**REMOTE PHOTOPLETHYSMOGRAPHY**

**NAME**: **MATRICULATION NUMBER**:

VENKATA SAI TARAK PADARTHI 1536988

**MOTIVATION:**

Remote Photoplethysmography signals are used for analysis. The main motivation of the project is to clearly understand which methods are better suitable and try to get Heartrate measurement. For this purpose we use ground truth values obtained from Pulse oximeter which is based on Photoplethysmography sensor. We can also measure Heartrate using ECG and taken as ground truth value. We can proceed using face detection as first step where in we can detect skin pixels and choose Region of Interest(ROI) where it is tracked and certain set of pixels are averaged out in 3 different channels namely Red, Green and Blue channels. This is then converted to RPPG signal. Different datasets are already used for heartrate measurement in which they considered with different specifications. Datasets already used for research are given below:

PURE,LGI,UBFC,MAHNOB,COHFACE,vicarPPG-2,EatingSet rPPG datasets.

Methods found after research:

Independent Component Analysis Method(ICA), Plane Orthogonal to skin tone(POS),CHROM method e.t.c.

Two approaches which seems to be promising and these two approaches are shown in the figures below.

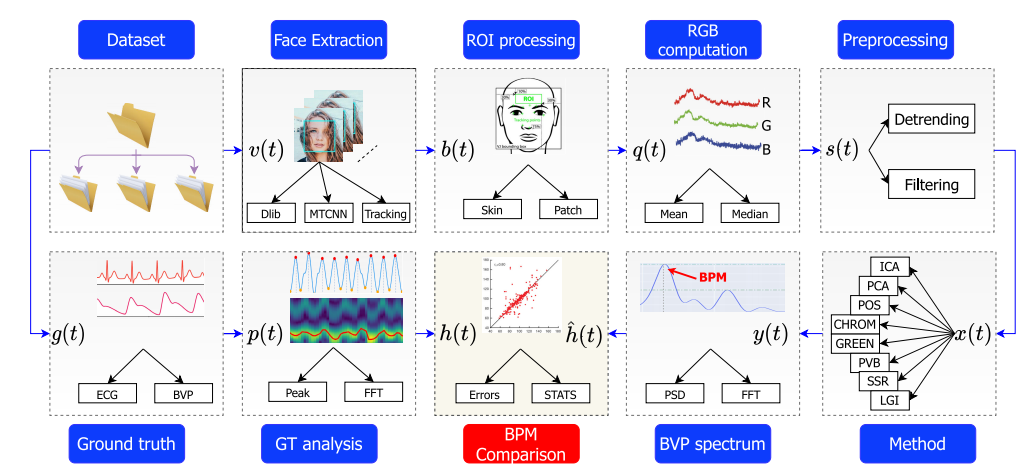


Fig1: Functional architecture of pyVHR framework

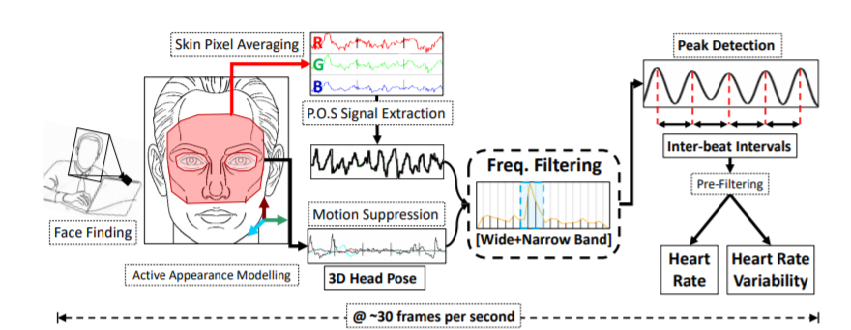


Fig2: Heart rate and Heart rate variability Estimation pipeline

**SELECTION OF DATASET:**

Datasets are selected based on availability from Research papers. Out of these papers some are listed private which cannot be issued to public in any case. But some of them are available for research.

|  |  |
| --- | --- |
| **Datsets** | **Availabilty** |
| StableSet rPPG dataset | Private |
| VicarPPG-2 dataset | Public(can be obtained on request) |
| EatingSet rPPG dataset | Private |
| CleanerPPG ground truth dataset | Public(can be obtained on request) |

ClaenerPPG ground truth dataset is used as ground truth value.

