1. Introduction

There are some disadvantages of FIR (Finite Impulse Response) filters, such as the delay and efficiency (sometimes it may need thousands of taps). In comparison, IIR (Infinite Impulse Response) filters can provide real-time performance and only require small amount of calculation resource. However, the IIR filters cannot provide linear phase response.

A IIR filter, is basically the discrete approximation of an analogue filter. There are several ways to transform an analogue filter into a digital IIR filter, but they are based on the relationship of Laplace transform and Z-transform.

IIR filters are usually designed as a chain of 2nd-order filters (as shown in Figure 1), each 2nd-order filter can be described as direct form1 and direct form2 (as shown in Figure 2 and Figure 3 respectively), which both satisfy the equation $h(z) = \frac{b0+b1*Z^{-1}+b2Z^{-2}}{1+a1*Z^{-1}+a2*Z^{-2}}$ (a1 and a2 are negative numbers in this case so the calculation are written as addition instead of subtraction).

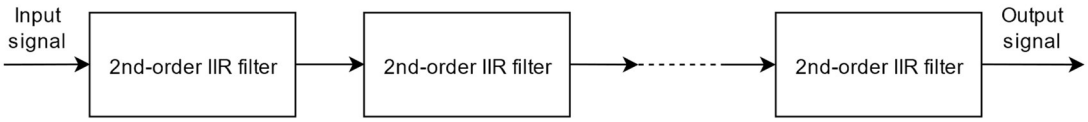

Figure 1 High order IIR filter implementation

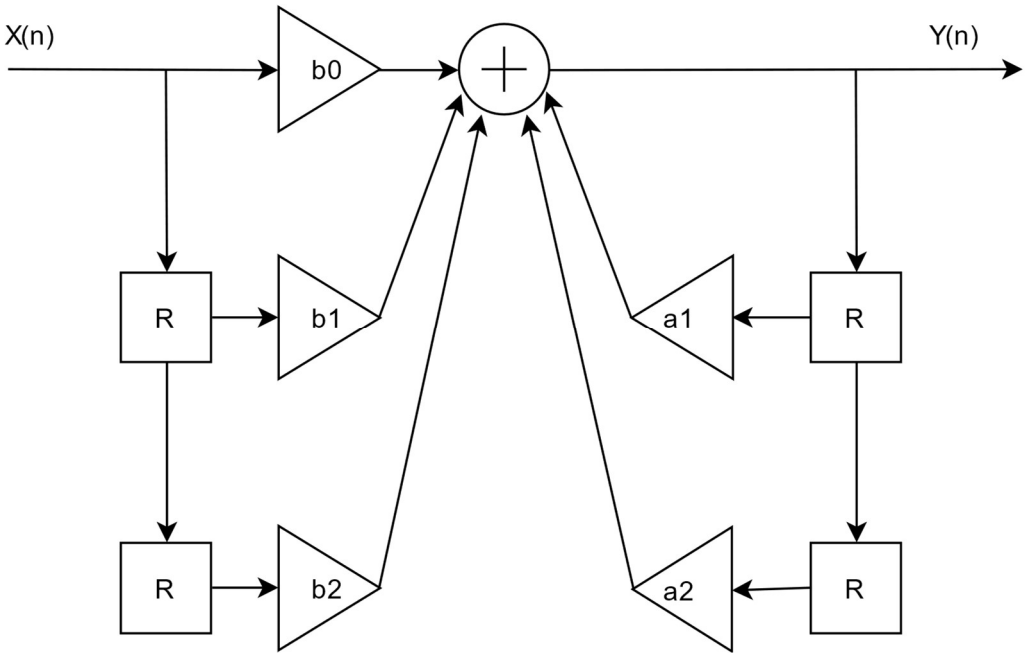

Figure 2 IIR 2nd-order filter direct form 1

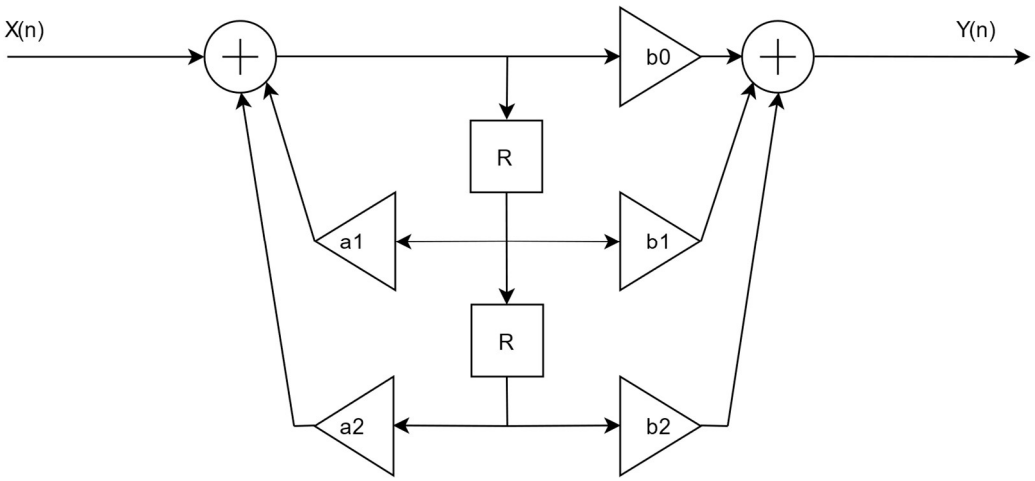

Figure 3 IIR 2nd-order filter direct form 2

Direct form 1 uses four delay buffers but one summation unit while direct form 2 uses two delay buffers but two summation unit. Considering the chance of getting overflow in fixed point system and memory usage, in this exercise, IIR filter is implemented as a 2nd-order filter chain based on direct form 2.

This exercise introduces an application problem where the output signal of the sensor is quite noisy, and it requires real-time digital filtering. In our case, the application is the detection of pulse using light dependent resistor, there would be interference (100Hz in this case) from the ambient light. Hence, a IIR filter, which eliminates DC and 100Hz noise, is designed and implemented in Python. The rest of the report would describe the system and the result.

2. System description

2.1 System overview

The system functional is shown in Figure 4. The LDR (Light dependent resistor) is firstly been biased and the corresponding signal would be amplified. Then the signal is sampled by a built-in ADC of the MCU (STM32 NUCLEO-L152RE). The MCU will pack the data and send it to the PC. The PC has an IIR filter which would filter the data. Finally, both original and filtered data are plotted in real-time. (All the software on PC are implemented using python script). The physical shot of the whole system is shown in Figure 5 where the USB cable goes to PC.

