Mobile ECG Measurement Device

Final Technical Report

Group 7: Adam Karaien (akaraien), Jesse Hu (jessehu), Megi Shahollari (mshaholl), Ross Richards (rossrich), Sartaj Chowdhury (sartajc)

Department of Electrical Engineering, University of Michigan, Ann Arbor, MI

EECS 452 - Winter 2024

April 24, 2024

Table of Contents

Background and Motivation..... 3
Materials..... 3
High-Level Overview..... 3
Technical Issues..... 5
Testing Methodology..... 6
Results..... 8
Recommendations for Future Work..... 8
References..... 10

Background and Motivation

Rapid advancements in signal processing have greatly influenced the health industry, especially in the development of electrocardiograms (ECG). An ECG is a medical test that measures the electrical activity of the heart using small electrodes that attach to a patient's skin; however, the original signals coming off the electrodes are very faint and noisy and need processing. To achieve the level of signal processing required, modern ECG machines are both bulky and expensive. The devices need their own carts for transportation, and the sport price tags of several thousand dollars. The goal of our project was to design and build an ECG device that is both mobile and inexpensive while still providing an accurate read of the patient's heart activity.

Materials

- ESP-32-WROVER E
- AA Batteries - 8
- 6V AA Battery Pack - 2
- AD620 - 3
- 100 Ohm Resistor - 3
- Breadboard - 3
- Hydrogel Foam Electrodes - 3
- Wires

High-Level Overview

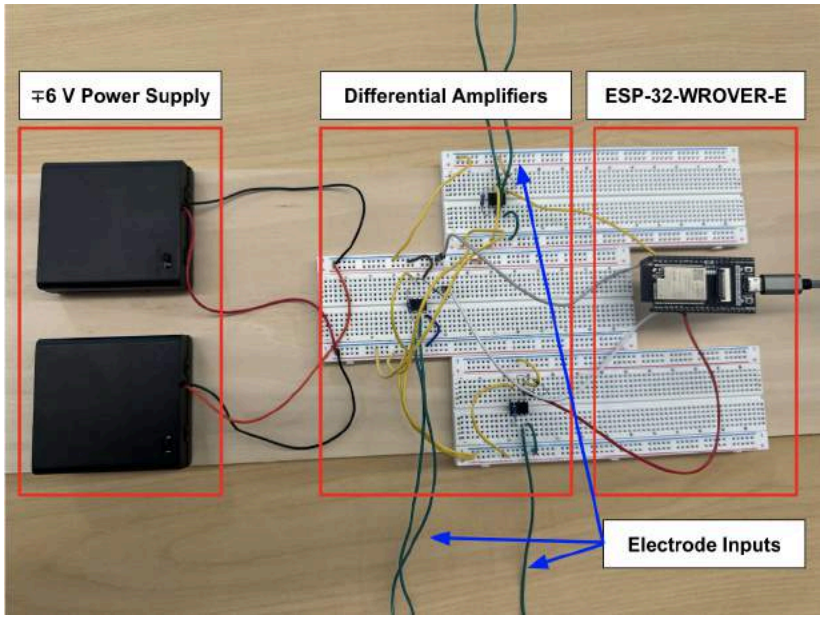

Figure 1. *The ECG device. The differential amplifiers, power supply, electrode inputs and ESP-32 are shown. The ESP-32 is connected to the device running the Python plotting script.*

Within our design, we had two main signal processing components: the analog circuit and the software on the ESP-32. The first step of our hardware design process involved collecting the raw ECG signal from the electrode leads and inputting it into our analog circuit. We had built

three circuits to follow Einthoven's law. Einthoven's Law states that lead I + lead III = lead II [1]; thus, the deflection in one lead can be predicted from the deflections in the other two. As shown in Figure 1, each circuit board had an AD620 instrumental amplifier, which was responsible for one individual lead [2]. The AD620 is great for improving signal-to-noise ratio, and it is very user-friendly, as you just need one external resistor to set the gain. In our design, we used a 100-ohm resistor to achieve a gain of 500, which gave us the highest amplification possible while staying within the 0-3.3V sampling range of the ESP-32 [3].