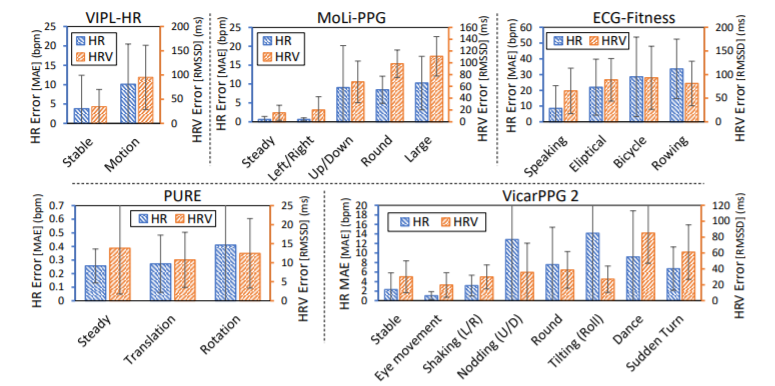


Fig3: rPPG performance for different subject movements

From the above graphs and availability of datasets we can choose VicarPPG-2 as our dataset and proceed further. The only drawback is for the case of vertical movement of head accuracy is low. But for horizontal movement of head it still gives a very good accuracy. Our case is for stable position of head or little or no movement of head. So for our case it works good enough and can be chosen.

**FACE TRACKING:**

I have applied 2 methods and tried to use the best one out of these 2 methods. The different methods has to be looked upon before the explanation of the chosen methods. The different algorithms are shown below:

**Tracker APIs:**

These are the trackers which are available for the OPENCV and are selected based on our choice and requirement.

**BOOSTING TRACKER:**

This tracker is slow. It doesn’t work well.

Pros:Well known algorithm.

Cons:Can’t find when tracking has failed.

**MIL TRACKER:**

Pros: This performs good and doesn’t drift as much as Boosting does.

Cons: It can’t comeback after full obstruction.

Once object tracking fails it won’t report.

**TLD TRACKER:**

This is called as Tracking, Learning and Detection tracker. This learning estimates the errors of detectors and can be avoided in future by updating.

Pros: Even in obstruction of frames tracking will be good.

Cons: Can provide false positives.

**MEDIANFLOW TRACKER:**

This tracker will track objects both in forward and backward directions in time and can measure all the discrepancies present between 2 trajectories.

Pros: It is very good in case of reporting failed tracking.

Cons: In case of fast moving objects it fails.

I have chosen MIL Tracker for this case of steadiness analysis. It works good enough for our case. The code for MIL Tracker is shown below.

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
  
tracker = cv2.TrackerMIL\_create() # Choose MIL Tracker   
cap = cv2.VideoCapture(0) # Capture webcam video  
ret,frame = cap.read()  
roi = cv2.selectROI(frame,False) # Select Region of Interest  
ret = tracker.init(frame,roi)   
while cap.isOpened():  
 ret,frame = cap.read() # Read Captured frame  
 success,roi = tracker.update(frame) # Update tracking position in every frame  
 (x,y,w,h) = tuple(map(int,roi)) # Coordinates with width and height  
 if success:  
 p1 = (x,y)   
 p2 = (x+w,y+h)  
 cv2.rectangle(frame,p1,p2,(0,255,0),3) # Draw rectangle with above chosen points  
 rgb = cv2.cvtColor(frame[p1[1]:p2[1],p1[0]:p2[0]],cv2.COLOR\_BGR2RGB)  
 r = rgb[:,:,0] # Red channel  
 g = rgb[:,:,1] # Green channel  
 b = rgb[:,:,2] # Blue channel  
 cv2.imshow("frame1",np.hstack([r,g,b])) # Stack all channels together  
 plt.plot(r)  
 plt.show()  
 else:  
 cv2.putText(frame,"Failure to detect Tracking",(100,200),cv2.FONT\_HERSHEY\_SIMPLEX,1,(0,255,0),3)  
 cv2.imshow('tracker',frame)  
 k = cv2.waitKey(1) & 0xFF  
 if k == 27:  
 break  
cap.release() # Capture release  
cv2.destroyAllWindows()

Fig4: Code for MIL Tracker

The code is used with MIL Tracker as choice and Red, Green, Blue channels are also considered.

