CONCEPTUAL DESIGN OF A TEST BED FOR MINER RESCUE

By

Rowshon Ara Mannan Munny, B.Sc.

A Project Submitted in Partial Fulfillment of the Requirements

For the Degree of

Master of Science

In

Electrical and Computer Engineering

University of Alaska Fairbanks

August 2019

APPROVED:

Dr. Michael Hatfield, Committee Chair
Dr. Richard Wies, Committee Member
Dr. Katrina Bossert, Committee member
Dr. Richard Wies, Chair
Department of Electrical and Computer Engineering

Table of Contents

1. Abstract

In the mining industry, miners are constantly exposed to various safety and health hazards associated with often unpredictable conditions. When an accident occurs, it is difficult for the rescue team to come up with a proper plan for the rescue mission without having adequate knowledge of the situation. One possible approach to managing these hazards is to provide the rescue team with situational awareness such as real-time data regarding the environment (fire, poisonous or explosive gasses), as well as the location and physical condition of the trapped miners. Before starting the rescue mission, and in order to eliminate or reduce the dangers of exposing more humans to the explosive mining environment for information collection, a combination of unmanned ground vehicles (UGVs) and unmanned aerial vehicles (UAVs) is proposed. In this project, a conceptual test bed is designed to collect one specific set of information about a trapped miner (in this case, heartrate data). This test bed collects the required data from a heart rate sensor on the trapped miner and transmits it wirelessly to a nearby UAV which will receive the data and send it back to the rescue team via a UGV.

2. Introduction

Mining accidents have been occurring from early days, particularly in underground mining. A total of 455 fatalities were reported in accidents that occurred from 2007 through to 2017 in the USA in both metal/nonmetal and coal mines [1]. In a hostile environment, most of the time these accidents involve a specially trained mine rescue team (MRT) who are also miners [2]. Heavy smoke, explosive gas and unstable ground conditions are some of the barriers that the MRT often encounters. These important environmental parameters, along with data on the health conditions of the trapped miner can help to inform critical and timely decisions as to where assistance is most needed considering the resources available.

Reliable communication is also required to ensure the data collected from the miner is received correctly to the MRT. As breakage of fiber optic cable is highly likely in such a chaotic environment, wireless communication will play a vital role in this situation [3]. To reduce the risk of human exposure in this dangerous environment, a system using an unmanned ground vehicle (UGV) and unmanned aircraft vehicle (UAV) is developed as shown in Figure 1 [4].

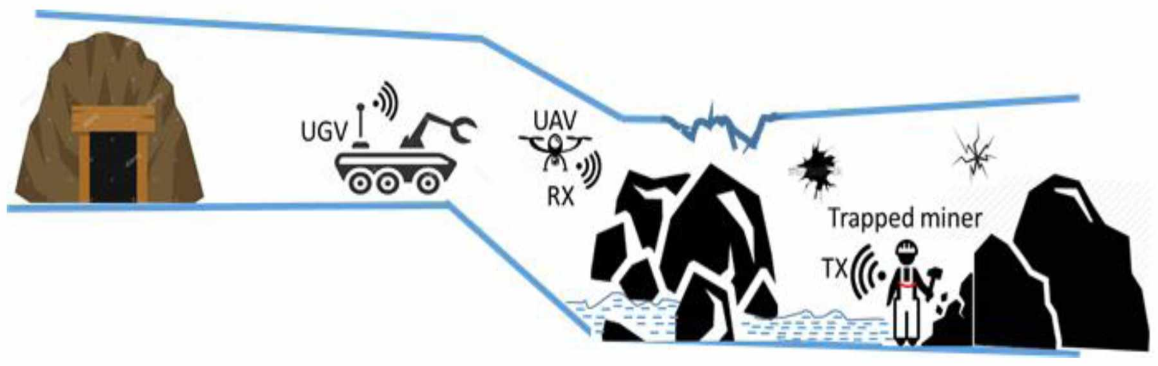


Figure 1: Trapped miner heart rate sensor system overview.

3. System Architecture

The overall system (see functional block diagram in Figure 2) consists of three sensors: gas, sound and heart rate. Within the scope of this project, only the heart rate detection sensor is implemented. Two Raspberry Pi 3 B+ are used to transmit and receive data and a micro SD card is used to store the received data. There are two separate parts of the design. The trapped miner holds or wears the transmitter portion of the test bed which has a heart rate sensor, one Raspberry Pi 3B+ and a micro SD card. The receiver portion of the system consists of a Raspberry Pi 3 B+ and micro SD card, which is attached to the UAV. Upon completion of the data transfer from the trapped miner to the UAV, the UAV returns to the ground station or UGV where the MRT personnel will collect and analyze the data.

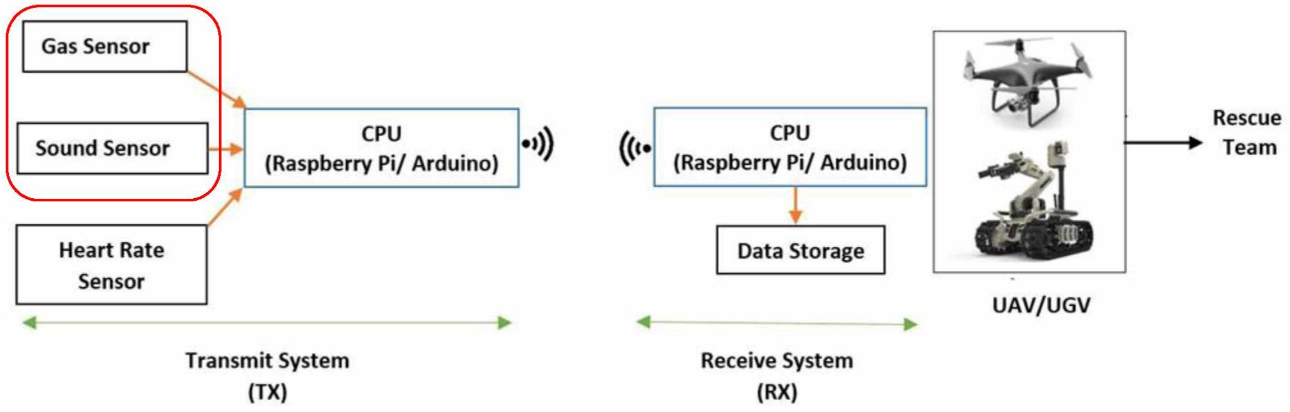


Figure 2: Functional block diagram of the test bed (red block is a conceptual portion).

4. Electrocardiogram

The test showing the electrical activity of the heart is known as an electrocardiogram (ECG). An ECG pulse can be divided into two intervals as illustrated in Figure 3. One is the PR interval and the other is the QT interval. The PR signal is created when the heart receives deoxygenated blood from the body through veins (see red sections in Figure 4), and the QT signal is generated when the heart pumps oxygenated blood to the body through arteries (see blue sections in Figure 4). There are three significant segments in an ECG pulse. First, the P wave segment represents the depolarization of the atria. Second, the QRS complex segment represents the depolarization of the ventricles. Lastly, the T wave segment represents the repolarization of the ventricles. For beat per minute (BPM) calculation, the segment of interest is the QRS complex. Here R is the peak of the QRS complex in Figure 3 (a) seen in the yellow section and the RR interval is the distance between consecutive R peaks shown in Figure 3 (b).

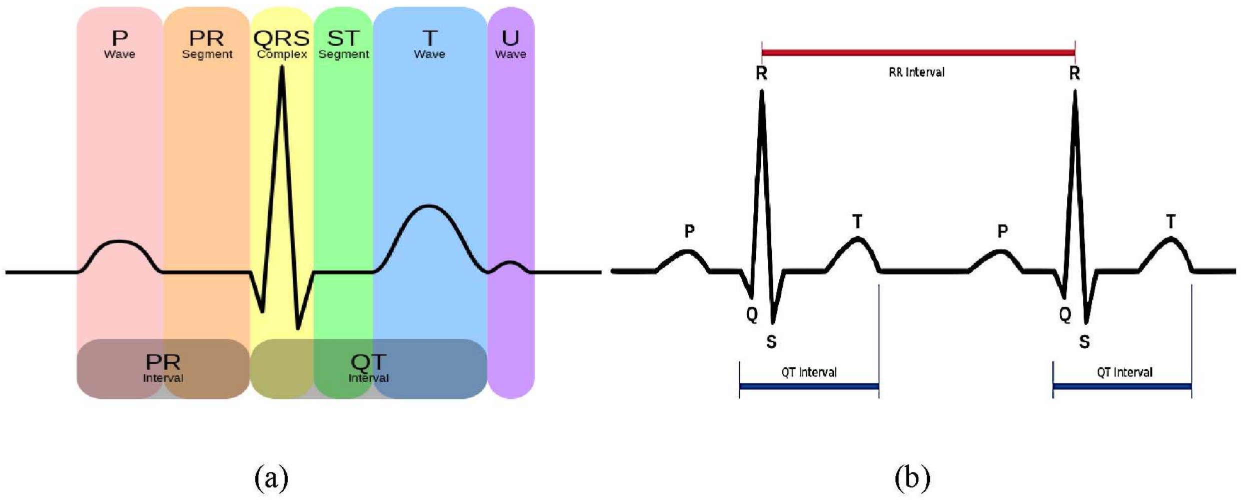


Figure 3: (a) Segments of an ECG pulse [5], (b) RR interval [6].

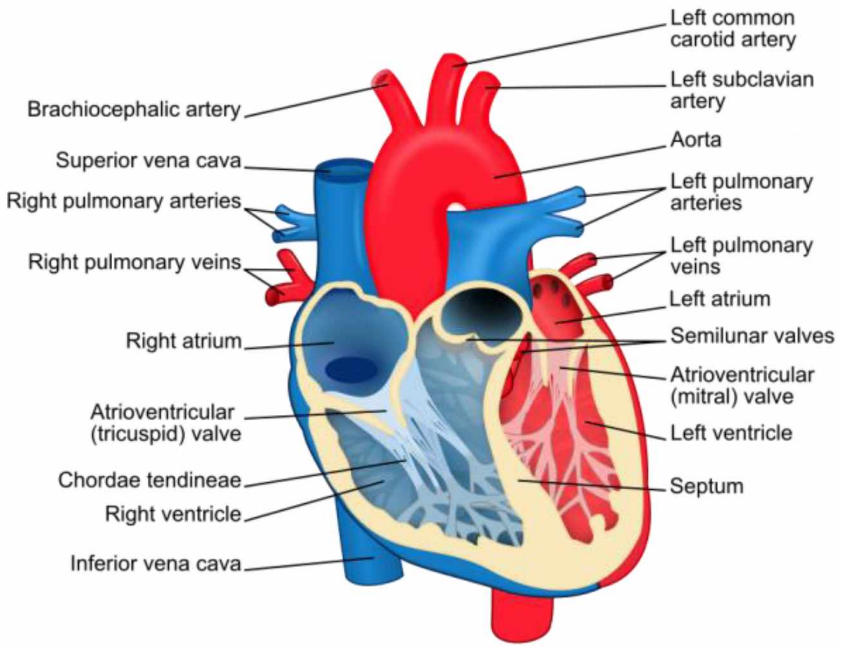


Figure 4: Cross section of a human heart showing atria (blue) and ventricles (red) [5].

5. Hardware implementation

5.1. Sensor

The AD8232 (see circuit level block diagram in Figure 5) is a cost-effective integrated signal conditioning module used for ECG measurement. It operates with a single supply source in the range of 2 - 3.5 V. To improve the common mode rejection from line frequency and other unexpected interferences, this module has a right leg drive amplifier. Another beneficial feature of this module is a fast-restoring function that helps to reduce the duration after a “lead off” condition occurs. A lead off condition indicates if any of the three leads is disconnected. Based on the position of switch S2 (see block diagram in Figure 5) it is even possible to say which one of the three leads is disconnected. Because of this feature, it is possible to resume valid measurements soon after having an interruption.

