DSP Assignment 2
FIR Filter

Miss Ramon Suwanban (2495594S)
Mr Wikara Gunawan (2397833G)

due 9 November 2020 (3PM)

Declaration of Originality and Submission Information
I affirm that this submission is my own / the groups original work in accordance with the University of Glasgow Regulations and the School of Engineering Requirements.

Student Number: 2495594S Student Name: Miss Ramon Suwanban
Student Number: 2397833G Student Name: Mr Wikara Gunawan

1 ECG Filtering

1.1 Introduction

Fast Fourier Transform is a transformation which requires the full recording data to perform, so it is not suitable for real-time processing. As a result, the finite-impulse response filter or FIR filter which is a causal digital filter that only takes finite number of sample from the whole data to perform was implemented. However, note that FIR filter only do semi-real time processing because its implementation involves buffer which introduces time delay.

ECG signal measured is subjected to 50 Hz noise and DC line shift which needs to be removed. In cases that almost real time filtering is needed, such as filtering the ECG signal to display on a patient monitor, FIR filter is used instead of Fast Fourier Transform. However, the result from Fast Fourier Transform would actually be more accurate because it takes more sample to process.

FIR filter is written as a function that takes 1 data in and outputs 1 data at a time, so it is used for semi-real time processing. The process of an FIR filter is to sum up the multiplication of the coefficients of the filter and the corresponding data stored in buffer which will be shifted every time a new data is fed. Furthermore, we are using ring buffer to reduce computation resource from normally shifting the array operation.

1.2 Method and Results

The implementation of this part consists of 2 python files, fir_filter.py and ecg_filter.py. fir_filter.py contains the FIR filter class which needs coefficient array as the input. It will automatically runs the **unittest()** if it is run in the **main()** function. ecg_filter.py contains 2 filters made from the FIR filter class(from fir_filter.py) and imports raw ecg data to filter. It saves the results as shortecg.dat(from ECG.dat) and shorteint.dat(from Einthoven_ii Walking).

1.2.1 FIR-filter Function

```python
import numpy as np
class FIR_filter:
    def __init__(self,coefficients):
        self.coeff = coefficients
        self.offset = 0
        self.buffer = np.zeros(len(coefficients))
```

Listing 1: Initialise Constructor

Import the libraries used by the code.

- **numpy** : array processing and mathematical calculations

The constructor of the class is initialised with the local variables inside the constructor function: **coeff**, **offset** and **buffer**. **coeff** consists of the input coefficients or the impulse response of the filter. It will be taken from the input when you instantiate the filter class. **offset** is the position of the current data on the buffer of the processed data which is used only inside the filter class. Finally, **buffer** is created by having an array of zeros with the size of the coefficients number to store all delayed data.

```
def dofilter(self,u):
        result = 0
        self.buffer[self.offset] = u
        #print("offset index:",self.offset)

        for i in range(self.offset+1):
            result = result + self.buffer[i]*self.coeff[self.offset-i]
            #print("buffer index:",i)
            #print("coeff index:",self.offset-i)

        #print("Second For Section")
        for i in range(self.offset+1,len(self.buffer),1):
            result = result + self.buffer[i]*self.coeff[len(self.buffer)-1+self.offset+1-i]
            #print("buffer index:",i)
            #print("coeff index:",len(self.buffer)-1+self.offset+1-i)

        self.offset+=1
        if self.offset>=len(self.buffer):
            self.offset=0

        return result
```

Listing 2: Ring Buffer Function

New input from the outside is put into the **offset** th position of the buffer. Note that the **offset** increments every time a new data was input. It then resets at the maximum size of the length of the buffer, which is the length of the coefficient. There are 2 **for-loop** to operate the multiplication process. The index of the coefficients and the buffer slots multiplied are shown in figure 1.

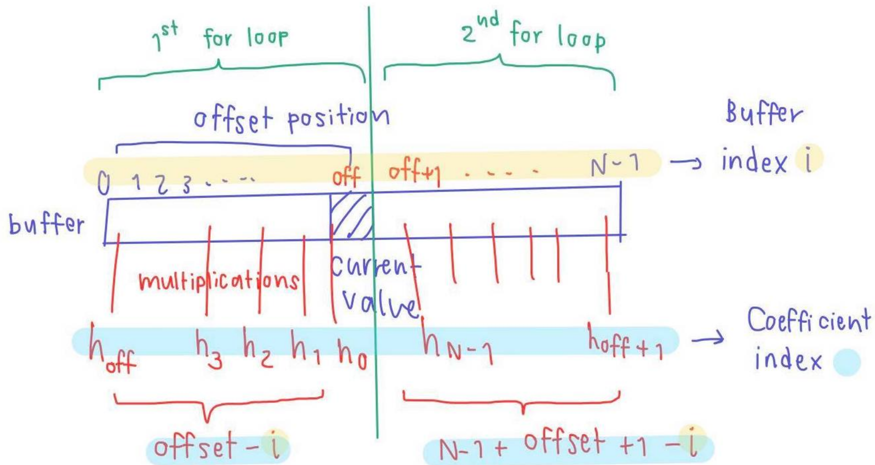

Figure 1: Ring Buffer Index Diagram

dofilterPrint is the exact same function as **dofilter**, but with the **print** lines uncommented. We use this **dofilterPrint** in **unittest()** only, so we can see which coefficient is being multiplied to which buffer. It also shows which pairs are being done on which **for-loop**.

```
    def dofilterPrint(self,u):      #index printing version for diagnosis purpose
        result = 0
        self.buffer[self.offset] = u
        print("offset index:",self.offset)

        for i in range(self.offset+1):
            result = result + self.buffer[i]*self.coeff[self.offset-i]
            print("buffer index:",i)
            print("coeff index:",self.offset-i)

        print("Second For Section")
        for i in range(self.offset+1,len(self.buffer),1):
            result = result + self.buffer[i]*self.coeff[len(self.buffer)-1+self.offset+1-i]
            print("buffer index:",i)
            print("coeff index:",len(self.buffer)-1+self.offset+1-i)

        self.offset+=1
        if self.offset>=len(self.buffer):
            self.offset=0

        return result
```

Listing 3: dofilterPrint

unittest() is testing with coefficient [0.5,0.5,0,0] and input [0,1,0,0,0,...](impulse). Because the solution is obvious[0,0.5,0.5,0,0,0,...], it can be used to check primarily if the filter is functioning correctly. The **print** function from the **dofilterPrint** also helps checking the indices.

```
def unittest():
        h = np.array([1/2,1/2,0,0,0])
        print("Coefficient:",h)
        f = FIR_filter(h)
        y= f.dofilterPrint(0)         #use the index printing version
        print("Input 0, Output",y)
        y= f.dofilterPrint(1)         #use the index printing version
        print("Input 1, Output",y)
        for i in range (20):
            y= f.dofilterPrint(0)     #use the index printing version
            print("Input 0, Output",y)

if __name__ == "__main__":
    unittest()
```

Listing 4: unittest()

1.2.2 ECGfilter.py

```
import numpy as np
from fir_filter import FIR_filter
from numpy import loadtxt
from matplotlib import pyplot

fs = 250     #Hz
M = 500      #loses 2 heart beat to warm up the filter (2seconds)
```

Listing 5: Import FIR_Filter Class

To initialize the code, **fir_filter.py** must be located in the same folder, so it could be imported to the code. The sampling frequency is 250 Hz, and the length of the coefficient is 500 elements(number of taps/number of delay buffer). The number of coefficient should be high enough to make frequency resolution becomes enough for the 0.5 Hz frequency cutoff in our DC filter design. Frequency resolution in this case equals 250 Hz / 500 taps = 0.5 Hz which is exactly how good resolution we need. The trade off for high coefficient number is that it takes time for the buffer to fill up before it can actually function.

```
# 50 Hz Notch Filter
k1 = int(49.5/fs * M)
k2 = int(50.5/fs * M)

Window50 = np.ones(M)
Coeff50 = np.ones(M)

Window50[k1:k2+1] = 0
Window50[M-k2:M-k1+1] = 0

```

```
pyplot.figure(1)
pyplot.plot(Window50)
pyplot.title('50 Hz Notch Filter - Ideal')
pyplot.xlabel('M (sample number)')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig1.eps', format='eps')

W50 = np.fft.ifft(Window50)
W50 = np.real(W50)

pyplot.figure(2)
pyplot.plot(W50)
pyplot.title('50 Hz Notch Filter - IFFT')
pyplot.xlabel('Coefficient Index')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig2.eps', format='eps')

Coeff50[0:int(M/2)] =  W50[int(M/2):M]
Coeff50[int(M/2):M] =  W50[0:int(M/2)]

pyplot.figure(3)
pyplot.plot(Coeff50)
pyplot.title('50 Hz Notch Filter - IFFT(fixed)')
pyplot.xlabel('Coefficient Index')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig3.eps', format='eps')

Filter50 = FIR_filter(Coeff50)
```

Listing 6: 50 Hz Bandstop Coefficient

To create coefficient array for the 50 Hz removal, an ideal notch filter(sequence of 0s and 1s) is created with targeting of 49.5 Hz(**k1**) to 50.5 Hz(**k2**) for the cutoff frequencies. The inverse fast-fourier transform is applied to the ideal filter array, then array manipulation is applied to shift the coefficients by half of the length of the coefficient to swap the positions to make it causal and intact.