**MEDIAPIPE:**

Mediapipe is another set of Machine learning algorithms. This is developed by Google which works very well in real world scenarios. But the problem is face detections are done only till eyebrows region. But can be chosen according to our interest by giving dimensions for region of interest. I used Mediapipe Face detection algorithm. The code is placed below with Red, Green , Blue channels under consideration.

import cv2  
import matplotlib.pyplot as plt  
import numpy as np  
import mediapipe as mp  
mp\_face\_detection = mp.solutions.face\_detection  
mp\_drawing = mp.solutions.drawing\_utils  
  
cap = cv2.VideoCapture(0)  
with mp\_face\_detection.FaceDetection(  
 min\_detection\_confidence=0.5) as face\_detection: # Confidence level in detection of face  
 while cap.isOpened(): # While Capturing is happening  
 success, image = cap.read()  
 if not success:  
 print("Ignoring empty camera frame.")  
 # If loading a video, use 'break' instead of 'continue'.  
 continue  
 # Flip the image horizontally for a later selfie-view display, and convert  
 # the BGR image to RGB.  
 image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB) # Convert color from BGR to RGB  
 # To improve performance, optionally mark the image as not writeable to  
 # pass by reference.  
 image.flags.writeable = False  
 results = face\_detection.process(image) # Processing Image  
 # Draw the face detection annotations on the image.  
 image.flags.writeable = True  
 image = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR)  
 if results.detections:  
 c = results.detections[0].location\_data.relative\_bounding\_box.width # Reading Width value  
 d = results.detections[0].location\_data.relative\_bounding\_box.height # Reading Height value  
 a = results.detections[0].location\_data.relative\_bounding\_box.xmin # Reading X coordinate  
 b = results.detections[0].location\_data.relative\_bounding\_box.ymin # Reading Y coordinate  
 for detection in results.detections: # For each detection  
 s,t,u = image.shape # Reading height, width and Channels  
 list = [] # Create Empty list  
 mp\_drawing.draw\_detection(image,detection) # Draw detection regions  
 (l,m,n,o) = a\*t,b\*s,a\*t+c\*t,b\*s+d\*s  
 list.append([l,m,n,o])  
 image = image[int(m):int(o),int(l):int(n)] # Select Region of Interest  
 print(image.shape)  
 red = image[:, :, 0] # Red Channel  
 green = image[:, :, 1] # Green Channel  
 blue = image[:, :, 2] # Blue Channel  
 plt.plot(red)  
 plt.show()  
 cv2.imshow("frame",np.hstack([red,green,blue])) # Stack Red, Green and Blue channels  
 cv2.imshow('MediaPipe Face Detection', image)  
 if cv2.waitKey(1) & 0xFF == 27:  
 break  
cap.release() # Release Capture  
cv2.destroyAllWindows()

Fig5: Code for Mediapipe Face detection

The above code is taken as reference from Mediapipe and made changes.

Other methods also include Camshift and Optical flow trackers.

**COMPARISION OF RESULTS OF TRACKING:**

Different methods are applied. The results obtained from MediaPipe is promising and more reliable because of the stabilization it has. The tracking methods tends to show less promising results because of accuracy or performance. Some tracking methods are good at accuracy but not at performance and some other tracking methods are good at performance and bad at accuracy. So, due to these reasons I have chosen MediaPipe Facemesh as the choice of preference. For the same the code is shown further down.

**Region of Interest for FACEMESH:**

Facemesh code is taken from MediaPipe and used for our case in the way we wanted. The tracking part is done by selecting the points and drawing contours in the region of interests chosen. The tracking is done at first and then the approach of masking with original image is done by blacking out the regions which are out of region of interests. In the foreground we can notice the region of interests on face and in the background we see black colored pixels. The Region of Interests are chosen as forehead region, Right cheeks and Left cheeks. So, for all these regions we consider and obtain towards a conclusion at the end of the project, for which region the better results are obtained has to be analysed.

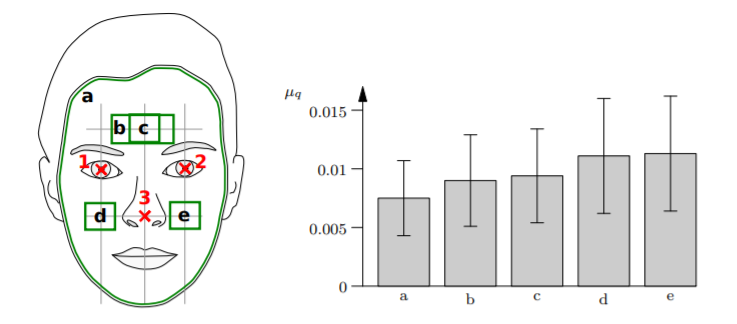


Fig6: Different regions of interests on X axis along with standard deviation and Mean signal quality taken on Y axis

import cv2  
import mediapipe as mp  
import numpy as np  
import matplotlib.pyplot as plt  
import datetime as dt  
mp\_drawing = mp.solutions.drawing\_utils  
mp\_face\_mesh = mp.solutions.face\_mesh  
  