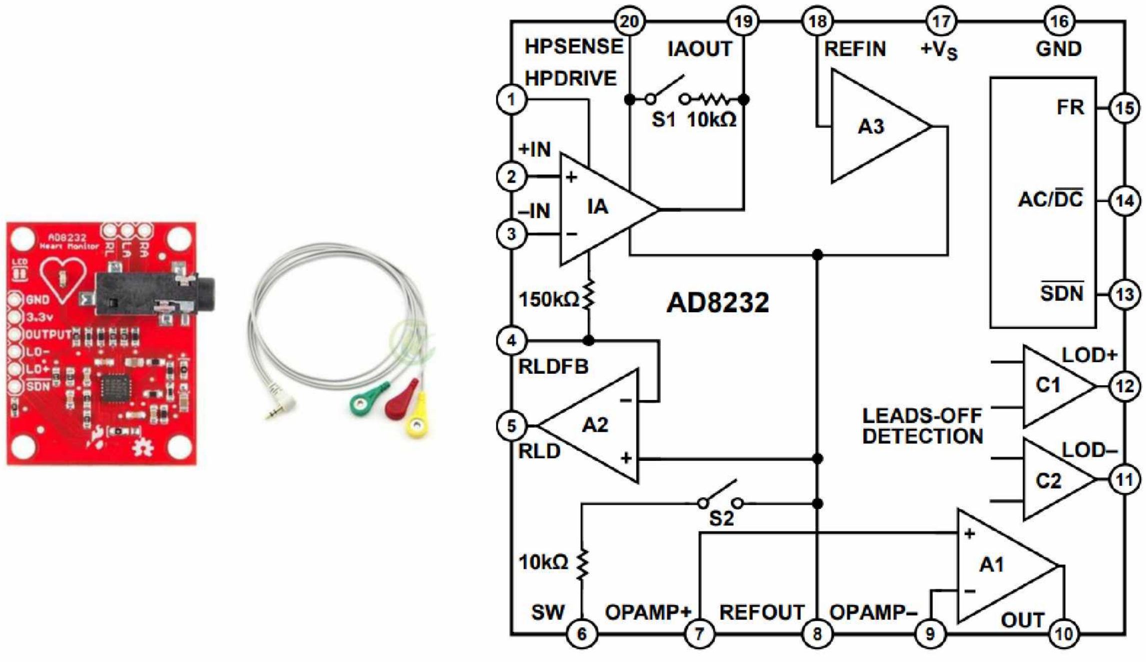
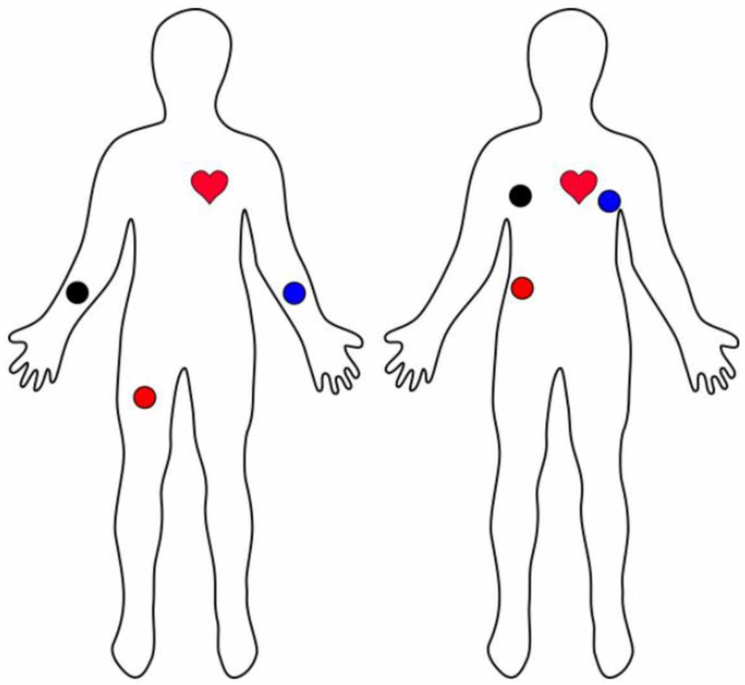


Figure 5: AD8232 heart rate sensor and its block diagram.

To collect the data, a user needs to wear the sensor pads close to the heart. Normally, the cables are color coded, which makes it easier to identify proper placement based on Einthoven's triangle as shown for two different sensors lead placement positions in Figure 6. The table in Figure 6 shows the color code for the sensor lead placement. The hookup guidelines are found in [5].



Cable Color	Signal
Black	RA (Right Arm)
Blue	LA (Left Arm)
Red	RL (Right Leg)

Figure 6: Typical sensor placement and color code.

5.2. CPU

The Raspberry Pi 3 B+ shown in Figure 7 includes a Broadcom BCM2837B0, consisting of a 1.4 GHz 64-bit SoC Cortex-A53 processor and 1 GB LPDDR2 SD RAM. The board also possesses dual-band 2.4 and 5 GHz wireless LAN transceivers meeting modular compliance certification standards. Other specifications of the Raspberry Pi 3 B+ are as follows:

- Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI
- 4 USB 2.0 ports
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5 V / 2.5 A DC power input

- Power-over-Ethernet (PoE) support - requires separate PoE Hardware Attached on Top (HAT) board

Figure 7: Raspberry Pi 3 B+.

5.3. Wireless module

Two Raspberry Pi 3 B+'s communicate via onboard Wi-Fi. The dual-band 2.4 and 5 GHz WLAN uses the IEEE 802.11.b/g/n/ac standard. The details of Wi-Fi communication used in this project are discussed in the software section.

5.4. MCP3008

The Raspberry Pi 3 B+ does not possess an on-board analog general purpose input/output (GPIO) port, therefore the MCP3008 (see pin configuration shown in Figure 8) is used to collect analog data from the AD8232 analog heart rate sensor. The MCP3008 is a successive approximation 10-bit analog-to-digital converter (ADC) with an onboard sample-and-hold circuit. This device is programmable and provides eight single ended inputs. Communication with this device is performed with a simple serial peripheral interface (SPI) protocol supporting a device conversion rate of up to 200 ksps. This device operates over a voltage range of 2.7 - 5.5 V. Low current design capacity permits the operation with a standby current of 5 nA and active current of 320 uA.

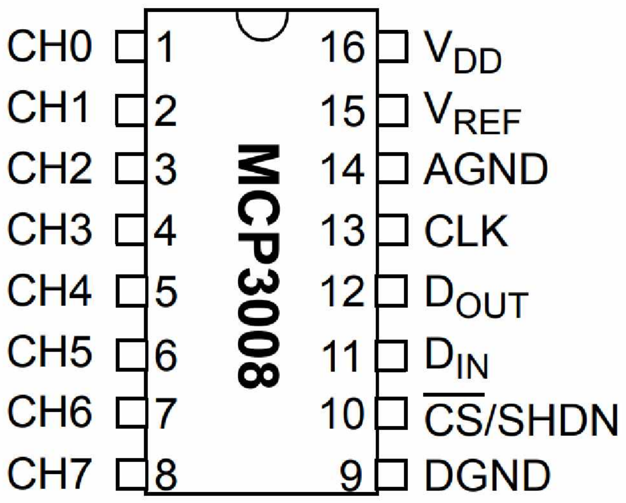


Figure 8: Pin configuration of MCP3008.

5.5. Data storage

A 32 GB micro SD card is used for storing local data and loading the operating system. After collecting raw data from the heart rate sensor, processed data and beats per minute (BPM) are stored in separate text files. The rescue team can access these text files for any specific set of data as a backup in case of failure of the real-time data system.

5.6. Final system and circuit diagram

The circuit diagram in Figure 9 below shows the overall system block diagram which has a transmitter side and a receiver side. The transmitter consists of a Raspberry Pi 3B+, an ADC module (MCP3008) and the ECG sensor (AD8232). The receiver consists of a Raspberry Pi 3 B+ and micro SD card. The Raspberry Pi 3B+'s on board Wi-Fi is used to communicate between transmitter and receiver. The MCP3008 and Raspberry Pi 3B+ on the transmitter side are connected via SPI (Serial Peripheral Interface).

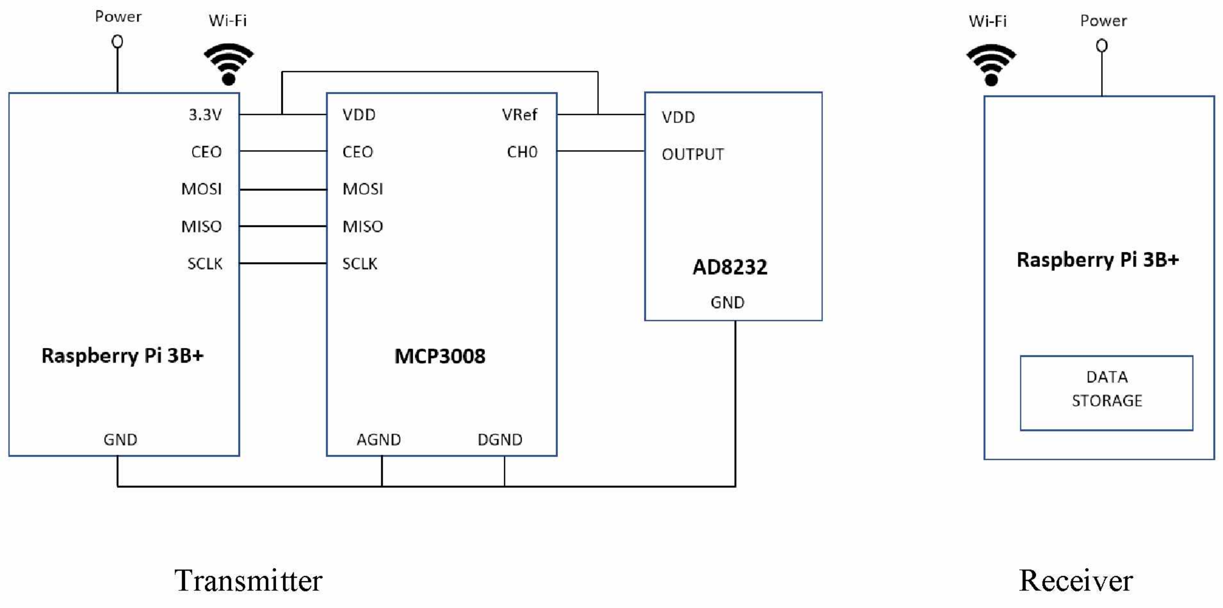


Figure 9: Circuit diagram of transmitter, receiver, and ADC system.

The actual test bed implemented is shown in Figure 10. This consists of two Raspberry Pi 3B+, MCP3008 and AD8232 with the three leads connected. Pads are also connected with the leads. During data collection, these pads are attached to the body with supplied glue gel.

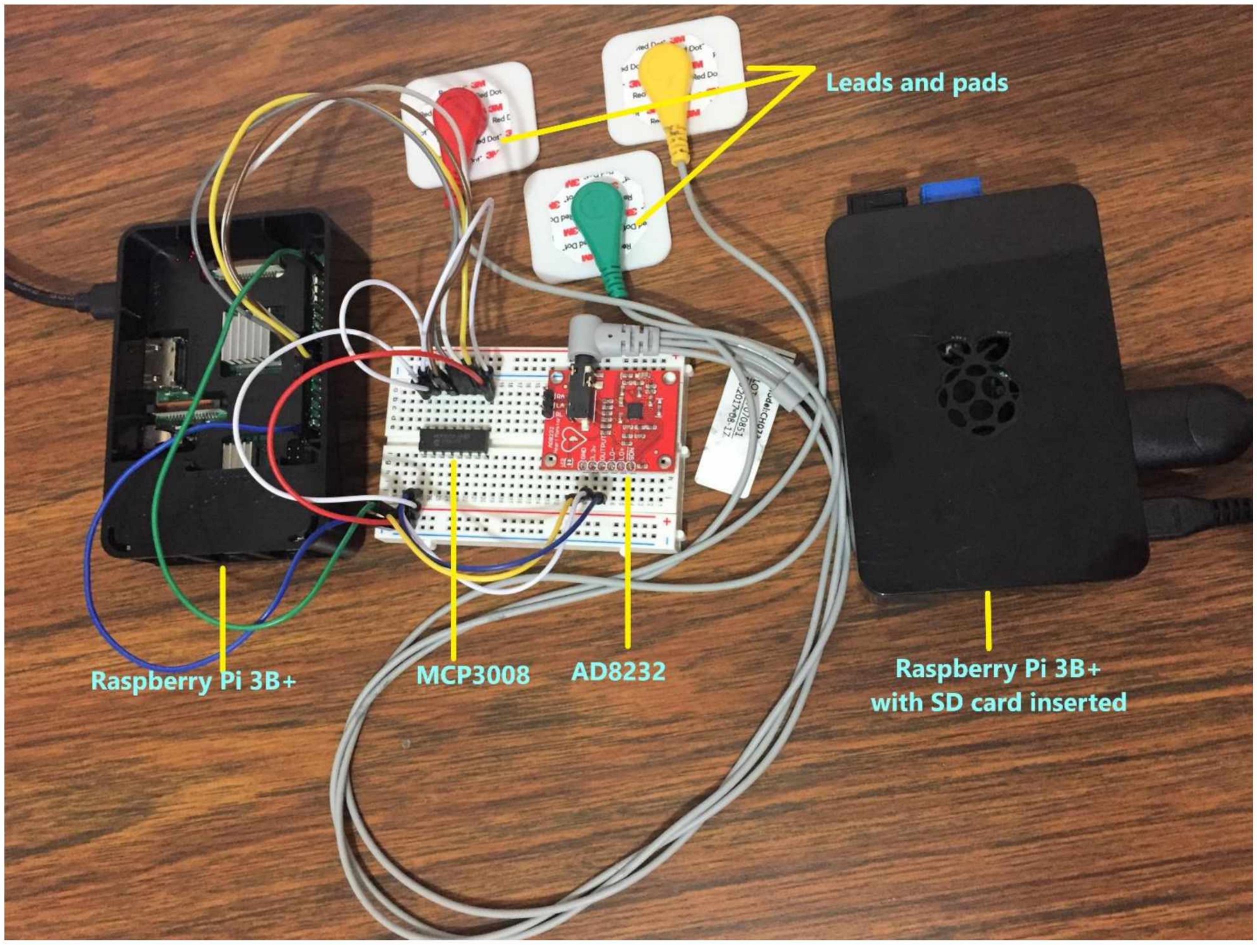


Figure 10: Actual test bed with transmitter, receiver, and ADC system.