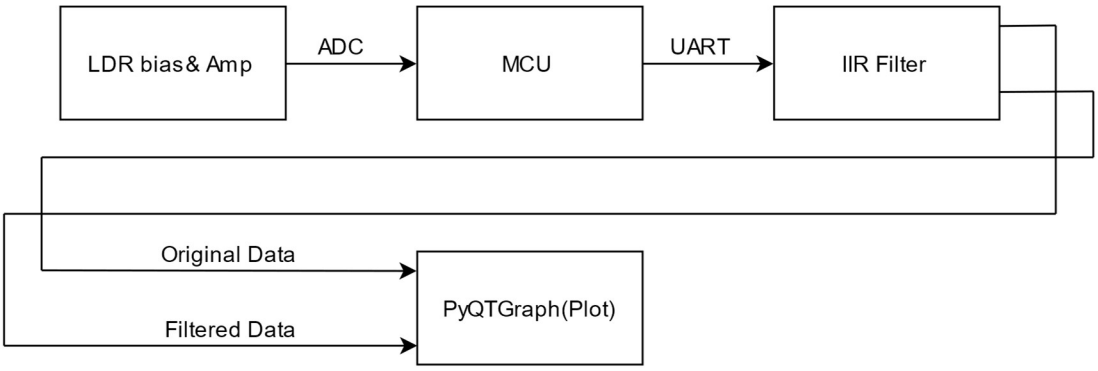

Figure 4 System overview

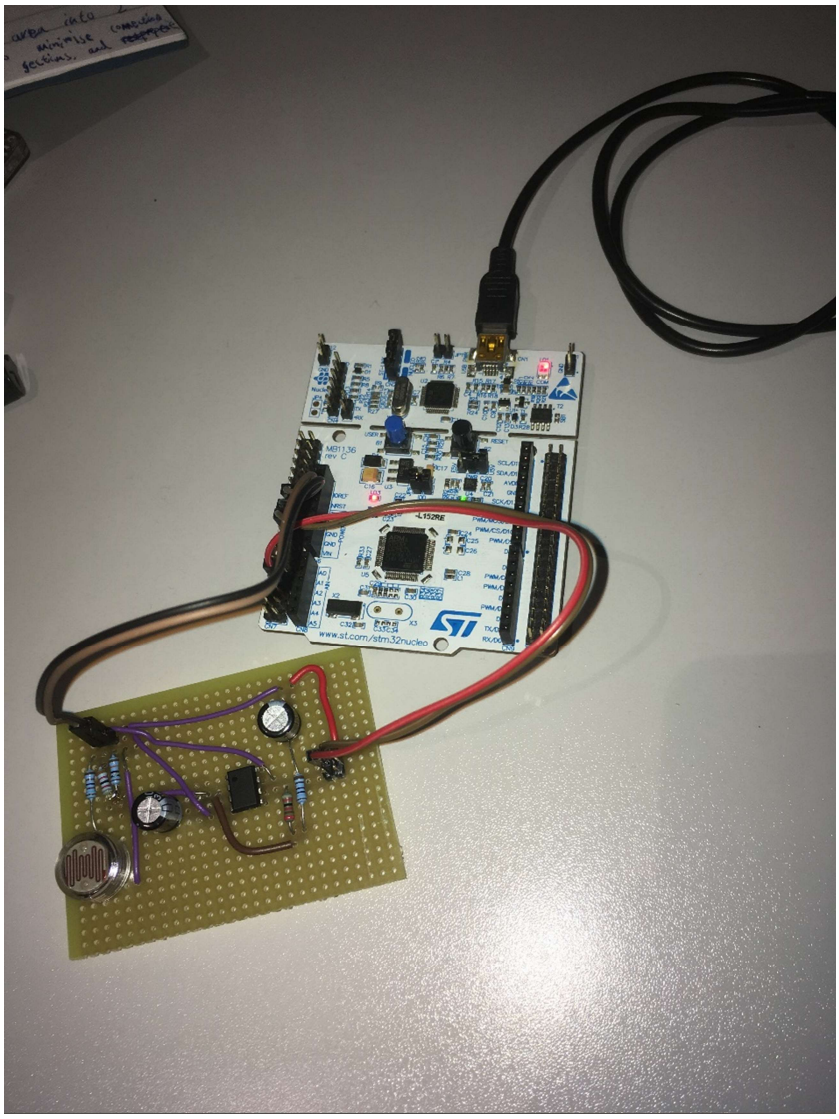

Figure 5 The physical set of the system

2.2 Analogue circuit

The objective is to measure the strength of light passed through a human finger. When the heartbeat happens, the light resistance of the finger would change. Based on this, a DC LED (from iPhone) is used as the light source, this LED is DC adjust or high frequency PWM adjust so it would not introduce any interference to the system. When one's finger is put between the light source and the LDR, the resistance of the LDR would change due to the light resistance of the finger.