drawing\_spec = mp\_drawing.DrawingSpec(thickness=1, circle\_radius=1)  
cap = cv2.VideoCapture(0)  
# Creating empty lists for storing mean values of all channels  
red\_forehead\_mean\_values = []  
green\_forehead\_mean\_values = []  
blue\_forehead\_mean\_values = []  
red\_right\_cheek\_mean\_values = []  
green\_right\_cheek\_mean\_values = []  
blue\_right\_cheek\_mean\_values = []  
red\_left\_cheek\_mean\_values = []  
green\_left\_cheek\_mean\_values = []  
blue\_left\_cheek\_mean\_values = []  
time = []  
with mp\_face\_mesh.FaceMesh(  
 min\_detection\_confidence=0.5,  
 min\_tracking\_confidence=0.5) as face\_mesh:  
 while cap.isOpened():  
 success, image = cap.read()  
 if not success:  
 print("Ignoring empty camera frame.")  
 # If loading a video, use 'break' instead of 'continue'.  
 continue  
 height,width,\_ = image.shape  
 # Flip the image horizontally for a later selfie-view display, and convert  
 # the BGR image to RGB.  
 image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR\_BGR2RGB)  
 # To improve performance, optionally mark the image as not writeable to  
 # pass by reference.  
 image.flags.writeable = False  
 results = face\_mesh.process(image)  
 # Draw the face mesh annotations on the image.  
 image.flags.writeable = True  
 # Convert the RGB image to BGR.  
 image = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR)  
 # Storing all the points data  
 forehead = [10,338,297,332,284,298,333,151,108,69,104,103,67,109,68,54,337,299,9]  
 left\_cheek = [165,203,36,101,50,147,132,58,172,186,92,165,177,215,138,213,215,186,187,206,216,205,207,192]  
 right\_cheek = [391,423,266,330,280,401,361,288,397,410,322,426,425,427,436,416,433,411,376]  
 forehead\_cheeks = []  
 forehead\_cheeks.extend(forehead)  
 forehead\_cheeks.extend(right\_cheek)  
 forehead\_cheeks.extend(left\_cheek)  
 # Connecting lines using these points  
 forehead\_line = [284,298,9,68,54,103,67,109,10,338,297,332,284]  
 right\_cheek\_line = [391,423,266,330,280,401,361,288,397,410,322,391]  
 left\_cheek\_line =[165,203,36,101,50,147,132,58,172,186,92,165]  
 forehead\_line\_xy = []  
 left\_cheek\_line\_xy = []  
 right\_cheek\_line\_xy = []  
 if results.multi\_face\_landmarks:  
 a = int((results.multi\_face\_landmarks[0].landmark[54].x) \* width)  
 b = int((results.multi\_face\_landmarks[0].landmark[284].x) \* width)  
 c = int((results.multi\_face\_landmarks[0].landmark[10].y) \* height)  
 d = int((results.multi\_face\_landmarks[0].landmark[8].y) \* height)  
 e = int((results.multi\_face\_landmarks[0].landmark[391].x) \* width)  
 f = int((results.multi\_face\_landmarks[0].landmark[361].x) \* width)  
 g = int((results.multi\_face\_landmarks[0].landmark[330].y) \* height)  
 h = int((results.multi\_face\_landmarks[0].landmark[397].y) \* height)  
 m = int((results.multi\_face\_landmarks[0].landmark[132].x) \* width)  
 n = int((results.multi\_face\_landmarks[0].landmark[165].x) \* width)  
 o = int((results.multi\_face\_landmarks[0].landmark[101].y) \* height)  
 p = int((results.multi\_face\_landmarks[0].landmark[172].y) \* height)  
 im = image.copy()  
 for face\_landmarks in results.multi\_face\_landmarks:  
 for i in forehead\_cheeks:  