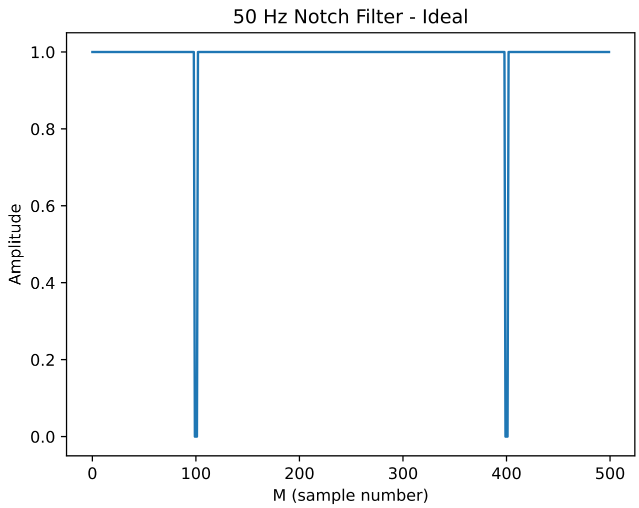

Figure 2: 50 Hz Notch Filter Ideal Frequency Domain Plot

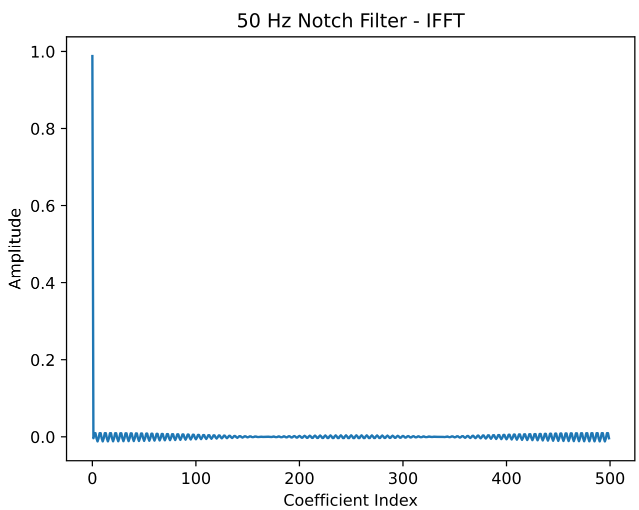

Figure 3: 50 Hz Notch Filter IFFT

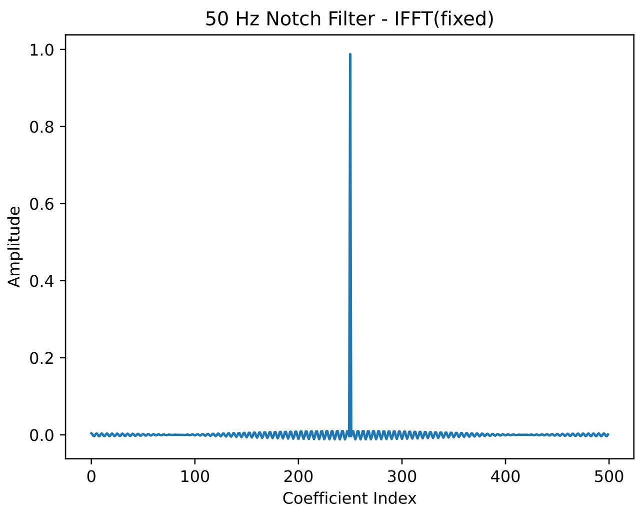

Figure 4: 50 Hz Notch Filter IFFT (fixed) - Coefficients

To remove the DC line shift from the data, a window function coefficient is created by targeting the 0.5 Hz as the cutoff frequency. The same procedure as the 50 Hz notch filter applies.

```
#DC filter
k3 = int(0.5/fs * M)
WindowDC = np.ones(M)
CoeffDC = np.ones(M)

WindowDC[0:k3+1] = 0
pyplot.figure(4)
pyplot.plot(WindowDC)
pyplot.title('DC Filter - Ideal')
pyplot.xlabel('M (sample number)')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig4.eps', format='eps')

WDC = np.fft.ifft(WindowDC)
WDC = np.real(WDC)

pyplot.figure(5)
pyplot.plot(WDC)
pyplot.title('DC Filter - IFFT')
pyplot.xlabel('Coefficient Index')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig5.eps', format='eps')

CoeffDC[0:int(M/2)] =  WDC[int(M/2):M]
CoeffDC[int(M/2):M] =  WDC[0:int(M/2)]

pyplot.figure(6)
pyplot.plot(CoeffDC)
```

```
pyplot.title('DC Filter - IFFT(fixed)')
pyplot.xlabel('Coefficient Index')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig6.eps', format='eps')

FilterDC = FIR_filter(CoeffDC)
```
Listing 7: DC Filter Coefficient

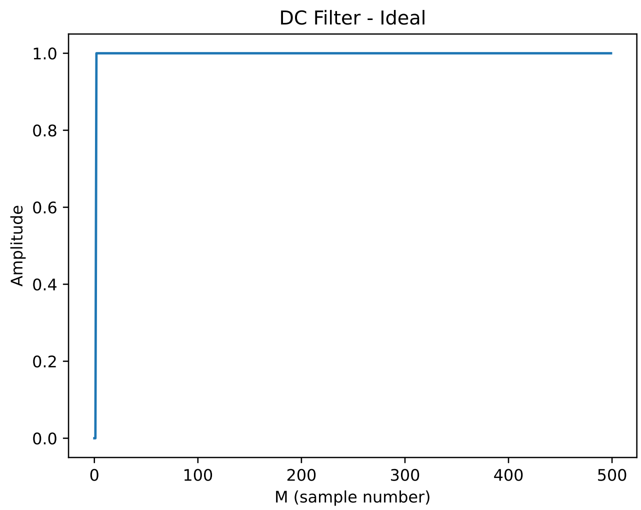

Figure 5: DC Removal Filter Ideal Frequency Domain Plot

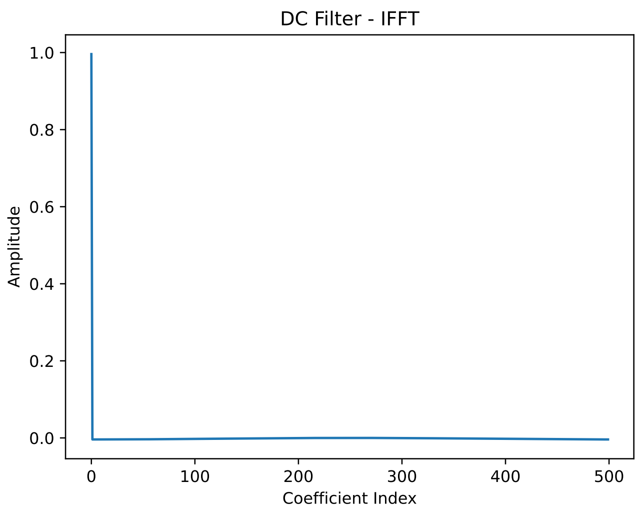

Figure 6: DC Removal Filter IFFT

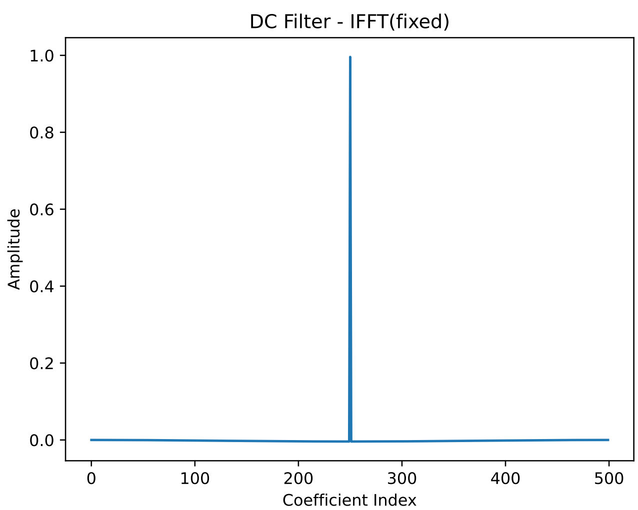

Figure 7: DC Removal Filter IFFT (fixed) - Coefficients

We load .dat ECG data to our file to perform semi-real time processing and plot them in time domain.

```
# load first ECG file
file1 = open('ECG.dat', 'r')     #line by line to use as real-time
    filter
```

```python

ecg = loadtxt("ECG.dat")        #for diagnosis plot

pyplot.figure(7)
print(len(ecg))
time = np.linspace(0,len(ecg)/fs,len(ecg))
pyplot.plot(time,ecg)
pyplot.title('ecg (raw)')
pyplot.xlabel('Time(s)')
pyplot.ylabel('Amplitude')
#pyplot.savefig('ecg_fig7.eps', format='eps')


# Investigate frequency domain
fx=np.fft.fft(ecg)
fxx = fx/len(ecg)             # Fourier Transform Normalised
dbs = 20*np.log10(abs(fxx))    # DB Conversion
pyplot.figure(8)
freq = np.linspace(0,fs,len(ecg))
pyplot.plot(freq,dbs)
pyplot.title('ecg (raw) - Frequency Domain')
pyplot.xlabel('Frequency (Hz)')
pyplot.ylabel('Amplitude')
#pyplot.savefig('ecg_fig8.eps', format='eps')

#load second ECG data
'''Initialise experiments from the files of einthoven'''
subject_number = 3
experiment = 'walking'
ecg_class = GUDb(subject_number, experiment)

'''Initialise experiments from the files of einthoven'''
chest_strap_V2_V1 = ecg_class.cs_V2_V1
einthoven_ii = ecg_class.einthoven_II

'''Filtered Data With Einthoven'''
ecg_class.filter_data()
einthoven_ii_filt = ecg_class.einthoven_II_filt

#define output array and intermediate variable
filterecg = np.zeros(len(ecg)+1)
filtereinthoven = np.zeros(len(einthoven_ii)+1)
intermediate = 0
```

```python
pyplot.figure(11)
#print(len(einthoven_ii))
time4 = np.linspace(0,len(einthoven_ii)/fs,len(einthoven_ii))
pyplot.plot(time4,einthoven_ii)
pyplot.title('einthoven_ii (raw)')
pyplot.xlabel('Time(s)')
pyplot.ylabel('Amplitude')
#pyplot.savefig('ecg_fig11.eps', format='eps')
```

