Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Теоретическая информатика и компьютерные технологии

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ ПО ДИСЦИПЛИНЕ «АЛГОРИТМЫ КОМПЬЮТЕРНОЙ ГРАФИКИ» НА ТЕМУ:

Определение ЧСС методом фотоплетизмографии в режиме реального времени

Студент ИУ9-52Б _____ Потребина В.В.
(Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ Брагин А.В.
(Подпись, дата) (И.О.Фамилия)

Оценка _____

2024 г.

Оглавление

ВВЕДЕНИЕ

Область компьютерной графики известна прежде всего своими приложениями в сфере развлечений, дизайна, виртуального моделирования и визуализации различной информации. Однако её потенциал выходит далеко за пределы этих традиционных областей. В этой курсовой работе рассматривается не совсем обычное применение алгоритмов компьютерной графики: извлечение физиологических данных из стандартных видеозаписей с веб-камеры, в частности, мониторинг сердечного ритма в реальном времени с использованием фотоплетизмографии (PPG).

В основе этой работы лежит предположение, что каждый кадр видео, снятый видеокамерой, содержит больше информации, чем кажется на первый взгляд. Используя алгоритмы компьютерной графики, становится возможным обрабатывать и анализировать эти кадры, чтобы обнаружить данные, которые не очевидны сразу. Такой подход потенциально превращает веб-камеру из простого средства связи в сложное устройство мониторинга здоровья. Целью данной работы является демонстрация того, как алгоритмы компьютерной графики могут быть адаптированы и оптимизированы для анализа незаметных глазу изменений цвета кожи, зафиксированных на видео обычной видеокамерой, которые указывают на частоту сердечных сокращений человека.

Применение алгоритмов компьютерной графики в этом контексте сопряжено с определёнными сложностями. Требуется точная калибровка для обнаружения мельчайших изменений цвета кожи и надежные возможности обработки для обработки потоков данных в реальном времени. Это исследование сосредоточено на разработке и совершенствовании этих алгоритмов в веб-приложении на основе JavaScript, обеспечивая доступность и эффективность системы.

В условиях растущего спроса на решения для телемедицины и удаленного мониторинга пациентов увеличивается и количество исследований и

практических работ, направленных на возможности перепрофилировать повседневные технологии, такие как обычные вебкамеры и камеры смартфонов, для сбора медицинских данных. Самые последние результаты по этой тематике опубликованы в Nature [Lee 2022], PubMed [Tohma 2021] и Springer [Lee 2023], что показывает их высокую научно-практическую ценность.

В рамках данной курсовой работы была рассмотрена задача программной реализации метода фотоплетизмографии для определения частоты сердечных сокращений (ЧСС) человека по видеоизображению.

1. Обзор предметной области и постановка задачи

Целью данной работы является определение ЧСС методом фотоплетизмографии [Мошкевич 1970]. Этот метод можно реализовать двумя способами: анализом видеокадров из файла, либо непосредственно с видеокамеры в реальном времени. Метод фотоплетизмографии основан на оптическом способе измерения изменений, возникающих при наполнении капилляров кровью. Обычно используются светодиоды и фотодиоды для регистрации пульсаций крови. Такие изменения также можно регистрировать, хоть и с определённой долей погрешности, при помощи обычной видеокамеры. Этот метод позволяет определить частоту сердечных сокращений (ЧСС) и оценить общую активность сердечно-сосудистой системы. Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Сбор данных: необходимо провести с использованием устройства PPG для сбора данных в различных условиях.
2. Преобразование данных: фильтрация и преобразование сырых данных для удаления шумов и артефактов.
3. Извлечение ЧСС: разработка алгоритмов для выделения пульсаций и определения времени между ними
4. Тестирование и отладка: провести тестирование алгоритма на различных данных, включая разные физиологические состояния и типы кожи.

Также в перспективе стоят следующие задачи:

1. Интеграция: внедрение алгоритма в устройства
2. Оценка точности: оценка точности разработанного метода сравнительно с другими методами измерения ЧСС (например, ЭКГ) для подтверждения его эффективности.

3. Обеспечение безопасности: учесть вопросы конфиденциальности и безопасности при сборе и обработке биометрических данных.

Эффективное определение ЧСС методом PPG имеет потенциал для широкого применения в медицине, фитнесе и технологиях самомониторинга.