 forehead\_cheeks\_x = int((face\_landmarks.landmark[i].x) \* width)  
 forehead\_cheeks\_y = int((face\_landmarks.landmark[i].y) \* height)  
 cv2.circle(im, (forehead\_cheeks\_x, forehead\_cheeks\_y), 2, (0, 255, 0), -1)  
 #cv2.putText(im,str(i),(forehead\_cheeks\_x,forehead\_cheeks\_y),0,0.2,(0,0,255))  
 for i in forehead\_line:  
 # Taking XY coordinate points on forehead  
 forehead\_line\_x = int((face\_landmarks.landmark[i].x) \* width)  
 forehead\_line\_y = int((face\_landmarks.landmark[i].y) \* height)  
 forehead\_line\_xy.append((forehead\_line\_x, forehead\_line\_y))  
 for point1, point2 in zip(forehead\_line\_xy, forehead\_line\_xy[1:]):  
 cv2.line(im, point1, point2, [0, 255, 0], 1)  
 for i in left\_cheek\_line:  
 # Taking XY coordinate points on left cheeks  
 left\_cheek\_line\_x = int((face\_landmarks.landmark[i].x) \* width)  
 left\_cheek\_line\_y = int((face\_landmarks.landmark[i].y) \* height)  
 left\_cheek\_line\_xy.append((left\_cheek\_line\_x, left\_cheek\_line\_y))  
 for point1, point2 in zip(left\_cheek\_line\_xy, left\_cheek\_line\_xy[1:]):  
 cv2.line(im, point1, point2, [0, 255, 0], 1)  
 for i in right\_cheek\_line:  
 # Taking XY coordinate points on right cheeks  
 right\_cheek\_line\_x = int((face\_landmarks.landmark[i].x) \* width)  
 right\_cheek\_line\_y = int((face\_landmarks.landmark[i].y) \* height)  
 right\_cheek\_line\_xy.append((right\_cheek\_line\_x, right\_cheek\_line\_y))  
 for point1, point2 in zip(right\_cheek\_line\_xy, right\_cheek\_line\_xy[1:]):  
 cv2.line(im, point1, point2, [0, 255, 0], 1)  
 # Collecting the contours at all the region of interests  
 forehead\_pts = np.array([forehead\_line\_xy], dtype=np.int32)  
 right\_cheek\_pts = np.array([right\_cheek\_line\_xy], dtype=np.int32)  
 left\_cheek\_pts = np.array([left\_cheek\_line\_xy], dtype=np.int32)  
 # Creating mask with zeros in array  
 forehead\_mask = np.zeros(image.shape[:2], np.int8)  
 right\_cheek\_mask = np.zeros(image.shape[:2], np.int8)  
 left\_cheek\_mask = np.zeros(image.shape[:2], np.int8)  
 # The regions are filled with the contours taken and filled with white color  
 cv2.fillPoly(forehead\_mask, [forehead\_pts], 255)  
 cv2.fillPoly(right\_cheek\_mask,[right\_cheek\_pts],255)  
 cv2.fillPoly(left\_cheek\_mask,[left\_cheek\_pts],255)  
 # Take the values of pixels that are not with the value of zero  
 maskimage\_forehead = cv2.inRange(forehead\_mask, 1, 255)  
 maskimage\_right\_cheek = cv2.inRange(right\_cheek\_mask, 1, 255)  
 maskimage\_left\_cheek = cv2.inRange(left\_cheek\_mask, 1, 255)  
 # Results of all the region of interests after combining the original image and masking  
 forehead\_result = cv2.bitwise\_and(image,image,mask = maskimage\_forehead)  
 right\_cheek\_result = cv2.bitwise\_and(image, image, mask=maskimage\_right\_cheek)  
 left\_cheek\_result = cv2.bitwise\_and(image, image, mask=maskimage\_left\_cheek)  
 # Taking mean of all the pixels at the region of interest  
 mean\_forehead = cv2.mean(forehead\_result,mask = maskimage\_forehead)  
 mean\_right\_cheek = cv2.mean(right\_cheek\_result, mask=maskimage\_right\_cheek)  
 mean\_left\_cheek = cv2.mean(left\_cheek\_result, mask=maskimage\_left\_cheek)  
  
