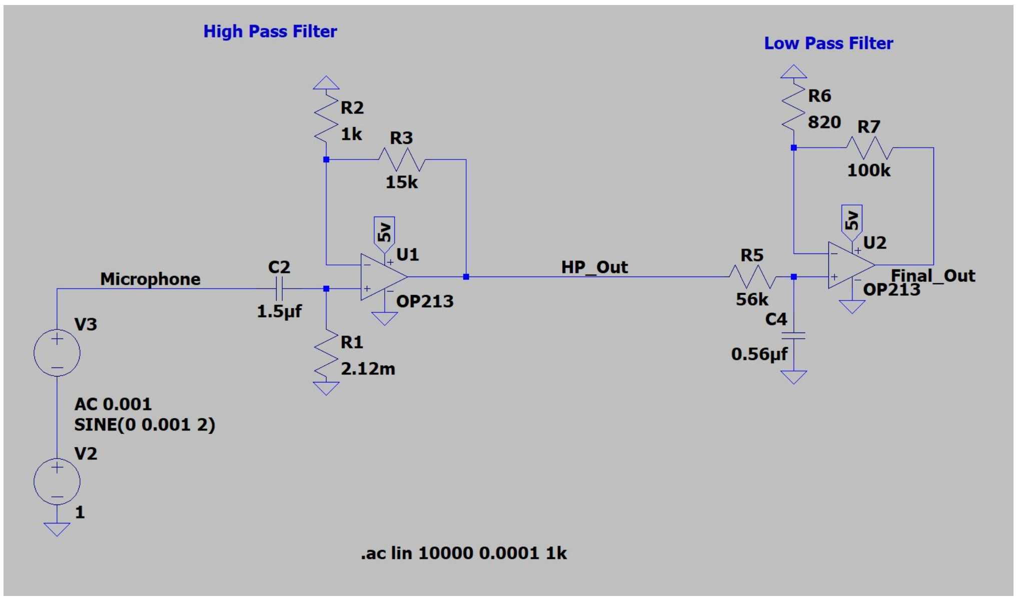


Figure 3. LTSPICE Circuit Diagram

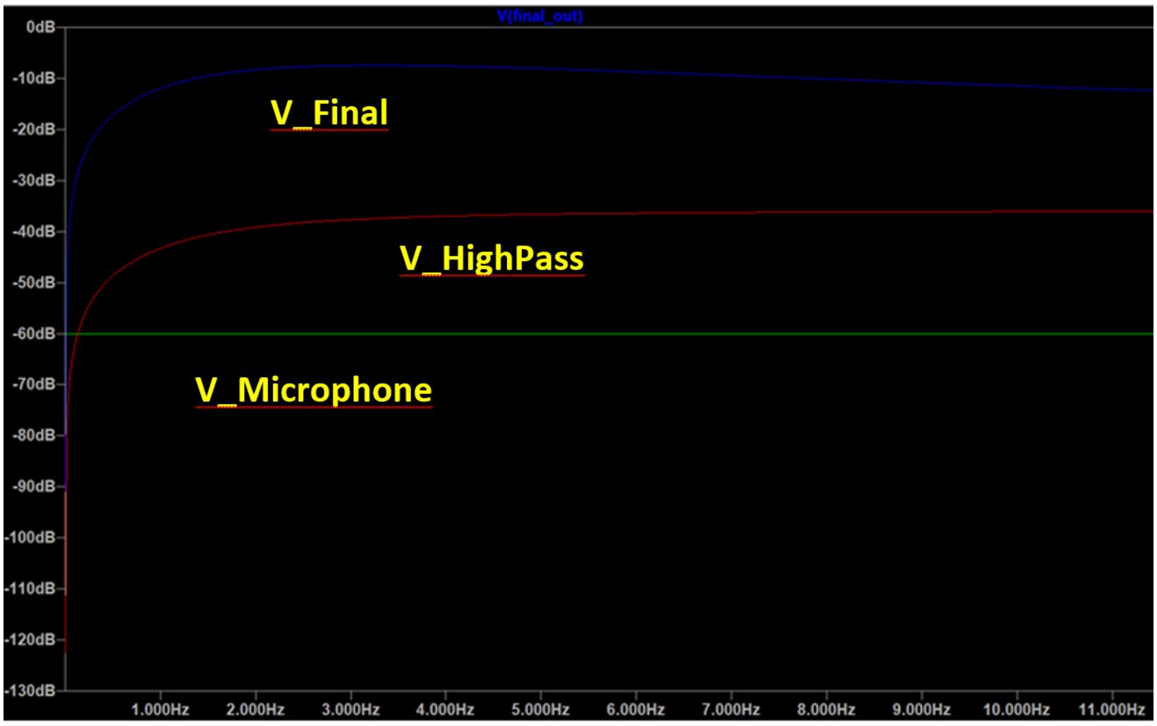


Figure 4. AC Frequency Power Simulation

From the simulation, the circuit has a peak frequency centered around 1Hz, and attenuates frequencies around it. This is what we want as the desired signal is 0.6-1 Hz.

Signal Processing

Signal Capture

The output of the low pass filter was passed to an analog pin of the Arduino. The Arduino was constantly sampling the voltage of the pin at a theoretical rate of 20KHz and sending the data over serial to the computer for signal processing. The process or capturing, sending and saving the data however meant the actual sampling rate of the data was approximately 3.1KHz. Realterm was used to capture the serial data and save to a text file. A capture interval of 10 seconds was used.

MATLAB Processing

The data was imported into MATLAB, converted from ADC to voltage values (Figure 3):