2. Алгоритм определения положения лица

Для определения положения черт лица на видео в режиме реального времени была использована библиотека компьютерного зрения face-api.js[1]. Библиотека реализует несколько CNN (сверточных нейронных сетей), которые позволяют нам обнаружить, распознать лица и его ориентиры. Основными этапами распознавания лица будем считать:

1. Загрузка данных модели
2. Обнаружение положения черт лица

Важно отметить, что данная реализация может определять только одно лицо на кадре. При использовании программы несколькими пользователями, данные с высокой вероятностью будут некорректными.

2.1. Загрузка данных моделей

В библиотеке face-api.js есть файлы с весами моделей, которые были квантованы для уменьшения размера файла модели на 75% по сравнению с исходной моделью, чтобы клиентский компьютер мог загружать только необходимые данные. В папке models были загружены файлы моделей, которые понадобятся в дальнейшей реализации нашей задачи: модель обнаружения и распознавания положения черт лица, а также их выражение. Модели будут использоваться в качестве статических ресурсов для нашего веб-приложения.

```
Promise.all([
   faceapi.nets.tinyFaceDetector.loadFromUri('/models'),
   faceapi.nets.faceLandmark68Net.loadFromUri('/models'),
   faceapi.nets.faceRecognitionNet.loadFromUri('/models'),
   faceapi.nets.faceExpressionNet.loadFromUri('/models')
]).then(startVideo)
```

Листинг 1 - Загрузка моделей из папки /models

[1] Исходный код библиотеки доступен по адресу: https://github.com/justadudewhohacks/face-api.js

2.2. Обнаружение положения черт лица

Face-api.js предоставляет нам функцию detectSingleFace, которая принимает, в нашем случае, видео в качестве первого элемента и аргумента необязательных опций, который мы можем передать детектору нейронной сети. Полное описание лица содержит результат обнаружения (ограничивающий прямоугольник + оценка), положения черт лица, выражение лица, а также вычисленный дескриптор. Затем визуализируем обнаружение, рисуя ограничивающую рамку на холсте и возвращаем полное описание лица, которое будет использовано при распознавании лица.

```
const detection = await faceapi.detectSingleFace(video,
    new faceapi.TinyFaceDetectorOptions())
    .withFaceLandmarks()
    .withFaceExpressions()
```

Листинг 2 - Определение лица на видео

За отрисовку контейнера и основных точек на лице (из также называют ландмарками) отвечают функции faceapi.draw.drawDetections() и faceapi.draw.drawFaceLandmarks(), которым на вход передаем холст, на котором и будет происходить отрисовка, и размер переопределенного дисплея. При верном использовании библиотеки мы получим следующий результат.