6. Software description

The ECG sensor provides an analog output. To process the sensor data in the Raspberry Pi 3 B+, it needs to be converted in digital values. To do so, the MCP3008 10-bit ADC is used to convert the analog sensor output to digital before sending the data to the Raspberry Pi 3 B+. The sampling rate is set to 360 Hz for both real-time and offline analysis.

To collect data from the MCP3008, an SPI is initiated utilizing the common SpiDev driver. For initiating the SPI, first, the SpiDev object is created. After that, a port is opened, and then the device is selected with the chip select option. With the driver-supported speeds, 3.9 MHz is set for the bus communication. After this is setup, the SPI transaction is performed with the xfer2 command. To use the SPI, one needs to first make sure the SPI is enabled from the raspi-config tool in Raspberry Pi 3 B+.

The wireless communication is established between the transmitter and the receiver using a socket software interface which communicates between two processes over the internet. A process sends and receives messages to and from the network, respectively, through the socket interface. Here the underlying transport protocol used by the processes is Transmission Control Protocol (TCP). A socket is also referred to as an Application Programming Interface (API) between the application and network.

TCP is a connection-oriented protocol. In this protocol, the client and server need to handshake and establish a TCP connection before they start to send data to each other. Here one end of the protocol connection attaches to the client and other end to the server. At the beginning of the connection the socket associates with the client socket address (IP address and port number) and the server socket address (IP address and port number). With the connection established it is now ready to send data from one side to other via its socket.

7. Data processing

The raw ECG data is contaminated with muscle noise, power line interference, base line wonder, T-wave interference and motion artifacts. For BPM measurement these noise and interference elements need to be removed from the ECG signal. A band pass filter with cascaded low pass and high pass filters is implemented to remove the noise and interference mentioned above. The cut-off frequency of the lowpass filter is 15 Hz and the high pass filter is 5 Hz. After filtering this signal is differentiated to provide the slope information of the QRS complex. This signal is then squared point by point so that all data points become positive and nonlinearly amplified, making the signal ready for integration.

8. Results and analysis

8.1. Real-time data implementation

After collecting the ECG signal at a 360 Hz sampling rate, the signal is denoised using a band pass filter which is cascaded with a low pass and high pass filter. Figure 11 shows the raw data collected in real time (top plot) then passed through the bandpass filter (second plot from top). This clearly shows that all the high and low frequency noise has been significantly removed. After this the derivative of the filtered signal is shown which reveals the slope information (third plot from top). After squaring sample by sample the R peaks are significant, and all other peaks get suppressed (bottom plot). Calculating the slope of the R wave only is not a perfect way for the identification of a QRS event, as an abnormal QRS complex with long duration and large amplitudes may not be possible to detect with slope information only. To overcome this, a moving-window integration technique is applied for extraction of additional features. To illustrate the above-mentioned steps, the set of major equations used is:

b, a = scipy. signal.butter(N, Wn, btype='low', analog=False). (1)

where b, a are filter coefficients (b the numerator and a the denominator, N is order of the filter, Wn the length of the two-sequence critical frequency, btype the type of the filter (eg, low pass, high pass, band pass).

Using the coefficients, the raw ECG data is filtered along one dimension implementing a direct II transposed structure. The equation used is:

a[0] * y[n] = b[0] * x[n] + b[1] * x[n-1] + ...

$$+ b[M] * x[n-M] - a[1] * y[n-1] - \ldots - a[N] * y[n-N]. \qquad (2)$$

where y is the filtered data buffer, x is the raw ECG data buffer, n is the sample number, and M and N are the degrees of the numerator and denominator, respectively.

The first difference is calculated by the equation below which is possible to use recursively as well.

$$\text{diff\_out}[n]= y[n+1] - y[n]. \qquad (3)$$

where diff_out is difference of back to back sample

A point by point squaring is done by

$$\text{sqr\_sig}[n] = (\text{diff\_out}[n])^\wedge 2. \qquad (4)$$

where sqr_sig is the sample-by-sample square of difference output.

The moving window integration is implemented with the difference equation below

$$\text{MWI}[n] = (1/N) * (\text{sqr\_sig}[n - (N-1)] + \text{sqr\_sig}[n - (N-2)] + \ldots\ldots + \text{sqr\_sig}[n]) \qquad (5)$$

where MWI is moving window integration buffer and N is the number of samples in the moving window width.

Now, a calibrated threshold is applied according to [7] to find the actual number of R peaks or 60 seconds.

From this an RR interval time is calculated and used to determine the BPM.

The RR interval is calculated as

$$RR = R_{peaks}/\ f_s *1000, \qquad (6)$$

where f_s is the sampling rate (360 Hz in this case) multiplied by 1000 to get the RR interval in milliseconds and Rpeaks is the number of samples between consecutive R peaks.

The BPM is calculated using the RR interval as

$$\text{BPM} = 60 * 1000 / \text{RR}. \tag{7}$$

This device is tested with three different users. One of them is a 26 year old female. Applying the algorithm, the experimental BPM reading fluctuates from 83 - 89 which is within the normal resting BPM rate range from 60 - 100 [8]. Figure 11 shows the real-time ECG signal from a single user including the raw data (top plot), filtered data (second), derivative data (third) and squared data (bottom). From visual inspection, 8 beats were found for every 5 seconds consistently as the raw data shown in figure 11. This represents 96 beats for every 60 seconds (96 BPM). The experimental reading is then compared with the visually estimated reading. The equation used to calculate the error rate is:

$$\%\, error = \frac{|Experimental\ val - Expected\ val|}{Expected\ val} \times 100 \tag{8}$$

where Experimental val is the BPM reading found by experiment. Using the applied algorithm, minimum and maximum BPMs are 83 and 89, respectively. Expected val is the estimated BPM found from visual inspection which is 96 in this case. Plugging these values in (8), this system gives 7 - 13% error.

Figure 12 illustrates the terminal output of the real-time system. With the present experimental setup it is hard to improve this error because of the tradeoff between CPU execution cycle and the sampling rate. In this system an external ADC module is used because our CPU doesn't have one. So, the sampling rate in our system was limited by SPI bus speed, external module analog to digital sample, hold and conversion time. But while choosing the sampling rate 360 Hz for the system it was made sure that it meets the Nyquist criterion. According to Nyquist criterion, for proper digitization of the analog signal, sampling frequency must be twice or greater than the maximum frequency present in the analog signal. The standard bandwidth of the ECG used in clinical setups is 0.05 - 100 Hz [9]. Therefore, 360 Hz was considered to be a reasonable choice for the system.

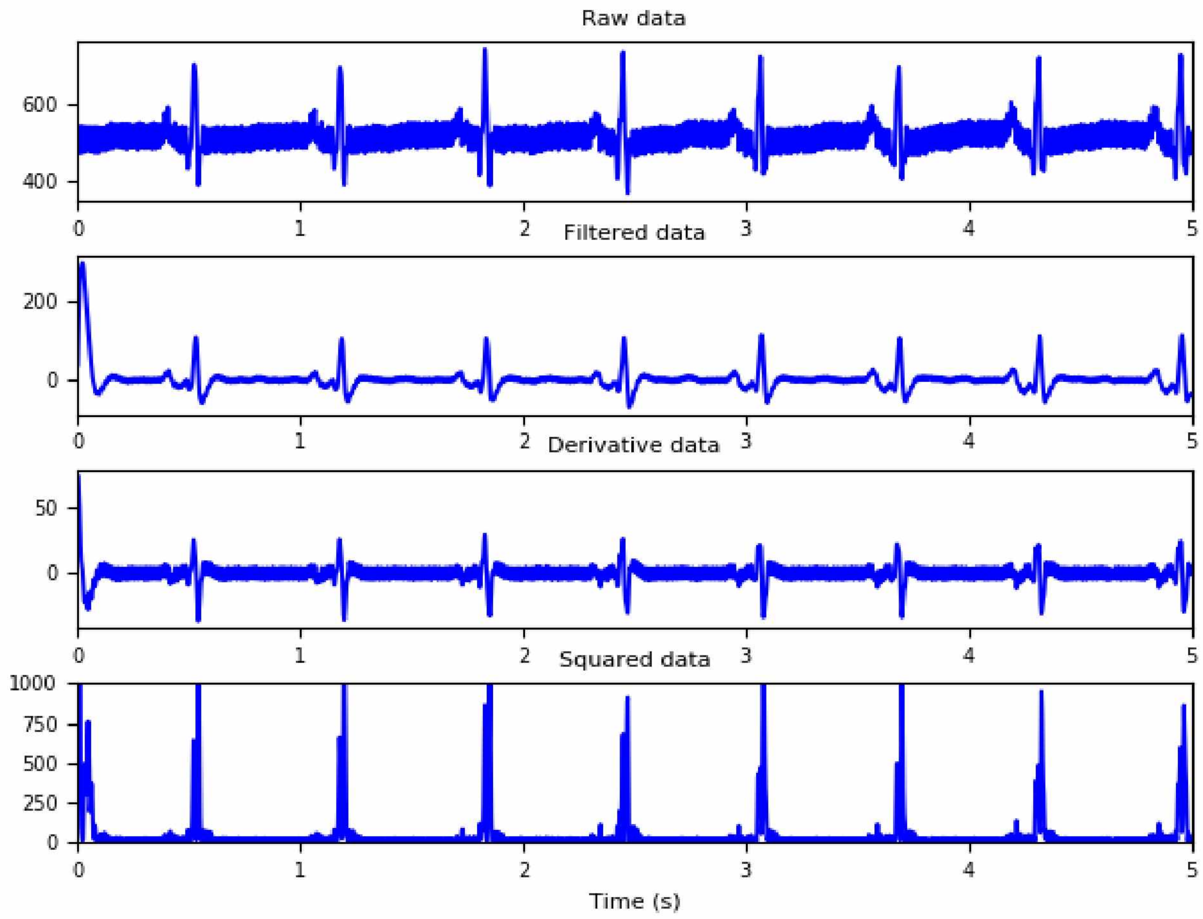
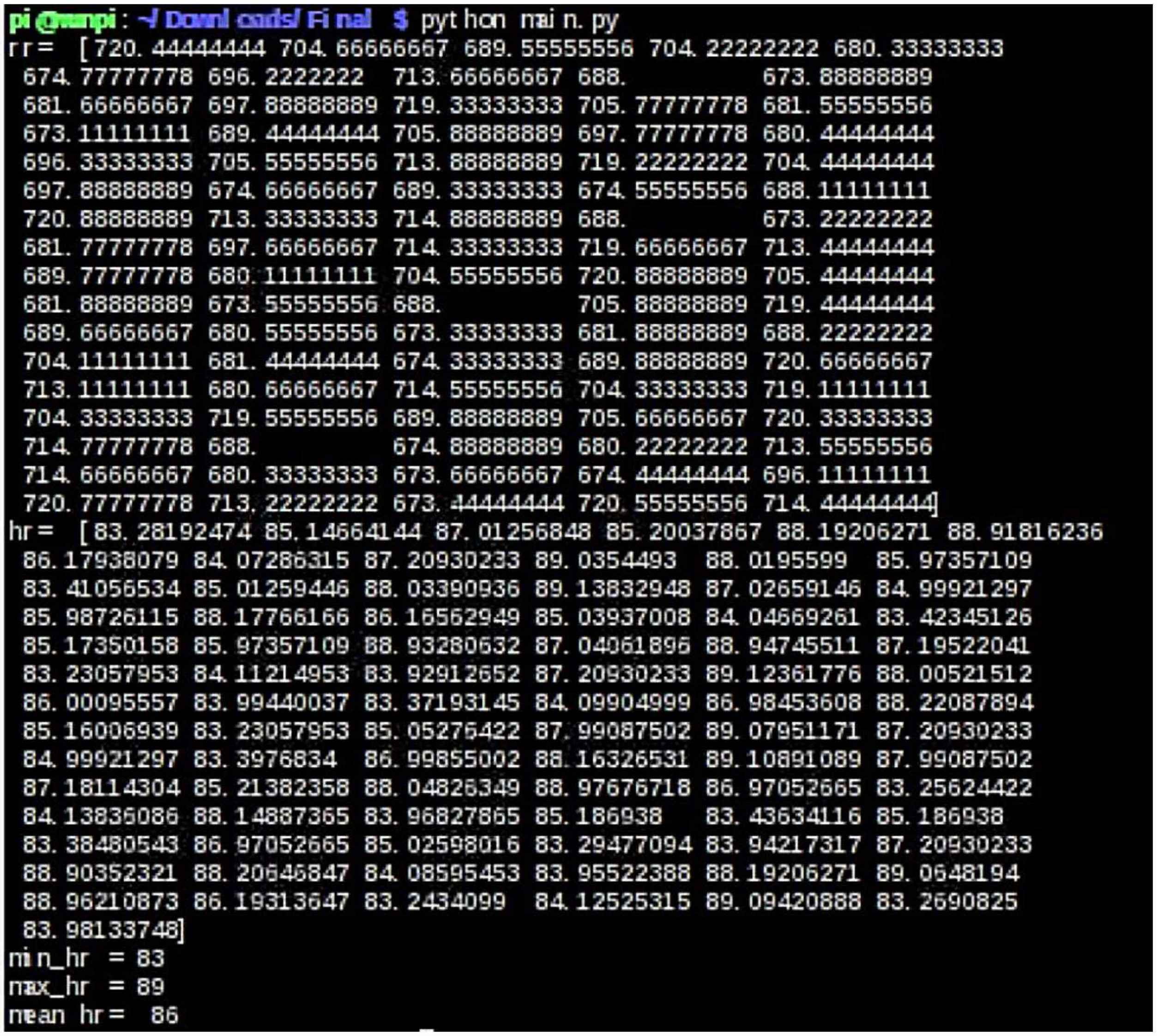


