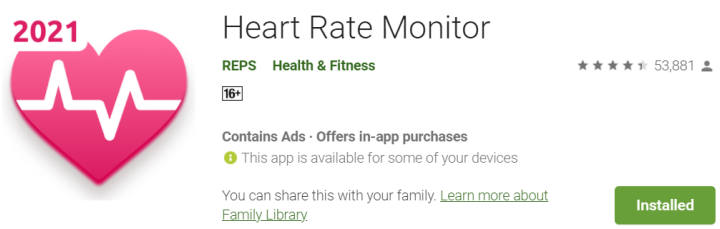
CS6650 Homework 1

V S S ANIRUDH SHARMA | EE18B036

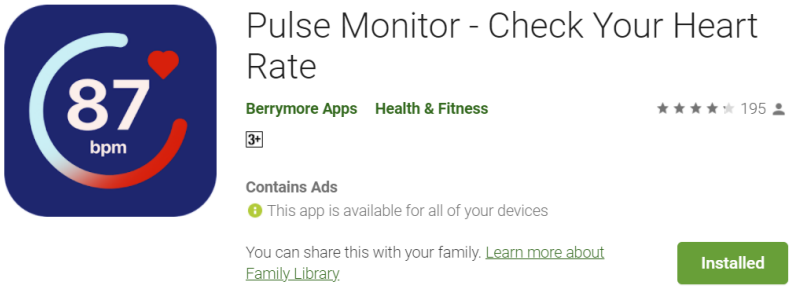
Task 1: SURVEYING

(1)



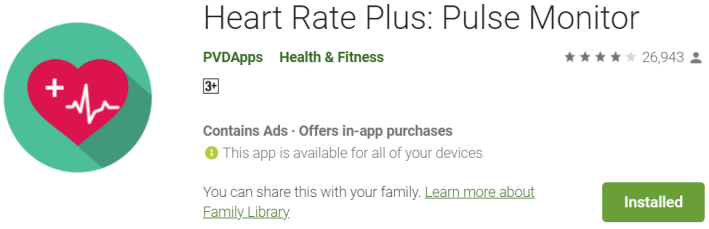
Using the Heart Rate Monitor app, you can measure and monitor your heart rate.
To use this heart rate monitor app, just put your finger on the phone's camera and stay still, the heart rate should be shown after several seconds.

(2)



This health app uses photoplethysmography (PPG), a non-invasive optical technique, to help you gain insights into your heart health and cardio fitness level.

(3)



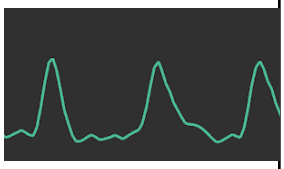
The Heart Rate Plus app will measure your heartbeat using your smartphone's camera by processing the images of your pulse on your finger!

a) All the three mentioned above give a consistent value to the pulse rate. The ratings of such apps on the playstore are more to do with the UI and ease of use than accuracy. The comments for low rated apps didn't complain about inaccuracy but UI alone. We observe that all the three detect very similar waveforms.

b)

Comparing with (1):

1. (2) is much more smooth/filtered
2. (3) is inverted horizontally

	1. Heart Rate Monitor	2. Pulse Monitor	3. Heart Rate Plus
Waveform			
Starting time	7.38 sec	6.23 sec	6.45 sec
Total time to evaluate	12.68	16.16	15.41

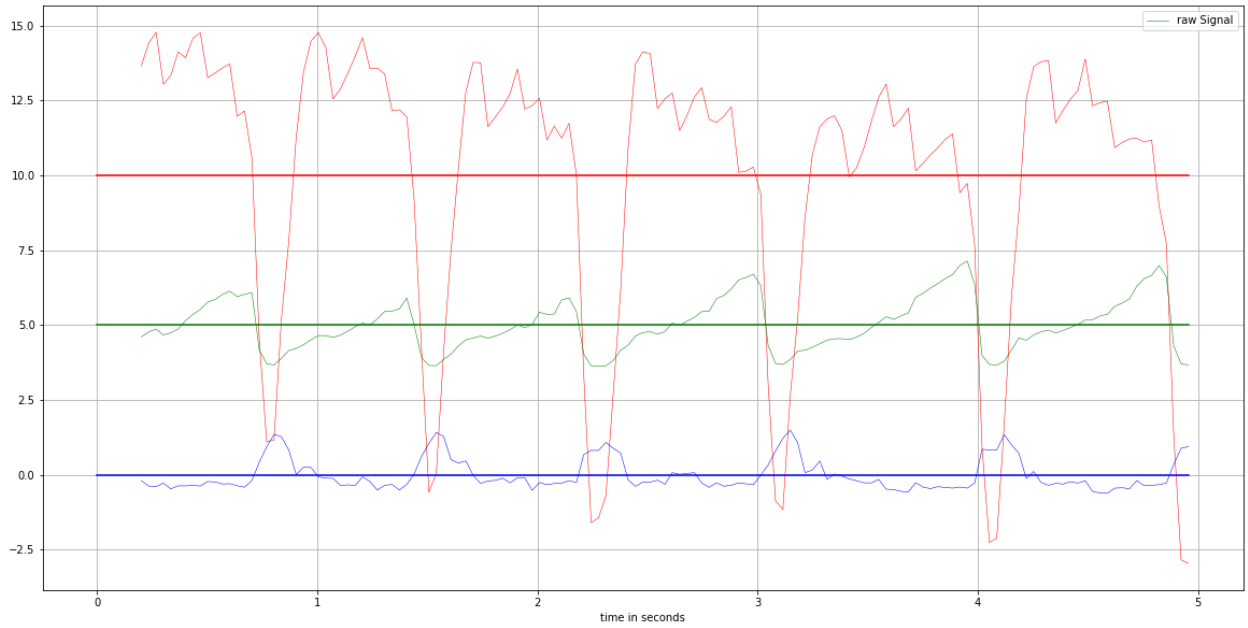
c) Apps (1) and (2) reset on disturbing the finger about the camera. Also, app (2) waveform seems to have perfected filters as it almost always keeps the magnitude of the waves to the same lengths.
1 takes a long time to start showing pulse rate (window of computation is larger) but quickly ends the analysis (Number of windows averaged over is lower)

d) (3) doesn't recognise disturbing movements of the finger about the camera. It calculates and displays a high pulse rate, unlike (1) and (2) which immediately detect this and reset/show error