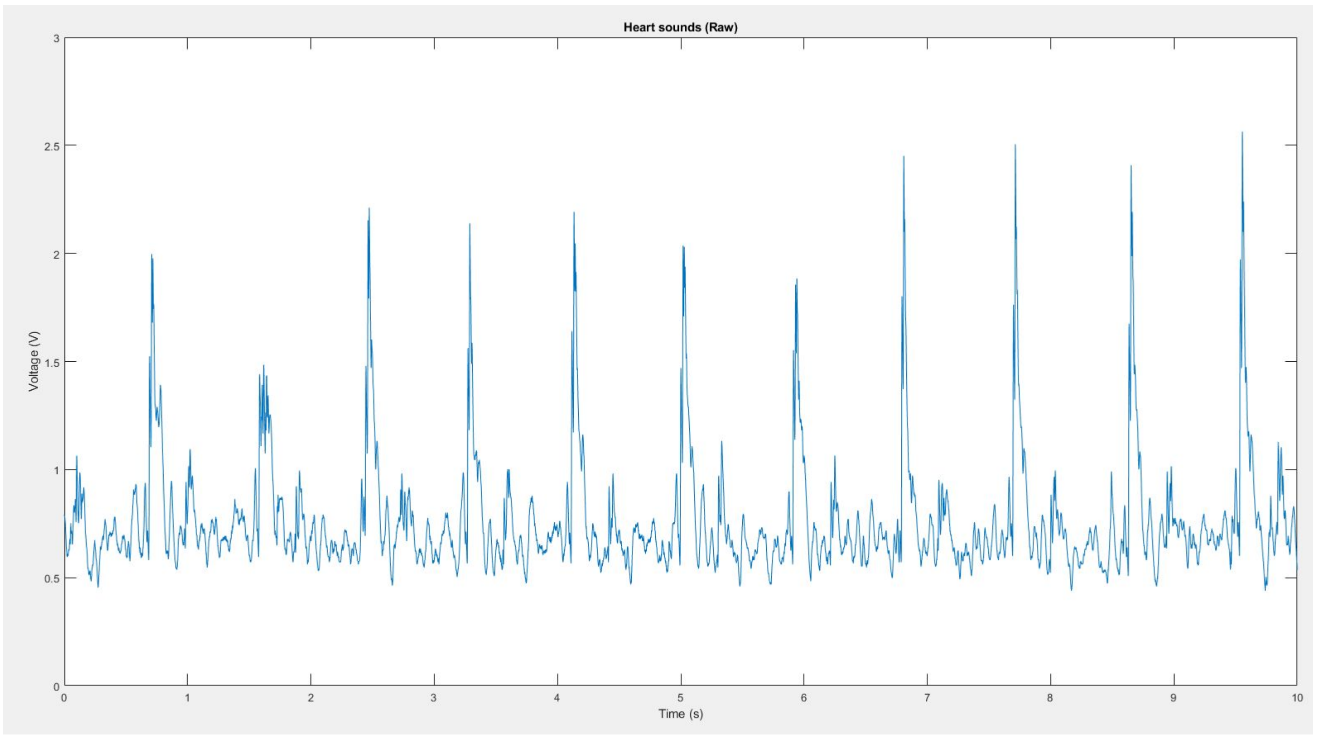


Figure 5. Data before signal processing

The data was first offset to zero by taking the mean of all the data and subtracting this from each data point. A 100 order FIR filter with a cutoff frequency of 3Hz was then used to further attenuate any frequencies not within operating frequency band.This improved the signal quality without affecting the underlying heartbeat sounds. The valve sound locations were identified by using MATLAB’s findpeaks with minimum interval (to not pick up surrounding voltage values) and minimum height parameters. From previous data, the amplitude of atrioventricular valve was typically above 