Выполняли: Прасолова Виолетта и Корбанова Татьяна БМТ1-43Б

Задание: Используя данные материалы научиться выделять из видео сигнал видеоплетизмограммы, оценить похожесть с фотоплетизмограммой.

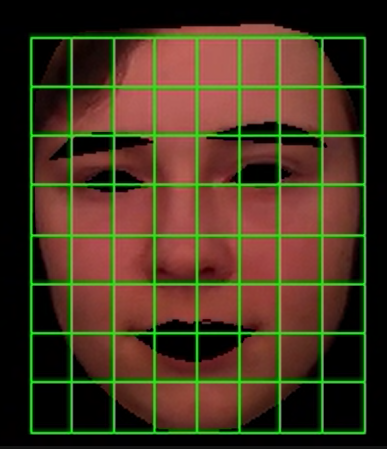


Рисунок 1 –Исходный кадр. Рисунок 2 – Обработанный кадр.

С помощью кода была проделана обработка видео: выделена только кожа лица с помощью комбинированного метода. Для расчёта сигнала видеоплетизмограммы(ВПГ) область размечалась сеткой. В каждом секторе сетки находилось значение VPG, отсеивались неинформативные. Окончательно VPG кадра рассчитывалось как среднее значение по всем секторам сетки.

Где i-номер кадра видеоизображения; j, k- координаты пикселей, принадлежащих выделенной области; R, G, B-цветовые координаты соответствующих пикселей.

Значение сигнала ВПГ сохранены в Results.csv.