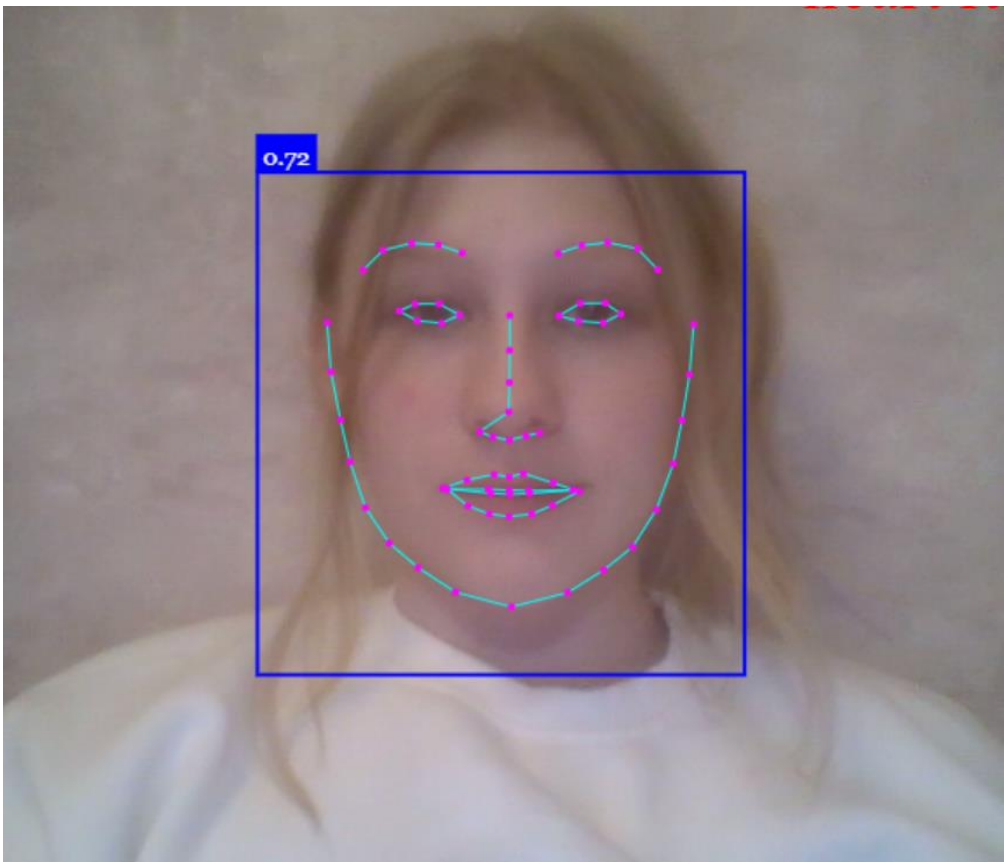


Рисунок 1 - Обнаружение положения черт лица

3. Определение графика сигналов

3.1. Определение области сигналов

При сердцебиении кровь циркулирует через сосуды, что может оказывать воздействие на цвет лица. Область щек является одной из областей лица, которая активно меняет цвет при сердцебиении из-за расширения сосудов и увеличения притока крови. Именно поэтому было принято решения взять область щек для определения сигналов.

На ранних этапах были определены, с помощью библиотеки Face-api.js, область лица и ее основные точки. По данным этих точек можно определить набор координат вертикальной линии носа и линий правой и левой брови. Массив линии носа представляет собой массив координат x и y длиной в 8 элементов (8 точек). Массив каждой брови также представляет собой массив координат длиной в 5 элементов. Выбрав оптимальное расстояние между точками на правой, левой брови и носу, можно проецировать линии и их пересечение и будет той самой нужной нам областью.

```
function getPieceOfFace(detection, canvas) {
   const noseLine = [detection.landmarks.getNose()[1],
detection.landmarks.getNose()[2]]
   const upLine = makePerpendicularLine(noseLine[0], noseLine)
   const downLine = makePerpendicularLine(noseLine[1], noseLine)

   const left_brow = [detection.landmarks.getLeftEyeBrow()[1],
detection.landmarks.getLeftEyeBrow()[3]]
   const right_brow = [detection.landmarks.getRightEyeBrow()[1],
detection.landmarks.getRightEyeBrow()[3]]

   const leftCheekPoints = [...left_brow.map((point) => pointProjection(point,
upLine)), ...left_brow.map((point) => pointProjection(point, downLine)).reverse()]
   const rightCheekPoints = [...right_brow.map((point) => pointProjection(point,
upLine)), ...right_brow.map((point) => pointProjection(point, downLine)).reverse()]

   return [leftCheekPoints, rightCheekPoints]
}
```