Task 2: DATA ACQUISITION

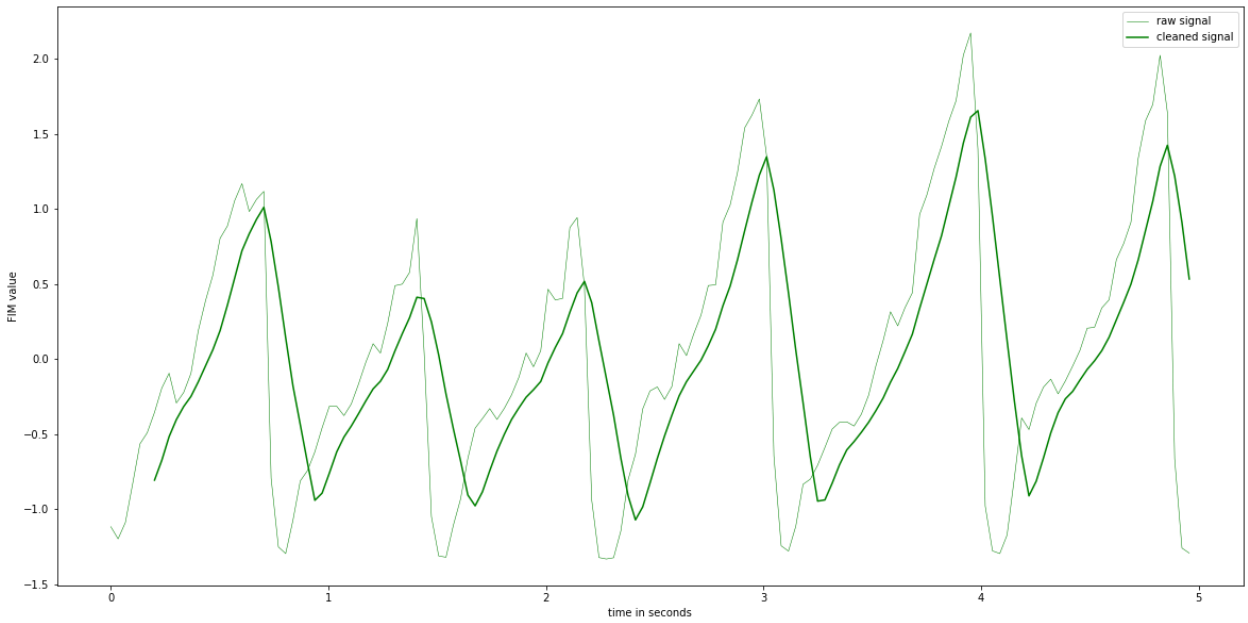
(a) Frame intensity metric

Considering the kind of variation observed in the R,G and B values averaged over the frame for each frame, green was chosen to best represent the pulse.

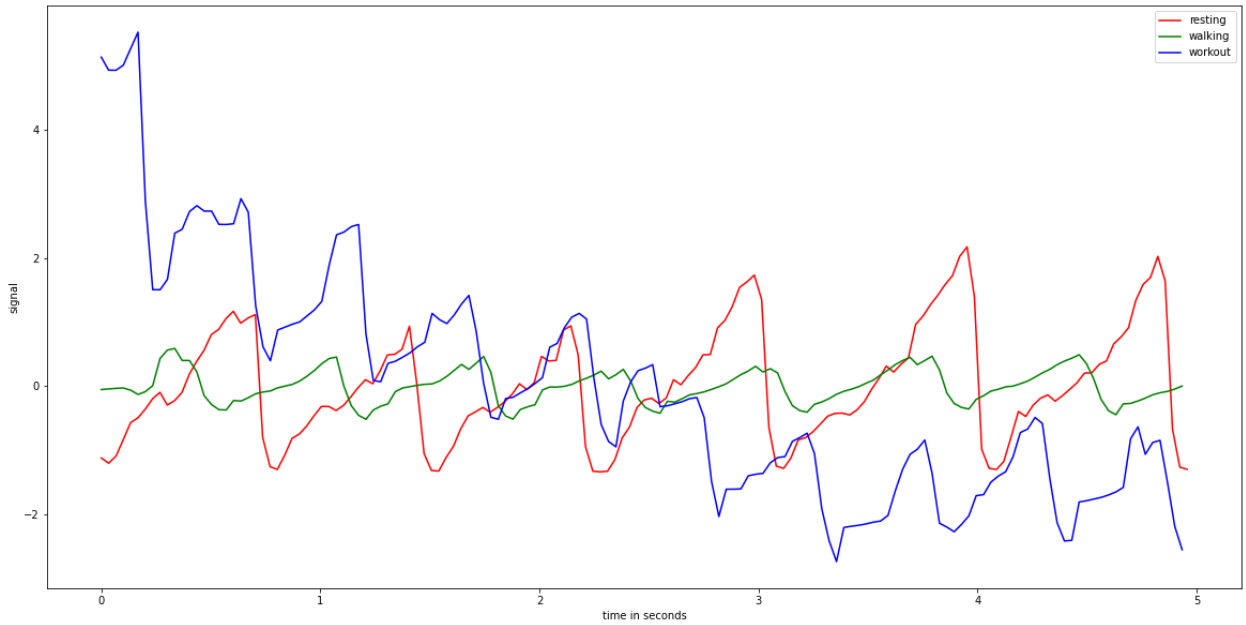


$$FIM(n) = \frac{\Sigma_i \Sigma_j g_{ij}(n)}{l*b}$$

Where $g_{ij}(n)$ is the green pixel value at (i,j) position in the nth frame. We use moving average to filter out the noise and then detect peaks



(b) Plotting FIR for 3 states



An interesting observation to make with respect to our FIM and the plots from the 3 applications, is that (1) and (2) might have most probably used red or blue values to compute the pulse whereas (3) used green, considering the shape of the waveform.