The software portion of our project was implemented on the Arduino IDE and flashed on the ESP-32-WROVER-E. After our ECG signals were amplified with the analog circuit, we sampled them using the ADC of the ESP-32. Because the frequency range of the signal was 0.5–150 Hz, we sampled the signal at 300 Hz to obey the Nyquist–Shannon sampling theorem. After sampling the signal, we applied an IIR bandpass filter using the Arduino Filters library [4]. The bandpass filter's coefficients were generated using the Filter Designer from MATLAB and had a low cutoff frequency at 0.5 Hz and a high cutoff frequency at 150 Hz. This was done intentionally to remove the inherent high-frequency noise and the low-frequency noise from movement while maintaining the integrity of the signal that exists from 0.5 to 150 Hz. The bode plot of this bandpass filter can be seen in Figure 2. After the signal was filtered through the bandpass filter, we then ran it through a 60 Hz notch filter. The notch filter was set to remove 60 Hz as it is a common noise source, which could have been picked up from our breadboards. To design our filter, we used the fir1 MATLAB function to generate a 201 tap FIR notch filter with a notch width of 2. We analyzed the bode plots of notch filters of various lengths and selected 201 taps because they reduced the noise at 60 Hz by ~27 dB while not significantly attenuating frequencies outside of the stopband, as shown in Figure 3.

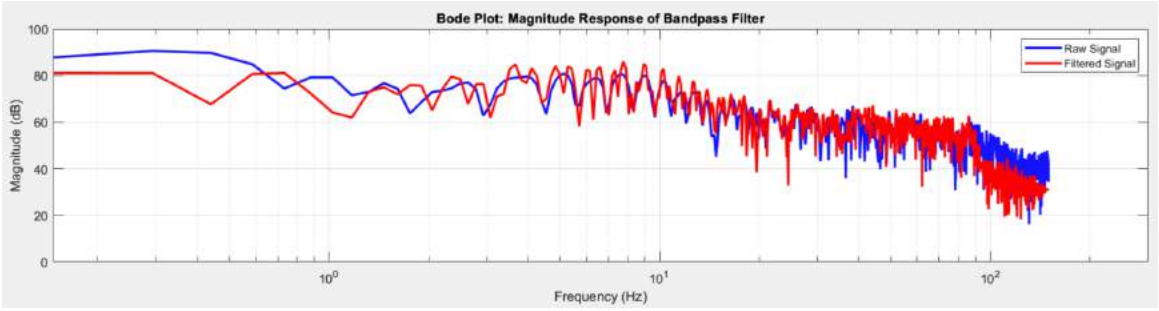

Figure 2. *Bode plot of our IIR bandpass filter.*

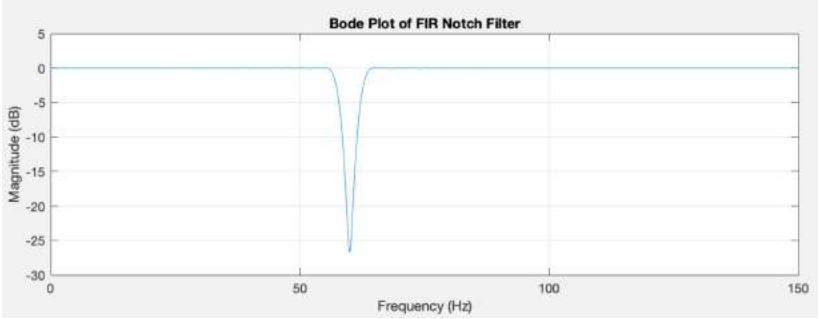

Figure 3. *Bode plot of our FIR notch filter.*

The final stage of our design process was our peak detection algorithm and the subsequent heart rate calculation script. We implemented the Pan Tompkins Algorithm, which is an algorithm that specializes in detecting the R-peaks of the QRS complexes in the ECG signal [5]. It involves performing a series of differentiation, squaring, and moving window integrations to accentuate the peaks in the signal. Afterwards, a threshold estimation function is applied to count R-peaks and continuously update the noise and signal thresholds of the signal as the data is processed. Finally, we use the number of peaks counted from the Pan-Tompkins algorithm and measure the distance between the peaks in samples to calculate the test subject's heart rate [6].