Листинг 3 - Определение координат ландмарков линии носа, бровей

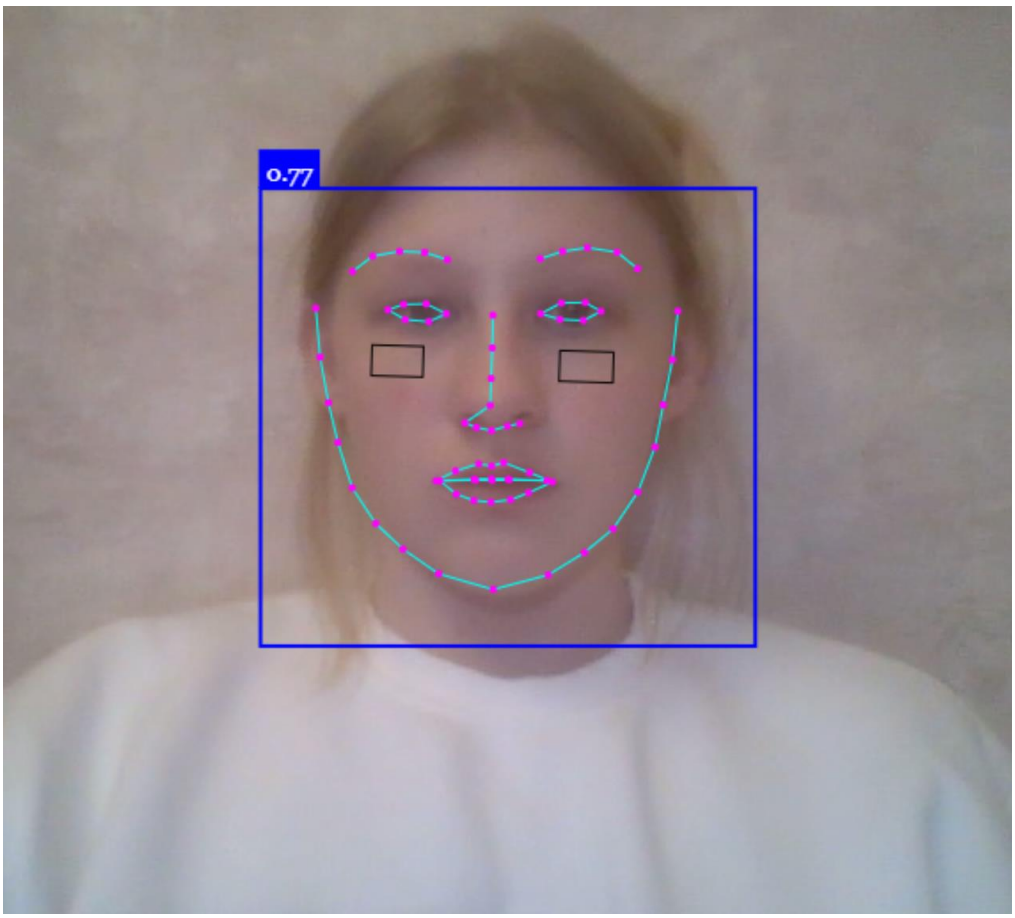


Рисунок 2 – Области сигналов

3.2. Алгоритм получения массива цветов в области

Для получения массива цветов RGB в конкретной области был использован алгоритм построчного сканирования. Построчное сканирование (также известное как растровая развертка) является фундаментальной концепцией в компьютерной графике. Алгоритм построчного сканирования с упорядоченным списком ребер основан на том факте, что любое горизонтальное сечение контура многоугольника состоит из четного числа точек, следовательно, для каждой строки сканирования необходимо:

1. Растеризация всех негоризонтальных ребер многоугольника и помещение полученных точек в списки x-координат для каждой строки сканирования; упорядочить их вдоль строки сканирования;
2. Для каждой строки сканирования:

2.1. Список точек пересечения с ребрами упорядочивается по возрастанию x;

2.2. Заполняются все промежутки вида $[x_{2i-1}; x_{2i})$ – соглашения по подсчету точек пересечения строк сканирования с ребрами многоугольника гарантируют, что в каждом списке содержится четное число элементов.

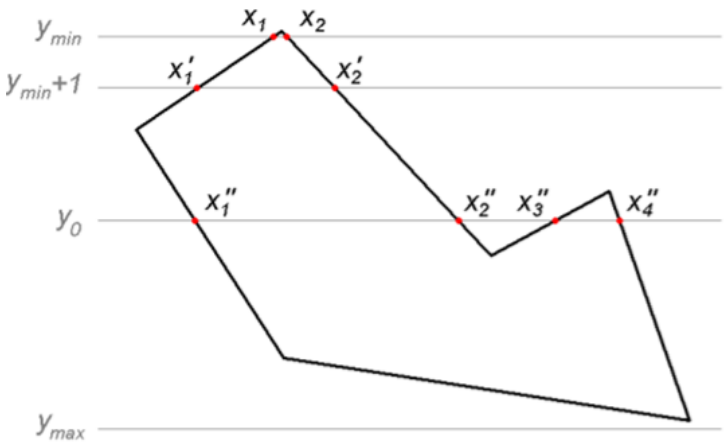
Также важно обратить внимание на особые случаи определения точек пересечения строки сканирования с ребрами многоугольника.

1. Горизонтальные ребра многоугольника - не учитываются;
2. Строка сканирования проходит через вершину многоугольника (можно использовать один из двух подходов):

   2.1. Смещение всех строк сканирования на ½ пикселя (таким образом, вершины заведомо не будут лежать на строке сканирования);

   2.2. Вершина не учитывается (учитывается дважды), если оба смежных ребра лежат по одну сторону от строки сканирования, и учитывается один раз, если смежные ребра лежат по разные стороны от строки сканирования.