Figure 11: Real-time processed ECG signal showing from top to bottom the raw data, filtered data, derivative data and squared data.



```
pi@wanpi:~/Downloads/Final $ python main.py
rr=  [720.44444444 704.66666667 689.55555556 704.22222222 680.33333333
 674.77777778 696.2222222  713.66666667 688.         673.88888889
 681.66666667 697.88888889 719.33333333 705.77777778 681.55555556
 673.11111111 689.44444444 705.88888889 697.77777778 680.44444444
 696.33333333 705.55555556 713.88888889 719.22222222 704.44444444
 697.88888889 674.66666667 689.33333333 674.55555556 688.11111111
 720.88888889 713.33333333 714.88888889 688.         673.22222222
 681.77777778 697.66666667 714.33333333 719.66666667 713.44444444
 689.77777778 680.11111111 704.55555556 720.88888889 705.44444444
 681.88888889 673.55555556 688.         705.88888889 719.44444444
 689.66666667 680.55555556 673.33333333 681.88888889 688.22222222
 704.11111111 681.44444444 674.33333333 689.88888889 720.66666667
 713.11111111 680.66666667 714.55555556 704.33333333 719.11111111
 704.33333333 719.55555556 689.88888889 705.66666667 720.33333333
 714.77777778 688.         674.88888889 680.22222222 713.55555556
 714.66666667 680.33333333 673.66666667 674.44444444 696.11111111
 720.77777778 713.22222222 673.44444444 720.55555556 714.44444444]
hr=  [83.28192474 85.14664144 87.01256848 85.20037867 88.19206271 88.91816236
 86.17938079 84.07286315 87.20930233 89.0354493  88.0195599  85.97357109
 83.41056534 85.01259446 88.03390936 89.13832948 87.02659146 84.99921297
 85.98726115 88.17766166 86.16562949 85.03937008 84.04669261 83.42345126
 85.17350158 85.97357109 88.93280632 87.04061896 88.94745511 87.19522041
 83.23057953 84.11214953 83.92912652 87.20930233 89.12361776 88.00521512
 86.00095557 83.99440037 83.37193145 84.09904999 86.98453608 88.22087894
 85.16006939 83.23057953 85.05276422 87.99087502 89.07951171 87.20930233
 84.99921297 83.3976834  86.99855002 88.16326531 89.10891089 87.99087502
 87.18114304 85.21382358 88.04826349 88.97676718 86.97052665 83.25624422
 84.13836086 88.14887365 83.96827865 85.186938   83.43634116 85.186938
 83.38480543 86.97052665 85.02598016 83.29477094 83.94217317 87.20930233
 88.90352321 88.20646847 84.08595453 83.95522388 88.19206271 89.0648194
 88.96210873 86.19313647 83.2434099  84.12525315 89.09420888 83.2690825
 83.98133748]
min_hr  = 83
max_hr  = 89
mean hr=  86
```

Figure 12: Terminal output showing the heart rate (mean_hr is converted into integer) which indicates the beats per minute for the real-time ECG data implementation.

The two other individuals that helped with real-time testing of the system are a 25 year old male and a 24 year old female. Their beat pattern and terminal output showing heart rate after applying the algorithm are shown in Figures 16-21 in appendix 11.2. For subject A the minimum, maximum and mean BPM were determined using the algorithm are 64, 68 and 66, respectively. The BPM with visual inspection is 72. Using (8) an error range of 5 - 11% exists between the range of experimental values and the visually determined expected value for subject A. For subject B the minimum, maximum and mean BPM determined using the algorithm are 85, 91 and 88, respectively. The BPM with visual inspection is 98. Using (8) an error range of 7 - 13% exists between the range of experimental values and the visually determined expected value for subject B.

8.2. Existing data implementation

An ECG data set from the MIT/BIH database [10] was acquired and then filtered using the same algorithm developed for the real-time system to validate the process. The data set from the MIT/BIH database was digitized followed by a band pass filtered signal to limit anti-aliasing effects and saturation of the ADC. Figure 13 shows the 60 seconds of raw data from the database (top plot), followed by filtered data (second), derivative data (third) and squared data (bottom). It is clear that the processing stages remove the baseline wander, motion artifact and powerline noise. Figure 14 shows only the first 10 seconds of the data to better visualize the beats. Finally, using the same algorithm BPM is calculated and the mean heart rate is displayed as 70 beats per minute on the terminal screen capture shown in Figure 15. From these results it can be concluded that the device properly detects the BPM from the 60 seconds of ECG data.

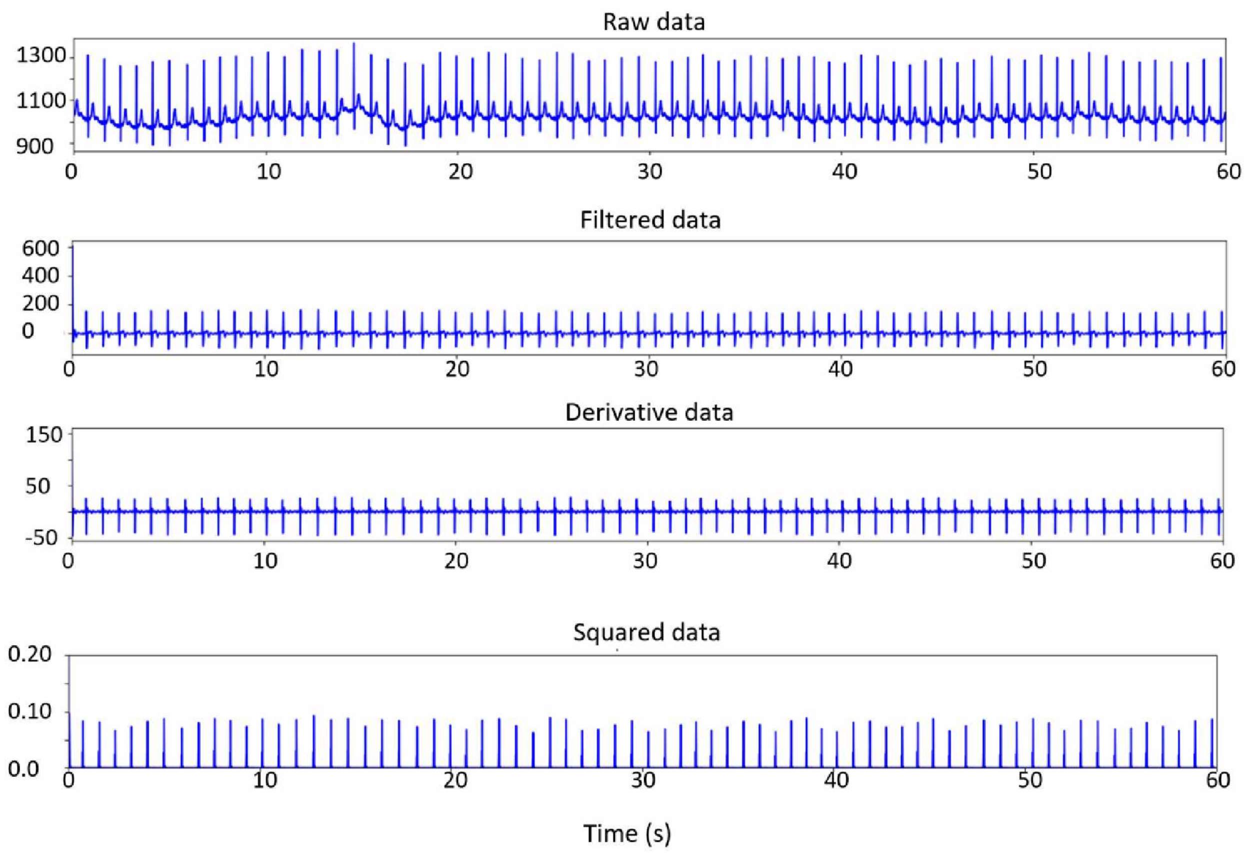


Figure 13: 60 seconds of MIT/BIH ECG raw, filtered, derivative and squared data to verify the system implemented.