Listing 8: Load .dat Files and Time Domain Plot

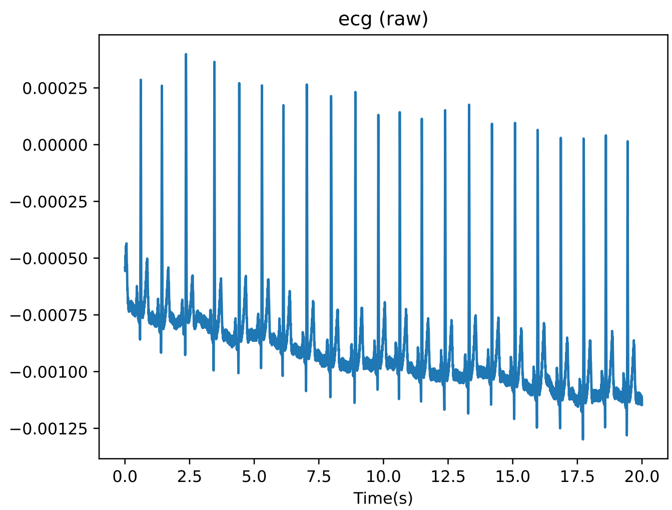

Figure 8: Raw ECG from ECG.dat

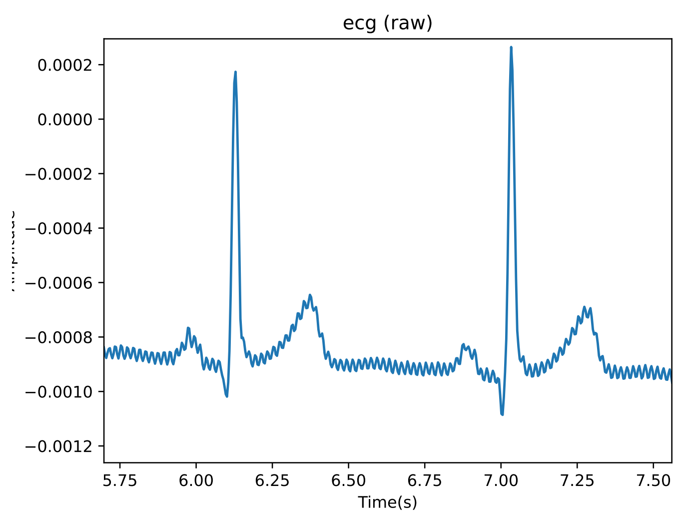

Figure 9: Partly Raw ECG from ECG.dat

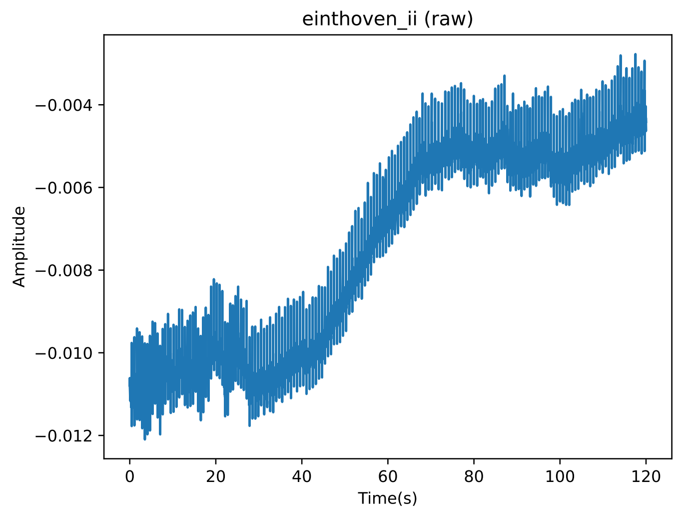

Figure 10: Raw ECG from Einthoven_ii Walking

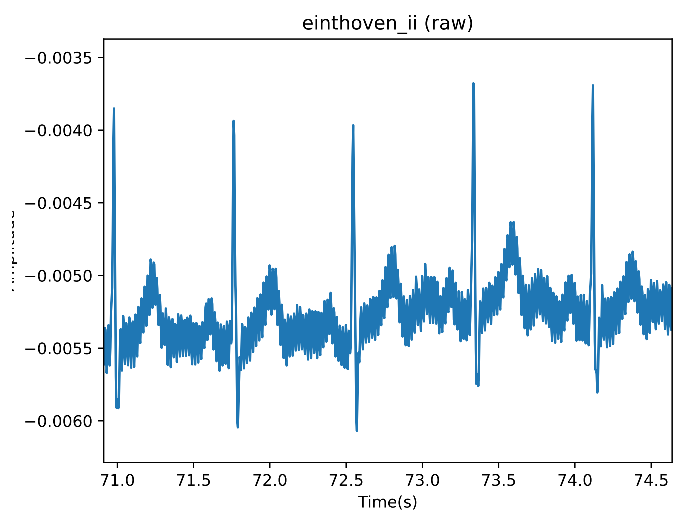

Figure 11: Partly Raw ECG from Einthoven_ii Walking

We perform both 50 Hz notch filter and DC filter to both data sets and plot the filtered ECG.

```
#filter ecg for both data sets
count = 0
```

```
for line in file1:
    count += 1
    ecg1 = line.strip()
    #print(ecg1)
    intermediate = Filter50.dofilter(ecg1)
    filterecg[count] = FilterDC.dofilter(intermediate)

for i in range(len(einthoven_ii)):
    intermediate = Filter50.dofilter(einthoven_ii[i])
    filtereinthoven[i] = FilterDC.dofilter(intermediate)

#print(count)

#plot the filtered data
pyplot.figure(9)
time2 = np.linspace(0,len(filterecg)/fs,len(filterecg))
pyplot.plot(time2,filterecg)
#pyplot.ylim(-0.002,0.002)
pyplot.title('ecg (filtered)')
pyplot.xlabel('Time(s)')
pyplot.ylabel('Amplitude')
#pyplot.savefig('ecg_fig9.eps', format='eps')

pyplot.figure(10)
time3 = np.linspace(0,len(filtereinthoven)/fs,len(filtereinthoven))
pyplot.plot(time3,filtereinthoven)
pyplot.ylim(-0.002,0.002)
pyplot.title('einthoven (filtered)')
pyplot.xlabel('Time(s)')
pyplot.ylabel('Amplitude')
#pyplot.savefig('ecg_fig10.eps', format='eps')
```

Listing 9: Filter operation and Filtered Time Domain Plot

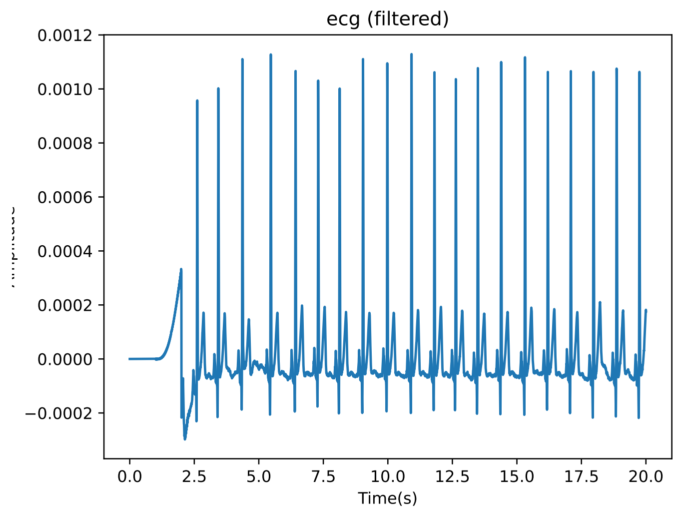

Figure 12: Filtered ECG from ECG.dat

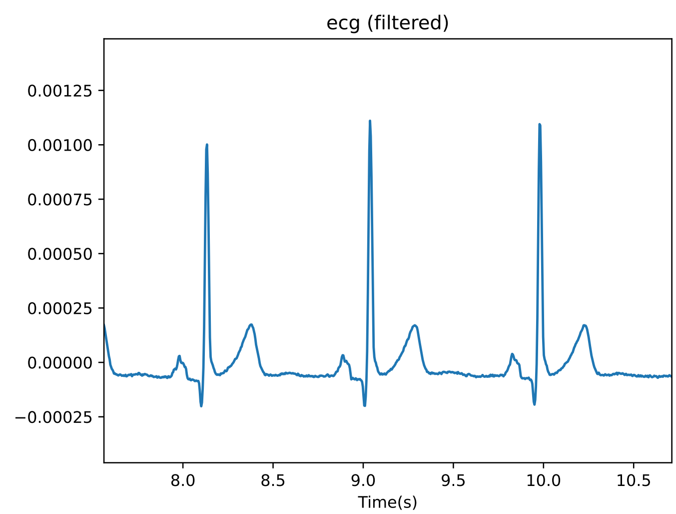

Figure 13: Partly ECG from ECG.dat

From figure 13, the ECG from the filter does still look intact.

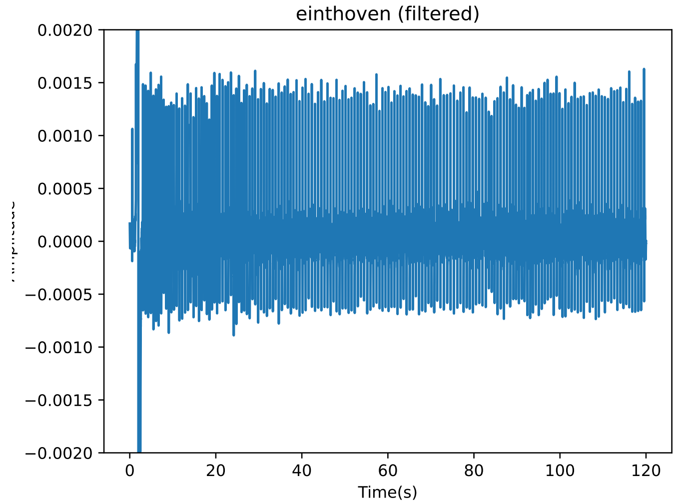

Figure 14: Filtered ECG from Einthoven_ii Walking

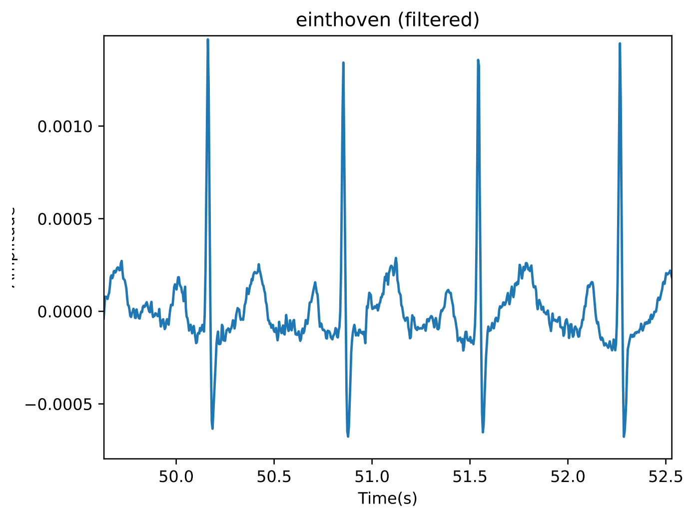

Figure 15: Partly Filtered ECG from Einthoven_ii Walking

We can see that the filtered plots are a lot cleaner and no DC line visible.

The first part of both filtered ECGs are invalid because the filter is being filled up as mentioned earlier. There are about 2 heart beat loss.

We saved the processed ECG data into another .dat file to be used for heart rate detection.

```
#save filtered data
np.savetxt('shortecg.dat',filterecg)
np.savetxt('shorteint.dat',filtereinthoven)
```

Listing 10: Save .dat Files

1.3 Discussion

We can use ring buffer to reduce computation steps required from shifting operation.

```
'''
import numpy as np
# Inefficient way
class FIR_filter:
    def __init__(self,coefficients):
        self.coeff = coefficients
        self.offset = 0
        self.buffer = np.empty(len(coefficients))

    def dofilter(self,u):
        result = 0
        for i in range(len(self.buffer)-1,0,1): #range(start,end,step)
            self.buffer[i+1] = self.buffer[i]
        self.buffer[0] = u
        for i in range(len(self.buffer)-1):
            result = result + self.coeff[i] * self.buffer[i]
        return result
'''
```

Listing 11: Not Using Ring Buffer Version

The fundamental frequency of the ECG starts around 1 Hz, so the cutoff frequency should be below 1 Hz to avoid eliminating the ECG signal itself.

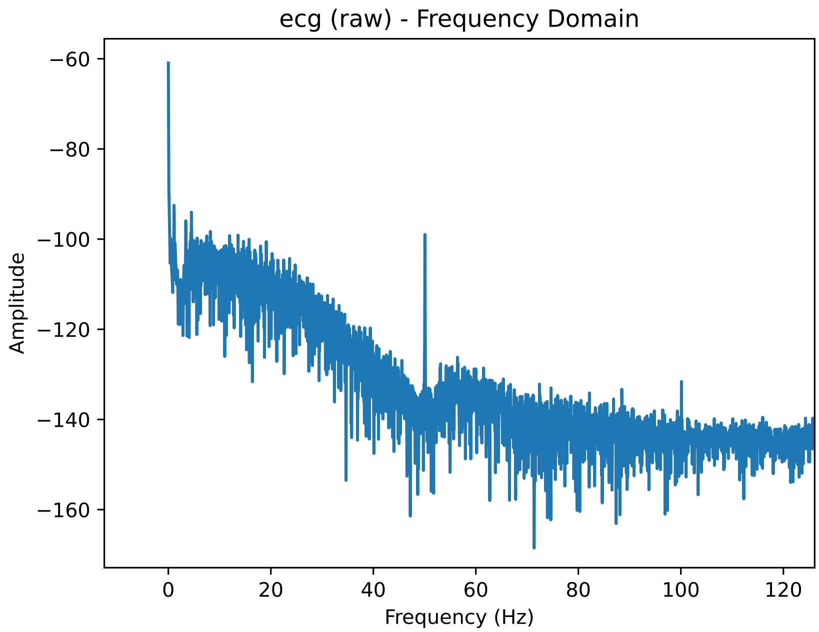

Figure 16: Raw ECG in Frequency Domain from ECG.dat

From figure 16, we could observe peak at 0 Hz (DC) and a peak at 50 Hz noise. The other spectrum amplitude is high near lower frequencies which implies that the ECG signal composes of lower frequencies. To conclude, from this frequency domain plot of the spectrum, we could decide on which cutoff frequencies should be used in our filter design.

2 HeartRate Detection

2.1 Introduction

Detecting heart-rate peaks from the ECG data requires a DC free signal, so we used the data processed from the previous ecg_filter.py. The template is created by trimming one of a PQRST from the corresponding ECG. Then inverse of the template is used as the coefficient of the FIR Filter. An anti-noise heartbeat algorithm is implemented to the code to prevent noise detected as an extra irregular heartbeat. Time difference from one peak to another is processed to get the beats per minute.

Einthoven ii walking ECG recording, which provides greater noise as a benchmark test of the noise filtering, is loaded to test the heart-rate detection program.