Task 3: ALGORITHMS

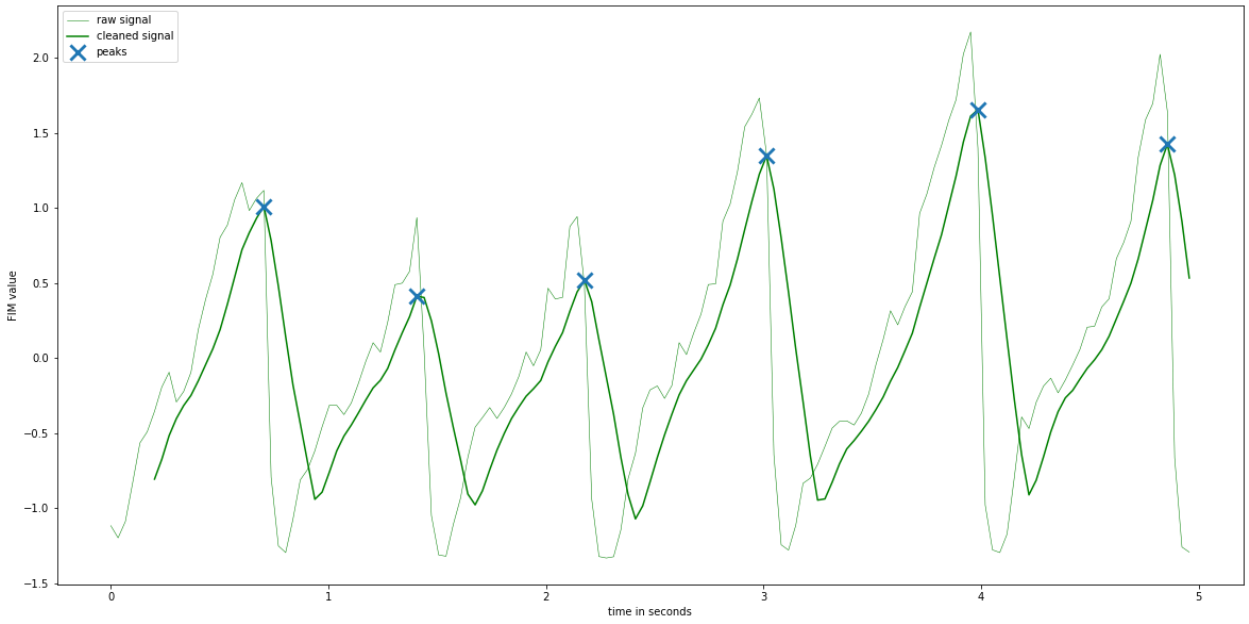
The most basic function we will use is the cleanSignal, which is basically the running average of a signal, smoothing curves.

```
def cleanSignal(x,w):
    return np.convolve((x-np.mean(x)), np.ones(w), 'valid')/w
```

Time Series Technique

We take the signal, clean it, count peaks and divide it by timespan

```
#return number of peaks in cleaned green per given time
def peakFreq(green, fps_act = 30, w=7):
    l = len(green)
    signal = cleanSignal(green,w)
    peaks, _ = find_peaks(signal,distance = 5)
    BPM_calc = 60*len(peaks)*fps_act/l
    return BPM_calc
```

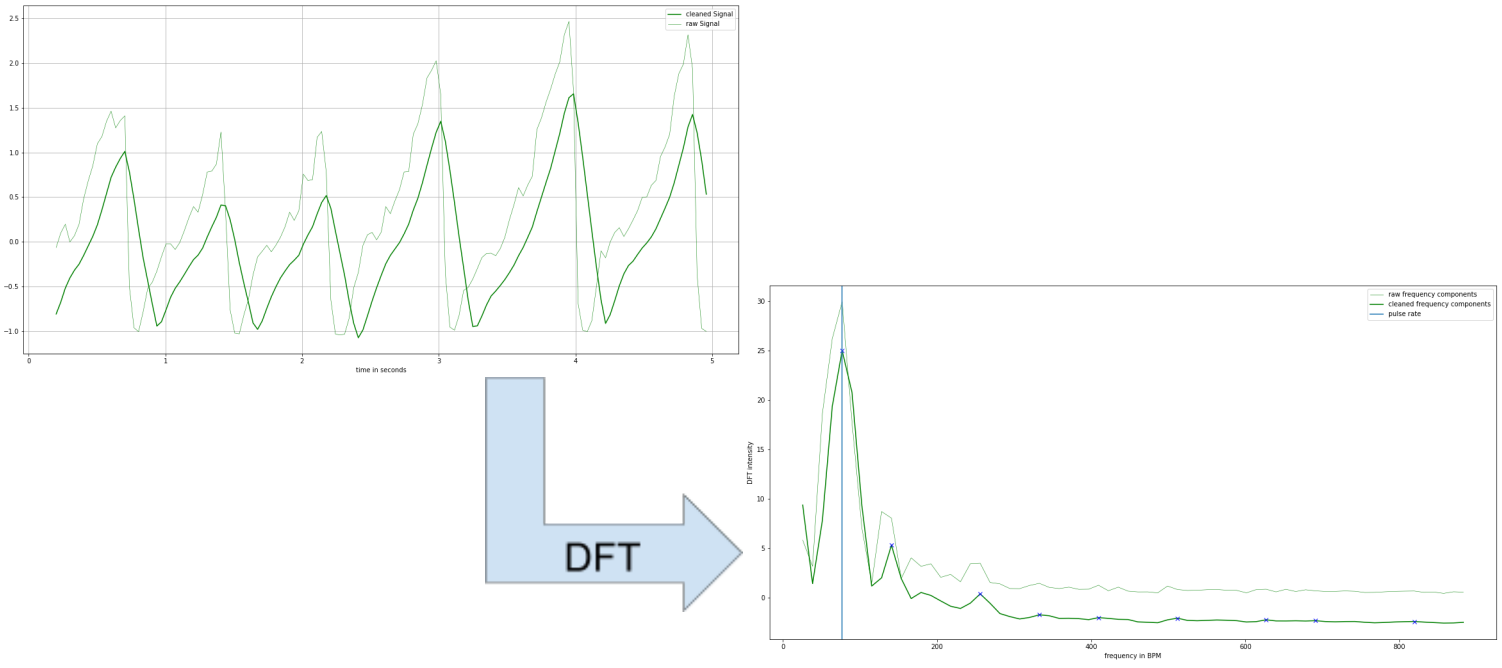


DFT Technique [Ref 1, 2]