 #All region of interest masks for visualisation  
 all\_roi\_mask = np.zeros(image.shape[:2], np.int8)  
 cv2.fillPoly(all\_roi\_mask, [forehead\_pts], 255)  
 cv2.fillPoly(all\_roi\_mask,[right\_cheek\_pts],255)  
 cv2.fillPoly(all\_roi\_mask,[left\_cheek\_pts],255)  
 masking\_entire\_regions = cv2.inRange(all\_roi\_mask, 1, 255)  
 result = cv2.bitwise\_and(image,image,mask = masking\_entire\_regions)  
  
 # All region of interests are taken into consideration  
 ROI1 = forehead\_result[c:d,a:b]  
 ROI2 = right\_cheek\_result[g:h,e:f]  
 ROI3 = left\_cheek\_result[o:p,m:n]  
 # RGB regions for forehead are taken into consideration  
 blue\_forehead = ROI1[:,:,0]  
 green\_forehead = ROI1[:,:,1]  
 red\_forehead = ROI1[:,:,2]  
 # Assign the mean values of forehead region for RGB channels separately  
 blue\_forehead\_mean = mean\_forehead[0]  
 green\_forehead\_mean = mean\_forehead[1]  
 red\_forehead\_mean = mean\_forehead[2]  
 # Append the mean values of forehead region to the list  
 red\_forehead\_mean\_values.append(red\_forehead\_mean)  
 green\_forehead\_mean\_values.append(green\_forehead\_mean)  
 blue\_forehead\_mean\_values.append(blue\_forehead\_mean)  
 # Time is taken to print on X axis  
 time.append(dt.datetime.now().strftime('%H:%M:%S.%f'))  
 # RGB regions for right cheeks are taken into consideration  
 blue\_right\_cheek = ROI2[:,:,0]  
 green\_right\_cheek = ROI2[:,:,1]  
 red\_right\_cheek = ROI2[:,:,2]  
 # Assign the mean values of right cheek region for RGB channels separately  
 blue\_right\_cheek\_mean = mean\_right\_cheek[0]  
 green\_right\_cheek\_mean = mean\_right\_cheek[1]  
 red\_right\_cheek\_mean = mean\_right\_cheek[2]  
 # Append the mean values of right cheek region to the list  
 red\_right\_cheek\_mean\_values.append(red\_right\_cheek\_mean)  
 green\_right\_cheek\_mean\_values.append(green\_right\_cheek\_mean)  
 blue\_right\_cheek\_mean\_values.append(blue\_right\_cheek\_mean)  
 # RGB regions for left cheeks are taken into consideration  
 blue\_left\_cheek = ROI3[:,:,0]  
 green\_left\_cheek = ROI3[:,:,1]  
 red\_left\_cheek = ROI3[:,:,2]  
 # Assign the mean values of left cheek region for RGB channels separately  
 blue\_left\_cheek\_mean = mean\_left\_cheek[0]  
 green\_left\_cheek\_mean = mean\_left\_cheek[1]  
 red\_left\_cheek\_mean = mean\_left\_cheek[2]  
 # Append the mean values of left cheek region to the list  
 red\_left\_cheek\_mean\_values.append(red\_left\_cheek\_mean)  
 green\_left\_cheek\_mean\_values.append(green\_left\_cheek\_mean)  
 blue\_left\_cheek\_mean\_values.append(blue\_left\_cheek\_mean)  
 # Stacking all the regions RGB channels for better visualization  
 forehead\_channels = np.hstack([red\_forehead,green\_forehead,blue\_forehead])  
 right\_cheek\_channels = np.hstack([red\_right\_cheek,green\_right\_cheek,blue\_right\_cheek])  
 left\_cheek\_channels = np.hstack([red\_left\_cheek,green\_left\_cheek,blue\_left\_cheek])  
 # Images are shown and printed for visualization  
 cv2.imshow('MediaPipe FaceMesh', im)  
 cv2.imshow('image', masking\_entire\_regions)  
 cv2.imshow('Masking with image', result)  
 cv2.imshow('Forehead region',forehead\_channels)  
 cv2.imshow('Right cheek region',right\_cheek\_channels)  
 cv2.imshow('Left cheek region',left\_cheek\_channels)  
 # Frame rate can be given in the waitKey command  
 if cv2.waitKey(1) & 0xFF == 27:  
 fig,axis = plt.subplots(1,3)  
 # Plot average RGB signals at all the region of interests  
 plt.subplot(1,3,1)  
 plt.plot(time,red\_forehead\_mean\_values,'r')  
 plt.plot(time,green\_forehead\_mean\_values,'g')  
 plt.plot(time,blue\_forehead\_mean\_values,'b')  
 plt.title("Forehead")  
 plt.xlabel('Time')  
 plt.ylabel('Average RGB channels for forehead region')  
 plt.subplot(1,3,2)  
 plt.plot(time,red\_right\_cheek\_mean\_values,'r')  
 plt.plot(time,green\_right\_cheek\_mean\_values,'g')  
 plt.plot(time,blue\_right\_cheek\_mean\_values,'b')  
 plt.title("Right cheeks")  
 plt.xlabel('Time')  
 plt.ylabel('Average RGB channels for right cheek region')  
 plt.subplot(1,3,3)  
 plt.plot(time,red\_left\_cheek\_mean\_values,'r')  
 plt.plot(time,green\_left\_cheek\_mean\_values,'g')  
 plt.plot(time,blue\_left\_cheek\_mean\_values,'b')  
 plt.title("Left cheeks")  
 plt.xlabel('Time')  
 plt.ylabel('Average RGB channels for left cheek region')  
 # This command below is used for visulization of the plots  
 plt.show()  
 break  
cap.release()  
cv2.destroyAllWindows()

Fig7: Code for Mediapipe Face mesh with region of interests selected and signal extraction

**RESULTS OF REGION OF INTERESTS:**

We plot different results of Region of interests chosen. The masking is done and added with orginal image with bitwise and operator. So, all the results are shown below. The results include contours at all the region of interests chosen. The masking with white color filled at region of interests. The result of final masking along with original image using bitwise and operator. The results also include different region of interests and plots of face.

