



New chat



# Apply a bandpass filter to the signal buffer to remove noise and extract the heart rate frequency

```
b, a = signal.butter(3, [0.7 / (fps / 2), 3.5 / (fps / 2)], btype='band')
filtered_signal = signal.filtfilt(b, a, signal_buffer)
peak_indices = signal.find_peaks(filtered_signal, height=0.1)[0]
heart_rate = len(peak_indices) * 60 / buffer_size * fps
```

```
# Display the frame with the ROI rectangle and heart rate information
cv2.putText(frame, f"Heart rate: {int(heart_rate)} BPM", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, roi_color, 2)
cv2.imshow('frame', frame)
```



This code snippet applies a bandpass filter to the signal buffer to remove noise and extract the heart rate frequency. It then finds the peak indices in the filtered signal and calculates the heart rate based on the number of peaks detected. Finally, it displays the frame with the region of interest (ROI) rectangle and heart rate information.

The bandpass filter used is a Butterworth filter of order 3, with cutoff frequencies of 0.7 Hz and 3.5 Hz, normalized by the sampling frequency (fps/2). The filter type is set to 'band' to allow only frequencies within the specified range to pass through and reject other frequencies outside this range.

The filtered signal is obtained by applying the filter twice in a forward-backward manner using the `filtfilt` function to eliminate any phase distortion. The `find_peaks` function from the `signal` module is then used to detect the peaks in the filtered signal above a certain threshold (`height=0.1`).

The heart rate is calculated as the number of detected peaks divided by the buffer size, then multiplied by 60 (to convert to BPM) and the sampling frequency (fps).

Finally, the heart rate is displayed on the frame using OpenCV's `putText` function, with the ROI color used as the text color. The frame is then displayed using OpenCV's `imshow` function

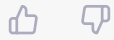




New chat



Finally, the heart rate is displayed on the frame using OpenCV's putText function, with the ROI color used as the text color. The frame is then displayed using OpenCV's imshow function.

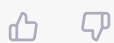


```
face_cascade = cv2.CascadeClassifier(cv2.data.harcascades +  
'haarcascade_frontalface_default.xml')  
roi_color = (0, 255, 0) # green  
roi_top = 150  
roi_bottom = 350  
roi_left = 150  
roi_right = 450
```



This code snippet initializes a Haar Cascade classifier to detect faces in an image. The classifier is loaded from the XML file 'haarcascade\_frontalface\_default.xml' which is included in the OpenCV data folder.

The ROI (region of interest) color is set to green (0, 255, 0). The top, bottom, left, and right coordinates of the ROI are specified as 150, 350, 150, and 450, respectively. These coordinates define the rectangular region of interest where the heart rate will be detected.



```
# Define the sampling frequency and buffer size for the heart rate signal  
fps = 30  
buffer_size = fps * 5 # 5 seconds of data  
signal_buffer = np.zeros(buffer_size)  
signal_index = 0
```





New chat



included in the OpenCV data folder.

The ROI (region of interest) color is set to green (0, 255, 0). The top, bottom, left, and right coordinates of the ROI are specified as 150, 350, 150, and 450, respectively. These coordinates define the rectangular region of interest where the heart rate will be detected.



```
# Define the sampling frequency and buffer size for the heart rate signal  
fps = 30  
buffer_size = fps * 5 # 5 seconds of data  
signal_buffer = np.zeros(buffer_size)  
signal_index = 0
```



This code snippet defines the sampling frequency and buffer size for the heart rate signal.

The sampling frequency (fps) is set to 30 Hz, which means that 30 samples will be taken per second.

The buffer size is set to 5 seconds of data, which means that the signal buffer will store  $5 \times 30 = 150$  samples at a time.

A numpy array of zeros is initialized as the signal buffer with the length of buffer\_size. signal\_index is also initialized as 0, which will be used to keep track of the current position in the buffer.



WHAT IS BUFFER SIZE





New chat



signal\_index is also initialized as 0, which will be used to keep track of the current position in the buffer.