The extracted FIM array is sent into majorFreq, where the dft of a cleaned array is again smoothened and the frequency between 40BPM and 200BPM (normal human pulse range) corresponding to the highest peak value in that ranis returned, which would be our pulse rate

```
# returns DFT of a given vector after cleaning it
def dft_util(green, w=10):
    signal = cleanSignal(green,w)
    # compute DFT with optimized FFT
    fs = np.fft.fftfreq(signal.shape[0])
    sp = np.fft.fft(signal)
    freq = np.fft.fftfreq(signal.shape[-1])
    wh = np.where(freq>0)
    return freq[wh],abs(sp)[wh]
```

```
#return highest peak in the dft of a signal in range of 40BPM-200BPM
def majorFreq(green,fps_act = 30,w1 = 7,w2=2, minPulse = 40, maxPulse =
200):
    W,Y = dft_util(green,w1)
    cleaned = cleanSignal(Y,w2); W1 = W[w2-1:]
    minFreq = minPulse/(60*fps_act)
    maxFreq = maxPulse/(60*fps_act)
    normal = np.where(np.logical_and(W1>minFreq , W1<maxFreq))
    cleaned2 = cleaned[normal]
    W2 = W1[normal]
    peaksFreq, _ = find_peaks(cleaned2,distance = 5)
    BPM_calc = W2[max(peaksFreq, key=lambda p: cleaned2[p])]*60*fps_act
    return BPM_calc
```



Task 4: EVALUATION

As a reference to the exact value, Omron HEM-7113 has been used. Here's the rate detected in videos 1/2/3.mp4 respectively. Motorola G5S+ has been used for capturing videos (13MP camera, ~30fps video)

With about 20 seconds of footage evaluated in each case (rest, walk, workout) the algorithms give the following results:

	1. Rest	2. Walk	3. Workout
From Omiron HEM-7113 (Checks for over a minute for pulse)	73	80	107
From Time series technique (~20 sec)	71.4342771578723	80.08690912731224	106.36839669421487
From DFT technique (~20 sec)	74.87074393231626	80.86319580706002	113.40240400667778
From Time series technique (first 5 sec)	72.13497476551277	84.55121318339552	108.70418918918918
From DFT technique (first 5 sec)	75.16161706336645	88.12379965593335	125.88591549295776

Rescaling frame resolution and frame rate

```
def rescale_frame(frame, percent=100):
    width = int(frame.shape[1] * percent/ 100)
    height = int(frame.shape[0] * percent/ 100)
    dim = (width, height)
    return cv2.resize(frame, dim, interpolation =cv2.INTER_AREA)
```

```
    i = 0
    while(cap.isOpened() and i<N):
      # Capture frame-by-frame
      ret, frame = cap.read()

      if ret == True:
        i += 1
        # Display the resulting frame
        # setting values for base colors

        if i%fpsScale == 0:
            frame2 = rescale_frame(frame, rescale)
            g = frame2[:, :, 1]

            # computing the mean
            g_mean = np.mean(g)
            green.append(g_mean)
```

Here, ***fpsScale*** is an integer parameter and our new fps become:

$$fps_{new} = \frac{fps_{actual}}{fpsScale}$$

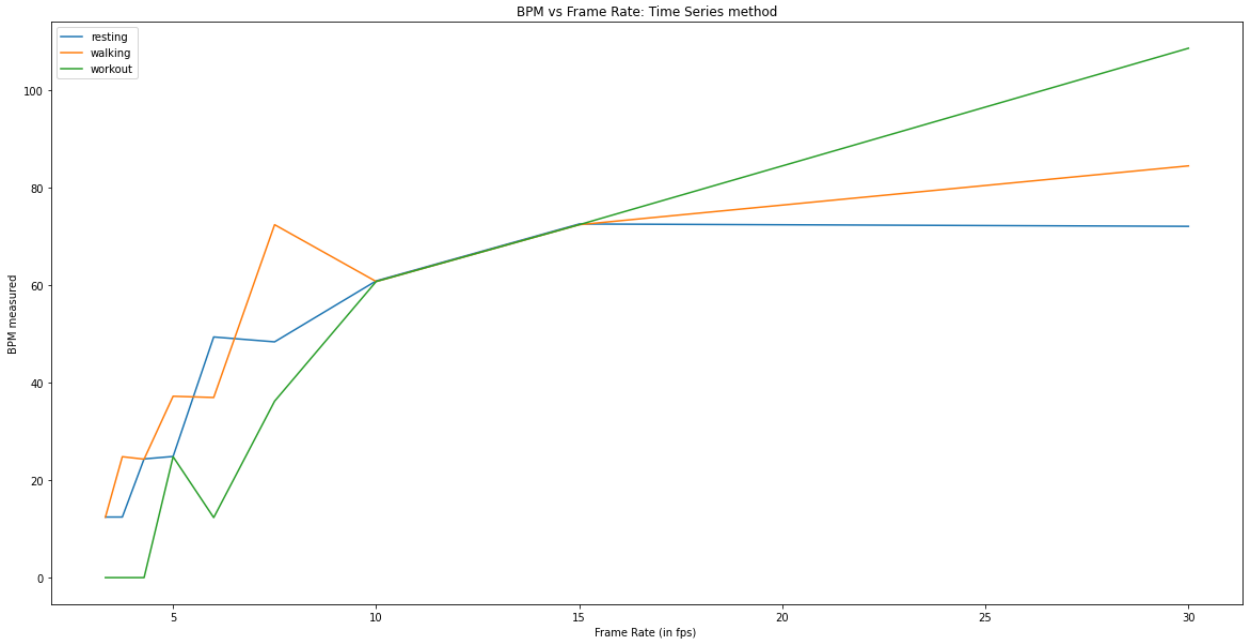
And for frame resolution, the new width and height of the frame become:

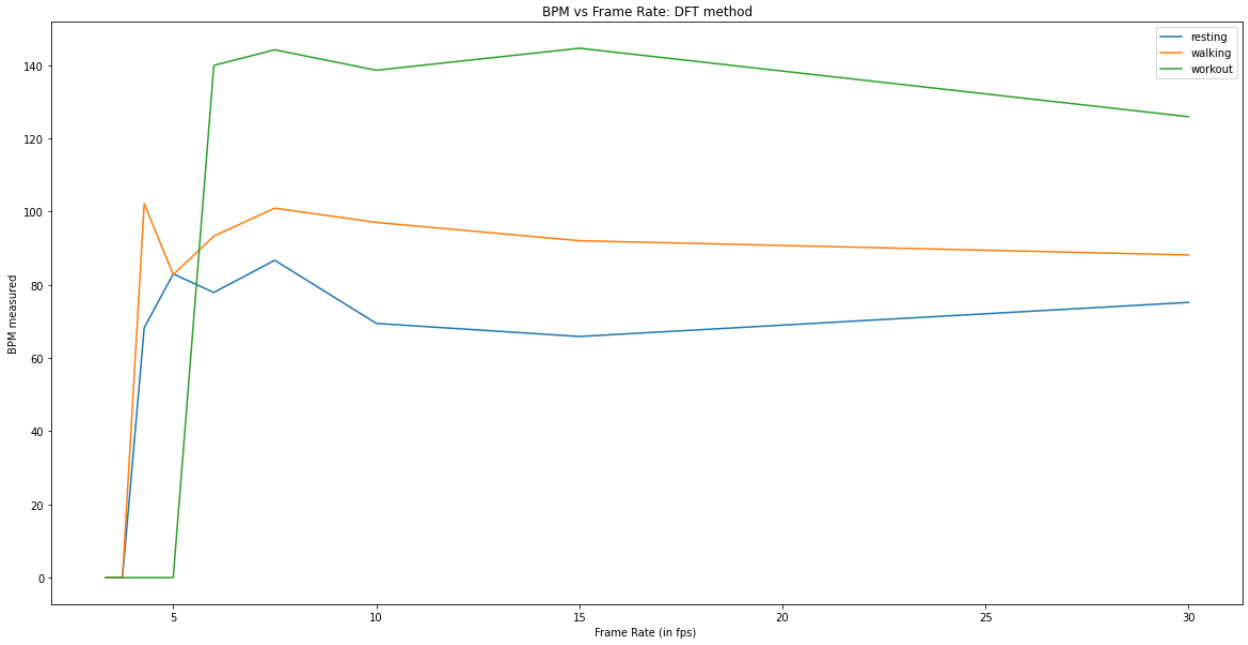
$$width_{new} = width * rescale/100$$

$$height_{new} = height * rescale/100$$

Where ***rescale*** is a parameter between 0 and 100.