2.2 Method and Results

```python
import numpy as np
from matplotlib import pyplot
from numpy import loadtxt
from fir_filter import FIR_filter

Case = 0
#Case = int(input("Case 0 For shortecg.dat. Case 1 For Einthoven_ii
    Walking. Please enter case number: "))
for Case in range (2):
    if Case == 0:
        cleanecg = loadtxt("shortecg.dat")
    if Case == 1:
        cleanecg = loadtxt("shorteint.dat")

    fs=250      #sampling frequency of the data

    pyplot.figure(8*Case+1)
    time = np.linspace(0,len(cleanecg)/fs,len(cleanecg))
    pyplot.plot(time,cleanecg)
    pyplot.title('ecg (filtered)')
    pyplot.xlabel('Time(s)')
    pyplot.ylabel('Amplitude')
    # if Case == 0:
    #     pyplot.savefig('hr_fig1.eps', format='eps')
    # if Case == 1:
    #     pyplot.savefig('hr_fig9.eps', format='eps')

    #ecg templates for each data set
    if Case == 0:
        template=cleanecg[775:975]
    if Case == 1:
        template=cleanecg[7050:7225]

    #plot template
    pyplot.figure(8*Case+2)
    time2 = np.linspace(0,len(template)/fs,len(template))
    pyplot.plot(time2,template)
    pyplot.title('1 ecg')
    pyplot.xlabel('Time(s)')
    pyplot.ylabel('Amplitude')
    # if Case == 0:
    #     pyplot.savefig('hr_fig2.eps', format='eps')
    # if Case == 1:
    #     pyplot.savefig('hr_fig10.eps', format='eps')

    #inverse the template
    fir_coeff = template[::-1]
    pyplot.figure(8*Case+3)
    pyplot.plot(time2,fir_coeff)
    pyplot.title('reversed 1 ecg')
    pyplot.xlabel('Time(s)')
    pyplot.ylabel('Amplitude')
    # if Case == 0:
    #     pyplot.savefig('hr_fig3.eps', format='eps')
    # if Case == 1:
```

```
#       pyplot.savefig('hr_fig11.eps', format='eps')
```

Listing 12: Obtaining Coefficient for FIR filter

Two cases are implemented to switch between the shortecg.dat and the Einthoven II walking ECG. The templates were eye-observed from the graphs, and then trimmed for the length of one PQRST Complex. Then the template is inversed corresponding the x-axis.

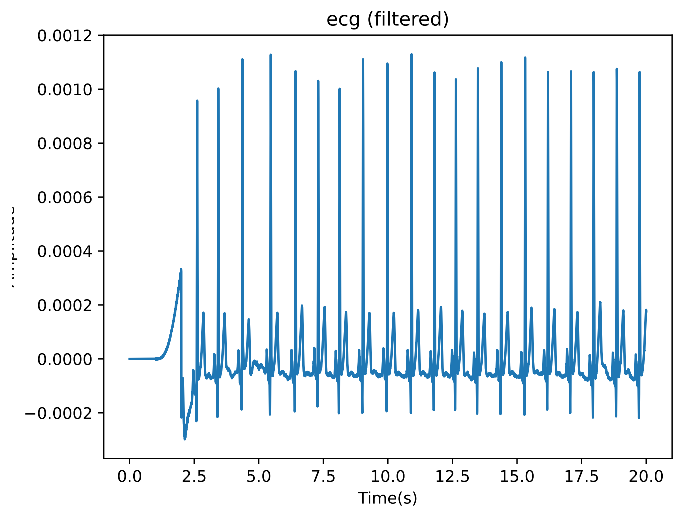

Figure 17: Filtered ECG from ECG.dat

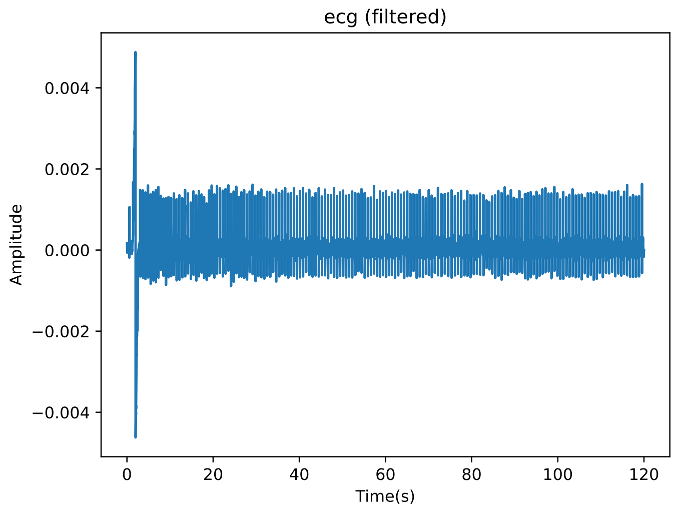

Figure 18: Filtered ECG from Einthoven_ii Walking

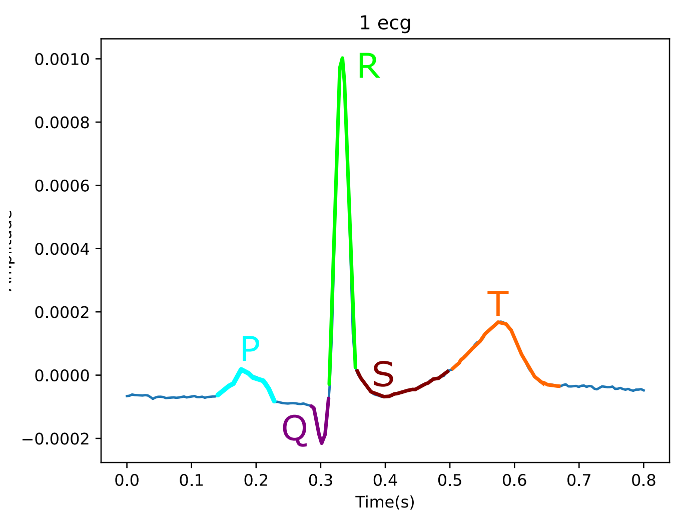

Figure 19: 1 ECG from ECG.dat

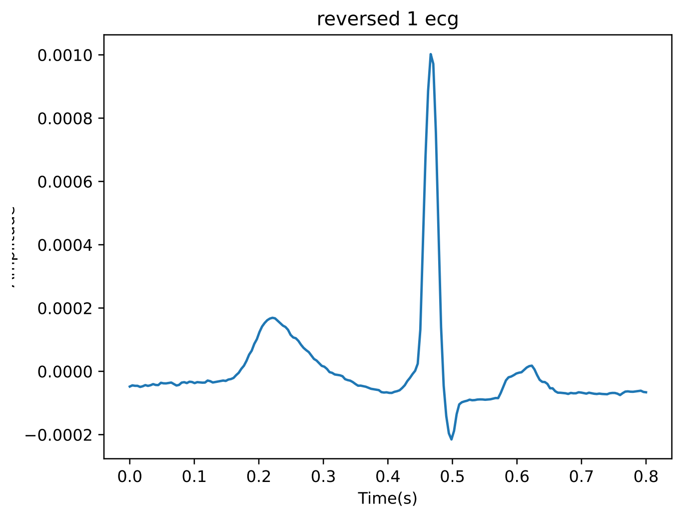

Figure 20: Reversed 1 ECG from ECG.dat

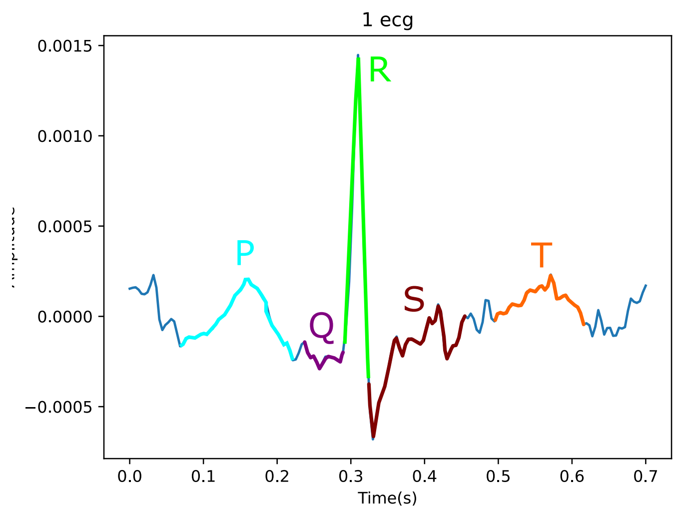

Figure 21: 1 ECG from Einthoven_ii Walking

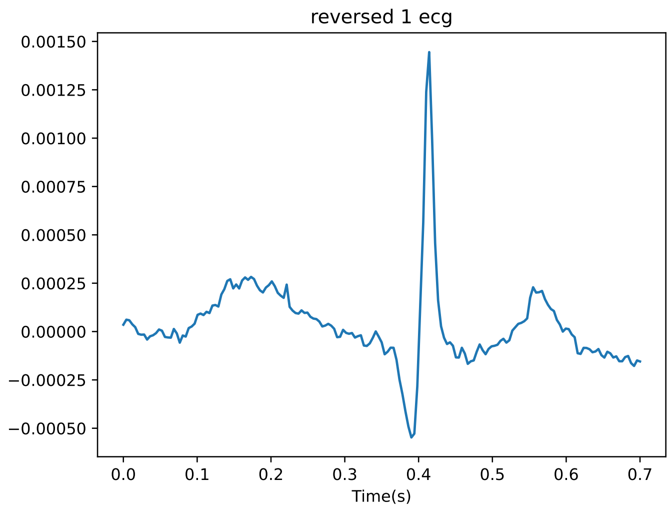

Figure 22: Reversed 1 ECG from Einthoven_ii Walking

```
#matched filtered data
matchfilt = FIR_filter(fir_coeff)

matchresult = np.zeros(len(cleanecg))
for i in range(len(cleanecg)):
    matchresult[i] = matchfilt.dofilter(cleanecg[i])

pyplot.figure(8*Case+4)
pyplot.plot(time,matchresult)
pyplot.title('Matched Filtered ecg')
pyplot.xlabel('Time(s)')
pyplot.ylabel('Amplitude')
# if Case == 0:
#     pyplot.savefig('hr_fig4.eps', format='eps')
# if Case == 1:
#     pyplot.savefig('hr_fig12.eps', format='eps')
print(len(matchresult))
```

Listing 13: Matched Filter Implementation

FIR Filter is used to perform matched filter. The coefficients are the inverted template, which is the **fir-coeff**.