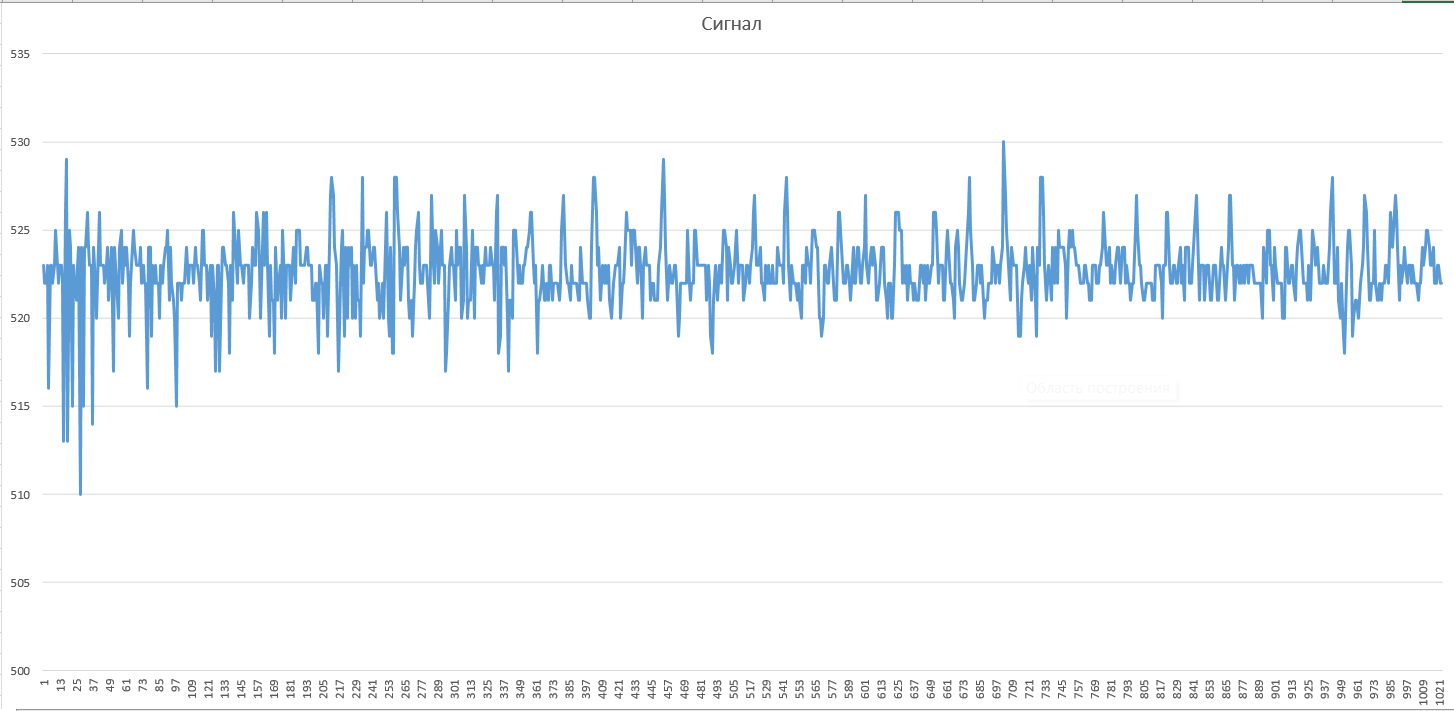


Рисунок 3 – Сигнал фотоплетизмограммы.

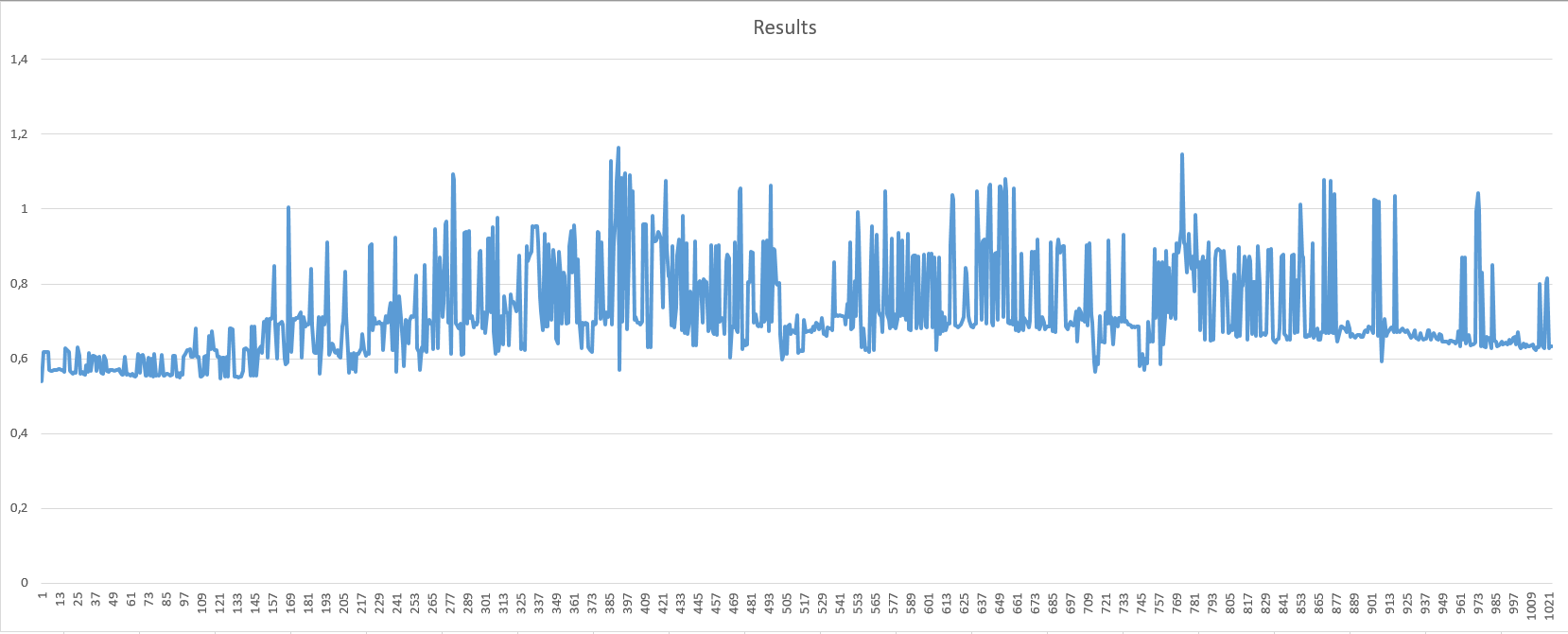


Рисунок 4 – Сигнал видеоплетизмограммы.