Technical Issues

We had a slew of technical issues that we encountered while building our ECG device. Our original plan was to build an analog circuit that incorporated both signal amplification and bandpass filtering. This would make the signal passed into the ESP-32 have its wandering baseline removed, and we wouldn't have to expend processing time on software bandpass filtering [7] and [8]. The problem was that the bandpass filter we created in hardware did not work properly, and it essentially made the signal too noisy to interpret. We rebuilt the bandpass filter multiple times without seeing the correct attenuation, so we pivoted to software bandpass filtering. We found a filtering library that allowed us to implement the bandpass filter in our software without violating our timing. This allowed us to read data at our 300 Hz sampling frequency and output fully processed data at the same rate.

For anyone trying to replicate our work, it is a good idea to try and implement all filters in software rather than a physical circuit. This isn't just because the physical circuits are more complicated to get right, but also because breadboards are like antennas, and they pick up large amounts of noise. We also weren't using shielded wires for any of our circuit testing. We were using jumper cables and feeder wires. These pieces of hardware are not impermeable to outside noise, so implementing the filters in software to reduce the amount of hardware that could pick up noise is the best bet.

Another issue we had was getting our serial data plotted in real time. To plot our data, we were using a Python script that read in serial data that our microcontroller was outputting. There was talk of trying to maybe get the data through a Wi-Fi channel because we switched from the Arduino R3 to the ESP-32, but we already had a semi-working script when we changed microcontrollers, so we decided to keep using wires. The problem with plotting in real time was that we were outputting data at a rate of 300 samples per second, and the library we were using, matplotlib, was not the greatest in terms of speed. An early version of the code that purely plotted the data with no optimizations plotted 0.1 seconds of data in one real-time second, and everything was way too delayed for a live demo. Once we realized the shortcomings of the matplotlib library, we tried to implement blipping, deques, and batches to help improve the plotting speed. Only when we implemented deques and blipping and then implemented the FuncAnimation library were we able to effectively plot in real time with a moving window. Getting our real-time plotting working was crucial to the success of our project because it allowed us to view our signal in real-time without an oscilloscope.

When implementing software filtering, it is important to create filters that have a fast transition from the passband to the stopband frequencies so that the unwanted frequencies are properly filtered out. To do so, we used MATLAB to generate filters, tested each filter on the collected

test data, and used bode plots to verify the quality of the filter. We found that elliptic or Butterworth IIR bandpass filters gave us better results as opposed to FIR bandpass filters. To implement the IIR bandpass filter into the software, we used the Arduino-Filters library to apply the filter coefficients. One thing to look out for when using the IIR filters is testing the coefficients not just in MATLAB but also in the microcontroller, as there is a high possibility of integer overflow or undefined values appearing when the filter is applied to the data. We fixed this issue by lowering the order of the filter as much as possible and keeping the attenuation lower so that the filter calculations don't exceed the maximum value of the data types. By doing this, we were able to stay within the bounds of our data type without sacrificing the effectiveness of the filter.

Testing Methodology

As for testing, we tested each component individually prior to integration. For the hardware portion, we used the oscilloscope as our input to the AD620 instrumental amplifier. We used the wave generator and set it to output a cardiac signal with an amplitude of 20 mV. The 20 mV was done intentionally, as it was the closest amplitude we could obtain that resembled a real cardiac signal. We connected the ground to the inverting input of the AD620, and we connected the positive signal to the non-inverting input of the AD620. The purpose of this was to verify the gain of the circuit and to observe the relative expected output of our circuit once we began testing with electrodes.