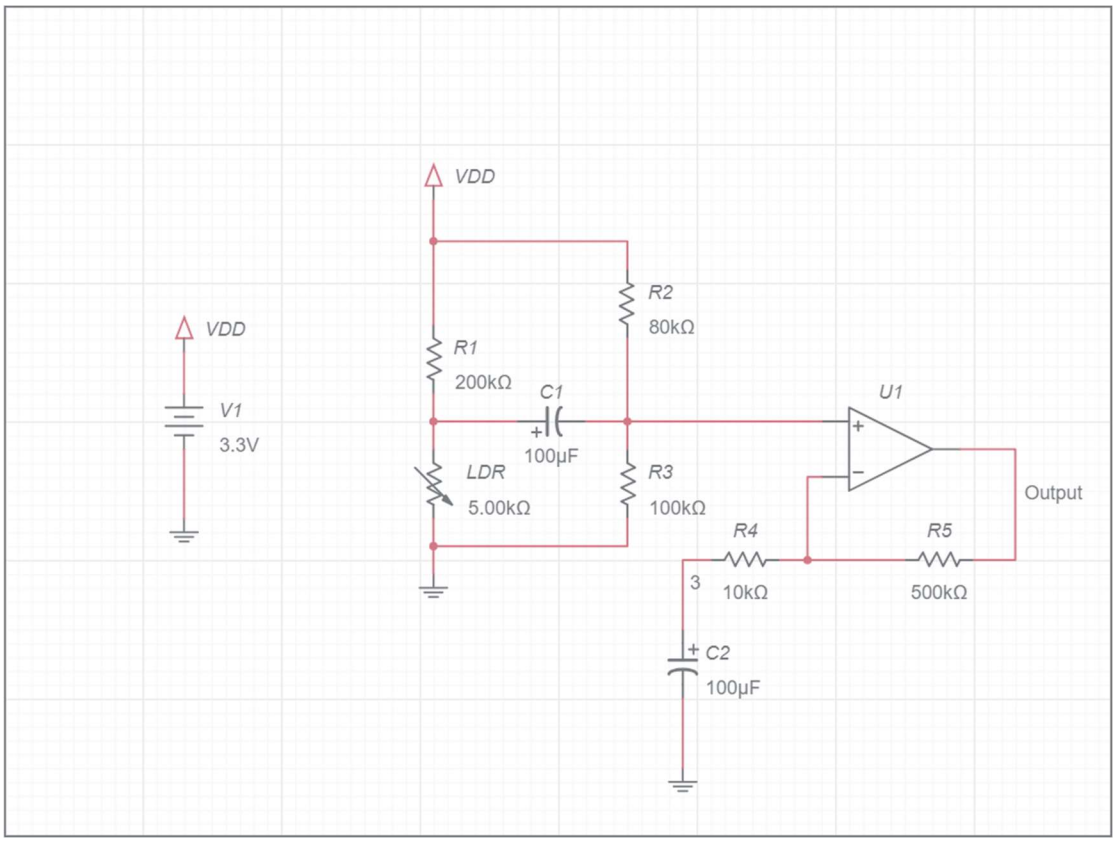

Figure 6 Analogue circuit

The schematic of the circuit is shown in Figure 6. The input of the circuit is the light change upon the LDR, where the LDR is biased using a voltage source and a constant resistor R1. Then the change of light strength would become the voltage change on LDR. However, because the change of the light strength is very small, then the voltage change on LDR is very small as well. In our case, the LDR would vary its resistance from 5Kohms to 1Mohms depending on the input light. Practically in this case, when the voltage source is set to 3.3v, the peak-peak value of the corresponding signal (voltage change on LDR) is roughly below 10 mv. Therefore, an amplifier is needed.

As shown in the diagram, the signal would firstly go through the capacitor (C1) and the lift, then be amplified. For DC part, Vin+, Vin- and output would all be the same due to C2 (DC would go through R5 and directly into Vin-). In this circuit, it is set to about 1.8V. For the AC part, when set the Vin+ as input and Output as output, the transfer function can be easily written as $H(s) = \frac{\frac{1}{SC_2}+R4+R5}{\frac{1}{SC_2}+R4} = \frac{\frac{1}{C_2}+(R4+R5)*S}{\frac{1}{C_2}+R4*S}$. Plugin the values of C2, R4 and R5, the gain is about 34dB (51), and the peak-peak value of the AC part of the output signal is about 1V. To summarise, the input signal (the light change upon the LDR) would be transferred and amplified and the output signal would be the pulse signal (voltage) where the peak values of the output signal is about 1.8Vdc + 0.5Vac.