Вывод: Сигнал ВПГ отличается от ФПГ предположительно из-за плохой освещенности или неэффективном выборе метода.

КОД:

import cv2  
import numpy as np  
import dlib  
import csv  
  
# Открываем видеофайл  
video\_path = 'C:/Users/Таня/PycharmProjects/paresis\_detection/практика/Measurementsmp4/Andreeva\_1\_27.04.19.mp4'  
cap = cv2.VideoCapture(video\_path)  
  
# Загрузка предобученной модели детекции лица (модель HOG)  
detector = dlib.get\_frontal\_face\_detector()  
# Загрузка предобученной модели для поиска 68 точек контуров лица  
predictor = dlib.shape\_predictor('models/shape\_predictor\_81\_face\_landmarks.dat')  
  
# Получаем информацию о видео (ширина, высота, количество кадров в секунду и т. д.)  
frame\_width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))  
frame\_height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))  
fps = cap.get(cv2.CAP\_PROP\_FPS)  
  
# Создаем видеовыход для сохранения результата  
#output\_path = 'path/to/save/output.mp4'  
output = cv2.VideoWriter("outputt.mp4", -1, fps, (frame\_width, frame\_height), isColor=True)  
  
VPG\_result = []  
  
# Обработка каждого кадра видео  
while True:  
 # Чтение кадра  
 ret, frame = cap.read()  
 if not ret:  
 break  
  
 # Преобразование изображения в пространство цветов YCrCb  
 ycrcb = cv2.cvtColor(frame, cv2.COLOR\_BGR2YCrCb)  
  
 # Определение диапазона цвета кожи в пространстве YCrCb  
 lower\_skin = np.array([0, 133, 77], dtype=np.uint8)  
 upper\_skin = np.array([255, 173, 127], dtype=np.uint8)  
  
 # Создание маски цветовой сегментации  
 skin\_mask = cv2.inRange(ycrcb, lower\_skin, upper\_skin)  
  
 # Применение морфологических операций для удаления шума  
 kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (11, 11))  
 skin\_mask = cv2.morphologyEx(skin\_mask, cv2.MORPH\_OPEN, kernel)  
  
 # Нахождение контуров на маске  
 contours, \_ = cv2.findContours(skin\_mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 # Поиск контура с максимальной площадью (предполагаемое лицо)  
 max\_area = 0  
 max\_contour = None  
 for contour in contours:  
 area = cv2.contourArea(contour)  
 if area > max\_area:  
 max\_area = area  
 max\_contour = contour  
  
 face\_mask = np.zeros\_like(frame)  
 if max\_contour is not None:  
 cv2.drawContours(face\_mask, [max\_contour], 0, (255, 255, 255), -1)  
  
 # Применение маски лица к изображению  
 masked\_image = cv2.bitwise\_and(frame, face\_mask)  
  
 # Преобразование кадра в оттенки серого  
 gray = cv2.cvtColor(masked\_image, cv2.COLOR\_BGR2GRAY)  
  
 # Детекция лиц на кадре  
 faces = detector(gray)  
  
 for face in faces:  
 # Поиск контуров лица  
 landmarks = predictor(gray, face)  
 landmarks\_points = []  
 for i in range(81):  
 x = landmarks.part(i).x  
 y = landmarks.part(i).y  
 landmarks\_points.append((x, y))  
  
 # Создание маски для лица  
 mask = np.zeros\_like(gray)  
 points = np.array(landmarks\_points, np.int32)  
 convexhull = cv2.convexHull(points)  
 cv2.fillConvexPoly(mask, convexhull, 255)  
  
 kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (5, 5))  
 skin\_mask\_closed = cv2.morphologyEx(mask, cv2.MORPH\_CLOSE, kernel)  
 skin\_mask\_cleaned = cv2.morphologyEx(skin\_mask\_closed, cv2.MORPH\_OPEN, kernel)  
  
 # Обрезание видео по контуру лица  
 masked\_image = cv2.bitwise\_and(masked\_image, masked\_image, mask=skin\_mask\_cleaned)  
  