На рисунке 3 продемонстрирован алгоритм построчного сканирования.



y_{min}:	x_1, x_2
$y_{min}+1$:	x_1', x_2'
…	…
y_0:	x_1'', x_2'', x_3'', x_4''
…	…
y_{max}:	–

Рисунок 3 - Схема алгоритма построчного сканирования

3.3. Преобразование массива цветов области в сигнал

После получения массива цветов в области, встал вопрос, каким образом нужно преобразовывать его в сигнал. Функция makeSignal(color) решает поставленную задачу. Данная функция принимает на вход неограниченное количество массивов цветов. Цвет в каждой точке области суммируется друг с другом с определенными коэффициентами и нормализуется от нулевого до единичного значения, а после суммы всех элементов массива берем их среднее арифметическое. Экспериментальным способом были подобраны такие коэффициенты, которые бы давали наиболее чистый сигнал с наименьшим количеством шумов. Задача оптимизации коэффициентов преобразования цветов в сигнал можно посвятить отдельное исследование в дальнейшем.

3.4. График сигналов

Обычно FPS камеры составляет 30 кадров в секунду, то есть, физический лимит для получения нового кадра равен 33.3 миллисекундам. Следовательно, нет смысла обрабатывать кадры чаще этого лимита. Возьмем в качестве периода между взятием нового кадра PERIOD_TIME, равный 34 миллисекундам. Получаем кадр и кладем его в очередь для дальнейшей его обработки. Благодаря

такой очередности, все кадры будут обрабатываться последовательно, даже если время их обработки будет отличаться. Далее для каждого кадра мы получаем сигнал для нашей, определенной ранее, области и убираем кадр из очереди. Таким образом, мы получаем график сигналов, у которого ось y – сигнал, сформированный из RGB компонент на заданной области, а ось x – время в миллисекундах (кратное 34), в которое был получен кадр.

Также мы ограничиваем график сигнала по оси абсцисс так, чтобы его длина не превышала 10000 миллисекунд. Если мы получаем сигнал для большего времени, то график соответственно сдвигается.