0.6V (after offsetting). This function was used again to find the semilunar valve sounds with a threshold of around 0.2V. Since the signal has a periodic pattern of atrioventricular-semilunar-atrioventricular-semilunar…, we can isolate a heartbeat by taking a atrioventricular-semilunar segment, and attenuating the signal until the next atrioventricular-semilunar segment. The result of signal processing is shown in Figure 6, where the red crosses are identified atrioventricular sounds and blue circles are identified semilunar valves.

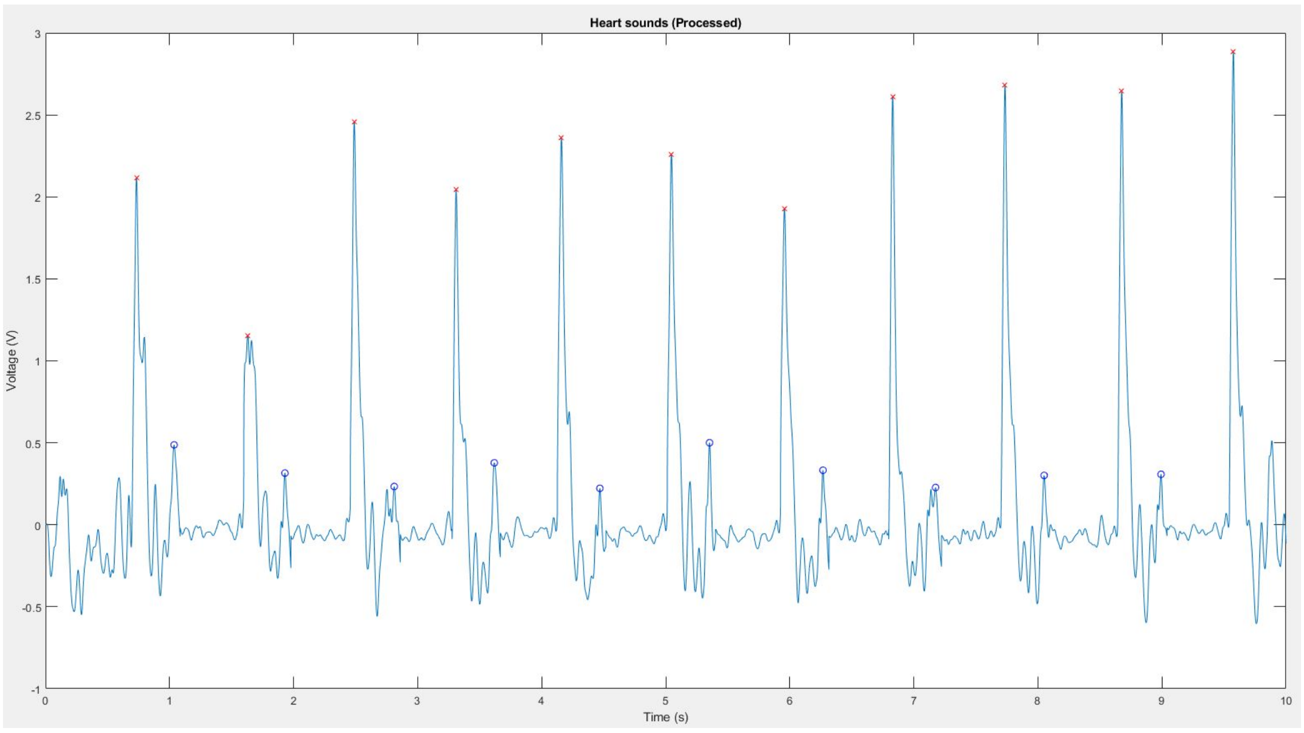
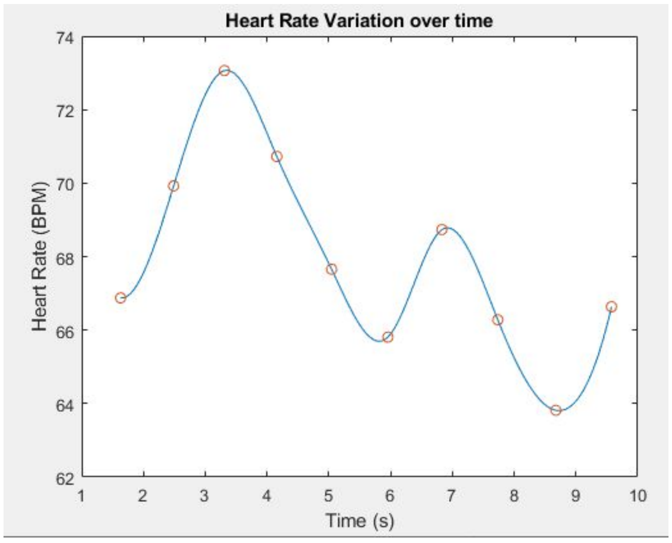
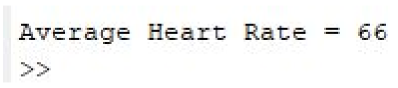


