

CS6650 Homework 1

V S S Anirudh Sharma EE18B036

Install Libraries

In [1]:

```
1 pip install tabulate
```

Requirement already satisfied: tabulate in c:\users\vanir\appdata\local\programs\python\python39\lib\site-packages (0.8.9)

Note: you may need to restart the kernel to use updated packages.

WARNING: You are using pip version 21.1.3; however, version 21.2.4 is available.

You should consider upgrading via the 'c:\users\vanir\appdata\local\programs\python\python39\python.exe -m pip install --upgrade pip' command.

Importing Libraries

In [2]:

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from scipy.signal import find_peaks, peak_prominences, butter, sosfilt
5 from scipy import fft
6 from tabulate import tabulate
```

Functions

In [3]:

```
1 def cleanSignal(x,w = 7):
2     return np.convolve((x-np.mean(x)), np.ones(w), 'valid')/w
```

In [15]:

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

Helping in Time series technique

In [4]:

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

Helping in DFT technique

In [5]:

```
1 # returns DFT of a given vector after cleaning it
2 def dft_util(green, w=7):
3     signal = cleanSignal(green,w)
4     # compute DFT with optimized FFT
5     fs = np.fft.fftfreq(signal.shape[0])
6     sp = np.fft.fft(signal)
7     freq = np.fft.fftfreq(signal.shape[-1])
8     wh = np.where(freq>0)
9     return freq[wh],abs(sp)[wh]
```

In [6]:

```
1 #return highest peak in the dft of a signal in range of 40BPM-200BPM
2 def majorFreq(green,fps_act = 30,w1 = 7,w2=2, minPulse = 40, maxPulse = 200):
3     W,Y = dft_util(green,w1)
4     cleaned = cleanSignal(Y,w2); W1 = W[w2-1:]
5     minFreq = minPulse/(60*fps_act)
6     maxFreq = maxPulse/(60*fps_act)
7     normal = np.where(np.logical_and(W1>minFreq , W1<maxFreq))
8     cleaned2 = cleaned[normal]
9     W2 = W1[normal]
10    peaksFreq, _ = find_peaks(cleaned2,distance = 5)
11    try:
12        BPM_calc = W2[max(peaksFreq, key=lambda p: cleaned2[p])]*60*fps_act
13    except:
14        return 0
15    return BPM_calc
```

In [7]:

```
1 #Computing pulse form a full video using DFT technique
2 def BPM_dft(videoName,w1 = 7, w2 = 2):
3     green = []
4     cap = cv2.VideoCapture(videoName)
5
6     # Read until video is completed
7     while(cap.isOpened()):
8         # Capture frame-by-frame
9         ret, frame = cap.read()
10        if ret == True:
11            #b = frame[:, :, :1]
12            g = frame[:, :, 1:2]
13            #r = frame[:, :, 2:]
14
15            # computing the mean
16            #b_mean = np.mean(b)
17            g_mean = np.mean(g)
18            #r_mean = np.mean(r)
19            green.append(g_mean)
20
21        # Break the Loop
22    else:
23        break
24
25    # When everything done, find fps of video and release the video capture object
26    fps_act = cap.get(cv2.CAP_PROP_FPS)
27    cap.release()
28    #print(Len(green))
29    ...
30    W,Y = func1(green,w1)
31    cleaned = cleanSignal(Y,w2); W1 = W[w2-1:]
32    peaksFreq, _ = find_peaks(cleaned,distance = 5)
33    BPM_calc = W1[max(peaksFreq, key=lambda p: cleaned[p])]*60*fps_act
34    ...
35
36    return majorFreq(green,fps_act,w1,w2)
37
38    #return BPM_calc
```

In [8]:

```
1 #Computing pulse form a full video using Time series technique
2 def BPM_ts(videoName,w = 7):
3     green = []
4     cap = cv2.VideoCapture(videoName)
5
6     # Read until video is completed
7     while(cap.isOpened()):
8         # Capture frame-by-frame
9         ret, frame = cap.read()
10        if ret == True:
11            b = frame[:, :, :1]
12            g = frame[:, :, 1:2]
13            r = frame[:, :, 2:]
14
15            # computing the mean
16            #b_mean = np.mean(b)
17            g_mean = np.mean(g)
18            #r_mean = np.mean(r)
19            green.append(g_mean)
20
21        # Break the Loop
22        else:
23            break
24
25    # When everything done, find fps of video and release the video capture object
26    fps_act = cap.get(cv2.CAP_PROP_FPS)
27    cap.release()
28
29    return peakFreq(green, fps_act,w)
```

Q2

In [9]:

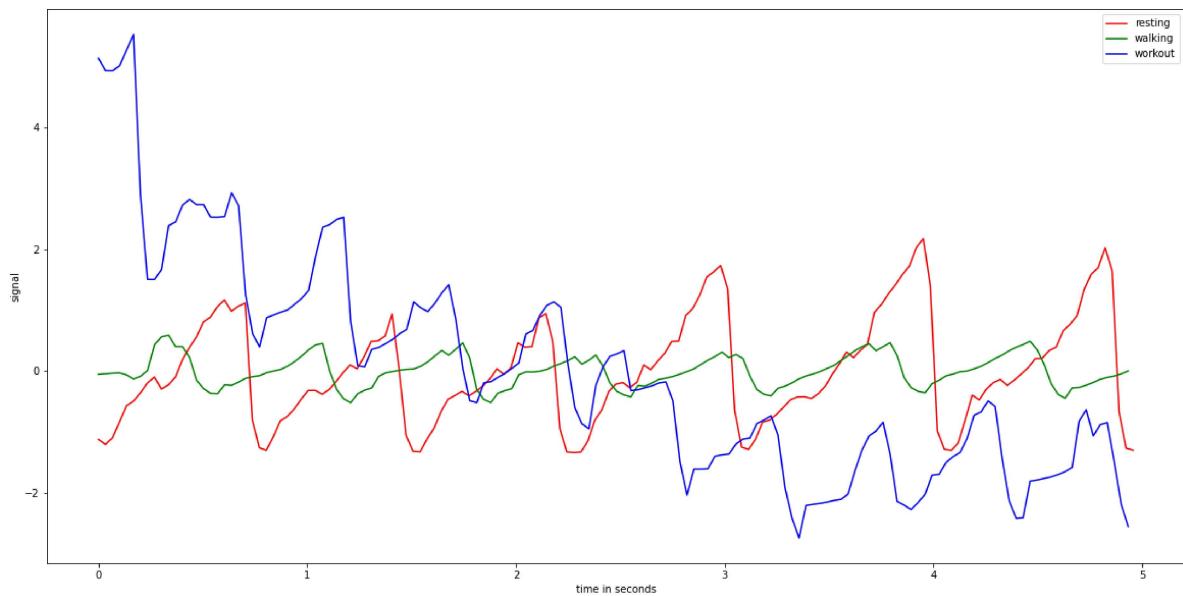
```
1 def plot_5sec(videoName):
2     green = np.zeros((0,1))
3     cap = cv2.VideoCapture(videoName)
4     fps_act = cap.get(cv2.CAP_PROP_FPS)
5     N = int(fps_act*5)
6     # Read until video is completed
7     i = 0
8     while(cap.isOpened() and i<N):
9
10         # Capture frame-by-frame
11         ret, frame = cap.read()
12         if ret == True:
13             b = frame[:, :, :1]
14             g = frame[:, :, 1:2]
15             r = frame[:, :, 2:]
16
17             # computing the mean
18             #b_mean = np.mean(b)
19             g_mean = np.mean(g)
20             #r_mean = np.mean(r)
21             green = np.append(green,[g_mean])
22             i += 1
23
24         # Break the Loop
25     else:
26         break
27
28     # When everything done, find fps of video and release the video capture object
29
30     cap.release()
31     l = green.shape[0]
32     return [i/fps_act for i in range(l)], green - np.mean(green)
```

In [10]:

```
1 t1, s1 = plot_5sec('EE18B036_1.mp4')
2 t2, s2 = plot_5sec('EE18B036_2.mp4')
3 t3, s3 = plot_5sec('EE18B036_3.mp4')
```

In [11]:

```
1 w = 7
2 plt.figure(figsize = (20,10))
3 plt.plot(t1,s1,'r',label = 'resting')#, linewidth = 0.5)
4 #plt.plot(t1[:-w+1],cleanSignal(s1,w), 'r',Label = 'cleaned resting')
5 plt.plot(t2,s2,'g',label = 'walking')#, linewidth = 0.5)
6 #plt.plot(t2[:-w+1],cleanSignal(s2,w), 'g',Label = 'cleaned walng')
7 plt.plot(t3,s3,'b',label = 'workout')#, linewidth = 0.5)
8 #plt.plot(t3[:-w+1],cleanSignal(s3,w), 'b',Label = 'cleaned workout')
9 plt.xlabel('time in seconds')
10 plt.ylabel('signal')
11 plt.legend()
12 plt.show()
```



In [12]:

```
1 #Return pulse rate for the first t seconds of video using time series technique
2 def BPM_sec_ts(videoName,t = 5,rescale = 100,fpsScale = 1, w = 7):
3     green = []
4     cap = cv2.VideoCapture(videoName)
5     #fps = cap.get(cv2.CAP_PROP_FPS)
6     #cap.set(cv2.CAP_PROP_FPS, fps*fpsScale/100)
7     fps = cap.get(cv2.CAP_PROP_FPS)
8     #print(fps)
9     bpm = 0
10    #fps = cap.get(cv2.CAP_PROP_FPS)
11    N = int(t*fps)
12    #result = cv2.VideoWriter('2_bpm.mp4', cv2.VideoWriter_fourcc(*'MP4V'),fps,(1920,16
13    # Read until video is completed
14    i = 0
15    font = cv2.FONT_HERSHEY_SIMPLEX
16    while(cap.isOpened() and i<N):
17        # Capture frame-by-frame
18        ret, frame = cap.read()
19
20        if ret == True:
21            i += 1
22            # Display the resulting frame
23            # setting values for base colors
24
25            if i%fpsScale == 0:
26                frame2 = rescale_frame(frame, rescale)
27                g = frame2[:, :, 1]
28
29                # computing the mean
30                g_mean = np.mean(g)
31                green.append(g_mean)
32
33
34
35            # Press Q on keyboard to exit
36            if cv2.waitKey(25) & 0xFF == ord('q'):
37                break
38
39
40
41            # Break the Loop
42        else:
43            break
44
45        # When everything done, release the video capture object
46        cap.release()
47        #result.release()
48
49        return peakFreq(green, fps/fpsScale)
```

In [13]:

```
1 #Return pulse rate for the first t seconds of video using time series technique
2 def BPM_sec_dft(videoName, t, rescale = 100, fpsScale = 1, w1 = 7, w2=2):
3     green = []
4     cap = cv2.VideoCapture(videoName)
5     bpm = 0
6     fps = cap.get(cv2.CAP_PROP_FPS)
7     N = int(t*fps)
8     #result = cv2.VideoWriter('2_bpm.mp4', cv2.VideoWriter_fourcc(*'MP4V'),fps,(1920,1600))
9     # Read until video is completed
10    i = 0
11    font = cv2.FONT_HERSHEY_SIMPLEX
12    while(cap.isOpened() and i<N):
13        # Capture frame-by-frame
14        ret, frame = cap.read()
15
16        if ret == True:
17            i += 1
18            # Display the resulting frame
19            # setting values for base colors
20            if i%fpsScale == 0:
21                frame2 = rescale_frame(frame, rescale)
22                g = frame2[:, :, 1]
23
24                # computing the mean
25                g_mean = np.mean(g)
26                green.append(g_mean)
27
28
29
30            # Press Q on keyboard to exit
31            if cv2.waitKey(25) & 0xFF == ord('q'):
32                break
33
34
35
36            # Break the Loop
37        else:
38            break
39
40        # When everything done, release the video capture object
41        cap.release()
42        #result.release()
43
44    return majorFreq(green,fps/fpsScale,w1,w2)
```

In [16]:

```
1 TS_5_1 = BPM_sec_ts('EE18B036_1.mp4',5)
2 TS_5_2 = BPM_sec_ts('EE18B036_2.mp4',5)
3 TS_5_3 = BPM_sec_ts('EE18B036_3.mp4',5)
4
5 DFT_5_1 = BPM_sec_dft('EE18B036_1.mp4',5)
6 DFT_5_2 = BPM_sec_dft('EE18B036_2.mp4',5)
7 DFT_5_3 = BPM_sec_dft('EE18B036_3.mp4',5)
8
```

In [17]:

```
1 # assign data
2 fiveSecData = [["Time Series", TS_5_1, TS_5_2, TS_5_3], ["DFT", DFT_5_1, DFT_5_2, DFT_5_3]]
3
4 # create header
5 head = ["Method", "Resting", "Walking", "Workout"]
6
7 # display table
8 print('in the first 5 seconds:')
9 print(tabulate(fiveSecData, headers=head, tablefmt="grid"))
```

in the first 5 seconds:

Method	Resting	Walking	Workout
Time Series	72.135	84.5512	108.704
DFT	75.1616	88.1238	125.886

In [18]:

```
1 TS_10_1 = BPM_sec_ts('EE18B036_1.mp4',10)
2 TS_10_2 = BPM_sec_ts('EE18B036_2.mp4',10)
3 TS_10_3 = BPM_sec_ts('EE18B036_3.mp4',10)
4
5
6 DFT_10_1 = BPM_sec_dft('EE18B036_1.mp4',10)
7 DFT_10_2 = BPM_sec_dft('EE18B036_2.mp4',10)
8 DFT_10_3 = BPM_sec_dft('EE18B036_3.mp4',10)
9
```

In [19]:

```
1 # assign data
2 tenSecData = [["Time Series", TS_10_1, TS_10_2, TS_10_3], ["DFT", DFT_10_1, DFT_10_2, DFT_10_3]]
3
4 # create header
5 head = ["Method", "Resting", "Walking", "Workout"]
6
7 # display table
8 print('in the first 10 seconds:')
9 print(tabulate(tenSecData, headers=head, tablefmt="grid"))
```

in the first 10 seconds:

Method	Resting	Walking	Workout
Time Series	66.1237	84.2665	108.338
DFT	73.6172	98.2903	116.715

In [20]:

```
1 TS_20_1 = BPM_sec_ts('EE18B036_1.mp4',20)
2 TS_20_2 = BPM_sec_ts('EE18B036_2.mp4',20)
3 TS_20_3 = BPM_sec_ts('EE18B036_3.mp4',20)
4
5
6 DFT_20_1 = BPM_sec_dft('EE18B036_1.mp4',20)
7 DFT_20_2 = BPM_sec_dft('EE18B036_2.mp4',20)
8 DFT_20_3 = BPM_sec_dft('EE18B036_3.mp4',20)
9
```

In [21]:

```
1 # assign data
2 twentySecData = [[{"Time Series": TS_20_1, TS_20_2, TS_20_3}, {"DFT": DFT_20_1, DFT_20_2, DFT_20_3}]
3
4 # create header
5 head = ["Method", "Resting", "Walking", "Workout"]
6
7 # display table
8 print('in the first 20 seconds:')
9 print(tabulate(twentySecData, headers=head, tablefmt="grid"))
```

in the first 20 seconds:

Method	Resting	Walking	Workout
Time Series	72.0141	81.1204	108.156
DFT	72.7453	81.9468	109.258

In [22]:

```
1 BPMdata = [[5, TS_5_1, TS_5_2, TS_5_3, DFT_5_1, DFT_5_2, DFT_5_3}, [10, TS_10_1, TS_10_2, TS_10_3, DFT_10_1, DFT_10_2, DFT_10_3]]
```

In [23]:

```
1 BPMhead = ["Time in sec", "TS Resting", "TS Walking", "TS Workout", "DFT Resting", "DFT Walking", "DFT Workout"]
```

In [24]:

```
1 print('Estimates of pulse rate over different periods of time by different algorithm')
2 print(tabulate(BPMdata, headers=BPMhead, tablefmt="grid"))
```

Estimates of pulse rate over different periods of time by different algorithm				
Time in sec	TS Resting	TS Walking	TS Workout	DFT Resting
DFT Walking	DFT Workout			
5	72.135	84.5512	108.704	75.1616
88.1238	125.886			
10	66.1237	84.2665	108.338	73.6172
98.2903	116.715			
20	72.0141	81.1204	108.156	72.7453
81.9468	109.258			

Q3

Playing video with pulse rate being displayed on it

In [25]:

```
1 #display video with pulse rate every second taken over 5 seconds using time series tech
2 def BPM_disp_ts(videoName,w = 7):
3     green = []
4     cap = cv2.VideoCapture(videoName)
5     bpm = 0
6     fps = cap.get(cv2.CAP_PROP_FPS)
7     N = int(5*fps)
8     #result = cv2.VideoWriter('2_bpm.mp4', cv2.VideoWriter_fourcc(*'MP4V'),fps,(1920,1600))
9     # Read until video is completed
10    i = 0
11    font = cv2.FONT_HERSHEY_SIMPLEX
12    while(cap.isOpened()):
13        # Capture frame-by-frame
14        ret, frame = cap.read()
15
16        if ret == True:
17            i += 1
18            # Display the resulting frame
19            # setting values for base colors
20
21            g = frame[:, :, 1:2]
22
23            # computing the mean
24            g_mean = np.mean(g)
25            green.append(g_mean)
26
27            if i>N and i%(N//5)==0:
28
29                bpm = peakFreq(green[-N:], fps, w)
30
31
32            cv2.putText(frame,
33                        '{}:PulseRate = {} BPM'.format(videoName,bpm),
34                        (50, 50),
35                        font, 1,
36                        (0, 255, 255),
37                        2,
38                        cv2.LINE_4)
39            cv2.imshow('Frame',frame)
40
41            #result.write(frame)
42
43
44
45            # Press Q on keyboard to exit
46            if cv2.waitKey(25) & 0xFF == ord('q'):
47                break
48
49
50
51            # Break the Loop
52            else:
53                break
54
55            # When everything done, release the video capture object
56            cap.release()
57            #result.release()
58
59            # Closes all the frames
```


In [26]:

```
1 #display video with pulse rate every second taken over 5 seconds using DFT technique
2 def BPM_disp_dft(videoName,w1 = 7,w2 = 2):
3     green = []
4     cap = cv2.VideoCapture(videoName)
5     bpm = 0
6     fps = cap.get(cv2.CAP_PROP_FPS)
7     N = int(5*fps)
8     #result = cv2.VideoWriter('2_bpm.mp4', cv2.VideoWriter_fourcc(*'MP4V'),fps,(1920,1600))
9     # Read until video is completed
10    i = 0
11    font = cv2.FONT_HERSHEY_SIMPLEX
12    while(cap.isOpened()):
13        # Capture frame-by-frame
14        ret, frame = cap.read()
15
16        if ret == True:
17            i += 1
18            # Display the resulting frame
19            # setting values for base colors
20
21            g = frame[:, :, 1:2]
22
23            # computing the mean
24            g_mean = np.mean(g)
25            green.append(g_mean)
26
27            if i>N and i%(N//5)==0:
28
29                bpm = majorFreq(green[-N:],fps,w1,w2)
30
31                cv2.putText(frame,
32                            '{}:PulseRate = {} BPM'.format(videoName,bpm),
33                            (50, 50),
34                            font, 1,
35                            (0, 255, 255),
36                            2,
37                            cv2.LINE_4)
38                cv2.imshow('Frame',frame)
39
40                #result.write(frame)
41
42
43
44            # Press Q on keyboard to exit
45            if cv2.waitKey(25) & 0xFF == ord('q'):
46                break
47
48
49
50            # Break the Loop
51            else:
52                break
53
54            # When everything done, release the video capture object
55            cap.release()
56            #result.release()
57
58            # Closes all the frames
59            cv2.destroyAllWindows()
```

In [27]:

```
1 #from Time Series Method
2 BPM_disp_ts('EE18B036_1.mp4')
3 BPM_disp_ts('EE18B036_2.mp4')
4 BPM_disp_ts('EE18B036_3.mp4')
```

In [28]:

```
1 #from DFT Method
2 BPM_disp_dft('EE18B036_1.mp4')
3 BPM_disp_dft('EE18B036_2.mp4')
4 BPM_disp_dft('EE18B036_3.mp4')
```

Q4

In [29]:

```
1 #Return pulse rate for the first 5 seconds of video using time series technique
2 def BPM_5sec_ts(videoName,rescale = 100,fpsScale = 1, w = 7):
3     green = []
4     cap = cv2.VideoCapture(videoName)
5     fps = cap.get(cv2.CAP_PROP_FPS)
6     bpm = 0
7     N = int(5*fps) #5 seconds
8     #result = cv2.VideoWriter('2_bpm.mp4', cv2.VideoWriter_fourcc(*'MP4V'),fps,(1920,1600))
9     # Read until video is completed
10    i = 0
11    while(cap.isOpened() and i<N):
12        # Capture frame-by-frame
13        ret, frame = cap.read()
14
15        if ret == True:
16            i += 1
17            # Display the resulting frame
18            # setting values for base colors
19
20            if i%fpsScale == 0:
21                frame2 = rescale_frame(frame, rescale)
22                g = frame2[:, :, 1]
23
24                # computing the mean
25                g_mean = np.mean(g)
26                green.append(g_mean)
27
28
29
30            # Press Q on keyboard to exit
31            if cv2.waitKey(25) & 0xFF == ord('q'):
32                break
33
34
35
36            # Break the Loop
37            else:
38                break
39
40            # When everything done, release the video capture object
41            cap.release()
42            #result.release()
43
44        return peakFreq(green, fps/fpsScale)
```

In [30]:

```
1 #Return pulse rate for the first 5 seconds of video using DFT technique
2 def BPM_5sec_dft(videoName,rescale = 100,fpsScale = 1, w1 = 7,w2=2):
3     green = []
4     cap = cv2.VideoCapture(videoName)
5     bpm = 0
6     fps = cap.get(cv2.CAP_PROP_FPS)
7     N = int(5*fps)
8     #result = cv2.VideoWriter('2_bpm.mp4', cv2.VideoWriter_fourcc(*'MP4V'),fps,(1920,1600))
9     # Read until video is completed
10    i = 0
11    font = cv2.FONT_HERSHEY_SIMPLEX
12    while(cap.isOpened() and i<N):
13        # Capture frame-by-frame
14        ret, frame = cap.read()
15
16        if ret == True:
17            i += 1
18            # Display the resulting frame
19            # setting values for base colors
20            if i%fpsScale == 0:
21                frame2 = rescale_frame(frame, rescale)
22                g = frame2[:, :, 1]
23
24                # computing the mean
25                g_mean = np.mean(g)
26                green.append(g_mean)
27
28
29
30            # Press Q on keyboard to exit
31            if cv2.waitKey(25) & 0xFF == ord('q'):
32                break
33
34
35
36            # Break the Loop
37            else:
38                break
39
40            # When everything done, release the video capture object
41            cap.release()
42            #result.release()
43
44        return majorFreq(green,fps/fpsScale,w1,w2)
```

In [31]:

```
1 frameScales = [i for i in range(20,101,20)]
```

In [32]:

```
1 fpsScales = [i for i in range(1,10)]
```

In [33]:

```
1 frameRates = [30/i for i in fpsScales]
```

In [34]:

```
1 ## TIME SERIES
```

In [35]:

```
1 BPM1_frsc_ts = [BPM_5sec_ts('EE18B036_1.mp4',i,1) for i in frameScales]
```

In [36]:

```
1 BPM2_frsc_ts = [BPM_5sec_ts('EE18B036_2.mp4',i,1) for i in frameScales]
```

In [37]:

```
1 BPM3_frsc_ts = [BPM_5sec_ts('EE18B036_3.mp4',i,1) for i in frameScales]
```

In [38]:

```
1 BPM1_fpssc_ts = [BPM_5sec_ts('EE18B036_1.mp4',100,i) for i in fpsScales]
```

In [39]:

```
1 BPM2_fpssc_ts = [BPM_5sec_ts('EE18B036_2.mp4',100,i) for i in fpsScales]
```

In [40]:

```
1 BPM3_fpssc_ts = [BPM_5sec_ts('EE18B036_3.mp4',100,i) for i in fpsScales]
```

In [41]:

```
1 # DFT
```

In [42]:

```
1 BPM1_frsc_dft = [BPM_5sec_dft('EE18B036_1.mp4',i,1) for i in frameScales]
```

In [43]:

```
1 BPM2_frsc_dft = [BPM_5sec_dft('EE18B036_2.mp4',i,1) for i in frameScales]
```

In [44]:

```
1 BPM3_frsc_dft = [BPM_5sec_dft('EE18B036_3.mp4',i,1) for i in frameScales]
```

In [45]:

```
1 BPM1_fpssc_dft = [BPM_5sec_dft('EE18B036_1.mp4',100,i) for i in fpsScales]
```

In [46]:

```
1 BPM2_fpssc_dft = [BPM_5sec_dft('EE18B036_2.mp4',100,i) for i in fpsScales]
```

In [47]:

```
1 BPM3_fpssc_dft = [BPM_5sec_dft('EE18B036_3.mp4',100,i) for i in fpsScales]
```

In [48]:

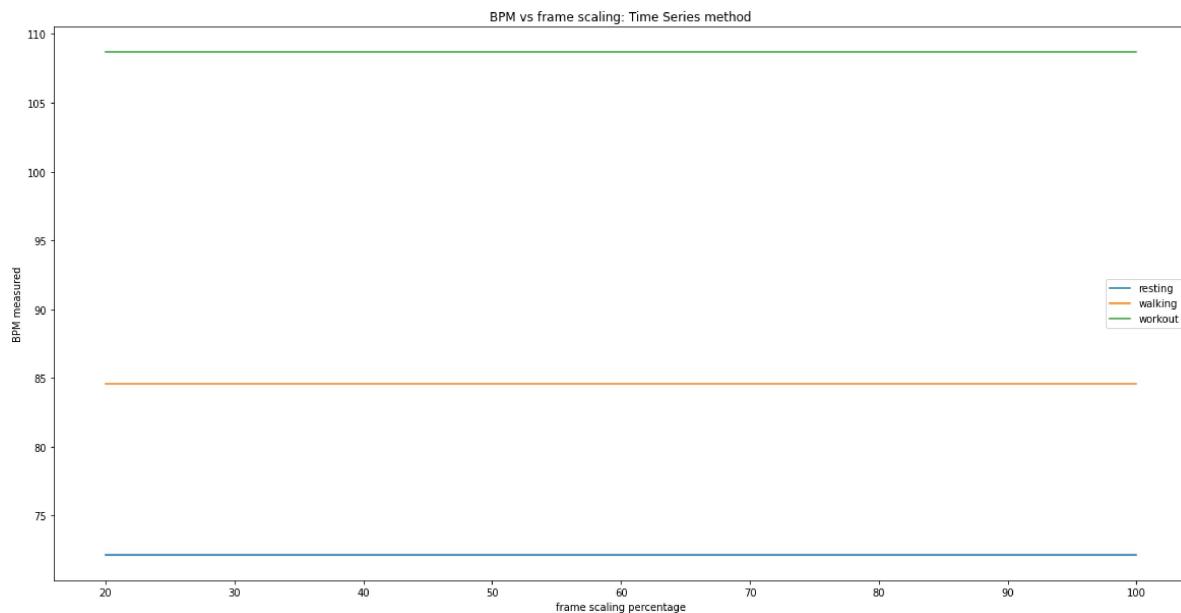
```
1 #plots
```

In [49]:

```
1 plt.figure(figsize = (20,10))
2 plt.plot(frameScales,BPM1_frsc_ts,label = 'resting')
3 plt.plot(frameScales,BPM2_frsc_ts,label = 'walking')
4 plt.plot(frameScales,BPM3_frsc_ts,label = 'workout')
5 plt.xlabel('frame scaling percentage')
6 plt.ylabel('BPM measured')
7 plt.title('BPM vs frame scaling: Time Series method')
8 plt.legend()
```

Out[49]:

```
<matplotlib.legend.Legend at 0x272425681c0>
```

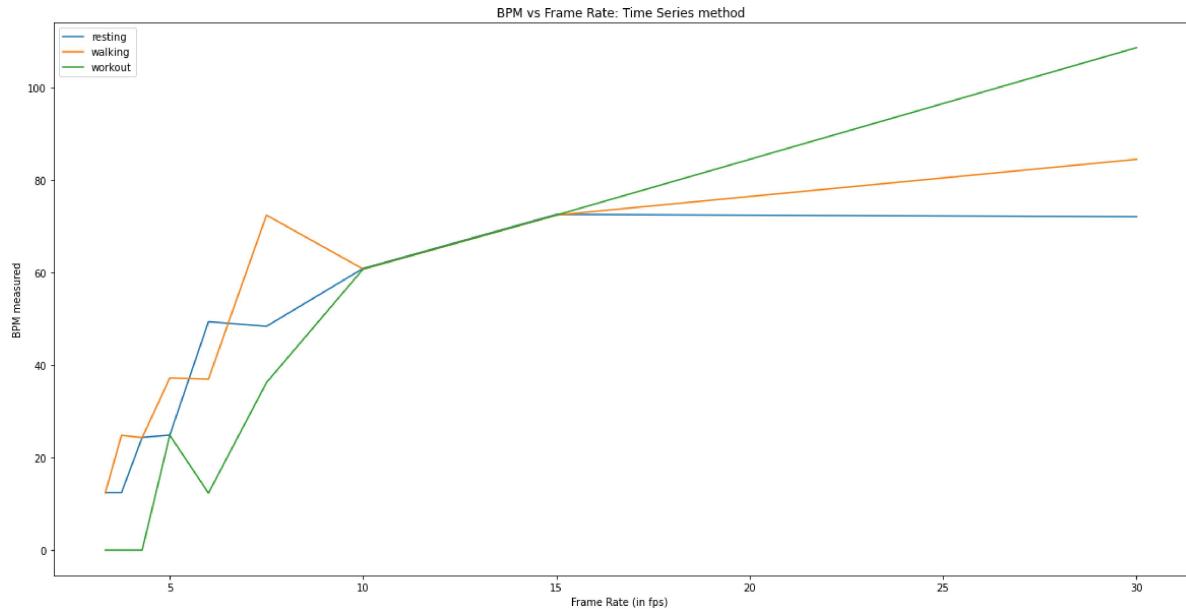


In [50]:

```
1 plt.figure(figsize = (20,10))
2 plt.plot(frameRates,BPM1_fpssc_ts,label = 'resting')
3 plt.plot(frameRates,BPM2_fpssc_ts,label = 'walking')
4 plt.plot(frameRates,BPM3_fpssc_ts,label = 'workout')
5 plt.xlabel('Frame Rate (in fps) ')
6 plt.ylabel('BPM measured')
7 plt.title('BPM vs Frame Rate: Time Series method')
8 plt.legend()
```

Out[50]:

```
<matplotlib.legend.Legend at 0x272445ab370>
```

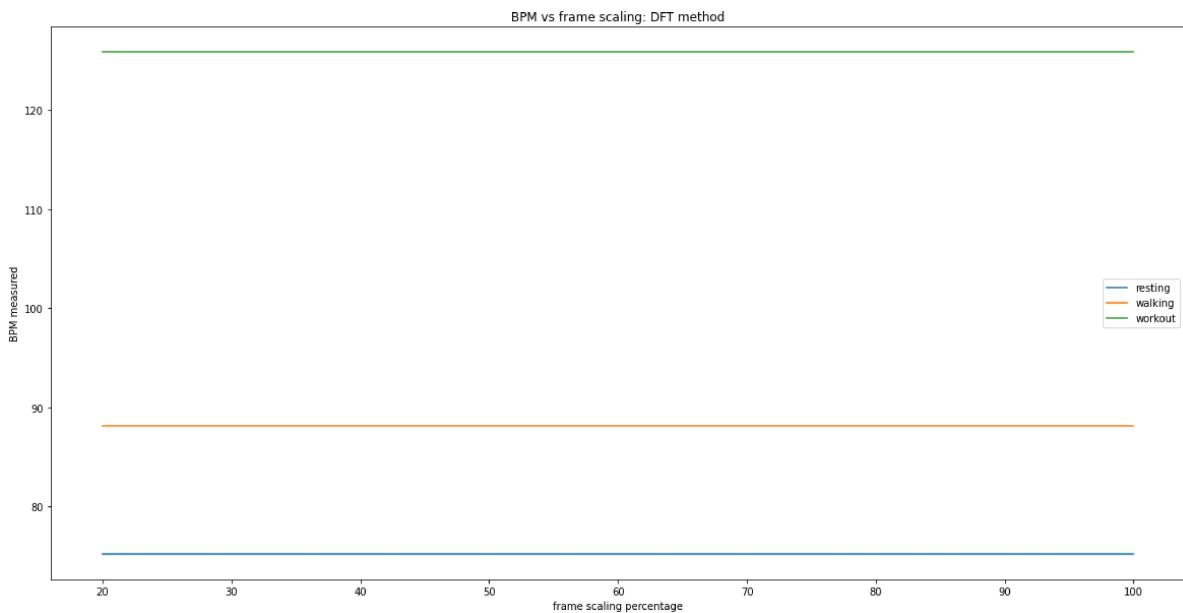


In [51]:

```
1 plt.figure(figsize = (20,10))
2 plt.plot(frameScales,BPM1_frsc_dft,label = 'resting')
3 plt.plot(frameScales,BPM2_frsc_dft,label = 'walking')
4 plt.plot(frameScales,BPM3_frsc_dft,label = 'workout')
5 plt.xlabel('frame scaling percentage')
6 plt.ylabel('BPM measured')
7 plt.title('BPM vs frame scaling: DFT method')
8 plt.legend()
```

Out[51]:

```
<matplotlib.legend.Legend at 0x27244619fa0>
```

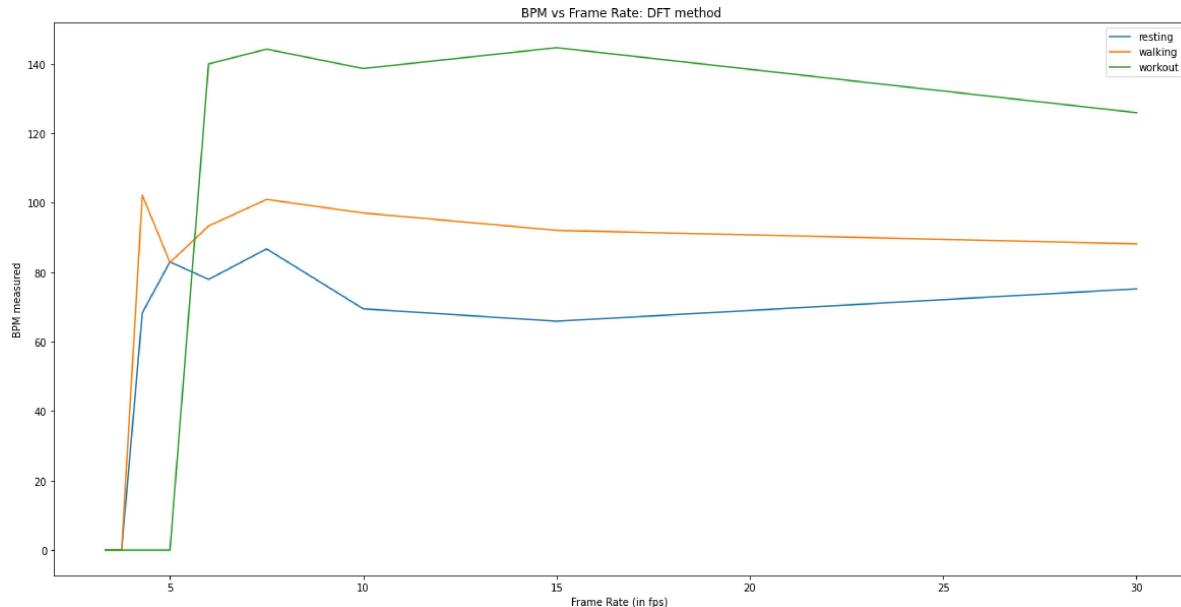


In [52]:

```
1 plt.figure(figsize = (20,10))
2 plt.plot(frameRates,BPM1_fpssc_dft,label = 'resting')
3 plt.plot(frameRates,BPM2_fpssc_dft,label = 'walking')
4 plt.plot(frameRates,BPM3_fpssc_dft,label = 'workout')
5 plt.xlabel('Frame Rate (in fps) ')
6 plt.ylabel('BPM measured')
7 plt.title('BPM vs Frame Rate: DFT method')
8 plt.legend()
```

Out[52]:

<matplotlib.legend.Legend at 0x272449439d0>



Finding only dynamic pixels in the frame to enhance SNR

In [53]:

```
1 def dynamicOnly(frame1, frame2,cutoff = 5):
2     f = abs(frame1 - frame2)
3     gmax = np.max((f[:, :, 1]))
4     select = np.where(abs(f[:, :, 1])>=gmax*cutoff/100)
5     return select
```

In [54]:

```
1 #Return pulse rate for the first t seconds of video using time series technique and usi
2 def BPM_better_ts(videoName,t = 5, cutoff = 10,w0 = 7, w1 = 7):
3     green = []
4     cap = cv2.VideoCapture(videoName)
5     bpm = 0
6     fps = cap.get(cv2.CAP_PROP_FPS)
7     N = int(fps)
8     #result = cv2.VideoWriter('2_bpm.mp4', cv2.VideoWriter_fourcc(*'MP4V'),fps,(1920,16
9     # Read until video is completed
10    i = 0
11    #print(cap.get(3))
12    prevFrame = np.zeros((int(cap.get(4)),int(cap.get(3)),3))
13    #print(prevFrame.shape)
14    font = cv2.FONT_HERSHEY_SIMPLEX
15    while(cap.isOpened() and i<t*N):
16        # Capture frame-by-frame
17        ret, frame = cap.read()
18        #print(frame.dtype, frame.shape)
19        if ret == True:
20            i += 1
21
22
23        if i< N:
24            g = frame[:, :, 1]
25
26            # computing the mean
27
28        else:
29            if i%N == 0:
30                select = dynamicOnly(prevFrame, frame,cutoff)
31
32            g = frame[:, :, 1][select]
33            #print(g.shape)
34
35
36            g_mean = np.mean(g)
37            green.append(g_mean)
38            prevFrame = frame
39
40
41        # Break the Loop
42        else:
43            break
44
45
46        # When everything done, release the video capture object
47        cap.release()
48
49        return peakFreq(cleanSignal(green,w0), fps,w1)
```

In [55]:

```
1 #Return pulse rate for the first t seconds of video using time series technique and usi
2 def BPM_better_dft(videoName,t = 5, cutoff = 20,w0 = 10, w1 = 7, w2 = 2):
3     green = []
4     cap = cv2.VideoCapture(videoName)
5     bpm = 0
6     fps = cap.get(cv2.CAP_PROP_FPS)
7     N = int(fps)
8     #result = cv2.VideoWriter('2_bpm.mp4', cv2.VideoWriter_fourcc(*'MP4V'),fps,(1920,16
9     # Read until video is completed
10    i = 0
11    #print(cap.get(3))
12    prevFrame = np.zeros((int(cap.get(4)),int(cap.get(3)),3))
13    #print(prevFrame.shape)
14    font = cv2.FONT_HERSHEY_SIMPLEX
15    while(cap.isOpened() and i<t*N):
16        # Capture frame-by-frame
17        ret, frame = cap.read()
18
19        if ret == True:
20            i += 1
21
22        if i< N:
23            g = frame[:, :, 1]
24
25            # computing the mean
26
27        else:
28            if i%N == 0:
29                select = dynamicOnly(prevFrame, frame,cutoff)
30
31            g = frame[:, :, 1][select]
32            #print(g.shape)
33
34
35            g_mean = np.mean(g)
36            green.append(g_mean)
37            prevFrame = frame
38
39
40        # Break the Loop
41        else:
42            break
43
44    # When everything done, release the video capture object
45    cap.release()
46
47    return majorFreq(cleanSignal(green,w0), fps,w1,w2)
```

In [91]:

```
1 dynamicTS_5_1 = BPM_better_ts('EE18B036_1.mp4',5,cutoff = 10,w0 = 7)
2 dynamicTS_5_2 = BPM_better_ts('EE18B036_2.mp4',5,cutoff = 10,w0 = 7)
3 dynamicTS_5_3 = BPM_better_ts('EE18B036_3.mp4',5,cutoff = 10,w0 = 7)
```

In [57]:

```
1 dynamicDFT_5_1 = BPM_better_dft('EE18B036_1.mp4',5,cutoff = 20, w0 = 10)
2 dynamicDFT_5_2 = BPM_better_dft('EE18B036_2.mp4',5,cutoff = 20, w0 = 10)
3 dynamicDFT_5_3 = BPM_better_dft('EE18B036_3.mp4',5,cutoff = 20, w0 = 10)
```

In [92]:

```
1 # assign data
2 fiveSecCroppedData = [[ "Time Series", dynamicTS_5_1, dynamicTS_5_2, dynamicTS_5_3], [ "Time Series", dynamicDFT_5_1, dynamicDFT_5_2, dynamicDFT_5_3]]
3
4 # create header
5 head = [ "Method", "Resting", "Walking", "Workout"]
6
7 # display table
8 print('in the first 5 seconds, when estimated with dynamic pixels only:')
9 print(tabulate(fiveSecCroppedData, headers=head, tablefmt="grid"))
```

in the first 5 seconds, when estimated with dynamic pixels only:

Method	Resting	Walking	Workout
Time Series	64.4371	90.0258	90.022
DFT	68.8981	82.5071	110.005

Cropping the inactive pixels out is best suiting to the DFT method, as can be observed from above table and below plots

In [59]:

```
1 #Plot the results for the performance of dynamic cropping
2 def plotClean(videoName, t = 5, w0 = 7, w1 = 7, w2 = 2, minPulse = 50, maxPulse = 200):
3     green_clean = []; green_norm = []
4     cap = cv2.VideoCapture(videoName)
5     bpm = 0
6     fps = cap.get(cv2.CAP_PROP_FPS)
7     N = int(fps)
8     #result = cv2.VideoWriter('2_bpm.mp4', cv2.VideoWriter_fourcc(*'MP4V'),fps,(1920,1600))
9     # Read until video is completed
10    i = 0
11    #print(cap.get(3))
12    prevFrame = np.zeros((int(cap.get(4)),int(cap.get(3)),3))
13    #print(prevFrame.shape)
14    font = cv2.FONT_HERSHEY_SIMPLEX
15    while(cap.isOpened() and i<t*N):
16        # Capture frame-by-frame
17        ret, frame = cap.read()
18        #print(frame.dtype, frame.shape)
19        if ret == True:
20            i += 1
21
22        if i< N:
23            g_clean = frame[:, :, 1]
24
25            # computing the mean
26
27        else:
28            if i%N == 0:
29                select = dynamicOnly(prevFrame, frame,20)
30
31            g_clean = frame[:, :, 1][select]
32            #print(g.shape)
33
34
35
36            g_mean_clean = np.mean(g_clean)
37            green_clean.append(g_mean_clean)
38
39            g_norm = frame[:, :, 1]
40            g_mean_norm = np.mean(g_norm)
41            green_norm.append(g_mean_norm)
42
43            prevFrame = frame
44
45        # Break the Loop
46        else:
47            break
48
49    # When everything done, release the video capture object
50    cap.release()
51    ts = [i/fps for i in range(len(green_clean))]
52
53    plt.figure(0,figsize = (20,10))
54    plt.plot(ts[6:],cleanSignal(green_clean,7),label = 'after cropping')
55    plt.plot(ts[6:],cleanSignal(green_norm,7),label = 'no crop')
56    plt.legend()
57    plt.xlabel('time in sec')
58    plt.ylabel('FIM value')
59
```

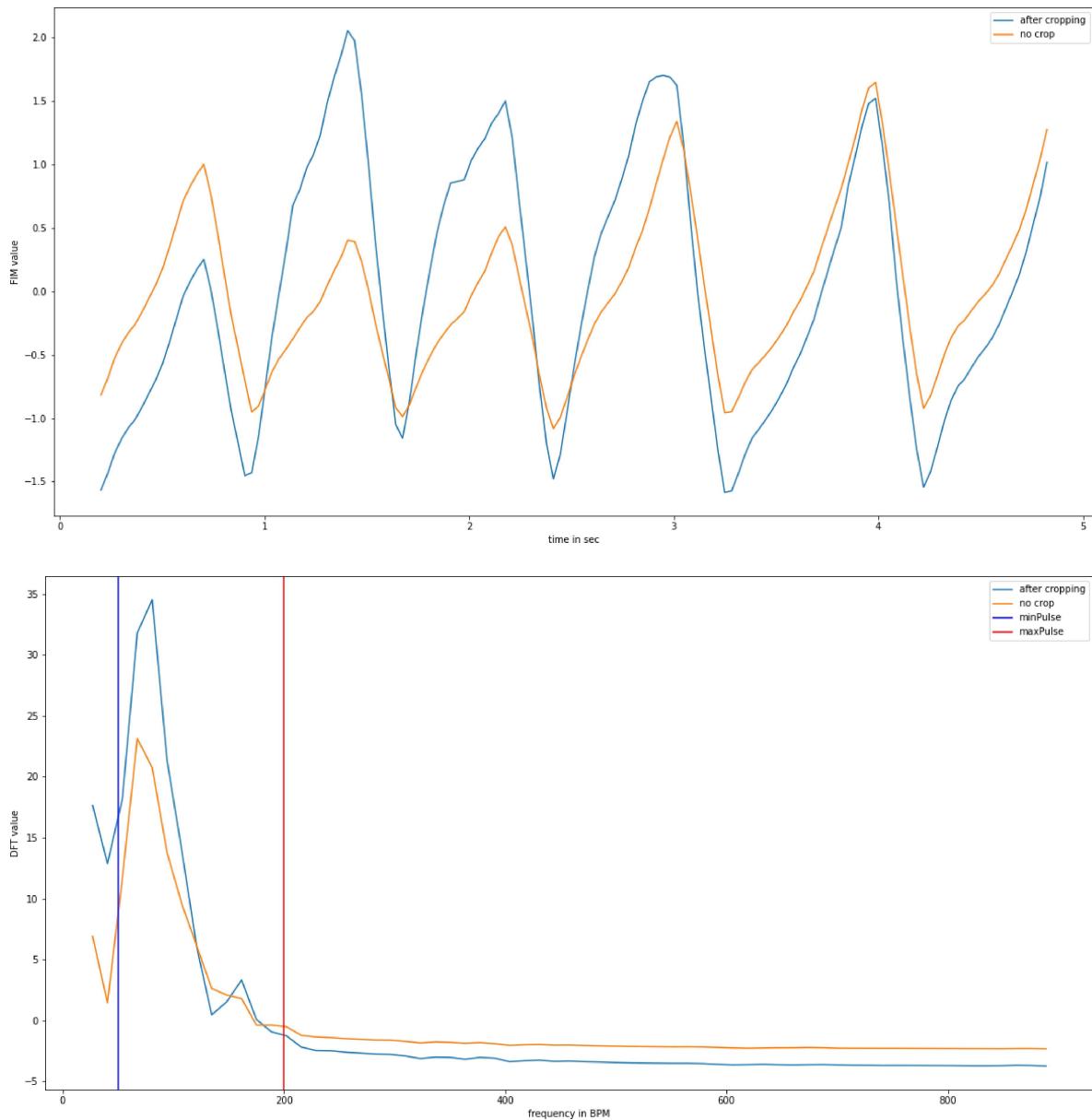
```

60 W1, Y1 = dft_util(cleanSignal(green_clean,w0),w1)
61 W2, Y2 = dft_util(cleanSignal(green_norm,w0),w1)
62
63 plt.figure(1, figsize = (20,10))
64 plt.plot(W1[1:]*60*fps,cleanSignal(Y1,w2),label = 'after cropping')
65 plt.plot(W2[1:]*60*fps,cleanSignal(Y2,w2),label = 'no crop')
66 plt.axvline(x = minPulse,color = 'b',label = 'minPulse')
67 plt.axvline(x = maxPulse,color = 'r',label = 'maxPulse')
68 plt.xlabel('frequency in BPM')
69 plt.ylabel('DFT value')
70 plt.legend()

```

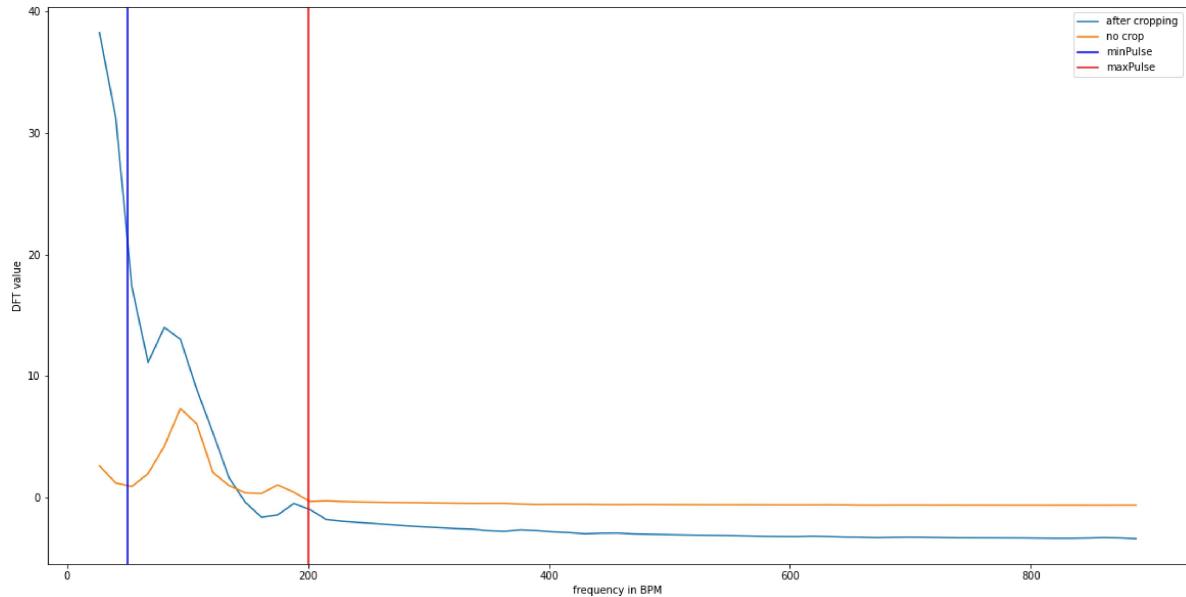
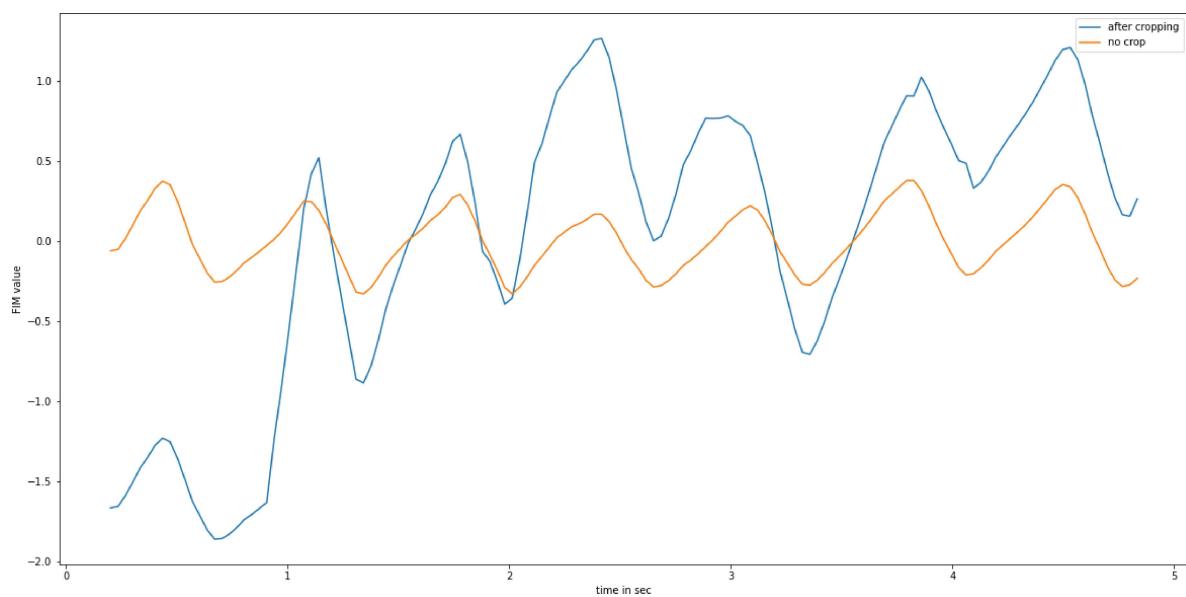
In [60]:

```
1 plotClean('EE18B036_1.mp4')
```



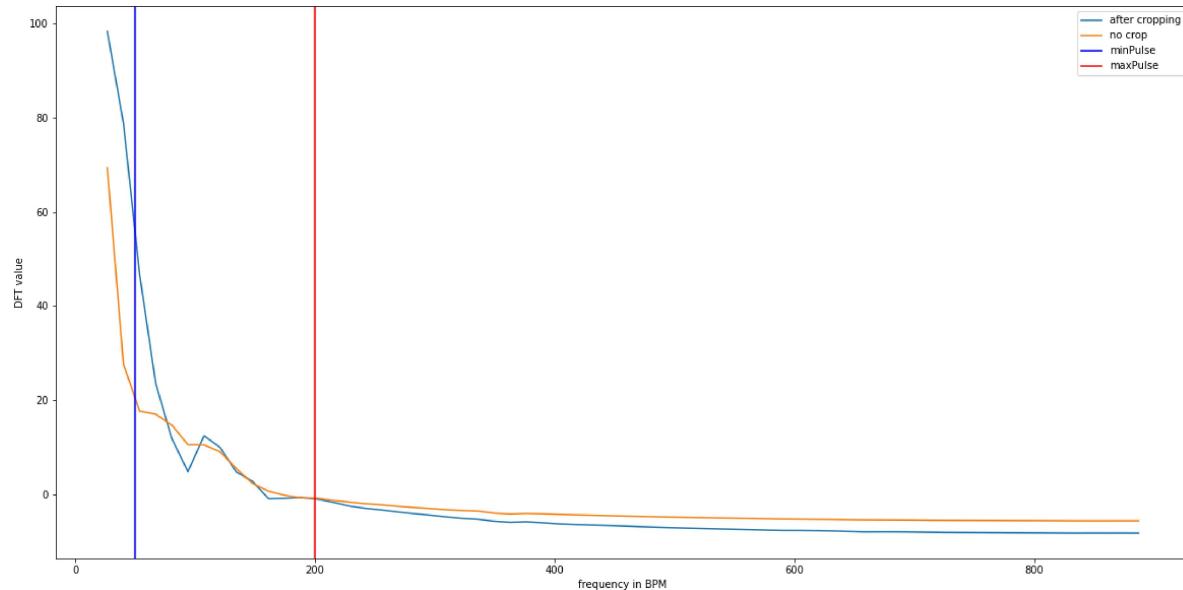
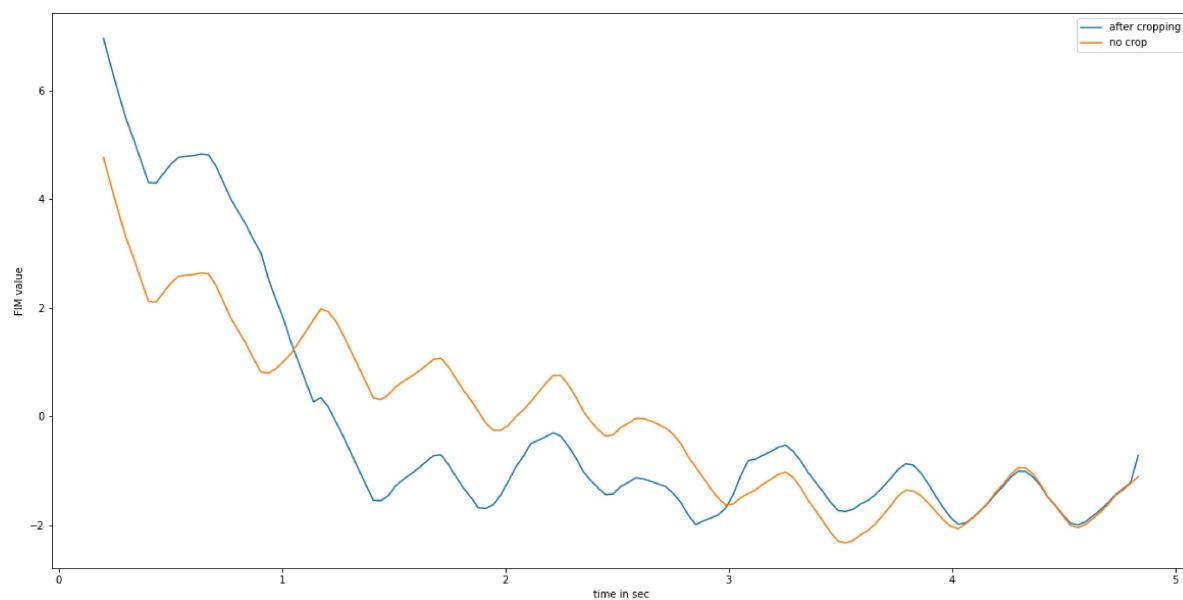
In [61]:

```
1 plotClean('EE18B036_2.mp4')
```



In [62]:

```
1 plotClean('EE18B036_3.mp4')
```



In []:

```
1
```