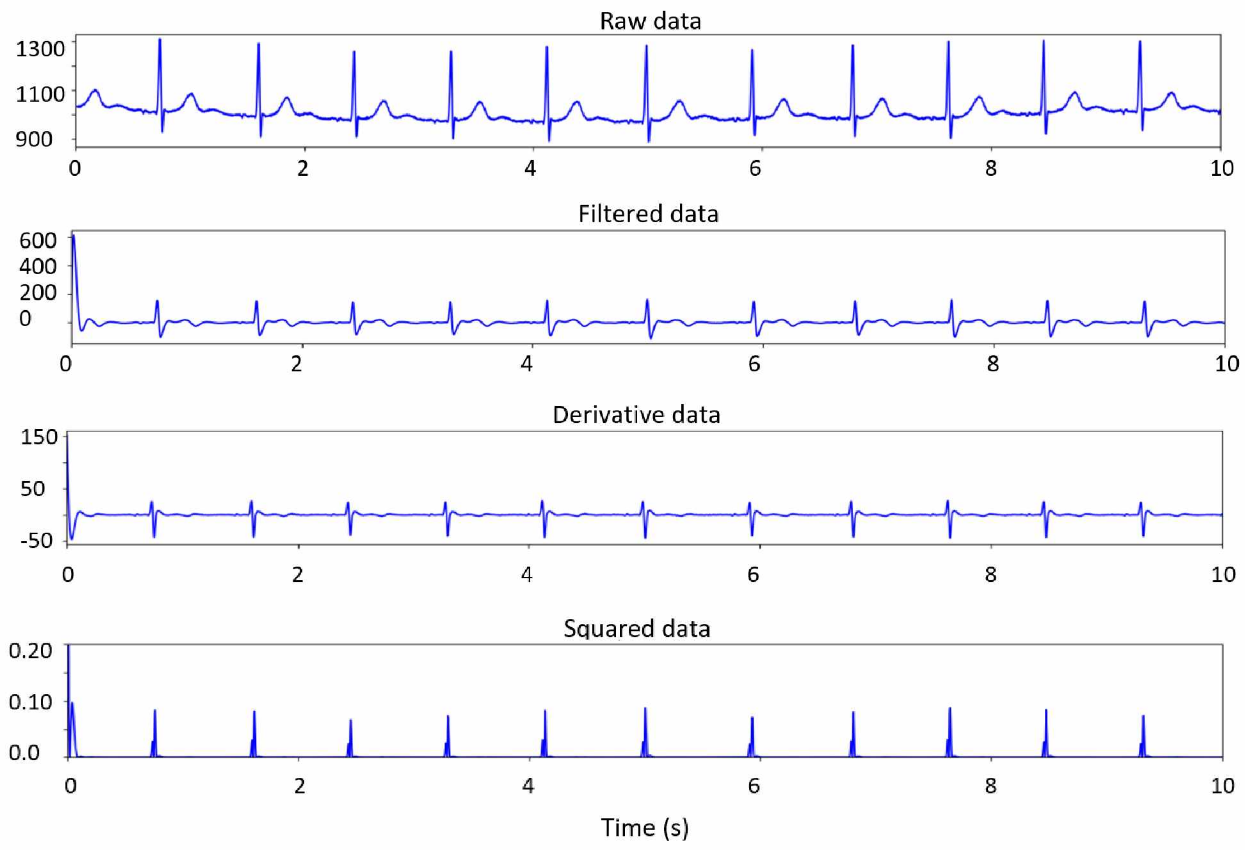
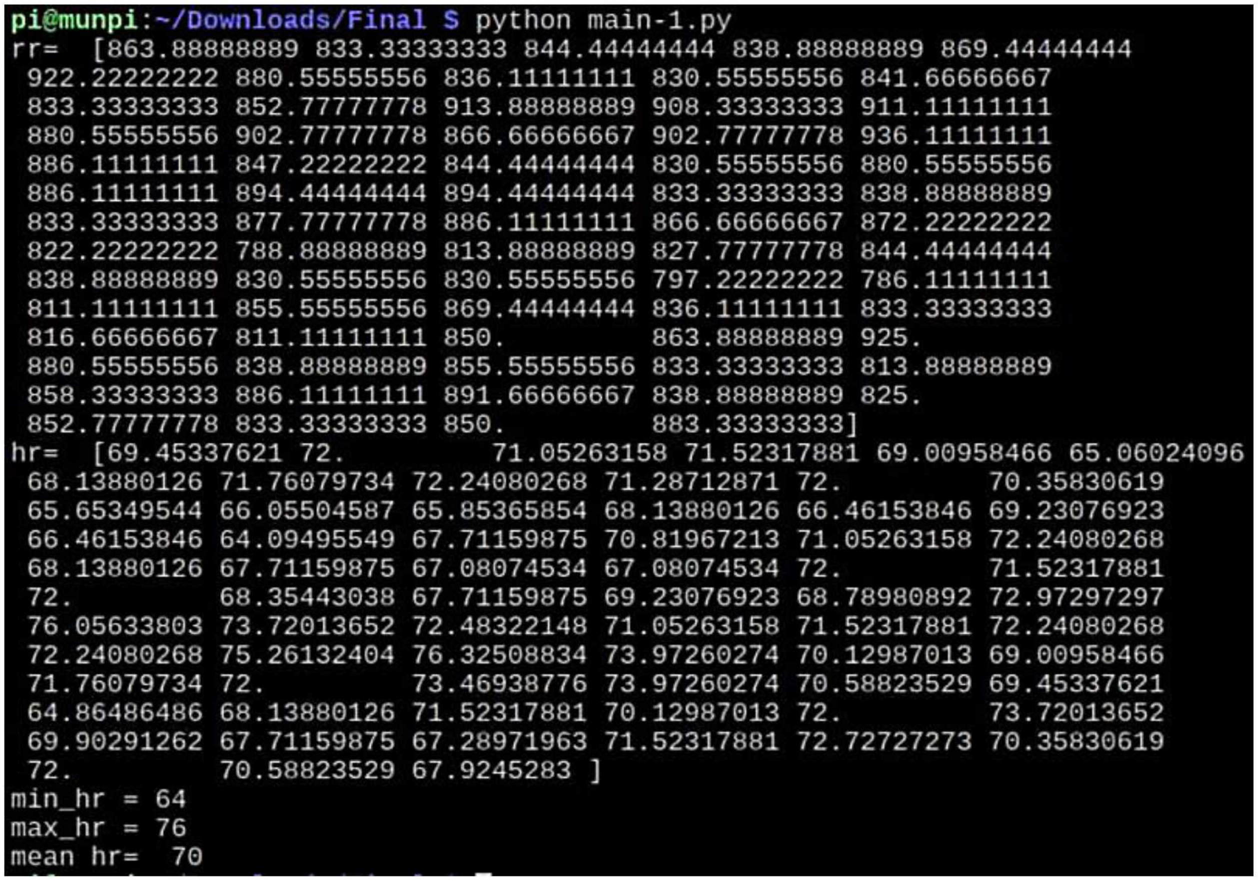


Figure 14: Ten seconds of MIT/BIH ECG raw, filtered, derivative and squared data to visualize the beat signatures more clearly.



```
pi@munpi:~/Downloads/Final $ python main-1.py
rr=  [863.88888889 833.33333333 844.44444444 838.88888889 869.44444444
 922.22222222 880.55555556 836.11111111 830.55555556 841.66666667
 833.33333333 852.77777778 913.88888889 908.33333333 911.11111111
 880.55555556 902.77777778 866.66666667 902.77777778 936.11111111
 886.11111111 847.22222222 844.44444444 830.55555556 880.55555556
 886.11111111 894.44444444 894.44444444 833.33333333 838.88888889
 833.33333333 877.77777778 886.11111111 866.66666667 872.22222222
 822.22222222 788.88888889 813.88888889 827.77777778 844.44444444
 838.88888889 830.55555556 830.55555556 797.22222222 786.11111111
 811.11111111 855.55555556 869.44444444 836.11111111 833.33333333
 816.66666667 811.11111111 850.         863.88888889 925.
 880.55555556 838.88888889 855.55555556 833.33333333 813.88888889
 858.33333333 886.11111111 891.66666667 838.88888889 825.
 852.77777778 833.33333333 850.         883.33333333]
hr=  [69.45337621 72.         71.05263158 71.52317881 69.00958466 65.06024096
 68.13880126 71.76079734 72.24080268 71.28712871 72.         70.35830619
 65.65349544 66.05504587 65.85365854 68.13880126 66.46153846 69.23076923
 66.46153846 64.09495549 67.71159875 70.81967213 71.05263158 72.24080268
 68.13880126 67.71159875 67.08074534 67.08074534 72.         71.52317881
 72.         68.35443038 67.71159875 69.23076923 68.78980892 72.97297297
 76.05633803 73.72013652 72.48322148 71.05263158 71.52317881 72.24080268
 72.24080268 75.26132404 76.32508834 73.97260274 70.12987013 69.00958466
 71.76079734 72.         73.46938776 73.97260274 70.58823529 69.45337621
 64.86486486 68.13880126 71.52317881 70.12987013 72.         73.72013652
 69.90291262 67.71159875 67.28971963 71.52317881 72.72727273 70.35830619
 72.         70.58823529 67.9245283 ]
min_hr = 64
max_hr = 76
mean hr=  70
```

Figure 15: Terminal output showing the heart rate (mean_hr is converted into integer) which indicates the beats per minute for the MIT/BIH data implementation.

9. Conclusion and Future Work.

Within the limited scope of this project, the concept of providing critical health and environmental data to the mine rescue team (MRT) has proven promising. In this effort, the calculated BPM was successfully sent wirelessly to the rescue UAV housing the receiver for real-time evaluation, as well as being stored locally on a micro SD card for post-event analysis. This test bed has shown to be useful in collecting data from multiple trapped miners to help the MRT assess the overall situation. Based on the number of miners alive and environmental conditions surrounding these, the MRT can finalize the rescue strategy.

The following is a list of additional research and development items to be further explored for mine environment sensing and data collection:

1. In addition to the heart rate sensor, gaseous particle, atmospheric pressure, temperature, and humidity are also required to provide an accurate assessment of the environment.
2. The test bed needs to be hardened for operations within an actual mine environment. Significant investment in research and development is required for mine sensor technology.
3. There is a need to create a simulation environment for testing the designed systems, particularly the sensors and range of the wireless data transmission in a mine environment.
4. The software is currently written in Python which is an integrated language. To improve the system speed and efficiency, writing the code in C is recommended.
5. It is recommended to use a CPU with an integrated ADC module which will increase the signal-to-noise ratio of the system.
6. The adhesive pads used to connect the sensor leads to the body cause itching sometimes. Care should be taken before using the pads during testing. This prototype system would need to be replaced with another which is more operationally suited for a miner exposed to this challenging environment over an extended period of time.
7. For advanced performance analysis, other market available similar category sensors can be compared. There exists a sharp learning curve of knowing different communication protocols such as Universal Asynchronous Receiver-Transmitter (UART), Inter-Integrated Circuit (I2C), Recommended Standard-232 (RS-232) etc. for data collection from sensors. Within the scope of this small project ADC and SPI protocols are used for data collection from the ECG sensor and a visual to algorithm got result is compared.

10. References

1. U. S. D. of L. MHSA, “Mine Safety and Health at a Glance: Calendar Year,” 2018. [Online]. Available: https://www.msha.gov/msha-glance
2. R. R. Murphy, J. Kravitz, S. L. Stover, and R. Shoureshi, “Mobile robots in mine rescue and recovery,” *IEEE Robot. Autom. Mag.*, vol. 16, no. 2, pp. 91–103, 2009.
3. J. McHugh, M. Collingridge, A. G. Sanchis, K. Munasinghe, and B. McGrath, “SDR based through the rubble communications for collapsed mines: A proof-of-concept,” in *Proc. of 9th International Conference on Signal Processing and Communication Systems (ICSPCS)*, 2015, pp. 1–8.
4. M. Hatfield, “Unmanned Vehicles for Mine Safety Operations and Training,” University of Alaska Fairbanks, 2015. [Online]. Available: http://mirl.uaf.edu/research/unmanned-vehicles-for-mine-safety- operations-and-training/
5. Image source: https://learn.sparkfun.com/tutorials/ad8232-heart-rate-monitor-hookup-guide/all
6. Image source: https://www.researchgate.net/figure/Typical-ECG-trace-with-QT-and-RR-intervals-labeled-Image-based-on_fig7_281176903
7. J. Pan and W. J. Tompkins, “A Real-Time QRS Detection Algorithm,” *IEEE Trans. on Biomedical Engineering*, vol. BME-32, no. 3, Mar. 1985. DOI: 10.1109/TBME.1985.325532. [Online]. Available: https://www.robots.ox.ac.uk/~gari/teaching/cdt/A3/readings/ECG/Pan+Tompkins.pdf
8. E. R. Laskowski, “Heartrate: What’s Normal?,” *mayoclinic.org,* Aug 29, 2018. [Online]. Available: https://www.mayoclinic.org/healthy-lifestyle/fitness/expert-answers/heart-rate/faq-20057979. [Accessed Aug 01, 2019].
9. E. Hartman, E. C. Bosch, “ECG Front-End Design is Simplified with MicroConverter,” *analog.com,* Nov 2003, Available: https://www.analog.com/en/analog-dialogue/articles/ecg-front-end-design-simplified.html. [Accessed Aug 01, 2019].
10. "MIT/BIH arrhythmia database-Tape directory and format specification," Document BMEC TR00, Mass. Inst. Technol., Cambridge, 1980. Database is available from Bioengineering Division KB-26, Beth-Israel Hospital, 330 Brookline Avenue, Boston, MA 02215.
11. V. X. Afonso, “ECG QRS detection,” in Biomedical digital signal processing: C-language examples and laboratory experiments for the IBM PC, W. J. Tompkins, Ed. NJ, USA: Prentice-Hall, Inc., 1993, pp. 236-264.