Figure 6. Data after signal processing

The heart rate of the patient can be calculated by counting the number of atrioventricular peaks detected and multiplying by 6, and the variation in BPM can be measured by calculating the time between these peaks. The heartbeat sound can be played from the data by calling soundsc(Data,fs).





```
Average Heart Rate = 66
>>
```

Figure 7. Calculated average BPM and BPM variation between heartbeats

Discussion

Even through all the filtering and processing, we still had some additional frequencies above the low-pass corner frequency of 5.6 Hz and FIR 3.5 Hz as shown in Figure 8.

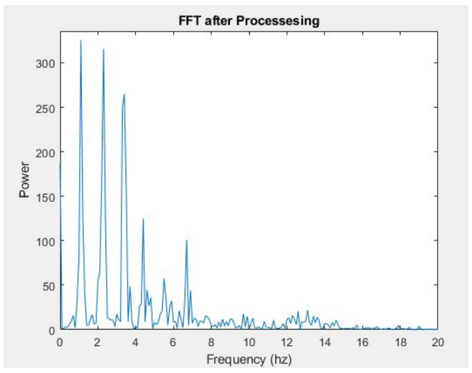


Figure 8. Frequency components after processing

An explanation for this is noise could have entered the signal after the low-pass filter, between the sampling of the Arduino. There could also be discrepancies between resistor and capacitor values, changing the calculated and measured cut-off frequencies. The filters also don’t attenuate all frequencies above the corner frequency to zero immediately- there is gradient of attenuation (which can be controlled using a second order filter). Changing the corner frequencies of the low-pass and high-pass filters to closer to desired frequencies often attenuated the output signal too much, making identification of heart sounds too difficult.

We often found it difficult to get consistent results. There was a lot of variability in the placement of the stethoscope which could affect the data drastically, making it hard for our processing algorithms to detect viable heartbeats. The data was also drastically different between patients due to the variability in body composition and heart strength. Where it was often impossible differentiating heartbeats from noise in the circuit. Even measuring data from the same patient was variable, depending on their recent activity (i.e. if they walked up the stairs recently, or have been sitting relaxed, or recently had a coffee).The voltage data from the circuit often changed between days of usage or locations, possibly a sign of the susceptibility to external noise.

Improvements

Instead of having two amplification stages, there should have been only one, after the low pass filter. Residual high frequency noise could have been entered the circuit before the first stage amplification, resulting in amplified noise. The low-pass filter would still have attenuated the high frequencies, however its power would still be higher than having one stage amplification. By having one amplification stage, the noise would be filtered out first, then the signal would be amplified.

More noise cancellation techniques could have been used in the circuit including:

- Decoupling capacitors could have been placed between the power rails.
- Trim down components and place them closer together.

- Twist wires of microphone lead to eliminate RF coupling.
- Doing measurements in a quieter room.
- Using 2nd order filters
- Dynamic algorithms for finding optimal peak thresholds

The corner frequency for the low-pass filter could have also been lowered from 5.6Hz to around 3Hz to reduce more noise. The amplification could have been increased. From figure 4, we can see the peak voltage reading was around 2.5V. By doubling the amplification we could have utilized the full range of the Arduino's ADC, making it easier to threshold the signal. Using real-time readings could have made finding the optimal placement of the stethoscope easier.