When we began testing with the electrodes, we started off by displaying the output on the oscilloscope. This was mainly to ensure that our circuit was actually reading data from the electrodes. Initially, we were using TENS/EMS electrodes, but we noticed we were not getting a clear output reading. This could be due to the fact that we had to shove a wire into the electrode, and the wire would frequently pop out. Regardless, it was not suitable for what we were trying to accomplish. We then proceeded to use hydrogel foam electrodes because they had a metal head and would give us a more secure connection with the use of alligator clips. We attached our electrodes to the circuit with alligator clips because we could not solder wires to the metal heads without melting the electrode pads. When testing our circuit with electrodes connected to a human test subject, we used the same person from trial to trial and attempted to keep our electrode placement the same. We did this because each person has a slightly different heart rate signal. We attached our electrodes to the circuit with alligator clips because we could not solder wires to the metal heads without melting the electrode pads, and we wanted to keep our testing as consistent as possible.

To test our software before we had access to live ECG data, we utilized raw ECG data from this database [9]. After creating our notch filter coefficients from MATLAB, we applied the filter to our example ECG data and compared the bode plot of the filtered data to our expected bode plot on MATLAB to verify that they were the same and the correct attenuation was occurring solely in the 60 Hz stopband. We followed a similar procedure to test our software bandpass filter, except we looked for zero attenuation in the 0.5–150 Hz passband and attenuation outside of the passband to verify that our bandpass filter was working properly.

Because we wrote our own derivative, squaring, and moving window integration functions for the Pan-Tompkins algorithm, we compared the output of our functions to theoretical values priced from MATLAB functions to ensure that our functions were producing accurate results. To

do this testing, we applied our Pan-Tompkins functions to an example ECG signal and plotted the resulting derivative, square, and moving window integration data from the serial output of the ESP-32. Our practical results from the Pan-Tompkins algorithm are shown in Figure 4.

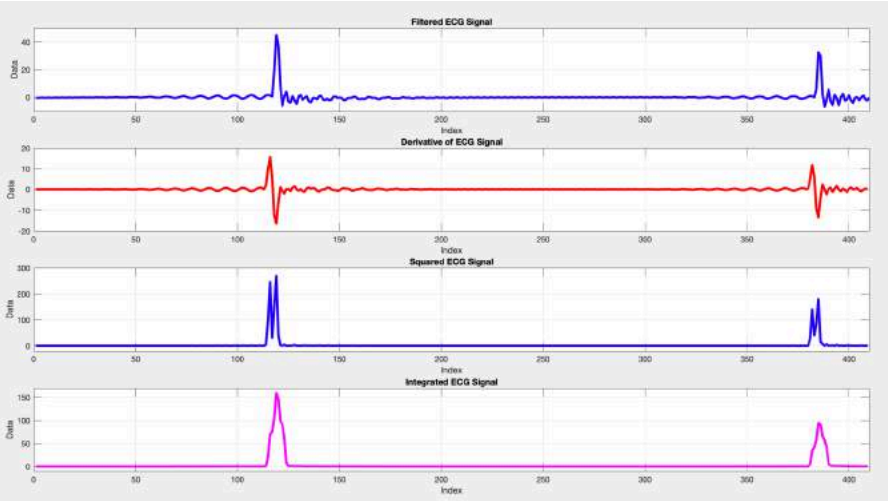

Figure 4. *Recorded filtered ECG Signal and the resulting differentiated, squared, and integrated data.*

We did not have access to live ECG data, so we used example ECG signals from the aforementioned database to test our threshold estimation and peak detection function from the Pan-Tompkins algorithm. We tested on data with several clear peaks and tuned our initial constants, such as the noise peak and signal peak, based on the characteristics of the signal. We tuned our initial conditions for the thresholds until we were able to consistently count each of the peaks in six sets of example signals. We also hand-calculated the heart rate based on the sample interval between the peaks of the example data to check the accuracy of our heart rate calculations.