11. Appendix:

11.1. Code

```
#------------------------------------------acquisition.py-----------------------------------------------------

#Data acquisition from the sensor to raspberry pi code

from spidev import SpiDev  # comment this line for stored data analysis from time import sleep

#Create SPI

spi = 0

def init_spi():

   spi = SpiDev()

   spi.open(0, 0)

   spi.max_speed_hz = 1350000              #Good enough for the project but possible to change

   return spi

def read_adc(adc_num, num_samples=500, sample_freq=500):

   spi= init_spi()

   data = []

   t = 1.0/sample_freq

   # read SPI data from the MCP3008, 8 channels in total

   if adc_num > 7 or adc_num < 0:

      return -1

   while (num_samples):

      r = spi.xfer2([1, 8 + adc_num << 4, 0])

      data.append(((r[1] & 3) << 8) + r[2])

      sleep(t)

      num_samples = num_samples - 1

   return data

#Data store in file save
```

```
def save_data(data, filename):
  f = open(filename, "a")
  for i in data:
     f.write(str(i) + "\n")
  f.close()


#----------------------------------------------------main.py ---------------------------------------------
#This code calculates the BPM and transmits the data to the server
import matplotlib.pyplot as plt
import numpy as np
from acquisition import init_spi, save_data, read_adc
from algorithm1 import MWA, searchBack, panPeakDetect, pan_tompkins_detector
from scipy.signal import butter, lfilter
from scipy.ndimage.filters import convolve1d
#from scipy.interpolate import spline
from subprocess import call
from time import sleep
from client import transmit


#sample_freq = 360
num_samples = 1800                                    #time = num_samples/sample_freq
how_many_times = 50                                  #how many times measuring the BPM
fs=360


if __name__ == "__main__":
   count = 0
```

```
 x = np.linspace(0, num_samples / fs, num_samples)                    #for the x axis

 while ( count < how_many_times):                          #Depending on the how_many_times
parameter, this will happen so many times
    raw_data = read_adc(adc_num=0, num_samples=num_samples, sample_freq=fs)  #sampling
frequency is in Hz
    plt.clf()                              #clear figure
    plt.subplot(4, 1, 1)                   #plotting the raw data
    plt.plot(x, raw_data, 'b')
    plt.xlim(0, num_samples / fs)
    plt.title("Raw data")
    plt.xlabel('Time(in s)')
    plt.grid()
    f1 = 5/fs                              # Highpass filter cutoff frequency
    f2 = 15/fs                             # Lowpass filter cutoff frequency

    b, a = butter(1, [f1*2, f2*2], btype='bandpass') # finding filter coefficients

    filtered_ecg = lfilter(b, a, raw_data)
    derivative_data = np.diff(filtered_ecg)        # slope information
    squared_data = derivative_data**2              #amplified signal



    plt.subplot(4, 1, 2)                           #plotting the filtered data
    plt.plot(x, filtered_ecg, 'b')
    plt.xlim(0, num_samples / fs)
    plt.title("Filtered data")


    plt.subplot(4, 1, 3)                          #plotting the derivative data

    plt.plot(x[1:], derivative_data, 'b')

    plt.xlim(0, num_samples / fs)

    plt.title("Derivative data")


    plt.subplot(4, 1, 4)                        #plotting the squared data

    plt.plot(x[1:], squared_data, 'b')

    plt.xlim(0, num_samples / fs)

    plt.ylim(0, 500)

    plt.title("Squared data")

    plt.xlabel('Time(in s)')


    rpeaks = pan_tompkins_detector(raw_data, squared_data, 360)

    rr = np.diff(rpeaks) / fs * 1000        # R-R interval in miliseconds

    hr = 60 * 1000 / rr                     #beats per minute

    bpm= hr

    print(int(np.mean(hr)))


    message= str(bpm)

    print(message)

    response = transmit(message)

    print("Sample "+ str(count) + " : " + str(bpm) +" bpm")

    save_data([str(count) + ":" + str(bpm)], "bpm.txt")     #save bpm data to file

    count = count + 1
```

```
        plt.subplots_adjust(hspace=0.35)

        plt.pause(0.5)

    plt.show()


#------------------------------------------algorithm1.py---------------------------------------------------

# This code is written following the Pan Tomkins algorithm described in [7] and [11]


import numpy as np

import pywt

import pathlib

from scipy.signal import butter, lfilter, find_peaks


def Moving_Window_Average(input, window_size):

    moving_window_avg = np.zeros(len(input))

    for i in range(len(input)):

        if i < window_size:

            section = input[0:i]

        else:

            section = input[i-window_size:i]


        if i!=0:

            moving_window_avg[i] = np.mean(section)

        else:

            moving_window_avg[i] = input[i]

    return moving_window_avg

def searchBack(find_peak, raw_ecg, search_samples):
    r_peaks = []
    window = search_samples
    for i in find_peak:
        if i<window:
            section = raw_ecg[:i]
            r_peaks.append(np.argmax(section))
        else:
            section = raw_ecg[i-window:i]
            r_peaks.append(np.argmax(section)+i-window)
    return np.array(r_peaks)

def PeakDetect(detection, fs):
    min_distance = int(0.25*fs)
    peaks, _ = find_peaks(detection, distance=min_distance)
    signal_peaks = []
    noise_peaks = []
    SPKI = 0.0
    NPKI = 0.0
    threshold_I1 = 0.0
    threshold_I2 = 0.0

    RR_missed = 0
    index = 0
    indexes = []
    missed_peaks = []
    for peak in peaks:
```

```
    if detection[peak] > threshold_I1:
        signal_peaks.append(peak)
        indexes.append(index)
        SPKI = 0.125*detection[signal_peaks[-1]] + 0.875*SPKI
        if RR_missed!=0:
            if signal_peaks[-1]-signal_peaks[-2]>RR_missed:
                missed_section_peaks = peaks[indexes[-2]+1:indexes[-1]]
                missed_section_peaks2 = []
                for missed_peak in missed_section_peaks:
                    if missed_peak-signal_peaks[-2]>min_distance and signal_peaks[-1]-
missed_peak>min_distance and detection[missed_peak]>threshold_I2:
                        missed_section_peaks2.append(missed_peak)

                if len(missed_section_peaks2)>0:
                    missed_peak =
missed_section_peaks2[np.argmax(detection[missed_section_peaks2])]
                    missed_peaks.append(missed_peak)
                    signal_peaks.append(signal_peaks[-1])
                    signal_peaks[-2] = missed_peak

    else:
        noise_peaks.append(peak)
        NPKI = 0.125*detection[noise_peaks[-1]] + 0.875*NPKI
    threshold_I1 = NPKI + 0.25*(SPKI-NPKI)
    threshold_I2 = 0.5*threshold_I1
    if len(signal_peaks)>8:
        RR = np.diff(signal_peaks[-9:])
        RR_ave = int(np.mean(RR))
        RR_missed = int(1.66*RR_ave)
    index = index+1
signal_peaks.pop(0)
return signal_peaks


def pan_tompkins_detector(Raw_ecg, squared, fs):
    N = int(0.2*fs)
    moving_window_avg = Moving_Window_Average(squared, N)
    moving_window_avg[:int(0.2*fs)] = 0
    moving_window_avg_peaks = PeakDetect(moving_window_avg, fs)
    r_peaks = searchBack(moving_window_avg_peaks, Raw_ecg, N)
    return r_peaks
#------------------------------------------------------client.py----------------------------------------------
# This code works as a connection point in the client side
import socket
from time import sleep


host = '192.168.0.06'   # Need to modify if changes on a new connection
port = 5560             # High value not to interrupt with other device in the network


def setupSocket():
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))
return s
```

```
def sendReceive(s, message):
    m = s.send(str.encode(message))
    s.close()
    return m
def transmit(message):
    s = setupSocket()
    response = sendReceive(s, message)
    return response
#----------------------------------------------------server.py ------------------------------------------------
#This code runs in the second raspberry pi which is considered as Server (receiver)here
import socket
from time import sleep
host = ''
port = 5560
storedValue = "Checking!"
def setupServer():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print("Socket created.")
    try:
        s.bind((host, port))
    except socket.error as msg:
        print(msg)
    print("Socket bind complete.")
    return s
def setupConnection():
    s.listen(1)                                  # one connection at a time.
    conn, address = s.accept()
    #print("Connected to: " + address[0] + ":" + str(address[1]))
    return conn
def dataTransfer(conn):
    # Sending or receiving data until told not to do so
    while True:
        # Receive the data
        data = conn.recv(1024)
        data = data.decode('utf-8')
        dataMessage = data.split(' ', 1)
        command = dataMessage[0]
        reply = str(command)
        print("BPM=" + reply)
        return reply
    conn.close()
s = setupServer()
while True:
    try:
        conn = setupConnection()
        dataTransfer(conn)
        sleepTime = 2
    except:
        break
```

11.2. Real-time implementation of the system for two different users

Subject A (Figures 16-18): 25 year old male (BPM 66)

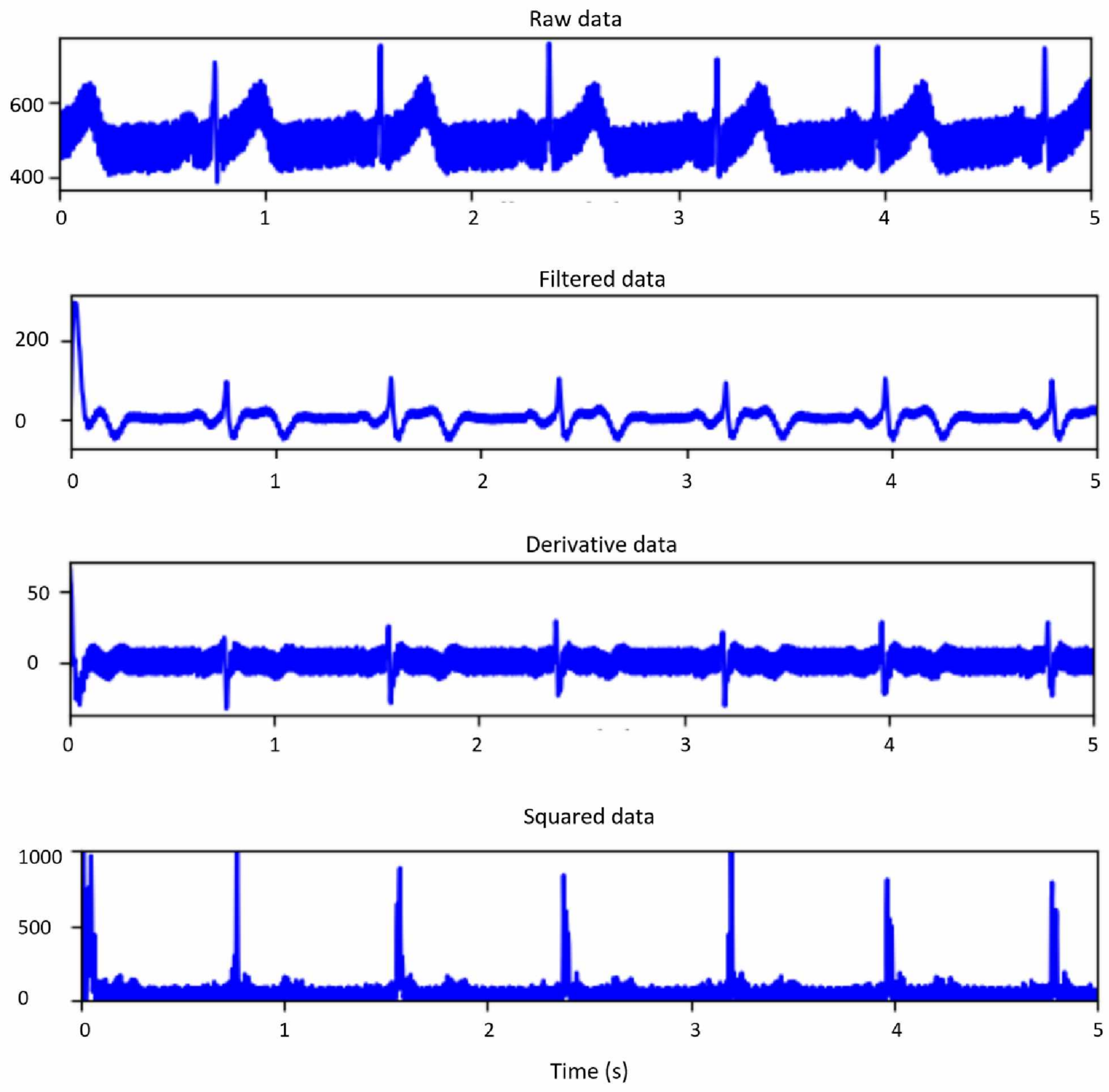


Figure 16: Real-time processed subject A ECG signal showing from top to bottom the raw data, filtered data, derivative data and squared data.