WHAT IS BUFFER SIZE



Buffer size refers to the number of data samples that are stored in a buffer at any given time. In the context of digital signal processing, a buffer is a temporary storage area in memory where incoming signal samples are stored before processing.

The size of the buffer determines how much data can be processed at a time. A larger buffer size allows for more data to be processed in one go, which can result in more accurate analysis of the signal. However, a larger buffer size can also result in increased latency, which can be a problem for real-time applications.

In the code snippet provided, the buffer size is set to 5 seconds of data. This means that 150 samples (30 samples per second \* 5 seconds) will be stored in the signal buffer at any given time. The heart rate will be calculated based on these 150 samples.



```
# Define a function to process the ROI and extract the average pixel intensity
def process_roi(roi):
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    return np.mean(gray)
```



This code snippet defines a function named `process_roi` that takes an ROI (region of interest) image as input and returns the average pixel intensity of the grayscale version of the image.



The ROI image is first converted to grayscale using OpenCV's `cvtColor` function with the





New chat



This code snippet defines a function named `process_roi` that takes an ROI (region of interest) image as input and returns the average pixel intensity of the grayscale version of the image.

The ROI image is first converted to grayscale using OpenCV's `cvtColor` function with the `COLOR_BGR2GRAY` flag. This converts the ROI image from BGR (Blue-Green-Red) color space to grayscale, which is a single-channel image.

The `mean` function from NumPy is then used to calculate the average pixel intensity of the grayscale image. This value is returned by the function. The average pixel intensity can be used as a measure of the blood flow in the region of interest, which can be used to estimate the heart rate.



while True:

```
# Read a frame from the camera
ret, frame = cap.read()
```

```
# Convert the frame to grayscale
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
# Detect faces in the grayscale frame
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30), flags=cv2.CASCADE_SCALE_IMAGE)
```

```
# Draw a rectangle around the faces and extract the ROI for heart rate measurement
for (x, y, w, h) in faces:
```

```
    cv2.rectangle(frame, (x, y), (x + w, y + h), roi_color, 2)
    roi = frame[y+roi_top:y+roi_bottom, x+roi_left:x+roi_right]
```

```
# Process the ROI and add the result to the signal buffer
intensity = process_roi(roi)
signal_buffer[signal_index] = intensity
```





New chat



```
# Process the ROI and add the result to the signal buffer
intensity = process_roi(roi)
signal_buffer[signal_index] = intensity
signal_index = (signal_index + 1) % buffer_size
```



This code snippet reads frames from a video camera and detects faces in the frames using a Haar Cascade classifier. It draws a rectangle around the detected faces and extracts the region of interest (ROI) for heart rate measurement.

The code uses a loop to continuously read frames from the camera. Each frame is first converted to grayscale using `cv2.cvtColor` function. The Haar Cascade classifier `face_cascade` is then used to detect faces in the grayscale frame. The `detectMultiScale` function is used for this purpose, which returns a list of rectangles that bound the detected faces.

A rectangle is then drawn around each detected face using the `cv2.rectangle` function. The ROI for heart rate measurement is extracted by selecting a rectangular region inside the face rectangle using the coordinates of the ROI (specified in the earlier code snippet).

The `process_roi` function is called to process the ROI and extract the average pixel intensity. This value is added to the signal buffer at the current position (`signal_index`) and `signal_index` is incremented. If `signal_index` exceeds the buffer size, it is reset to 0 using the modulo operator (%). This allows the buffer to store a continuous stream of data by overwriting the oldest data with the newest.



# Apply a bandpass filter to the signal buffer to remove noise and extract the heart rate frequency

```
b, a = signal.butter(3, [0.7 / (fps / 2), 3.5 / (fps / 2)], btype='band')
filtered_signal = signal.filtfilt(b, a, signal_buffer)
peak_indices = signal.find_peaks(filtered_signal, height=0.1)[0]
heart_rate = len(peak_indices) * 60 / buffer_size * fps
```