 # Закрашивание глаз и губ  
 left\_eye\_points = landmarks\_points[36:42]  
 right\_eye\_points = landmarks\_points[42:48]  
 lips\_points = landmarks\_points[48:60]  
 left\_eyebrow\_points = landmarks\_points[17:22]  
 right\_eyebrow\_points = landmarks\_points[22:27]  
  
 cv2.fillConvexPoly(masked\_image, np.array(left\_eye\_points), (0, 0, 0))  
 cv2.fillConvexPoly(masked\_image, np.array(right\_eye\_points), (0, 0, 0))  
 cv2.fillConvexPoly(masked\_image, np.array(lips\_points), (0, 0, 0))  
 cv2.fillConvexPoly(masked\_image, np.array(left\_eyebrow\_points), (0, 0, 0))  
 cv2.fillConvexPoly(masked\_image, np.array(right\_eyebrow\_points), (0, 0, 0))  
  
 # Расчет размера сетки  
 grid\_size = 8 # Размер сетки (количество рядов и столбцов)  
  
 # Создание сетки на лице  
 grid = np.zeros((grid\_size + 1, grid\_size + 1, 2), dtype=np.int32)  
 for i in range(grid\_size + 1):  
 for j in range(grid\_size + 1):  
 x = int((landmarks\_points[16][0] - landmarks\_points[0][0]) \* (i / grid\_size) + landmarks\_points[0][0])  
 y = int((landmarks\_points[8][1] - landmarks\_points[70][1]) \* (j / grid\_size) + landmarks\_points[70][1])  
 grid[j, i] = [x, y]  
  
 # Расчет видеоплетизмограммы для каждого сектора сетки  
 VPG\_values = []  
 for i in range(grid\_size):  
 for j in range(grid\_size):  
 x1, y1 = grid[i, j]  
 x2, y2 = grid[i, j + 1]  
 x3, y3 = grid[i + 1, j]  
 x4, y4 = grid[i + 1, j + 1]  
 cv2.line(masked\_image, (x1, y1), (x2, y2), (0, 255, 0), 1)  
 cv2.line(masked\_image, (x1, y1), (x3, y3), (0, 255, 0), 1)  
 cv2.line(masked\_image, (x2, y2), (x4, y4), (0, 255, 0), 1)  
 cv2.line(masked\_image, (x3, y3), (x4, y4), (0, 255, 0), 1)  
  
 # Извлечение области изображения в секторе сетки  
 sector\_image = masked\_image[y1:y4, x1:x4]  
  
 # Вычисление цветовых компонентов в секторе  
 R = sector\_image[:, :, 2]  
 G = sector\_image[:, :, 1]  
 B = sector\_image[:, :, 0]  
  
 # Вычисление значения видеоплетизмограммы  
 if np.sum(R) + np.sum(B) != 0:  
 VPG = np.sum(G) / (np.sum(R) + np.sum(B))  
 VPG\_values.append(VPG)  
  
 #print(VPG\_values)  
  
 mean = np.mean(VPG\_values) # Среднее значение  
 std = np.std(VPG\_values) # СКО  
  
 #print(mean)  
 #print(std)  
  
 filtered\_VPG = []  
 for v in VPG\_values:  
 if (v <= mean + std) and (v >= mean - std):  
 filtered\_VPG.append(v)  
  
 VPG\_result.append(np.mean(filtered\_VPG))  
  
 # Запись обработанного кадра в выходное видео  
 output.write(masked\_image)  
  
 # Отображение кадра  
 cv2.imshow('Face Detection', masked\_image)  
  
 # Выход из цикла при нажатии клавиши 'q'  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
# Путь к файлу CSV  
csv\_file = 'Resultss.csv'  
  
# Запись массива в файл CSV  
with open(csv\_file, 'w', newline='') as file:  
 writer = csv.writer(file)  
 writer.writerow(['Значение VPG']) # Запись заголовка  
 writer.writerows(zip(VPG\_result)) # Запись значений  
print(f"Массив успешно записан в файл CSV: {csv\_file}")  
  
# Освобождение ресурсов  
cap.release()  
output.release()  
cv2.destroyAllWindows()