There are different camera based measuring techniques. Of those one is color based methods and the other is motion based methods. The color based method is rPPG based method which extracts Heart rate from the slight changes in the color of skin due to heartbeat. The motion based method is based on BCG which measures Heart rate from tiny motions of the body accompanying cardiac activity.

****

Fig8: Contours at all the region of interests on face



Fig9: Masking with filling of white color at region of interests



Fig10: Making and original image combined



Fig11: Region of Interests for forehead region in the order of red, green and blue channels

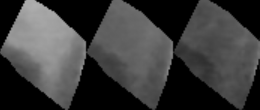


Fig12: Region of Interests for right cheek region in the order of red, green and blue channels

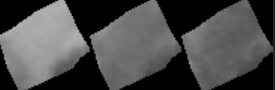


Fig13: Region of Interests for left cheek region in the order of red, green and blue channels

**RGB SIGNAL EXTRACTION:**

As shown in the above code RGB signals are extracted by using averaging method. We have chosen the region of interests and as can be noticed from above, there are black pixels in the region of interest chosen. We just need to ignore pixels which are out of our interest zone. So, while averaging only skin pixels are considered and the results of RGB signal extracted from one trial can be shown below. The plotting is done as time on X axis and average of pixels on red, green and blue channels on Y axis. Signal extraction are shown for reference below. The signal extraction is taken on the basis of fast motion of the face and one is in normal still position. The graphs extracted are versus real time. So, on X axis we can’t notice time clearly due to scatter. The results are shown below.