To test our live plotting, we used a recorded signal and a loop to repeatedly output serial data for our Python script to plot. This would ensure that we were testing on good data, so we could focus entirely on the speed and quality of the live plotting. There were a few things that we looked for that would immediately tell us if the plotting was working or not. The first characteristic is whether the window was traversing at the correct rate. Then, we looked at whether we were getting the correct number of samples plotted in the window size we allotted. There were a few versions of our code that traversed in real time, but only plotted around 30–50 of the 300 samples being fed through the serial monitor every second. The window traversal speed was correct, but the data wasn't being plotted quickly enough. Lastly, we looked at the quality of the signal being plotted. Our data fluctuated from point to point, regardless of the fact that we were sampling at 300 Hz. If the real-time plot had any horizontal lines in between the consecutive data points, we knew that there was aliasing occurring.

In the end, when both software and hardware were integrated together, we were mainly testing the positions of the leads on the human subject. The main test was how well we should be grounded since we use a 3-lead system but have a fourth leading for common ground. Some interferences were different types of shoes and whether having the shoes on or off would give us

the best result. Of course, this was all tested in an actual lab, so we needed to do one final test in the actual setting of our mock expo. This was mainly concerning since the room is carpeted, unlike the lab. We mainly tested how the grounded lead would be affected and ended up going with shoes on since it still worked and gave us good results.

Results

Our project aimed to design and build a mobile and effective ECG device capable of accurately measuring heart activity. We successfully integrated an analog circuit and software components on the ESP-32 platform to process raw ECG signals. The analog circuit, utilizing Einthoven's Law and AD620 instrumental amplifiers, effectively amplified the signals with a gain of 500, improving the signal-to-noise ratio. In software, we implemented a bandpass filter and a 60 Hz notch filter to remove noise while preserving the signal. The Pan-Tompkins algorithm for peak detection and heart rate calculation further enhanced our system's functionality. Our testing method, including oscilloscope readings, simulated ECG data testing, and real-time plotting validation, ensured the system's reliability and performance.

In the end, our results were very positive. Our system was able to pick up and plot the subject's periodic cardiac signal as shown in Figure 5, and our peak detection/heart rate calculation scripts worked on recorded data perfectly but had some trouble with real-time calculations due to the wandering nature of the signal. Realistically, we did not expect to have a perfect cardiac signal like what they may have in hospitals and higher-level industries, but we were able to at least show a conclusive result of our subject’s signals being read in and displayed.

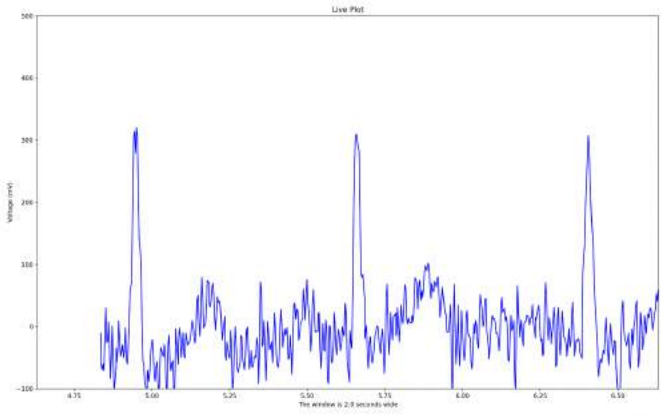

Figure 5: *Live plotting of ECG signal from our electrodes.*

Recommendations for Future Work

In the preliminary stage of our project, we realized the importance of a more strategic approach to ordering parts. We dealt with both supplier and shipping issues, and this became especially problematic with a third party placing orders on our behalf. Ideally, we believe that handling the orders personally would have reduced the possibility of delay. Additionally, we also should have been more diligent about verifying that the parts we ordered were the exact items we wanted, rather than just the individual parts of the components. This is a common error that could occur in any project group and has the potential to add further delays to project schedules.