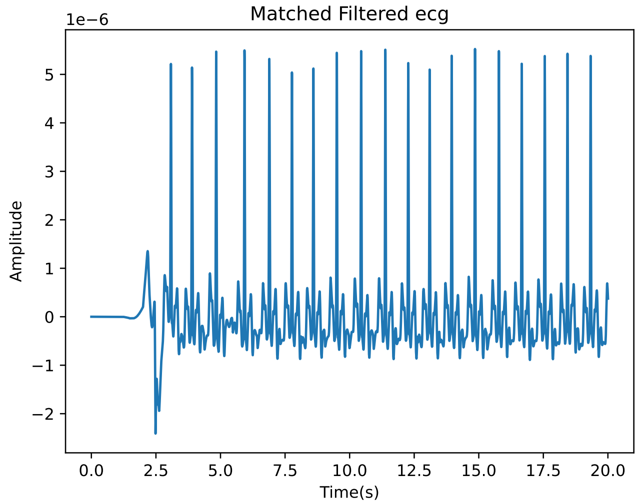

Figure 23: Match Filter Result from ECG.dat

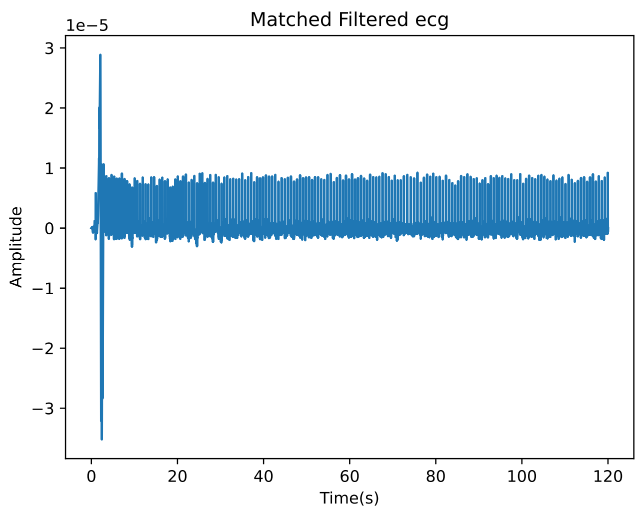

Figure 24: Match Filter Result from Einthoven_ii Walking

```
#squared matched filtered data
    matchresult = matchresult*matchresult
    pyplot.figure(8*Case+5)
    pyplot.plot(time,matchresult)            #
    pyplot.title('Squared Matched Filtered ecg')
```

```
    pyplot.xlabel('Time(s)')
    pyplot.ylabel('Amplitude')
    # if Case == 0:
    #     pyplot.savefig('hr_fig5.eps', format='eps')
    # if Case == 1:
    #     pyplot.savefig('hr_fig13.eps', format='eps')
```

Listing 14: Squaring the Results

By squaring the matched results post-filter, the outcome of the data produces higher peak with lower noise.

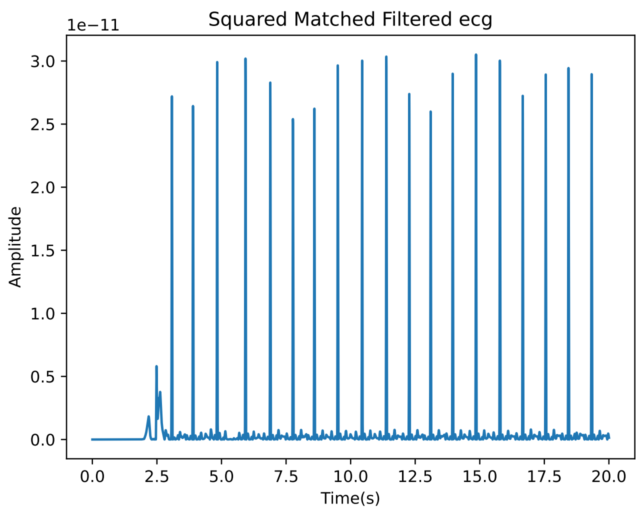

Figure 25: Squared Match Filter Result from ECG.dat

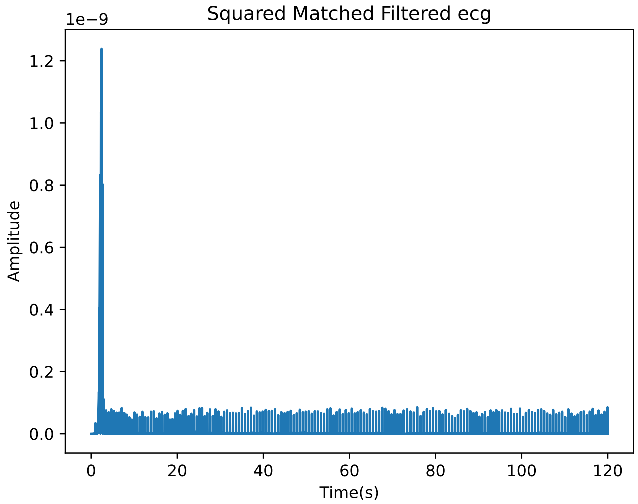

Figure 26: Squared Match Filter Result from Einthoven_ii Walking

```
    #using threshold to extract the peaks
    hr = np.zeros(len(matchresult))
    threshold = 0.00000000002
    for i in range(len(matchresult)):
        if matchresult[i]>threshold:
            hr[i]=1

    pyplot.figure(8*Case+6)
    pyplot.plot(hr)
    pyplot.title('Heart Beat Detection Sequence')
    pyplot.xlabel('Sample number')
    pyplot.ylabel('Amplitude')
    #pyplot.xlim(3,120)              #to delete the first part where
     the buffer is being filled/diagnosis purpose
    # if Case == 0:
    #     pyplot.savefig('hr_fig6.eps', format='eps')
    # if Case == 1:
    #     pyplot.savefig('hr_fig14.eps', format='eps')
```

Listing 15: Peak Threshold

The threshold used to detect the peaks of the data is around

$$2.0 * 10^{-}11$$

The value was observed from time domain plot of the squared matched result.

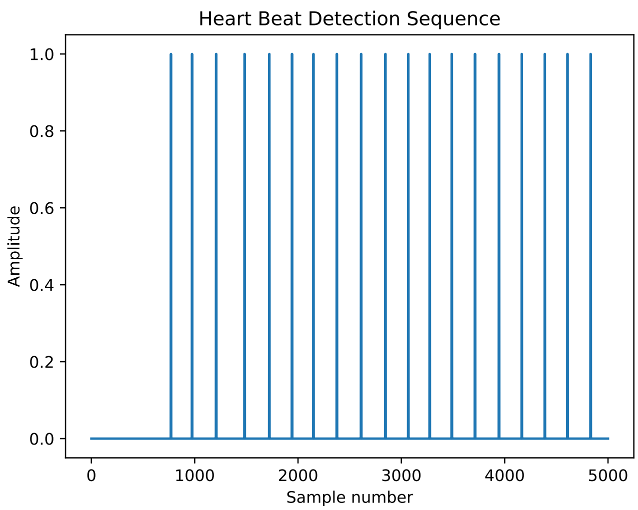

Figure 27: Heart Beat Extraction using Threshold from ECG.dat

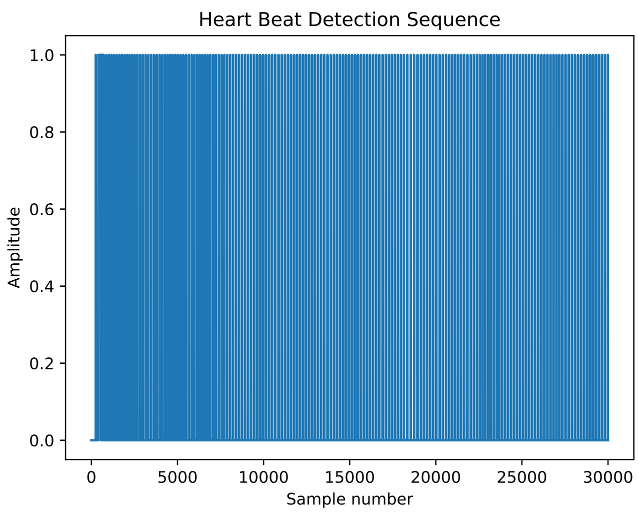

Figure 28: Heart Beat Extraction using Threshold from Einthoven_ii Walking

```
    #initiate counters
    peakCounter = 0   #count number of peaks > for array size
    detectIndex = 0   #detection index
    deltatIndex = 0   #deltat index

```

```python
    index = np.zeros(len(hr))
    threshT = int(1/(3.66/fs))    #3.66beat/s is the maximum heart rate(220bpm)

    for i in range(len(hr)):
        if hr[i] == 1:
            peakCounter+=1
            for n in range(int(threshT)):    #fix the impossible detected peak to 0
                if i+n+1 <= 30000:          #If the time is less than 30000
                    hr[i+n+1] = 0

    pyplot.figure(8*Case+7)
    pyplot.plot(hr)
    pyplot.title('Heart Beat Detection Sequence Fixed')
    pyplot.xlabel('Sample number')
    pyplot.ylabel('Amplitude')
    #pyplot.xlim(3,120)         #to delete the first part where the buffer is being filled/diagnosis purpose
    # if Case == 0:
    #     pyplot.savefig('hr_fig7.eps', format='eps')
    # if Case == 1:
    #     pyplot.savefig('hr_fig15.eps', format='eps')
```

Listing 16: Time Threshold

A time threshold **threshT** is used to delete unwanted/false 1s sequence between two peaks. A constant value of 3.66 beat/s is the highest heart-rate that would occur. The threshold is interpolated into the array index length. For any second detected 1s within that time threshold will be set to zero.

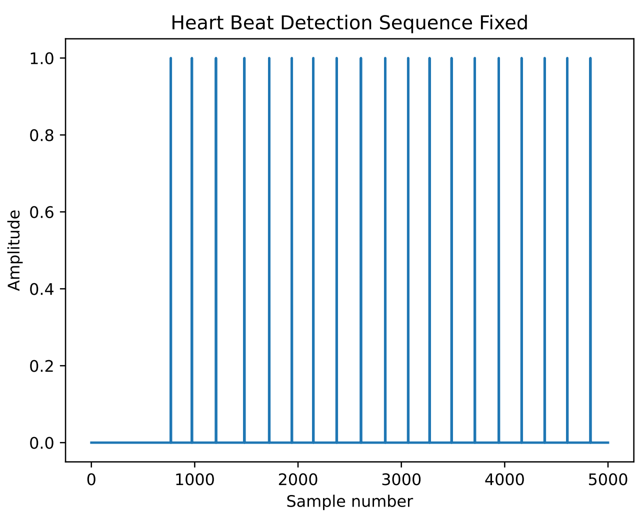

Figure 29: Heart Beat Extraction False 1s Deleted from ECG.dat

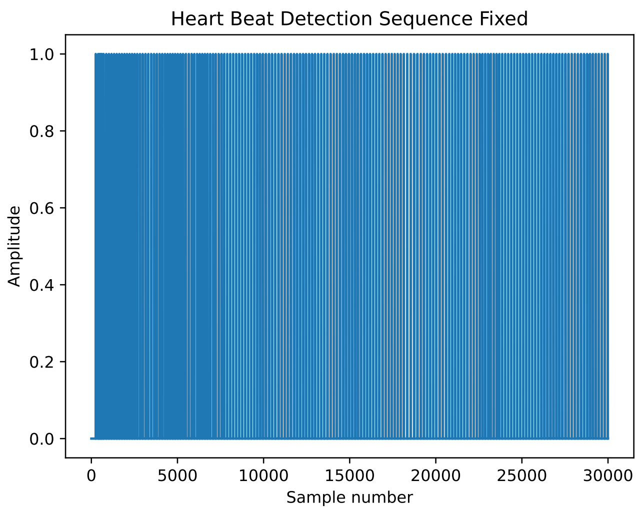

Figure 30: Heart Beat Extraction False 1s Deleted from Einthoven_ii Walking

From figure 31, we can see the 1s from the sequence is more than 1 due to the use of amplitude threshold. It's then proven to be fixed in figure 32 with the correction algorithm. Any other false peak in between time threshold would also get deleted.