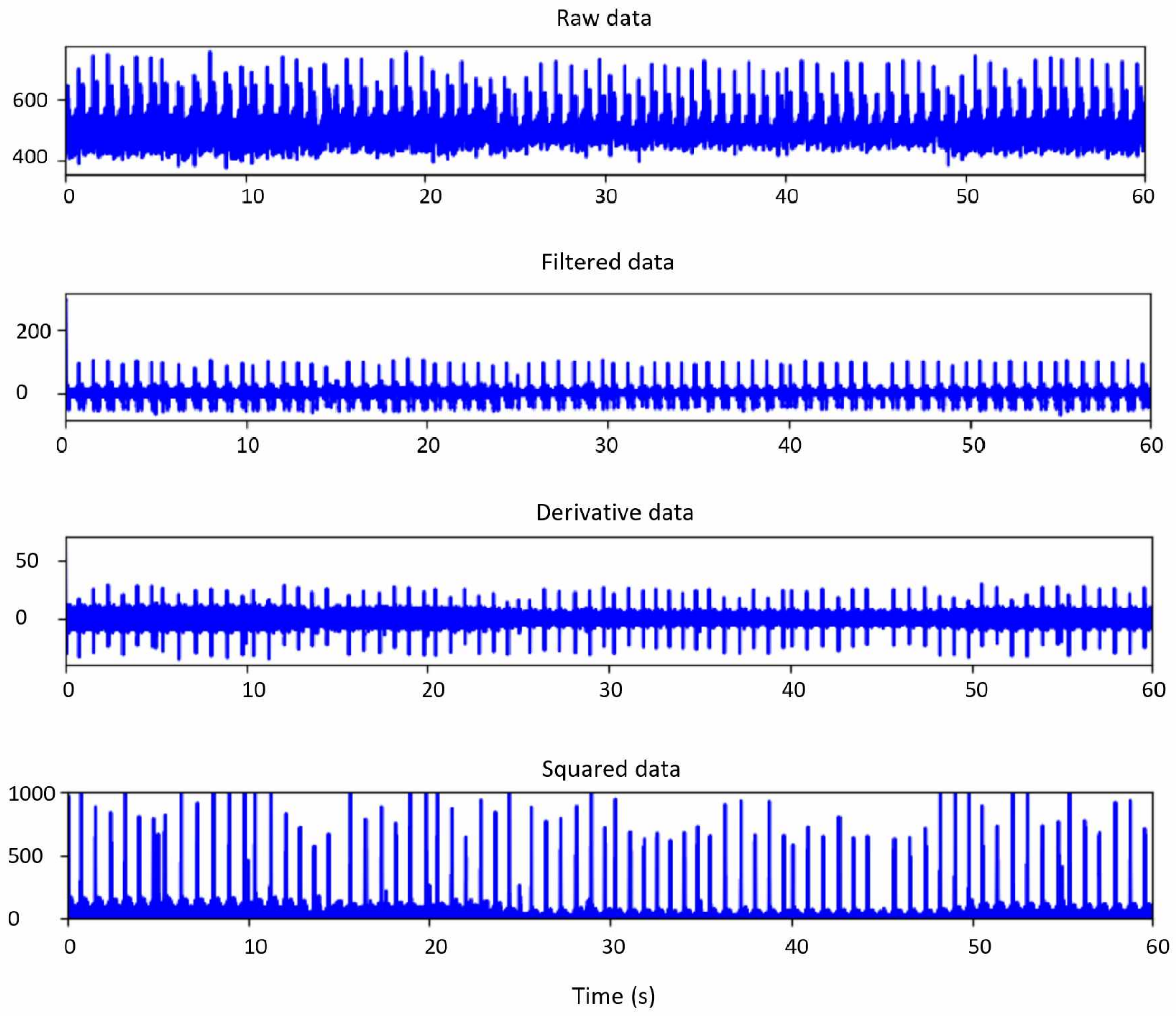
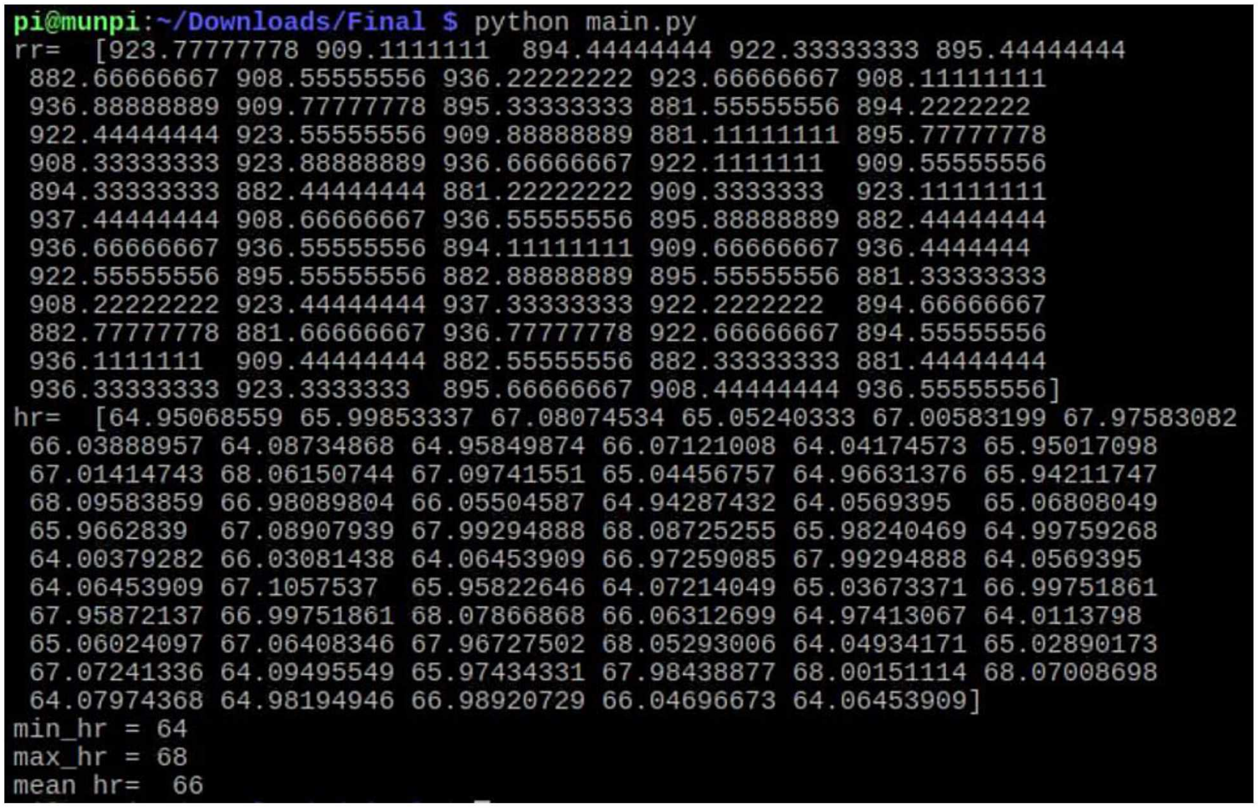


Figure 17: 60 seconds of subject A ECG raw, filtered, derivative and squared data.



```
pi@munpi:~/Downloads/Final $ python main.py
rr=  [923.77777778 909.1111111  894.44444444 922.33333333 895.44444444
 882.66666667 908.55555556 936.22222222 923.66666667 908.11111111
 936.88888889 909.77777778 895.33333333 881.55555556 894.2222222
 922.44444444 923.55555556 909.88888889 881.11111111 895.77777778
 908.33333333 923.88888889 936.66666667 922.1111111  909.55555556
 894.33333333 882.44444444 881.22222222 909.3333333  923.11111111
 937.44444444 908.66666667 936.55555556 895.88888889 882.44444444
 936.66666667 936.55555556 894.11111111 909.66666667 936.4444444
 922.55555556 895.55555556 882.88888889 895.55555556 881.33333333
 908.22222222 923.44444444 937.33333333 922.2222222  894.66666667
 882.77777778 881.66666667 936.77777778 922.66666667 894.55555556
 936.1111111  909.44444444 882.55555556 882.33333333 881.44444444
 936.33333333 923.3333333  895.66666667 908.44444444 936.55555556]
hr=  [64.95068559 65.99853337 67.08074534 65.05240333 67.00583199 67.97583082
 66.03888957 64.08734868 64.95849874 66.07121008 64.04174573 65.95017098
 67.01414743 68.06150744 67.09741551 65.04456757 64.96631376 65.94211747
 68.09583859 66.98089804 66.05504587 64.94287432 64.0569395  65.06808049
 65.9662839  67.08907939 67.99294888 68.08725255 65.98240469 64.99759268
 64.00379282 66.03081438 64.06453909 66.97259085 67.99294888 64.0569395
 64.06453909 67.1057537  65.95822646 64.07214049 65.03673371 66.99751861
 67.95872137 66.99751861 68.07866868 66.06312699 64.97413067 64.0113798
 65.06024097 67.06408346 67.96727502 68.05293006 64.04934171 65.02890173
 67.07241336 64.09495549 65.97434331 67.98438877 68.00151114 68.07008698
 64.07974368 64.98194946 66.98920729 66.04696673 64.06453909]
min_hr = 64
max_hr = 68
mean hr=  66
```

Figure 18: Terminal output showing subject A heart rate (mean_hr is converted into integer) which indicates the beats per minute for the real-time ECG data implementation.

Subject B (Figures 19-21): 24 years old female (BPM 88)

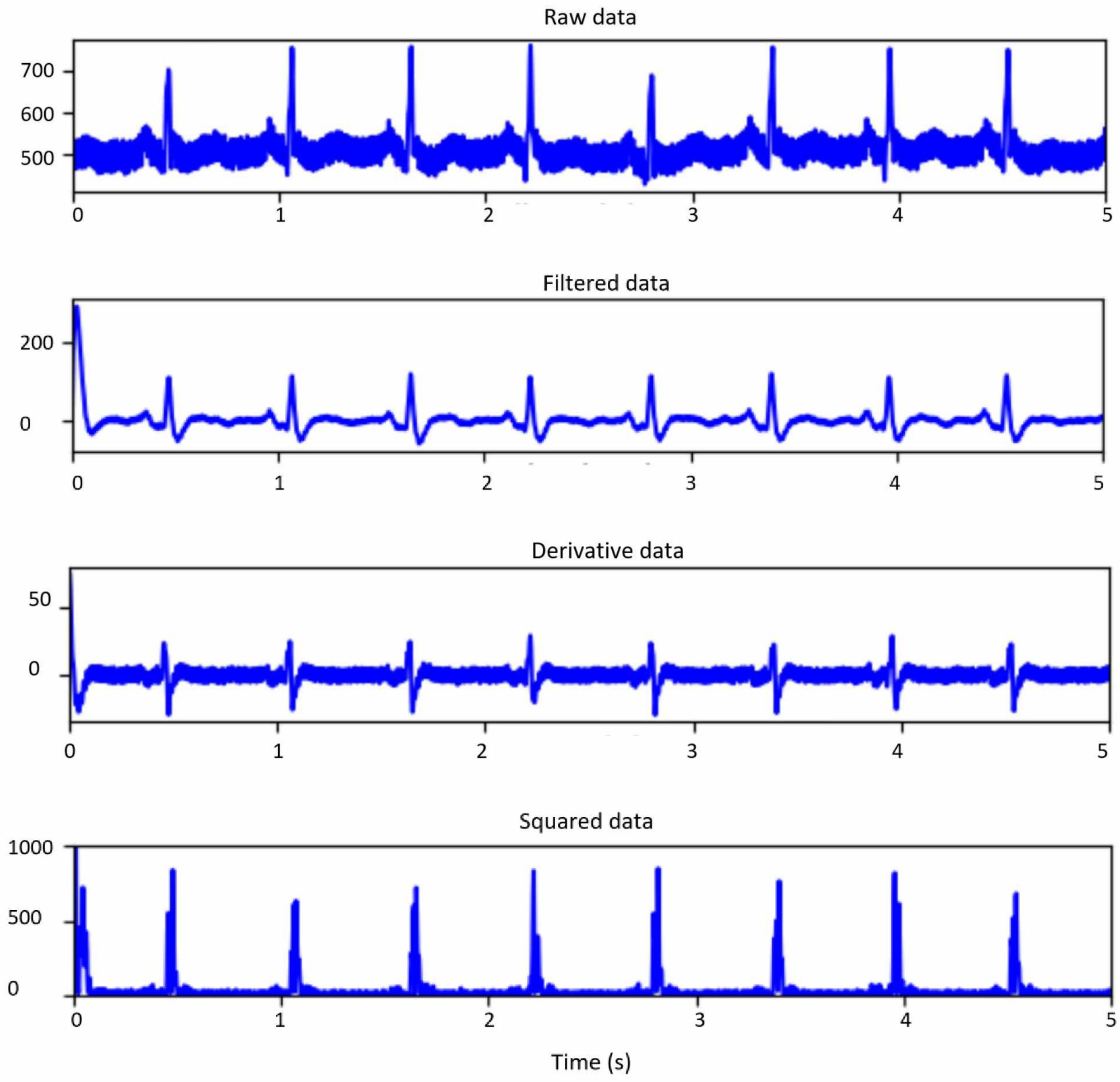


Figure 19: Real-time processed subject B ECG signal showing from top to bottom the raw data, filtered data, derivative data and squared data.