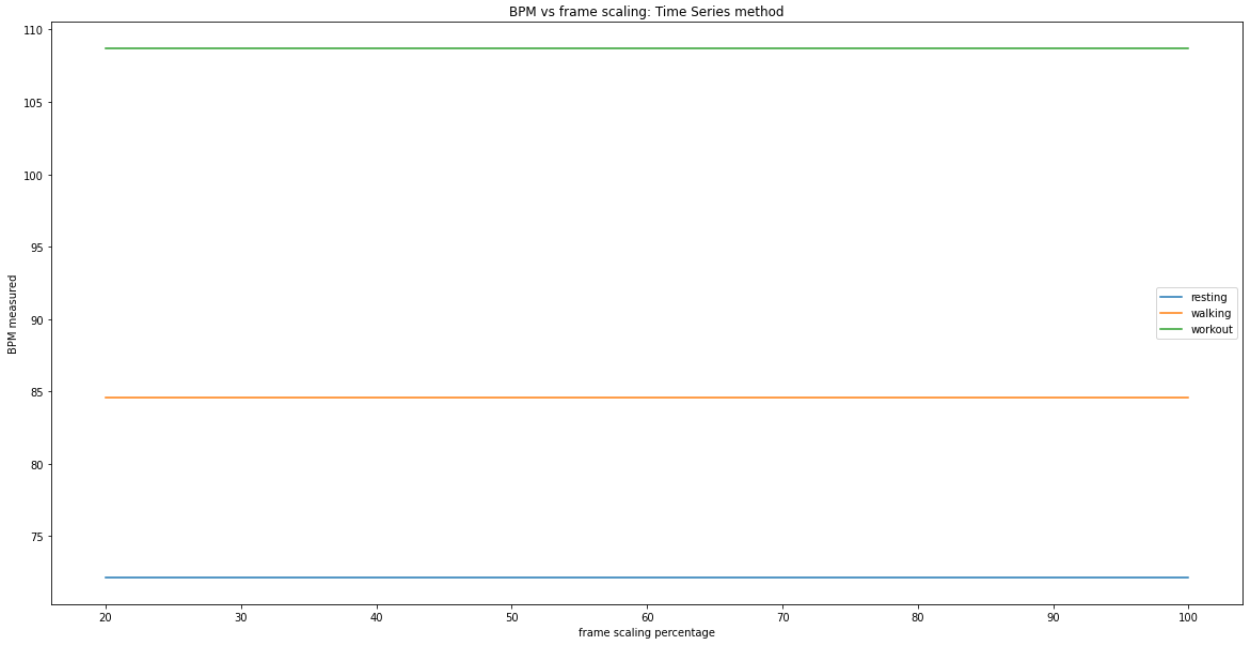
BPM vs Frame Rate

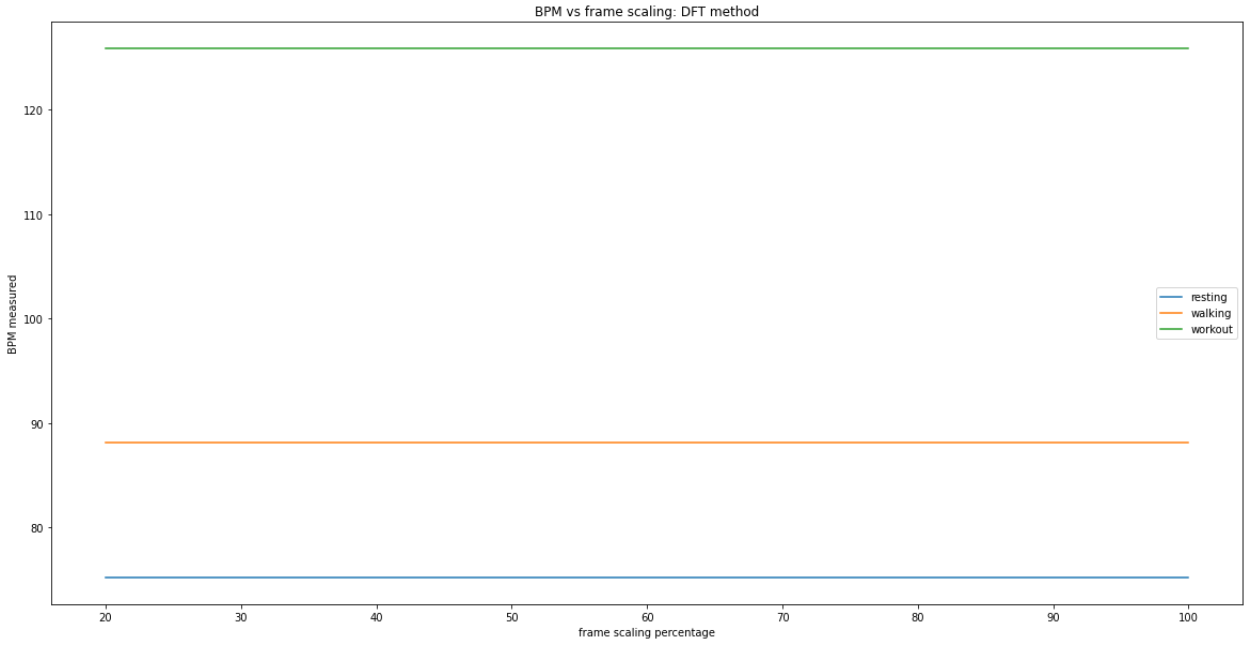




We can observe the DFT method to be very robust in terms of sensitivity to frame rate (fps).

BPM vs Frame Resolution [Ref 3]





Because of the averaging over the whole frame we do to get our FIM, rescaling of the frame didn't affect the pulse detected, as expected.

Cropping the frame

The following function takes in two frames and gives us the set of pixels which have magnitude of change above a certain fraction of the average.

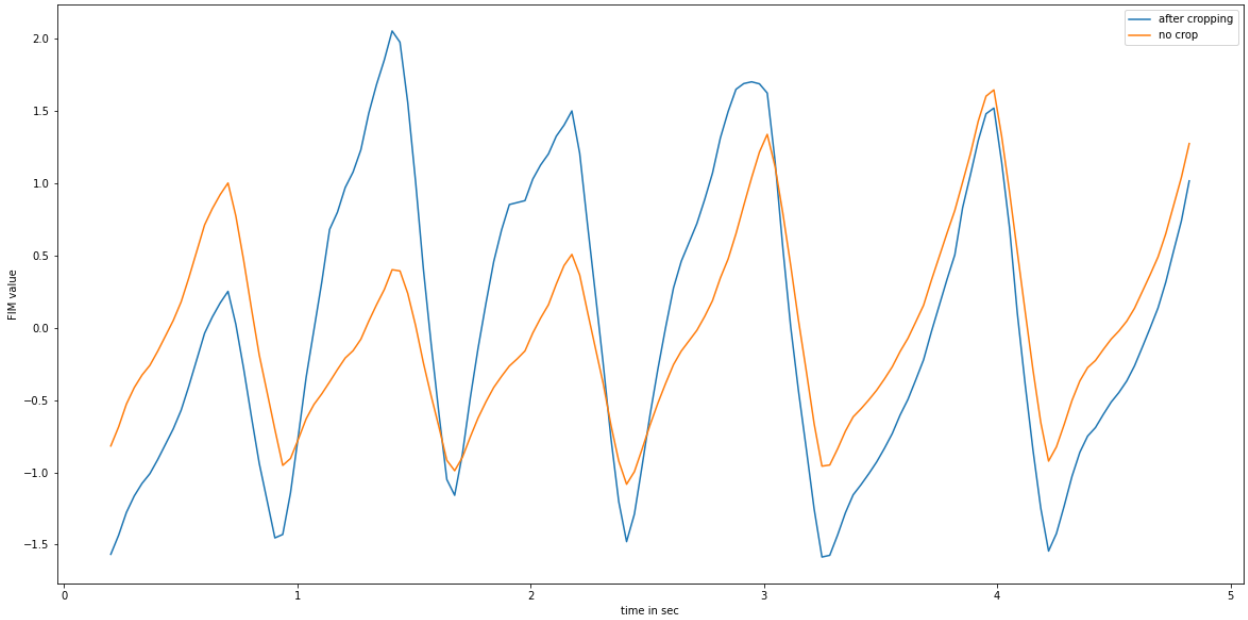
```
def dynamicOnly(frame1, frame2,cutoff = 20):
    f = abs(frame1 - frame2)
    gmax = np.max(f[:,:,1])
    select = np.where(f[:,:,1]>gmax*cutoff/100)
    return select
```

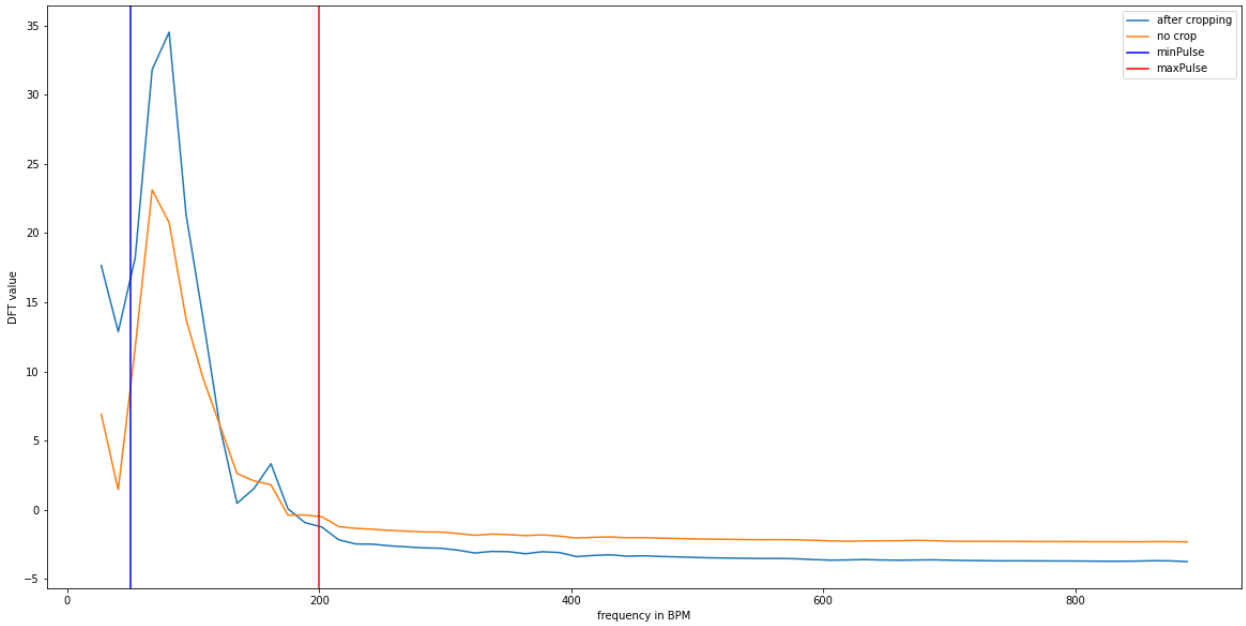
This is how we'll capture our frames. N = int(fps); t = duration in seconds

```
while(cap.isOpened() and i<t*N):
      # Capture frame-by-frame
      ret, frame = cap.read()
      if ret == True:
        i += 1
        if i< N:
            g = frame[:, :, 1]
            # computing the mean
        else:
            if i%N == 0:
                select = dynamicOnly(prevFrame, frame,cutoff)
            g = frame[:,:,1][select]

        g_mean = np.mean(g)
        green.append(g_mean)
        prevFrame = frame

      # Break the loop
      else:
        break
    # When everything done, release the video capture object
    cap.release()
```