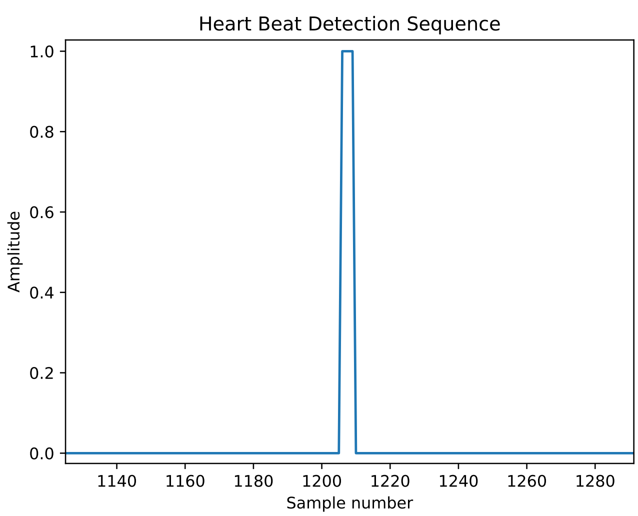

Figure 31: Partly Heart Beat Extraction using Threshold from ECG.dat

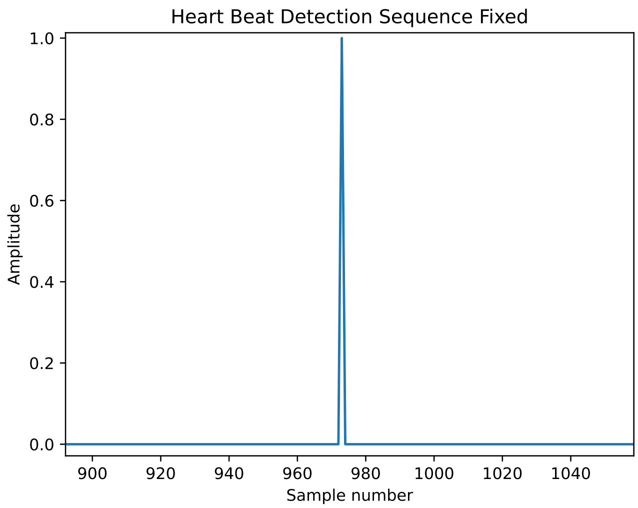

Figure 32: Partly Heart Beat Extraction False 1s Deleted from ECG.dat

```
    beatone = np.zeros(peakCounter)
    deltat = np.zeros(peakCounter)
    rate = np.zeros(peakCounter)

    print("Momentary Heart Rate(BPM)")
    for i in range(len(hr)):
        if hr[i] != 0:           # When element is not zero, input element to new array
            beatone[detectIndex] = i
            detectIndex += 1
            if detectIndex >= 0:           # To neglect the first detection
                deltat[deltatIndex] = (beatone[deltatIndex] - beatone[deltatIndex-1])/fs
                # print('ONE',beatone[deltatIndex])        #for diagnosis purpose
                # print('TWO',beatone[deltatIndex-1])
                # print('TIME',deltat[deltatIndex])
                rate[deltatIndex] = (1/deltat[deltatIndex])*60
                print(rate[deltatIndex])
                deltatIndex += 1

    pyplot.figure(8*Case+8)
    pyplot.plot(beatone/fs,rate)
    pyplot.title('Heart Rate Plot')
    pyplot.xlabel('Time(s)')
    pyplot.ylabel('Beats/Minute')
    #pyplot.xlim(4,max(beatone)/fs+1)         #to delete the first part where the buffer is being filled/diagnosis purpose
    # if Case == 0:
    #     pyplot.savefig('hr_fig8.eps', format='eps')
    # if Case == 1:
```

```
#       pyplot.savefig('hr_fig16.eps', format='eps')
```

Listing 17: Momentary Heart Rate Calculation

To calculate the Momentary Heart Rate (BPM), the time difference **deltat** is extracted from the sequence and then calculated into BPM.

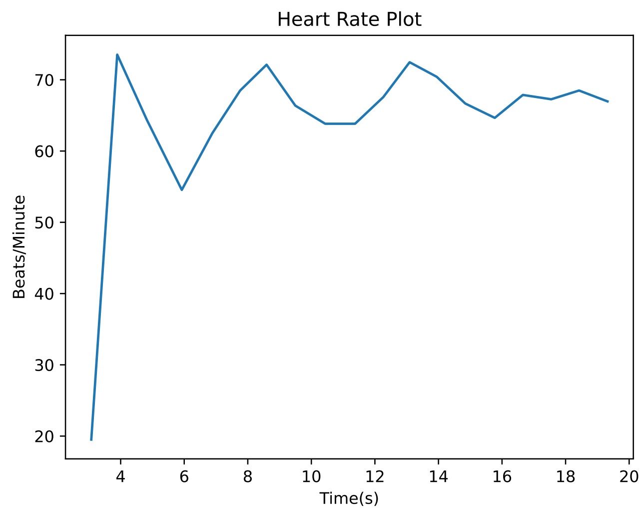

Figure 33: Momentary Heart Rate Plot from ECG.dat

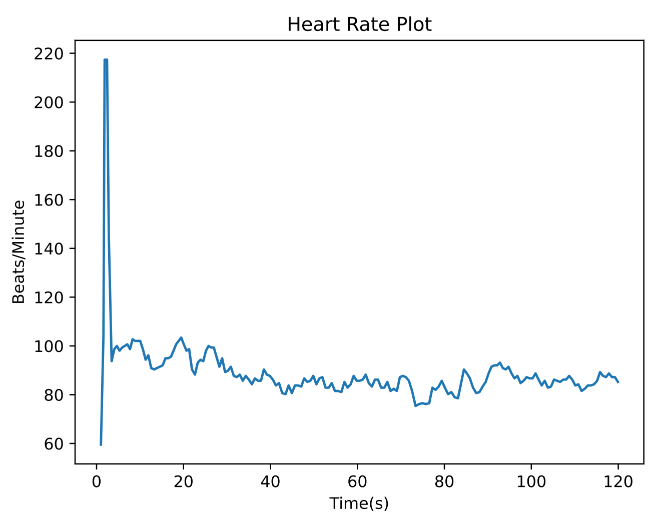

Figure 34: Momentary Heart Rate Plot from Einthoven_ii Walking

To comply with real-time processing, we have also implemented a real time heart rate detection python class using all above principle. Small changes are made to **deltat** and is replaced by **self.counter** which resets itself everytime a new peak is detected. The **threshold** also was lowered to make it be able to detect heart beat from eintoven ii data. Note that the data input into the filter instantiated from the class should be filtered by the 50 Hz notch and DC line elimination filter first. **And the file submitted would only run this part to display only the momentary rate graph. It also represents how it will be used in real application.** The result looks similar to figure 33 and figure 34, but only plotting at when the new heart beat has arrived and the momentary heart rate can be calculated.

```
class HR_filter:
    def __init__(self,):
        ecg = loadtxt("shortecg.dat")            #file which the template is extracted from
        template=ecg[775:975]                    #1ecg of the corresponding file
        fir_coeff = template[::-1]               #reverse template
        self.matchfilt = FIR_filter(fir_coeff)
        self.hr = 0
        self.hrBuffer = 0                        #keep previous hr value for peak starts checking
        self.counter = 0                         #counts time difference before next peak arrive

    def realtimeHR(self,data,fs):
        matchresult = self.matchfilt.dofilter(data)
        matchresult = matchresult*matchresult
        threshold = 0.00000000001                #observed by eye from graph

        #use threshold to extract peak area
        if matchresult>threshold:
            self.hr=1
        else:
            self.hr=0

        threshT = int(1/(3.66/fs))                #3.66beat/s is the maximum heart rate(220bpm)
        rate=0                                   #default argument

        #checks if the peak starts
        if self.hrBuffer==0 and self.hr==1:
            #peak=1      #for diagnosis purpose : peak starts
            if self.counter>threshT:
                rate = (fs/self.counter)*60
                self.counter=0                       #reset counter
            return rate
        else:
            #peak=0      #for diagnosis purpose : no peak starting
            self.counter+=1
            return None
        #return matchresult
        self.hrBuffer = self.hr                  #save value for
```

```
    future comparison


#Example Use
data = loadtxt("shortecg.dat")
data2 = loadtxt("shorteint.dat")
fs=250
mhr = np.zeros(len(data))
mhr2 = np.zeros(len(data2))
hrfilt = HR_filter()                                    #instantiate the
    HR_filter
for i in range(len(data)):
    mhr[i] = hrfilt.realtimeHR(data[i],fs)
for i in range(len(data2)):
    mhr2[i] = hrfilt.realtimeHR(data2[i],fs)

pyplot.figure(17)
pyplot.plot(np.linspace(0,len(data)/fs,len(data)),mhr)
pyplot.title('Momentary Heart Rate Plot from ECG.dat')
pyplot.xlabel('Time(s)')
pyplot.ylabel('Beats/Minute')
#pyplot.savefig('hr_fig17.eps', format='eps')

pyplot.figure(18)
pyplot.plot(np.linspace(0,len(data2)/fs,len(data2)),mhr2)
pyplot.title('Momentary Heart Rate Plot from einthoven II')
pyplot.xlabel('Time(s)')
pyplot.ylabel('Beats/Minute')
#pyplot.savefig('hr_fig18.eps', format='eps')
```
Listing 18: Momentary Heart Rate Calculation

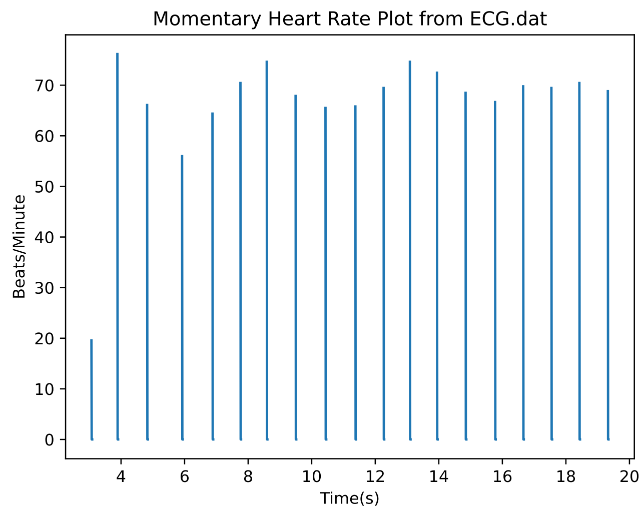

Figure 35: Momentary Heart Rate Plot from shortecg.dat using filter class implemented

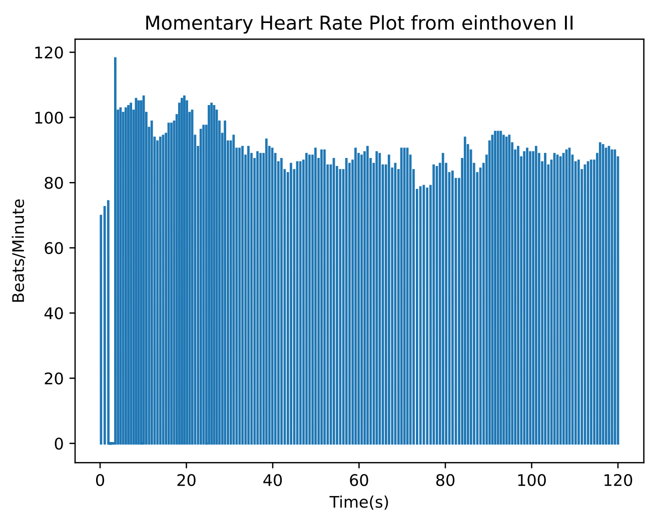

Figure 36: Momentary Heart Rate Plot from shorteint.dat using filter class implemented