2.3 MCU part

In this system, the MCU has two functions: 1. It would sample the output signal of the analogue circuit; 2. It would pack the data and send the data to the PC using a protocol. The code is attached in appendix.

For the sampling part, the MCU uses a 12-bit ADC and the ADC is triggered by a timer in which the frequency of the timer is 300Hz.

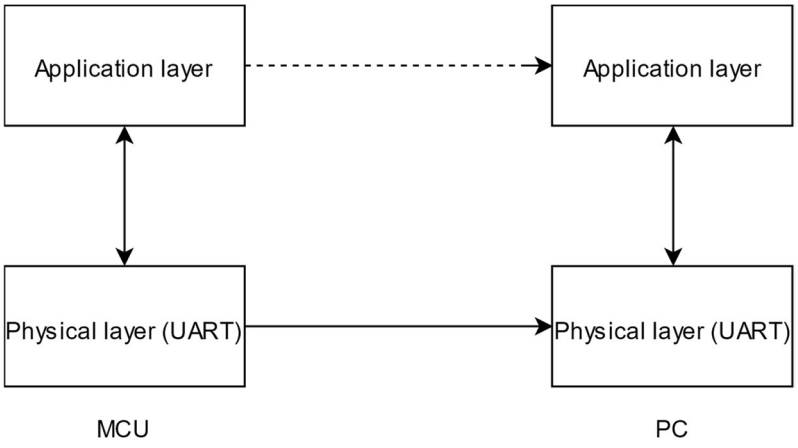

Figure 7 The protocol stack

Start flag	Frame counter	Payload
ASCII 'S'	8-bit Binary	4 Bytes ASCII number

Figure 8 Application layer specification

For the data transferring part, as shown in Figure 7, the whole protocol can be divided into two layers. The bottom layer is UART, where the data length is eight bits, baudrate is 115200 bps, stop bit is one bit and no parity bit. The top layer is the application layer. As shown in Figure 8, each frame contains a start flag (ASCII ‘s’), a frame counter (one-byte hex number) and four bytes of payload (ASCII numbers). The frame counter would accumulate each time and it allows the PC side to calculate the number of lost packet. The MCU here works as a transmitter and the PC works as a receiver.

All the functions, including reading ADC value, converting the value into voltage, data packaging and UART transfer, are all called in the ADC conversion finish callback function. For the efficiency purpose, the UART is using a DMA unit to transfer the data from memory to UART TX register.

2.4 PC overview

The whole system consists of three threads, as shown in Figure 9. The communication thread would read and unpack the data from MCU, and the main thread would read the data from communication thread, call IIR filter function to filter the data and send both original and filtered data in to real-time plot thread.