For improving efficiency in future projects, we recommend early stage-by-stage testing during the development process. This would involve testing each component separately before integrating them with other components. Doing this would have reduced the time spent identifying and resolving issues that arose during integration. Our original plan was to test each individual component more thoroughly before integration, but this was not possible due to getting our parts later than we wanted. By completely debugging each part before integration, we would have had more time for testing the main components, such as the electrodes.

Additionally, to improve efficiency, we should have conducted more research to understand the expected signal characteristics when passing our raw signal through the AD620 amplifier. In the initial phase of our project, there was uncertainty regarding the proper visualization of our actual signal. It was occasionally difficult for us to differentiate between signals that were only noise and very noisy ECG signals, and this caused us to waste time analyzing signals that were just noise. Looking back, it would have been beneficial to study the AD620 amplifier in more detail to avoid the issue of analyzing noise.

In terms of software, our biggest area for improvement was our threshold estimation algorithm for peak detection. While our peak detection and the resulting heart rate calculations worked on test data, they had trouble working properly on live data as it was far too erratic for the algorithm to keep up with. Our peak detection function used a threshold estimation algorithm that adapted to the signal over time, but outlier behavior in the signal from movement would throw off the thresholding, which caused our function to ignore future signal peaks. For future work, we recommend using a peak detection method that is different from the one we used in the Pan-Tompkins algorithm. A more robust threshold estimation algorithm that is less sensitive to disturbances in the signal is needed in order to accurately detect peaks from live ECG data.

References

[1] R. Pura, “Einthoven’s Triangle: A Piece Of Medical Device History,” *ClearPath Medical*, Oct. 27, 2022. https://clearpathmedical.com/medical-device-history-einthovens-triangle-ecg-electrocardiogram-willem-einthoven/

[2] “AD620 Datasheet and Product Info | Analog Devices,” *Analog.com*, 2016. https://www.analog.com/en/products/ad620.html

[3] rcthurston35, “Automated ECG: Amplification and Filter Simulations Using LTspice,” *Instructables*. https://www.instructables.com/Automated-ECG-Amplification-and-Filter-Simulations/

[4] P. P, “tttapa/Arduino-Filters,” *GitHub*, Apr. 05, 2024. https://github.com/tttapa/Arduino-Filters

[5] J. Pan and W. J. Tompkins, “A Real-Time QRS Detection Algorithm,” *IEEE Transactions on Biomedical Engineering*, vol. BME-32, no. 3, pp. 230–236, Mar. 1985, doi: https://doi.org/10.1109/tbme.1985.325532.

[6] T. Moller, M. Voss, and L. Kaltwasser, “An Arduino based heartbeat detection device (ArdMob-ECG) for real-time ECG analysis,” *Researchgate*, Apr. 01, 2022. https://www.researchgate.net/publication/359709737_An_Arduino_based_heartbeat_detection_device_ArdMob-ECG_for_real-time_ECG_analysis (accessed Apr. 21, 2024).

[7] R. Khushaba, “Removing Baseline Wandering,” *www.youtube.com*, Dec. 06, 2022. https://www.youtube.com/watch?v=rRdmrY5e-N8 (accessed Apr. 21, 2024).

[8] Y. Luo *et al.*, “A Hierarchical Method for Removal of Baseline Drift from Biomedical Signals: Application in ECG Analysis,” *The Scientific World Journal*, vol. 2013, no. 896056, p. e896056, May 2013, doi: https://doi.org/10.1155/2013/896056.

[9] “A large scale 12-lead electrocardiogram database for arrhythmia study v1.0.0,” *physionet.org*. https://physionet.org/content/ecg-arrhythmia/1.0.0/WFDBRecords/01/010/#files-panel (accessed Apr. 21, 2024).