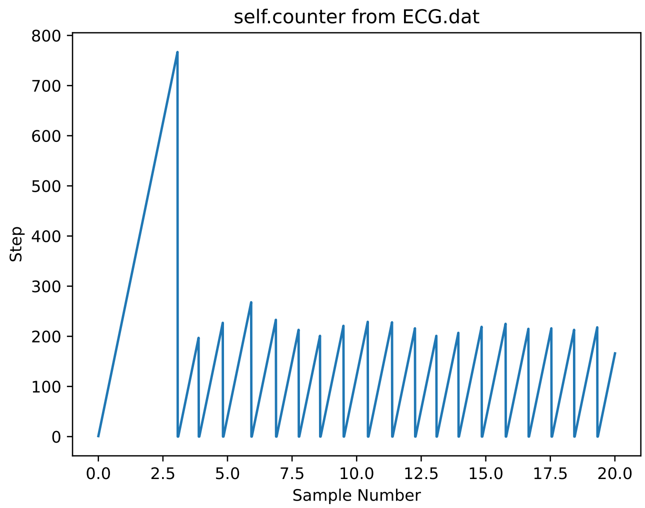

Figure 37: self.counter Uses of Detecting Heart Beat from shortecg.dat

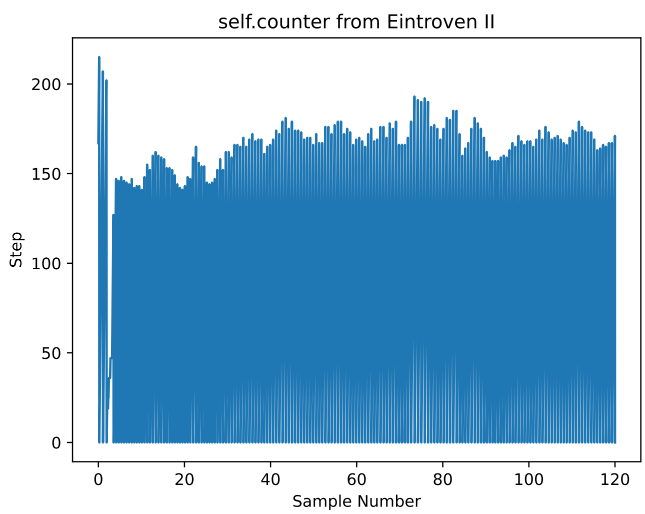

Figure 38: self.counter Uses of Detecting Heart Beat from shorteint.dat

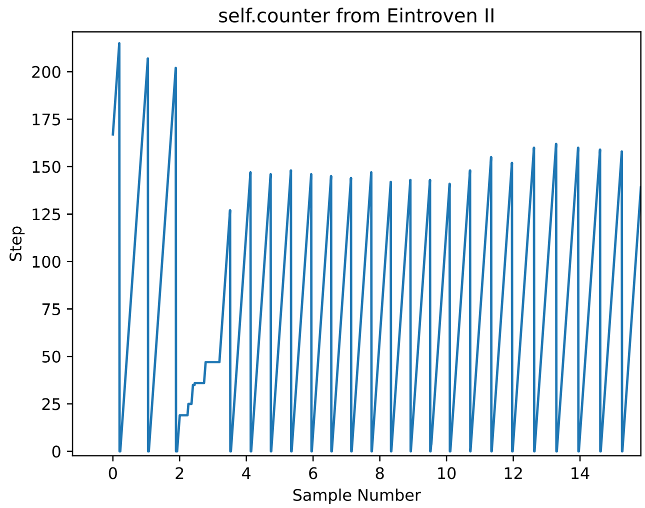

Figure 39: Partly self.counter Uses of Detecting Heart Beat from shorteint.dat

2.3 Discussion

Figure 20 shows the use of time-inversed template for the matched filter, as it indicates PQRST points on the template. The template is time-reversed to fulfil the requirements to perform convolution with the input signal. The template taken from one of the PQRST complex is then used as the coefficient to perform a matched filter to the data. Results of the match filter on figure 23 shows an increase of the ratio of the peaks to the noise. After squaring the data, a threshold is applied to the data to capture the peaks, which can be seen on figure 27. The detected sequence are in form of an array which consists of ones and zeros. The detected heart-rate plot on figure 32 shows the trends of the Beats Per Minute under 20 seconds of the sampled data.

The Einthoven II walking data is presumed to have greater noise occurring throughout the 50Hz, which may disrupt the R-peak detection. On figure 21 we indicate the PQRST complex on the template sampled from the Einthoven II which looks obviously more noisy than that of the shortecg.dat one. Figure 24 is the reversed template. Under the same process alike the shortecg.dat data, we use square matched filter and thresholds application to refine the peaks of the beats. After the calculation of the Beats/Minute, the momentary heart rate plot is shown on figure 28 within 120 seconds. Note that the array taken to complete the full buffer invalidate the value on the early times when the buffer is being filled up. Again, a trade-off is introduced between frequency resolution and the length of data being invalidated by the buffer.

We can actually use a same template for any data as demonstrated by the python class implemented. It doesn't matter which data the template is extracted from, but the template must represent an ECG signal(with all the ECG artefacts/PQRST).

As the outcome, the python class instantiated filter successfully detected the heartrate of the Einthoven II walking with lower **threshold**.

In addition, we tried to plot the raw vs. filtered data of a more noisy database. From figure 40, the raw data in orange obviously demonstrate a DC line shift, while the filtered one line shift is eliminated by the filters applied in ecg_filter.py. When it is used in hr_detect.py, the line shift will not affect the matched filter result.

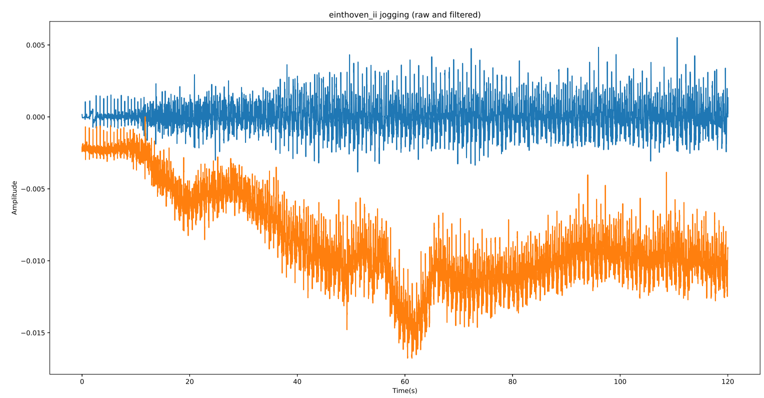

Figure 40: Raw vs. Filtered data from Einthoven_ii Jogging

3 Appendix

3.1 fir_filter.py

```
'''
import numpy as np
# Inefficient way
class FIR_filter:
    def __init__(self,coefficients):
        self.coeff = coefficients
        self.offset = 0
        self.buffer = np.empty(len(coefficients))

    def dofilter(self,u):
        result = 0
        for i in range(len(self.buffer)-1,0,1): #range(start,end,
    step)
            self.buffer[i+1] = self.buffer[i]
        self.buffer[0] = u
        for i in range(len(self.buffer)-1):
            result = result + self.coeff[i] * self.buffer[i]
```

```
        return result

'''
import numpy as np
class FIR_filter:
    def __init__(self,coefficients):
        self.coeff = coefficients
        self.offset = 0
        self.buffer = np.zeros(len(coefficients))

    def dofilter(self,u):
        result = 0
        self.buffer[self.offset] = u
        #print("offset index:",self.offset)

        for i in range(self.offset+1):
            result = result + self.buffer[i]*self.coeff[self.offset-i]
            #print("buffer index:",i)
            #print("coeff index:",self.offset-i)

        #print("Second For Section")
        for i in range(self.offset+1,len(self.buffer),1):
            result = result + self.buffer[i]*self.coeff[len(self.buffer)-1+self.offset+1-i]
            #print("buffer index:",i)
            #print("coeff index:",len(self.buffer)-1+self.offset+1-i)

        self.offset+=1
        if self.offset>=len(self.buffer):
            self.offset=0

        return result

    def dofilterPrint(self,u):      #index printing version for diagnosis purpose
        result = 0
        self.buffer[self.offset] = u
        print("offset index:",self.offset)

        for i in range(self.offset+1):
            result = result + self.buffer[i]*self.coeff[self.offset-i]
            print("buffer index:",i)
            print("coeff index:",self.offset-i)

        print("Second For Section")
        for i in range(self.offset+1,len(self.buffer),1):
            result = result + self.buffer[i]*self.coeff[len(self.buffer)-1+self.offset+1-i]
            print("buffer index:",i)
            print("coeff index:",len(self.buffer)-1+self.offset+1-i)

        self.offset+=1
        if self.offset>=len(self.buffer):
```

```python
            self.offset=0

        return result

def unittest():
        h = np.array([1/2,1/2,0,0,0])
        print("Coefficient:",h)
        f = FIR_filter(h)
        y= f.dofilterPrint(0)       #use the index printing version
        print("Input 0, Output",y)
        y= f.dofilterPrint(1)       #use the index printing version
        print("Input 1, Output",y)
        for i in range (20):
            y= f.dofilterPrint(0)   #use the index printing version
            print("Input 0, Output",y)

if __name__ == "__main__":
    unittest()
```

Listing 19: fir_filter.py

3.2 ecg_filter.py

```python
from ecg_gudb_database import GUDb
import numpy as np
from fir_filter import FIR_filter
from numpy import loadtxt
from matplotlib import pyplot

fs = 250     #Hz
M = 500      #loses 2 heart beat to warm up the filter (2seconds)/
    affect frequency resolution

# 50 Hz Notch Filter
k1 = int(49.5/fs * M)
k2 = int(50.5/fs * M)

Window50 = np.ones(M)
Coeff50 = np.ones(M)

Window50[k1:k2+1] = 0
Window50[M-k2:M-k1+1] = 0

pyplot.figure(1)
pyplot.plot(Window50)
pyplot.title('50 Hz Notch Filter - Ideal')
pyplot.xlabel('M (sample number)')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig1.eps', format='eps')

W50 = np.fft.ifft(Window50)
W50 = np.real(W50)

pyplot.figure(2)
pyplot.plot(W50)
pyplot.title('50 Hz Notch Filter - IFFT')
```

```
pyplot.xlabel('Coefficient Index')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig2.eps', format='eps')

Coeff50[0:int(M/2)] =  W50[int(M/2):M]
Coeff50[int(M/2):M] =  W50[0:int(M/2)]

pyplot.figure(3)
pyplot.plot(Coeff50)
pyplot.title('50 Hz Notch Filter - IFFT(fixed)')
pyplot.xlabel('Coefficient Index')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig3.eps', format='eps')

Filter50 = FIR_filter(Coeff50)


#DC filter
k3 = int(0.5/fs * M)
WindowDC = np.ones(M)
CoeffDC = np.ones(M)

WindowDC[0:k3+1] = 0
pyplot.figure(4)
pyplot.plot(WindowDC)
pyplot.title('DC Filter - Ideal')
pyplot.xlabel('M (sample number)')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig4.eps', format='eps')

WDC = np.fft.ifft(WindowDC)
WDC = np.real(WDC)

pyplot.figure(5)
pyplot.plot(WDC)
pyplot.title('DC Filter - IFFT')
pyplot.xlabel('Coefficient Index')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig5.eps', format='eps')

CoeffDC[0:int(M/2)] =  WDC[int(M/2):M]
CoeffDC[int(M/2):M] =  WDC[0:int(M/2)]

pyplot.figure(6)
pyplot.plot(CoeffDC)
pyplot.title('DC Filter - IFFT(fixed)')
pyplot.xlabel('Coefficient Index')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig6.eps', format='eps')

FilterDC = FIR_filter(CoeffDC)


# load first ECG file
file1 = open('ECG.dat', 'r')    #line by line to use as real-time
    filter