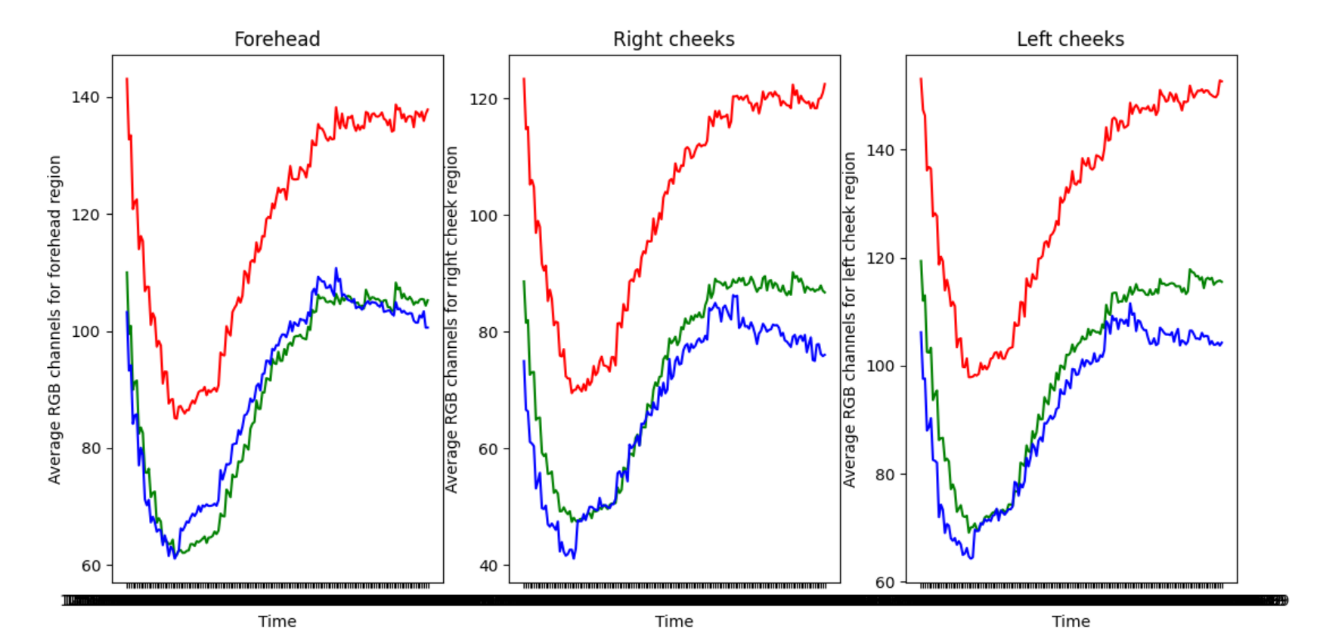


Fig14: RGB channels for all the regions in near to still position

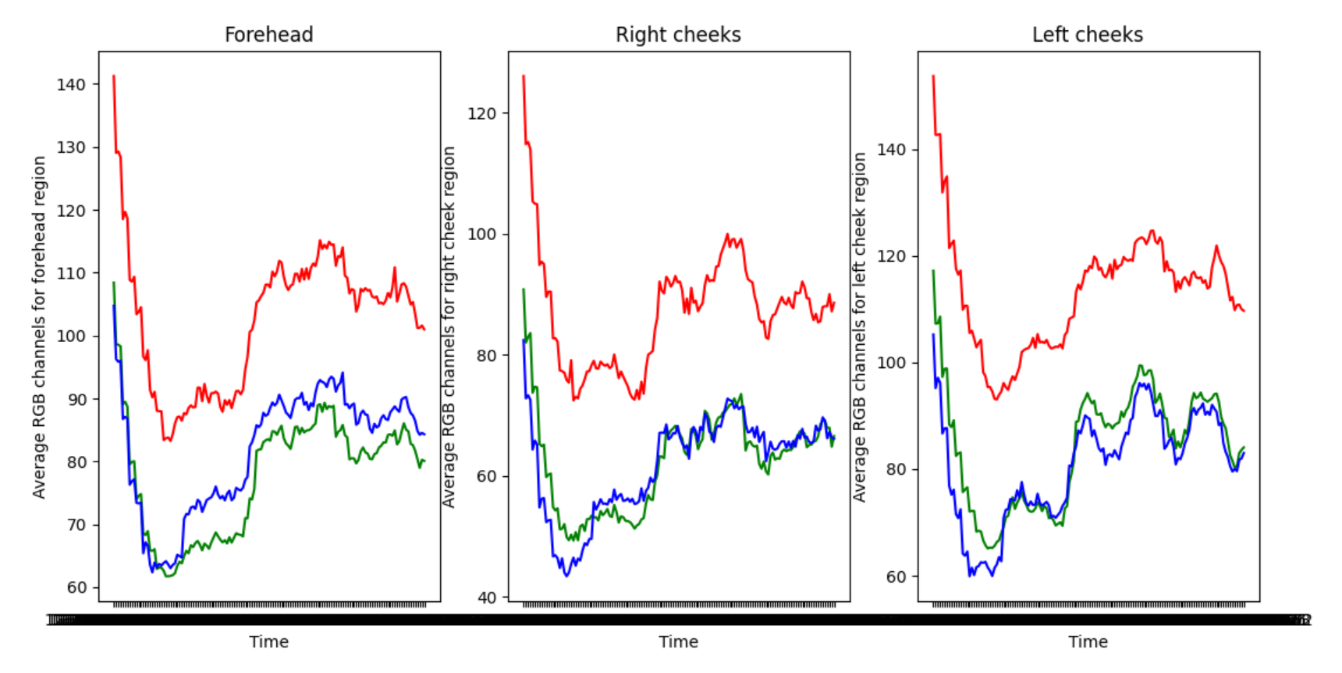


Fig15: RGB channels for all the regions in fast motion

**TASKS**:

1. Clear research on Remote Photoplethysmography and choose a method to be used based on analysis from research papers.
2. Get familiar with the programming part of Signal processing.
3. Implement the code in an organized pattern in a stepwise manner to get to good result.
4. A fine report has to be made at the end of the project.

**FIRST STEPS**:

1. Identify the dataset that is suitable for the case of low/no influence of motion and steady/stable conditions.
2. Find available libraries or implementation which are more promising for face tracking and signal extraction.
3. To combine all the solutions into one.

**REFERENCES**:

* Real-Time Webcam Heart-Rate and Variability Estimation with Clean Ground Truth for Evaluation

<https://www.mdpi.com/2076-3417/10/23/8630/pdf>

* An Open Framework for Remote-PPG Methods and Their Assessment

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9272290>

* Blind source separation: A review and analysis

<https://ieeexplore.ieee.org/document/6709849>

* An Overview of Remote Photoplethysmography Methods for Vital Sign Monitoring

<https://www.researchgate.net/publication/335474143_An_Overview_of_Remote_Photoplethysmography_Methods_for_Vital_Sign_Monitoring>

* Model-based Region of Interest Segmentation for Remote Photoplethysmography

<https://hal-univ-bourgogne.archives-ouvertes.fr/hal-02143091/document>

* Mediapipe reference code

<https://google.github.io/mediapipe/solutions/face_detection.html>

* ROI Selection for Remote Photoplethysmography

<https://link.springer.com/content/pdf/10.1007/978-3-642-36480-8_19.pdf>

* Refining ROI selection for real time rPPG using Adaptive skin detection

<https://www.researchgate.net/publication/329453135_Refining_ROI_selection_for_real-time_remote_photoplethysmography_using_adaptive_skin_detection>