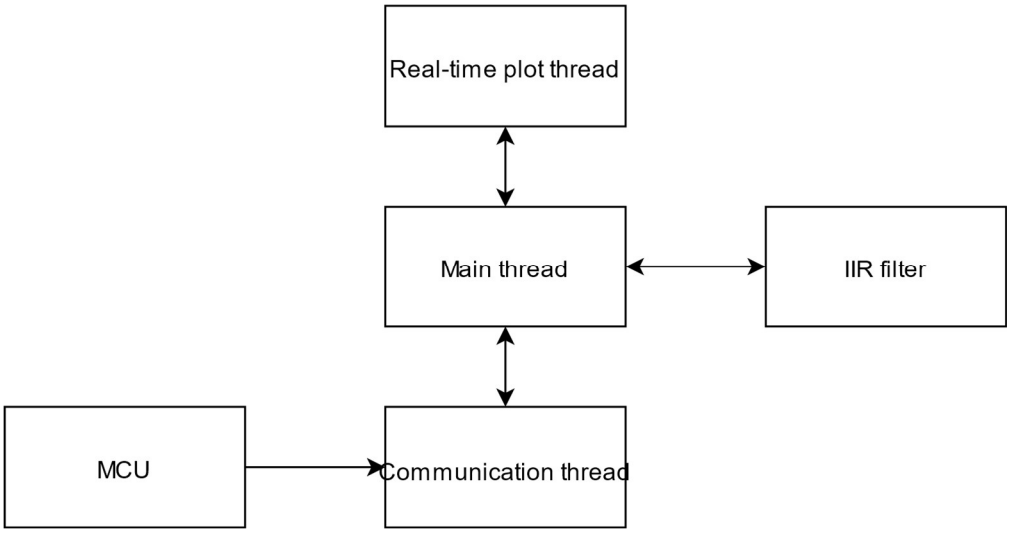

Figure 9 System overview of PC

2.5 PC Protocol

The protocol is the same as described in the section of MCU. The only difference is it's implemented in Python using the package "pyserial".

In this part, a polling thread is created to check if there is data in the UART receiver. If the start flag is detected, the thread would read the rest of the frame, calculate packet lost and convert the payload from ASCII numbers to integers. Finally, the thread would put the data into a queue and the main thread can read it.

2.5 PC IIR filter

The IIR filter part can be broken down to two parts, the design and the filter.

For the design part, the coefficients are calculated by calling functions from "scipy.signal". Because the IIR filter is implemented as a chain of direct form 2 2nd-order filter, so the output format of the function is set to 'sos'. In the case of this exercise, an 8th-order Butterworth band pass filter and a 2nd-order Butterworth high pass filter are used, where the cutoff frequencies are 100Hz and 0.01Hz, respectively. The code is shown as follow, where the last two instructions are used to connect the two filters together as a chain.

```
    # filter design
    bandstop = sig.butter(N = 8, Wn = [95/150, 105/150], btype =
'bandstop', output = 'sos')
    highpass = sig.butter(N = 2, Wn = 0.1/150, btype = 'highpass',
output = 'sos')

    sos = np.append(highpass, bandstop)
    sos = sos.reshape(int(len(sos)/6), 6)
```

For the filter part, two classes are implemented. The first one is the 2nd-order IIR filter, which requires coefficients when the user tries to create an instance. Besides, the filter function is designed as a direct form 2 IIR filter. The code is shown as below:

```
    def filter(self, x):
        # calculate the IIR part
        input_sum = x - self.__buffer0 * self.__a1 - self.__buffer1 *
self.__a2
        # calculate the FIR part
        output_sum = input_sum * self.__b0 + self.__buffer0 * self.__b1
+ self.__buffer1 * self.__b2

        # delay steps
```

```
        self.__buffer1 = self.__buffer0
        self.__buffer0 = input_sum

        return output_sum
```

The second class is the IIR filter which is a chain of the 2nd-order IIR filter. It would take the 'sos' form coefficients in the initialise function and create a set of instances of 2nd-order IIR filter. The filter function would call the filter function of each 2nd-order IIR filter, take the output value of previous one as the input of the next one. The code of the filter function is shown as below:

```
    def filter(self, x):
        for i in range (0, self.__length):
            if (i == 0):
                output = self.__filter_chain[i].filter(x)
            else:
                output = self.__filter_chain[i].filter(output)

        return output
```

2.6 PC real-time plot

To provide the real-time performance, "matplotlib" is not used due to its low efficiency. By instead, "Pyqtgraph" is used here. A thread is created to refresh the plots, where a timer would trigger the refreshing event 60 times per second, so the refresh rate of the graphs is 60 fps.

3. Conclusion

The result of the system is as expected, a video of demo was recorded and uploaded onto Youtube, the address is https://www.youtube.com/watch?v=ALNBNAd9Q6I.

The program would firstly plot the frequency response of the filter as shown in Figure 10, the frequency response at 0Hz (DC) is not shown due to the resolution of the testing frequency ("scipy.signal.sosfreqz()"). Then, the program would plot the signals in real-time. A screen shot (Figure 11) shows the plots of original and filtered signal.

There is still one problem in the python script. Theoretically, if the function "matplotlib.pyplot.show()" is called in a dedicated thread, it would not block the other threads. However, it will block the whole script even it is called in a dedicated process.