```

```
ecg = loadtxt("ECG.dat")          #for diagnosis plot

pyplot.figure(7)
#print(len(ecg))
time = np.linspace(0,len(ecg)/fs,len(ecg))
pyplot.plot(time,ecg)
pyplot.title('ecg (raw)')
pyplot.xlabel('Time(s)')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig7.eps', format='eps')


# Investigate frequency domain
fx=np.fft.fft(ecg)
fxx = fx/len(ecg)                # Fourier Transform Normalised
dbs = 20*np.log10(abs(fxx))      # DB Conversion
pyplot.figure(8)
freq = np.linspace(0,fs,len(ecg))
pyplot.plot(freq,dbs)
pyplot.title('ecg (raw) - Frequency Domain')
pyplot.xlabel('Frequency (Hz)')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig8.eps', format='eps')

#load second ECG data
'''Initialise experiments from the files of einthoven'''
subject_number = 3
experiment = 'walking'
ecg_class = GUDb(subject_number, experiment)

'''Initialise experiments from the files of einthoven'''
chest_strap_V2_V1 = ecg_class.cs_V2_V1
einthoven_ii = ecg_class.einthoven_II

'''Filtered Data With Einthoven'''
ecg_class.filter_data()
einthoven_ii_filt = ecg_class.einthoven_II_filt

#define output array and intermediate variable
filterecg = np.zeros(len(ecg)+1)
filtereinthoven = np.zeros(len(einthoven_ii)+1)
intermediate = 0

#filter ecg for both data sets
count = 0
for line in file1:
    count += 1
    ecg1 = line.strip()
    #print(ecg1)
    intermediate = Filter50.dofilter(ecg1)
    filterecg[count] = FilterDC.dofilter(intermediate)

for i in range(len(einthoven_ii)):
    intermediate = Filter50.dofilter(einthoven_ii[i])
    filtereinthoven[i] = FilterDC.dofilter(intermediate)

#print(count)
```

```

#plot the filtered data
pyplot.figure(9)
time2 = np.linspace(0,len(filterecg)/fs,len(filterecg))
pyplot.plot(time2,filterecg)
#pyplot.ylim(-0.002,0.002)
pyplot.title('ecg (filtered)')
pyplot.xlabel('Time(s)')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig9.eps', format='eps')

pyplot.figure(10)
time3 = np.linspace(0,len(filtereinthoven)/fs,len(filtereinthoven))
pyplot.plot(time3,filtereinthoven)
pyplot.ylim(-0.002,0.002)
pyplot.title('einthoven (filtered)')
pyplot.xlabel('Time(s)')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig10.eps', format='eps')


pyplot.figure(11)
#print(len(einthoven_ii))
time4 = np.linspace(0,len(einthoven_ii)/fs,len(einthoven_ii))
pyplot.plot(time4,einthoven_ii)
pyplot.title('einthoven_ii (raw)')
pyplot.xlabel('Time(s)')
pyplot.ylabel('Amplitude')
# pyplot.savefig('ecg_fig11.eps', format='eps')


#save filtered data
np.savetxt('shortecg.dat',filterecg)
np.savetxt('shorteint.dat',filtereinthoven)
```

Listing 20: ecg_filter.py

3.3 hr_detect.py

```
import numpy as np
from matplotlib import pyplot
from numpy import loadtxt
from fir_filter import FIR_filter
'''
Case = 0
#Case = int(input("Case 0 For shortecg.dat. Case 1 For Einthoven_ii
    Walking. Please enter case number: "))
for Case in range (2):
    if Case == 0:
        cleanecg = loadtxt("shortecg.dat")
    if Case == 1:
        cleanecg = loadtxt("shorteint.dat")

    fs=250      #sampling frequency of the data

    pyplot.figure(8*Case+1)
```

```
    time = np.linspace(0,len(cleanecg)/fs,len(cleanecg))
    pyplot.plot(time,cleanecg)
    pyplot.title('ecg (filtered)')
    pyplot.xlabel('Time(s)')
    pyplot.ylabel('Amplitude')
    # if Case == 0:
    #     pyplot.savefig('hr_fig1.eps', format='eps')
    # if Case == 1:
    #     pyplot.savefig('hr_fig9.eps', format='eps')

    #ecg templates for each data set
    if Case == 0:
        template=cleanecg[775:975]
    if Case == 1:
        template=cleanecg[7050:7225]

    #plot template
    pyplot.figure(8*Case+2)
    time2 = np.linspace(0,len(template)/fs,len(template))
    pyplot.plot(time2,template)
    pyplot.title('1 ecg')
    pyplot.xlabel('Time(s)')
    pyplot.ylabel('Amplitude')
    # if Case == 0:
    #     pyplot.savefig('hr_fig2.eps', format='eps')
    # if Case == 1:
    #     pyplot.savefig('hr_fig10.eps', format='eps')

    #inverse the template
    fir_coeff = template[::-1]
    pyplot.figure(8*Case+3)
    pyplot.plot(time2,fir_coeff)
    pyplot.title('reversed 1 ecg')
    pyplot.xlabel('Time(s)')
    pyplot.ylabel('Amplitude')
    # if Case == 0:
    #     pyplot.savefig('hr_fig3.eps', format='eps')
    # if Case == 1:
    #     pyplot.savefig('hr_fig11.eps', format='eps')

    #matched filtered data
    matchfilt = FIR_filter(fir_coeff)

    matchresult = np.zeros(len(cleanecg))
    for i in range(len(cleanecg)):
        matchresult[i] = matchfilt.dofilter(cleanecg[i])

    pyplot.figure(8*Case+4)
    pyplot.plot(time,matchresult)
    pyplot.title('Matched Filtered ecg')
    pyplot.xlabel('Time(s)')
    pyplot.ylabel('Amplitude')
    # if Case == 0:
    #     pyplot.savefig('hr_fig4.eps', format='eps')
    # if Case == 1:
    #     pyplot.savefig('hr_fig12.eps', format='eps')
    print(len(matchresult))
```

```
    #squared matched filtered data
    matchresult = matchresult*matchresult
    pyplot.figure(8*Case+5)
    pyplot.plot(time,matchresult)           #
    pyplot.title('Squared Matched Filtered ecg')
    pyplot.xlabel('Time(s)')
    pyplot.ylabel('Amplitude')
    # if Case == 0:
    #     pyplot.savefig('hr_fig5.eps', format='eps')
    # if Case == 1:
    #     pyplot.savefig('hr_fig13.eps', format='eps')

    #using threshold to extract the peaks
    hr = np.zeros(len(matchresult))
    threshold = 0.00000000002
    for i in range(len(matchresult)):
        if matchresult[i]>threshold:
            hr[i]=1

    pyplot.figure(8*Case+6)
    pyplot.plot(hr)
    pyplot.title('Heart Beat Detection Sequence')
    pyplot.xlabel('Sample number')
    pyplot.ylabel('Amplitude')
    #pyplot.xlim(3,120)              #to delete the first part where the buffer is being filled/diagnosis purpose
    # if Case == 0:
    #     pyplot.savefig('hr_fig6.eps', format='eps')
    # if Case == 1:
    #       pyplot.savefig('hr_fig14.eps', format='eps')

    #initiate counters
    peakCounter = 0   #count number of peaks > for array size
    detectIndex = 0   #detection index
    deltatIndex = 0   #deltat index

    index = np.zeros(len(hr))
    threshT = int(1/(3.66/fs))    #3.66beat/s is the maximum heart rate(220bpm)

    for i in range(len(hr)):
        if hr[i] == 1:
            peakCounter+=1
            for n in range(int(threshT)):    #fix the impossible detected peak to 0
                if i+n+1 <= 30000:          #If the time is less than 30000
                    hr[i+n+1] = 0

    pyplot.figure(8*Case+7)
    pyplot.plot(hr)
    pyplot.title('Heart Beat Detection Sequence Fixed')
    pyplot.xlabel('Sample number')
    pyplot.ylabel('Amplitude')
    #pyplot.xlim(3,120)          #to delete the first part where the buffer is being filled/diagnosis purpose
```

```
    # if Case == 0:
    #     pyplot.savefig('hr_fig7.eps', format='eps')
    # if Case == 1:
    #     pyplot.savefig('hr_fig15.eps', format='eps')

    beatone = np.zeros(peakCounter)
    deltat = np.zeros(peakCounter)
    rate = np.zeros(peakCounter)

    print("Momentary Heart Rate(BPM)")
    for i in range(len(hr)):
        if hr[i] != 0:          # When element is not zero, input element to new array
            beatone[detectIndex] = i
            detectIndex += 1
            if detectIndex >= 0:          # To neglect the first detection
                deltat[deltatIndex] = (beatone[deltatIndex] - beatone[deltatIndex-1])/fs
                # print('ONE',beatone[deltatIndex])       #for diagnosis purpose
                # print('TWO',beatone[deltatIndex-1])
                # print('TIME',deltat[deltatIndex])
                rate[deltatIndex] = (1/deltat[deltatIndex])*60
                print(rate[deltatIndex])
                deltatIndex += 1

    pyplot.figure(8*Case+8)
    pyplot.plot(beatone/fs,rate)
    pyplot.title('Heart Rate Plot')
    pyplot.xlabel('Time(s)')
    pyplot.ylabel('Beats/Minute')
    #pyplot.xlim(4,max(beatone)/fs+1)        #to delete the first part where the buffer is being filled/diagnosis purpose
    # if Case == 0:
    #     pyplot.savefig('hr_fig8.eps', format='eps')
    # if Case == 1:
    #     pyplot.savefig('hr_fig16.eps', format='eps')
'''

class HR_filter:
    def __init__(self,):
        ecg = loadtxt("shortecg.dat")           #file which the template is extracted from
        template=ecg[775:975]                   #1ecg of the corresponding file
        fir_coeff = template[::-1]              #reverse template
        self.matchfilt = FIR_filter(fir_coeff)
        self.hr = 0
        self.hrBuffer = 0                       #keep previous hr value for peak starts checking
        self.counter = 0                        #counts time difference before next peak arrive

    def realtimeHR(self,data,fs):
        matchresult = self.matchfilt.dofilter(data)
        matchresult = matchresult*matchresult
```

```
        threshold = 0.00000000001               #observed by eye from graph

        #use threshold to extract peak area
        if matchresult>threshold:
            self.hr=1
        else:
            self.hr=0

        threshT = int(1/(3.66/fs))               #3.66beat/s is the maximum heart rate(220bpm)
        rate=0                                  #default argument

        #checks if the peak starts
        if self.hrBuffer==0 and self.hr==1:
            #peak=1      #for diagnosis purpose : peak starts
            if self.counter>threshT:
                rate = (fs/self.counter)*60
                self.counter=0                      #reset counter
            return rate
        else:
            #peak=0      #for diagnosis purpose : no peak starting
            self.counter+=1
            return None
        #return matchresult
        self.hrBuffer = self.hr                 #save value for future comparison


#Example Use
data = loadtxt("shortecg.dat")
data2 = loadtxt("shorteint.dat")
fs=250
mhr = np.zeros(len(data))
mhr2 = np.zeros(len(data2))
hrfilt = HR_filter()                            #instantiate the HR_filter
for i in range(len(data)):
    mhr[i] = hrfilt.realtimeHR(data[i],fs)
for i in range(len(data2)):
    mhr2[i] = hrfilt.realtimeHR(data2[i],fs)

pyplot.figure(17)
pyplot.plot(np.linspace(0,len(data)/fs,len(data)),mhr)
pyplot.title('Momentary Heart Rate Plot from ECG.dat')
pyplot.xlabel('Time(s)')
pyplot.ylabel('Beats/Minute')
#pyplot.savefig('hr_fig17.eps', format='eps')

pyplot.figure(18)
pyplot.plot(np.linspace(0,len(data2)/fs,len(data2)),mhr2)
pyplot.title('Momentary Heart Rate Plot from einthoven II')
pyplot.xlabel('Time(s)')
pyplot.ylabel('Beats/Minute')
#pyplot.savefig('hr_fig18.eps', format='eps')
```

Listing 21: hr_detect.py

List of Figures

Listings