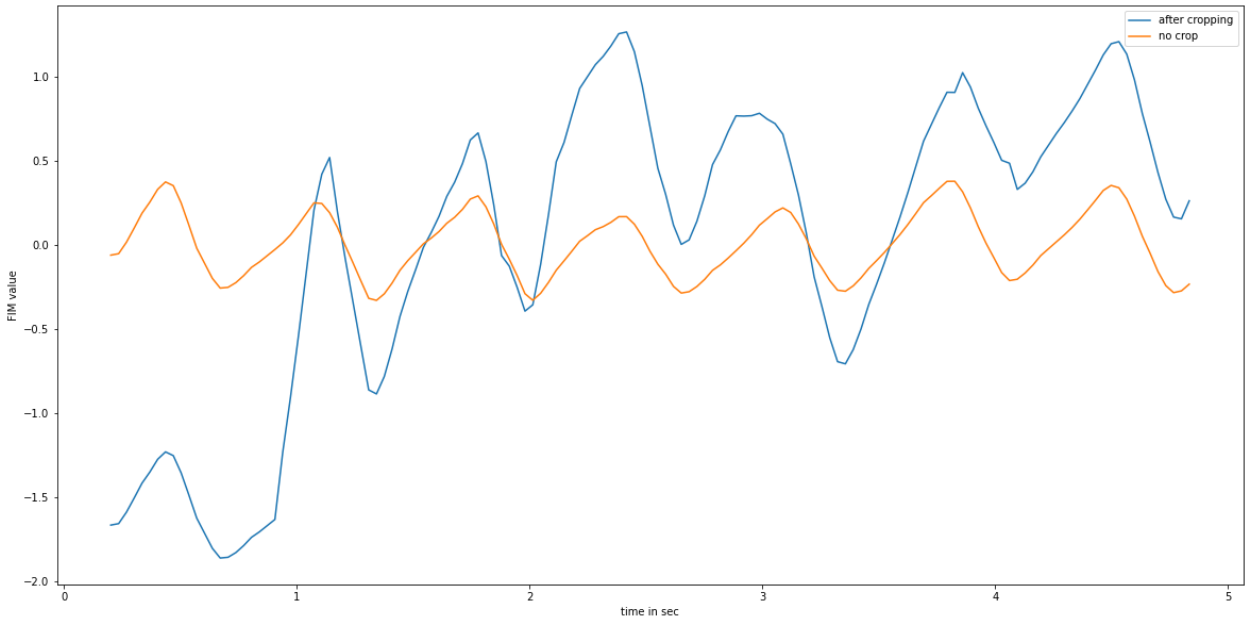
Here, instead of just sending the signal directly to peakFreq and majorFreq, we send a cleaned signal (cleanSignal(green,w)) since we observe the actual signal resulting from frame cropping to be very rough over the peaks.