In conclusion, we were able to successfully play and measure human heart beats through the use of a microphone. There are many improvements that can be made, mainly through better filtering and noise cancelling techniques.

Appendix

MATLAB Code

```
Data=importdata('C:\temp\capture.txt');
SamplePeriod=1/(length(Data)/10);
fs=1/SamplePeriod;
Time=(1:length(Data))*SamplePeriod;
figure;
plot(Time,Data*(5/1024))
ylabel('Voltage (V)')
xlabel('Time (s)')
title('Heart sounds (Raw)')
%Offset voltage to zero
Offset=mean(Data);
Data=Data-Offset;
Data=Data*(5/1024);
%FIR Filter
f = [0 (2*pi*3)/fs (2*pi*3.5)/fs 1];
m = [1 1 0 0];
b1 = fir2(100,f,m);
a = 1;
Data = filter(b1,a,Data);
Data = Data*8; %Scale Signal Up because of low pass filter attenuation
%Locating First and Second Valve Sounds
[ ~ , LocHeartbeat]=findpeaks(Data,'MinPeakDistance',0.2/SamplePeriod,'MinPeakHeight',0.55);
[ ~ , Loc2]=findpeaks(Data,'MinPeakDistance',0.2/SamplePeriod,'MinPeakHeight',0.2);
%Find Missing Beats
BeatSpacing(1) = LocHeartbeat(2) - LocHeartbeat(1);
for i = 1:(length(LocHeartbeat)-1)
   BeatSpacing(i) = LocHeartbeat(i + 1) - LocHeartbeat(i);
   if BeatSpacing(i) > 1.5*mean(BeatSpacing)
      n = LocHeartbeat(i+1) - BeatSpacing(i)/2;
      a = abs(Loc2 - n);
      a(a==0) = inf;
      [val,idx]=min(a);
      LocHeartbeat = [LocHeartbeat(1:i); Loc2(idx) ;LocHeartbeat(i+1:end)];
```

```
    end
end
%Refreshing Beat Spacing
PeaksHeartbeat = Data(LocHeartbeat);
for i = 1:(length(LocHeartbeat)-1)
    BeatSpacing(i) = LocHeartbeat(i + 1) - LocHeartbeat(i);
end
%Detecting Second Valve
for i = 1:(length(LocHeartbeat)-1)
    [Peak2(i), locValve(i)] = max(Data((LocHeartbeat(i)+round(mean(BeatSpacing)*0.15)):(LocHeartbeat(i + 1) -
round(mean(BeatSpacing)*0.15))));
    locValve(i) = locValve(i)+ (LocHeartbeat(i)+round(mean(BeatSpacing)*0.15));
end
%Attenuating in Between HeartBeats
for x=1:length(locValve)
    Data((locValve(x)+150):(LocHeartbeat(x+1)-100))= Data(locValve(x)+150:LocHeartbeat(x+1)-100)*0.25;
end
%Plotting and sounds
figure;
plot(Time,Data)
ylabel('Voltage (V)')
xlabel('Time (s)')
title('Heart sounds (Processed)')
%soundsc(Data,fs)
hold on
plot (LocHeartbeat*SamplePeriod,PeaksHeartbeat,'rx')
plot(locValve*SamplePeriod,Peak2,'bo')
%Calculating Heart Rate
BPM = 60./(BeatSpacing*SamplePeriod);
xTime = LocHeartbeat(2:end)*SamplePeriod;
%Plotting BPM
figure
xi = linspace(min(xTime), max(xTime), 150);  % Interpolation to lessen the effect of noise
yi = interp1(xTime, BPM, xi, 'spline', 'extrap');
plot(xi,yi);
title('Heart Rate Variation over time')
xlabel('Time (s)')
ylabel('Heart Rate (BPM)')
hold on
plot(xTime,BPM,'o')
%Frequency Analysis
figure
y = fft(Data);
n = length(Data);
f = (0:n-1)*(fs/n);
power = abs(y).^2/n;
plot(f,power);
axis([0 20 0 max(power)+10 ]);
title("FFT after Processesing");
xlabel("Frequency (hz)");
ylabel("Power")
fprintf("Average Heart Rate = %i\n",length(LocHeartbeat)*6)

soundsc(Data,fs)
```