New chat



# Apply a bandpass filter to the signal buffer to remove noise and extract the heart rate frequency

```
b, a = signal.butter(3, [0.7 / (fps / 2), 3.5 / (fps / 2)], btype='band')
filtered_signal = signal.filtfilt(b, a, signal_buffer)
peak_indices = signal.find_peaks(filtered_signal, height=0.1)[0]
heart_rate = len(peak_indices) * 60 / buffer_size * fps
```

```
# Display the frame with the ROI rectangle and heart rate information
cv2.putText(frame, f"Heart rate: {int(heart_rate)} BPM", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, roi_color, 2)
cv2.imshow('frame', frame)
```



This code snippet applies a bandpass filter to the signal buffer to remove noise and extract the heart rate frequency. It then calculates the heart rate and displays it on the frame along with the ROI rectangle.

The `signal.butter` function from the SciPy signal processing library is used to design a Butterworth filter. A Butterworth filter is a type of low-pass filter that has a flat frequency response in the passband and rolls off towards zero in the stopband. The filter is designed using a 3rd-order Butterworth filter design and a bandpass filter type. The passband frequencies are set to 0.7 Hz and 3.5 Hz, which are suitable for heart rate measurement.

The `signal.filtfilt` function is then used to apply the filter to the signal buffer. This function applies a filter to a signal in both the forward and reverse directions to eliminate phase distortion. This results in a filtered signal that has a smoother frequency response and reduced noise.

The `signal.find_peaks` function is then used to detect the peaks in the filtered signal. This function returns a tuple containing the indices of the peak locations and their corresponding peak heights. We are only interested in the peak indices, so we extract them using `[0]` indexing.





New chat



The `signal.filtfilt` function is then used to apply the filter to the signal buffer. This function applies a filter to a signal in both the forward and reverse directions to eliminate phase distortion. This results in a filtered signal that has a smoother frequency response and reduced noise.

The `signal.find_peaks` function is then used to detect the peaks in the filtered signal. This function returns a tuple containing the indices of the peak locations and their corresponding peak heights. We are only interested in the peak indices, so we extract them using `[0]` indexing.

The heart rate is then calculated by dividing the number of peaks by the buffer size and multiplying by the sampling frequency and 60 to convert to beats per minute. The heart rate is then displayed on the frame using the `cv2.putText` function. The font, font scale, and color are specified using the arguments `cv2.FONT_HERSHEY_SIMPLEX`, `1`, and `roi_color`, respectively. Finally, the frame is displayed using the `cv2.imshow` function.



```
import cv2
import numpy as np
import scipy.signal as signal
```

```
# Define the region of interest (ROI) on the face where heart rate will be measured
face_cascade = cv2.CascadeClassifier(cv2.data.harcascades +
'haarcascade_frontalface_default.xml')
roi_color = (0, 255, 0) # green
roi_top = 150
roi_bottom = 350
roi_left = 150
roi_right = 450
```

```
# Set up the video capture device (use 0 for the default camera)
cap = cv2.VideoCapture(0)
```

```
# Define the sampling frequency and buffer size for the heart rate signal
fps = 30
```







New chat



```
cv2.FONT_HERSHEY_SIMPLEX, 1, roi_color, 2)
    cv2.imshow('frame', frame)

    # Exit on 'q' key
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Clean up resources
cap.release()
cv2.destroyAllWindows()
```



This is a Python script that uses OpenCV and SciPy libraries to measure heart rate from a video stream captured from a camera.

The script starts by defining the region of interest (ROI) on the face where heart rate will be measured. It then sets up the video capture device and defines the sampling frequency and buffer size for the heart rate signal.

In the main loop, the script reads a frame from the camera, converts it to grayscale, and detects faces in the grayscale frame using a Haar cascade classifier. It then draws a rectangle around the faces and extracts the ROI for heart rate measurement.

The script then processes the ROI and adds the result to the signal buffer. It applies a bandpass filter to the signal buffer to remove noise and extract the heart rate frequency. It then finds the peak indices in the filtered signal and calculates the heart rate in beats per minute (BPM).

Finally, the script displays the frame with the ROI rectangle and heart rate information. It exits when the 'q' key is pressed.

Overall, this script provides a simple way to measure heart rate from a video stream using computer vision techniques.