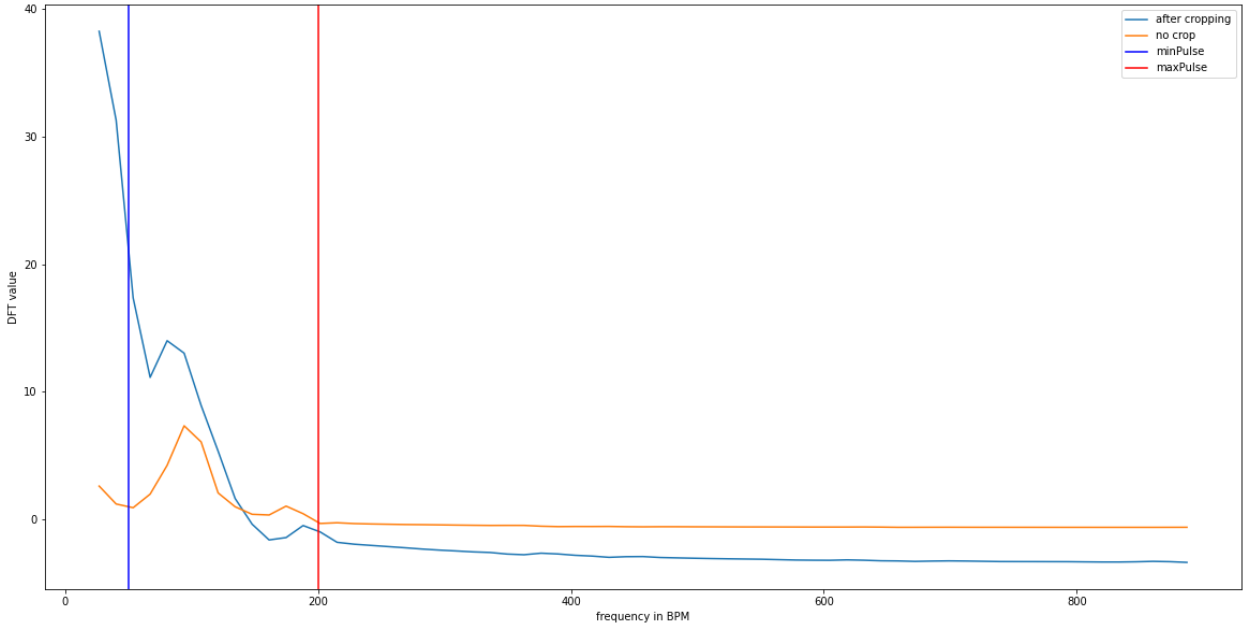
Resting


after cropping
no crop
FIM value
time in sec

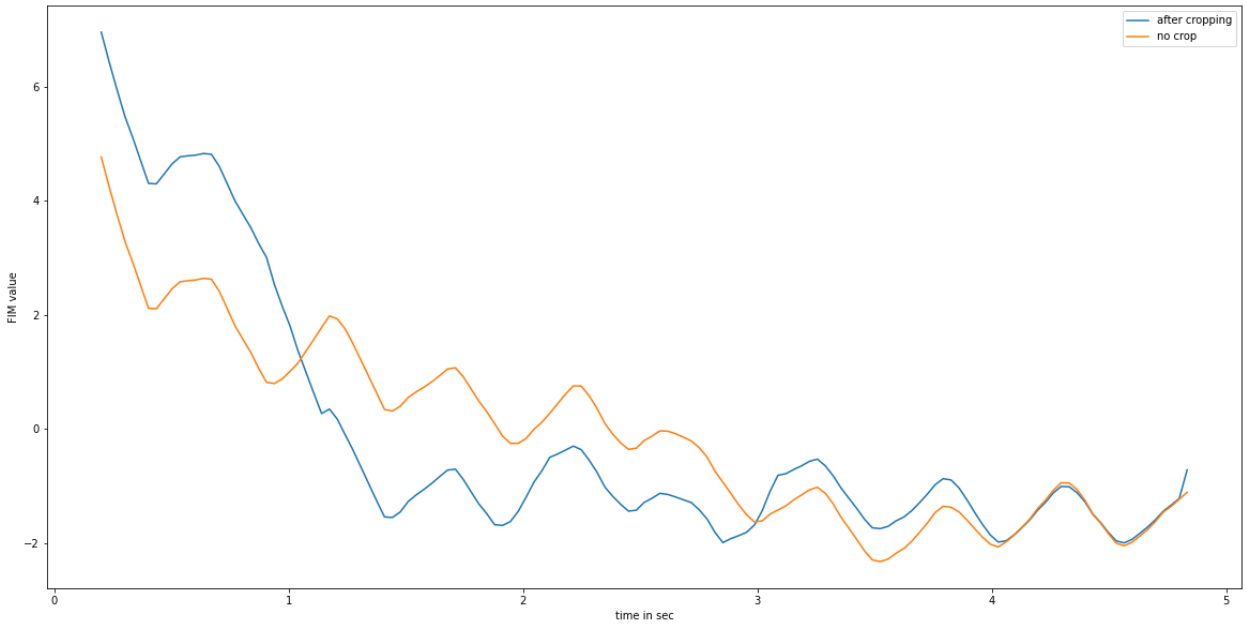

after cropping
no crop
minPulse
maxPulse
DFT value
frequency in BPM

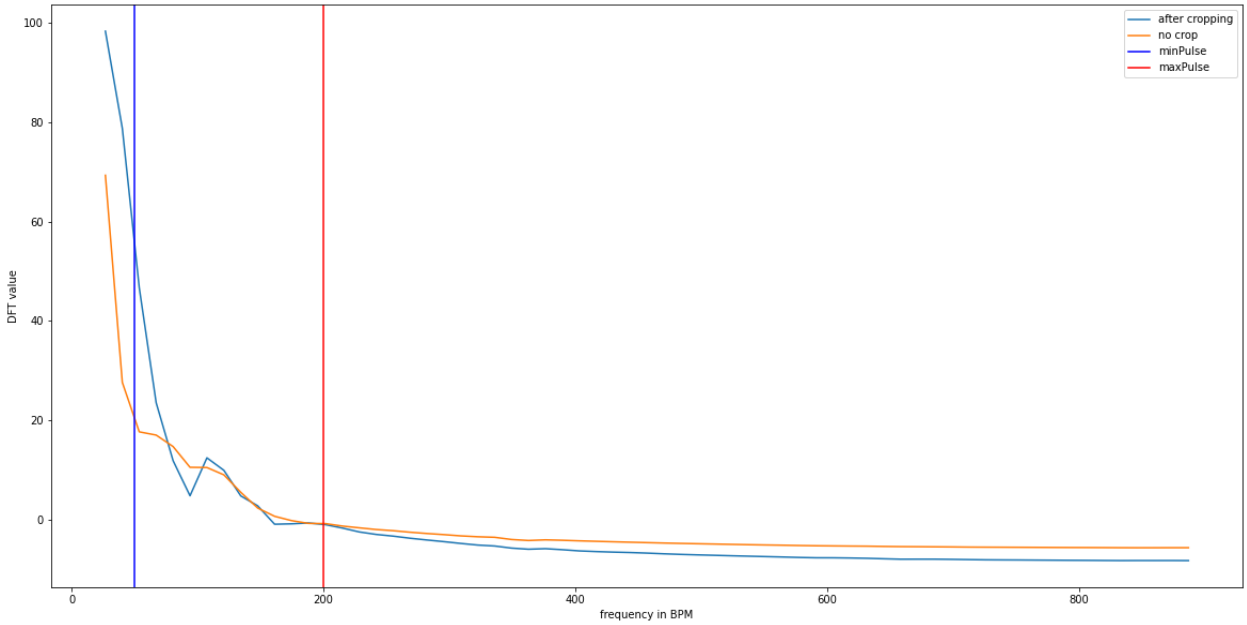
Walking


after cropping
no crop
FIM value
time in sec


after cropping
no crop
minPulse
maxPulse
DFT value
frequency in BPM

Workout





In all the three cases, we observe an increased SNR. But the downside is that some unwanted lower frequencies might just pop up with high value, giving us wrong estimates. Also, the said rough edges pose a challenge to the time series method by accounting for extra peaks.
Our results, above and in code, show that this method of cropping out inactive elements is best suited to the DFT technique, giving better estimates for shorter time windows.

Reference

1. S. Sukaphat, S. Nanthachaiporn, K. Upphaccha and P. Tantipatrakul, "Heart rate measurement on Android platform," 2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2016, pp. 1-5, doi: 10.1109/ECTICon.2016.7561442.
2. https://numpy.org/doc/stable/reference/generated/numpy.fft.fft.html
3. https://www.codingforentrepreneurs.com/blog/open-cv-python-change-video-resolution-or-scale