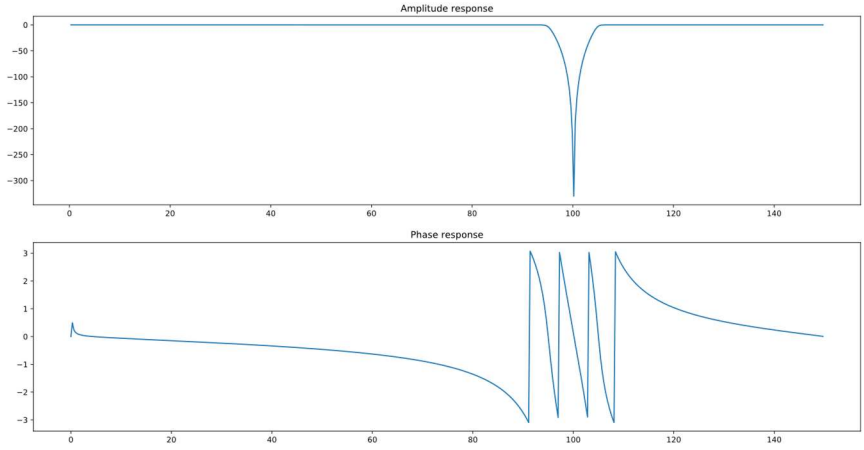

Figure 10 The frequency response of the IIR filter

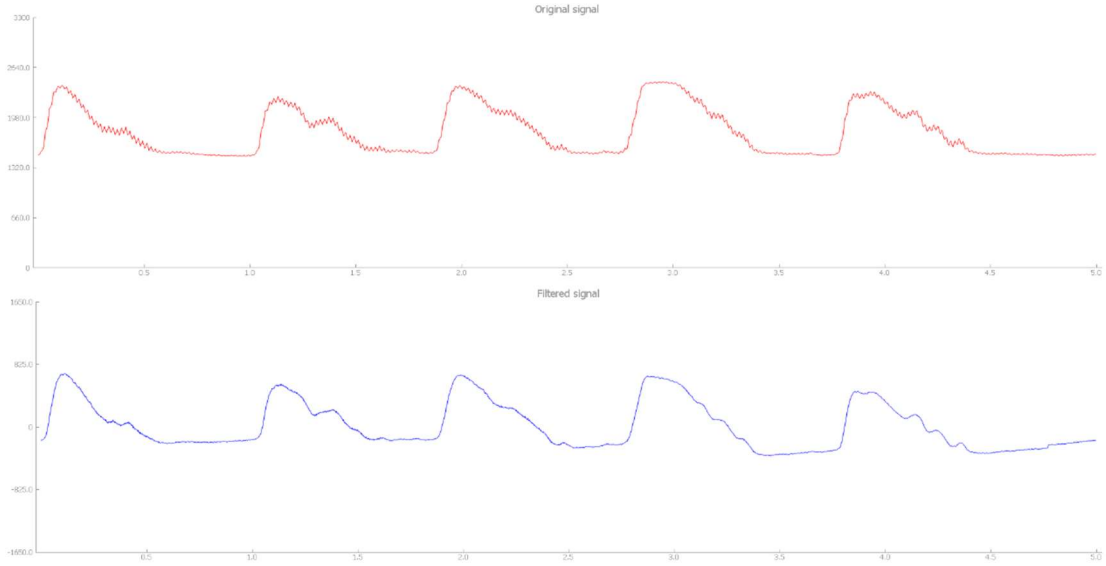

Figure 11 Screen shot of the real-time plots

To conclude, the whole system, especially the part of IIR filter, was successfully implemented. Comparing IIR filter with FIR filter, it uses only a few amounts of calculation resource and can provide real-time performance. In this exercise, it is purely designed in python, as in the last exercise, the FIR filter must be implemented in C to reduce the time spent on calculation. However, the phase response of the IIR filter is not linear, which means it may distort the signal. Therefore, the choice between FIR filter and IIR filter would depend on the requirement of the application.

4. Appendix

The link of the code is
https://github.com/MinghaoCheng/MSC_Work/tree/master/DSP/Assignment3