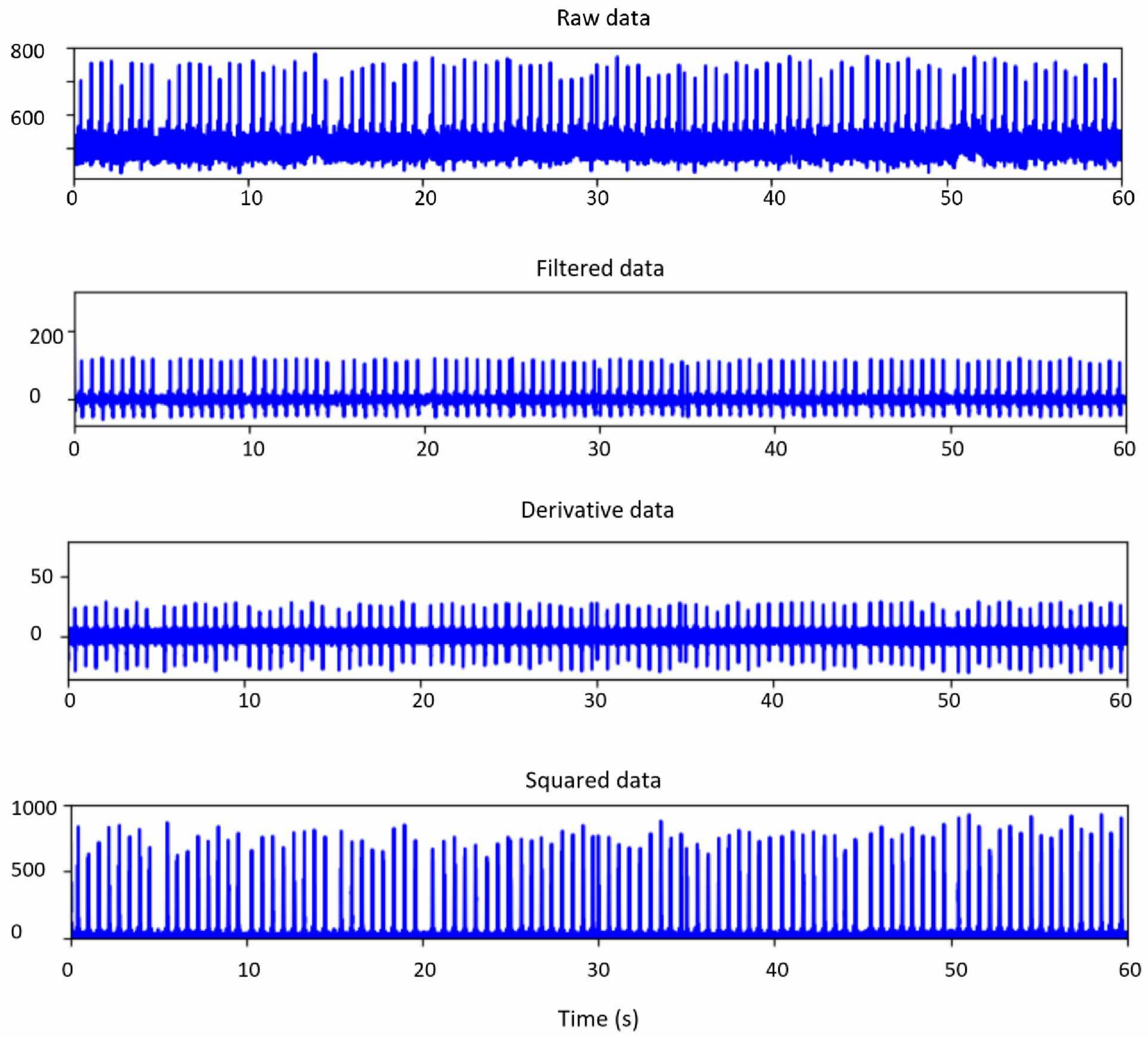
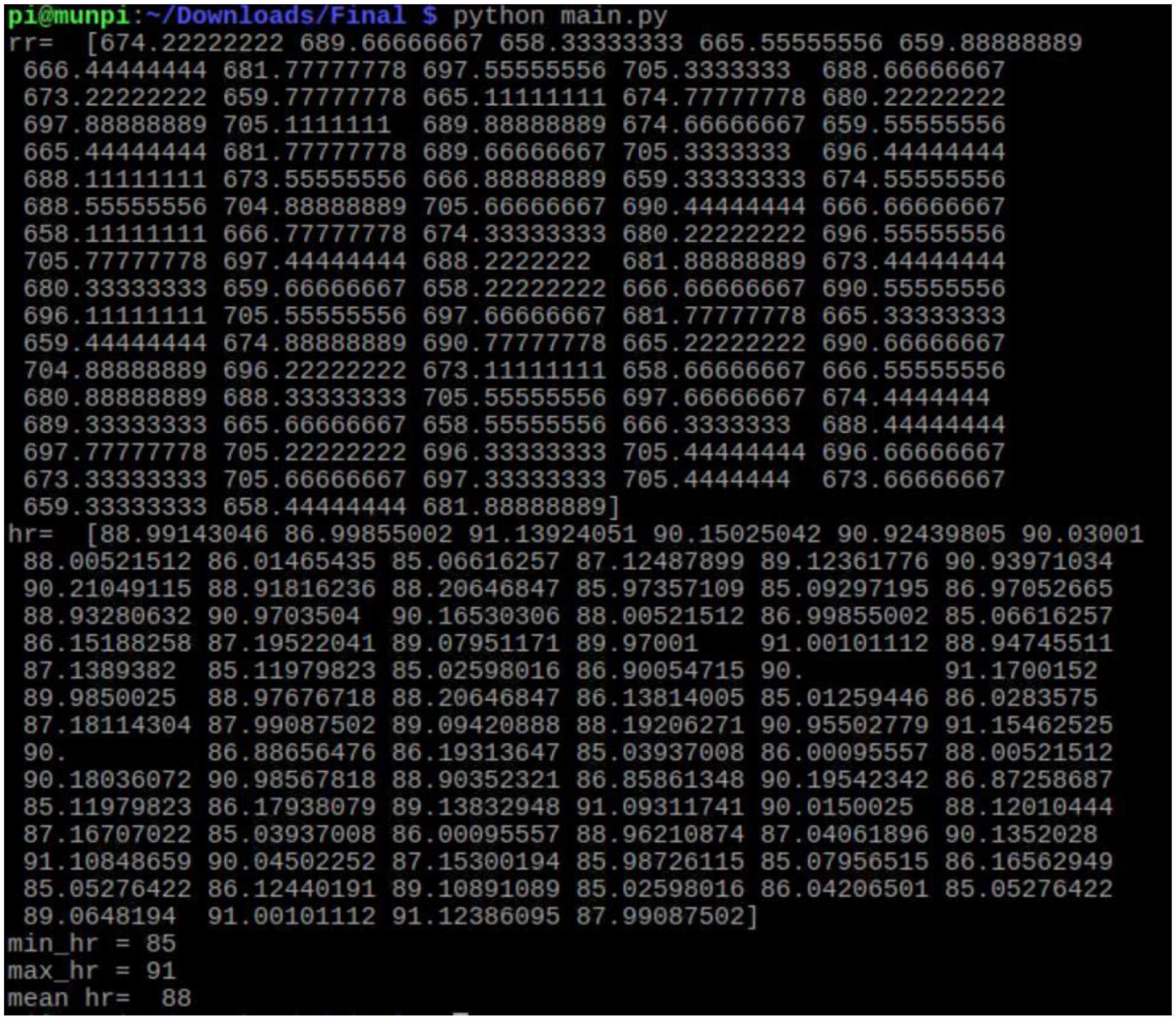


Figure 20: 60 seconds of subject B ECG raw, filtered, derivative and squared data.



```
pi@munpi:~/Downloads/Final $ python main.py
rr=  [674.22222222 689.66666667 658.33333333 665.55555556 659.88888889
 666.44444444 681.77777778 697.55555556 705.3333333  688.66666667
 673.22222222 659.77777778 665.11111111 674.77777778 680.22222222
 697.88888889 705.1111111  689.88888889 674.66666667 659.55555556
 665.44444444 681.77777778 689.66666667 705.3333333  696.44444444
 688.11111111 673.55555556 666.88888889 659.33333333 674.55555556
 688.55555556 704.88888889 705.66666667 690.44444444 666.66666667
 658.11111111 666.77777778 674.33333333 680.22222222 696.55555556
 705.77777778 697.44444444 688.2222222  681.88888889 673.44444444
 680.33333333 659.66666667 658.22222222 666.66666667 690.55555556
 696.11111111 705.55555556 697.66666667 681.77777778 665.33333333
 659.44444444 674.88888889 690.77777778 665.22222222 690.66666667
 704.88888889 696.22222222 673.11111111 658.66666667 666.55555556
 680.88888889 688.33333333 705.55555556 697.66666667 674.4444444
 689.33333333 665.66666667 658.55555556 666.3333333  688.44444444
 697.77777778 705.22222222 696.33333333 705.44444444 696.66666667
 673.33333333 705.66666667 697.33333333 705.4444444  673.66666667
 659.33333333 658.44444444 681.88888889]
hr=  [88.99143046 86.99855002 91.13924051 90.15025042 90.92439805 90.03001
 88.00521512 86.01465435 85.06616257 87.12487899 89.12361776 90.93971034
 90.21049115 88.91816236 88.20646847 85.97357109 85.09297195 86.97052665
 88.93280632 90.9703504  90.16530306 88.00521512 86.99855002 85.06616257
 86.15188258 87.19522041 89.07951171 89.97001    91.00101112 88.94745511
 87.1389382  85.11979823 85.02598016 86.90054715 90.         91.1700152
 89.9850025  88.97676718 88.20646847 86.13814005 85.01259446 86.0283575
 87.18114304 87.99087502 89.09420888 88.19206271 90.95502779 91.15462525
 90.         86.88656476 86.19313647 85.03937008 86.00095557 88.00521512
 90.18036072 90.98567818 88.90352321 86.85861348 90.19542342 86.87258687
 85.11979823 86.17938079 89.13832948 91.09311741 90.0150025  88.12010444
 87.16707022 85.03937008 86.00095557 88.96210874 87.04061896 90.1352028
 91.10848659 90.04502252 87.15300194 85.98726115 85.07956515 86.16562949
 85.05276422 86.12440191 89.10891089 85.02598016 86.04206501 85.05276422
 89.0648194  91.00101112 91.12386095 87.99087502]
min_hr = 85
max_hr = 91
mean hr=  88
```

Figure 21: Terminal output showing subject B heart rate (mean_hr is converted into integer) which indicates the beats per minute for the real-time ECG data implementation.