На рисунке 4 изображен полученный график сигналов

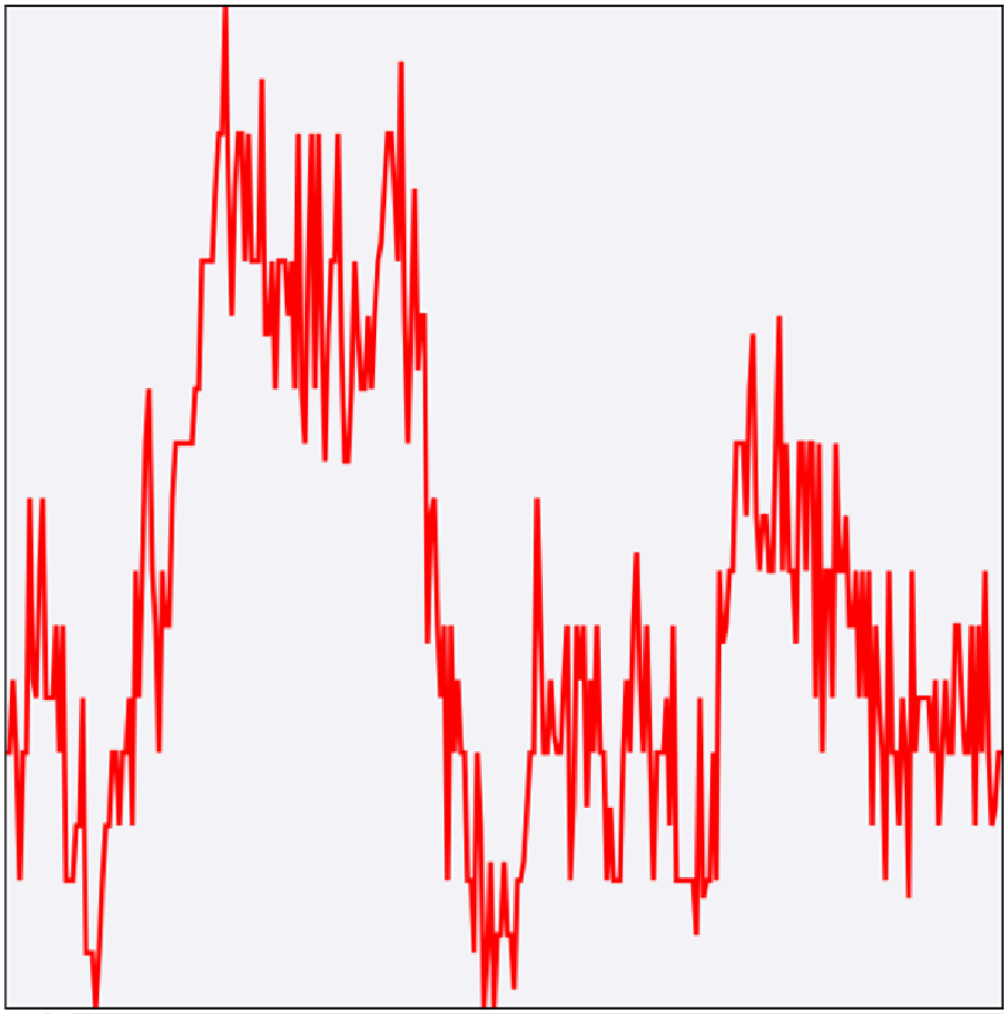


Рисунок 4 - График сигналов

4. Алгоритмы обработки графика сигналов

4.1. Алгоритм сглаживания сигнала

При обработке экспериментальных данных часто возникает необходимость их усреднения. С этой целью часто используют алгоритм арифметического скользящего среднего.

Скользящее среднее — общее название для семейства функций, значения которых в каждой точке определения равны среднему значению исходной функции за предыдущий период.

Скользящие средние обычно используются с данными временных рядов для сглаживания краткосрочных колебаний и выделения основных тенденций или циклов. Простое (арифметическое) скользящее среднее численно равно среднему арифметическому значений исходной функции за установленный период и вычисляется по формуле:

$$f_k = \frac{1}{h} \sum_{i=l}^{k} f_i,$$

где f_i – исходные значения рассматриваемой функции, $h = k - l$ – сглаживающий интервал – количество значений исходной функции для расчета скользящего среднего.

На рисунке 5 представлены функции скользящего среднего для исходной последовательности значений с разной величиной сглаживающего интервала. Чем шире сглаживающий интервал, тем более плавным будет график результирующей функции. Однако, с другой стороны, увеличение сглаживающего интервала приводит к временному сдвигу усредненной функции относительно исходной.

На рисунке 6 можно понаблюдать за тем, как меняется график сигнала при применении к нему алгоритма сглаживания.

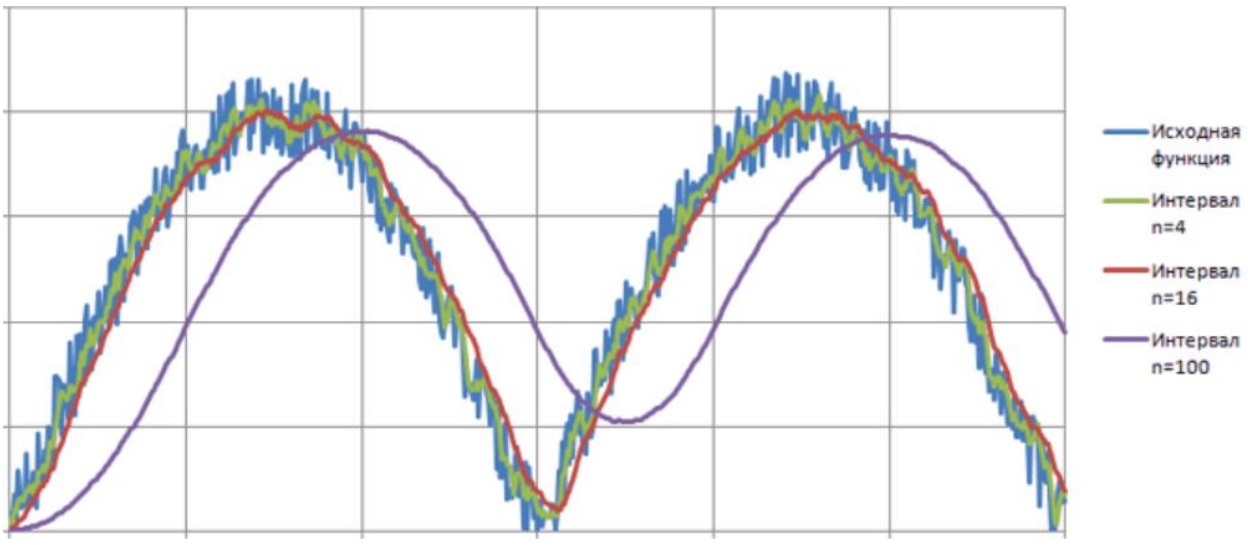


Рисунок 5 - Функции скользящего среднего

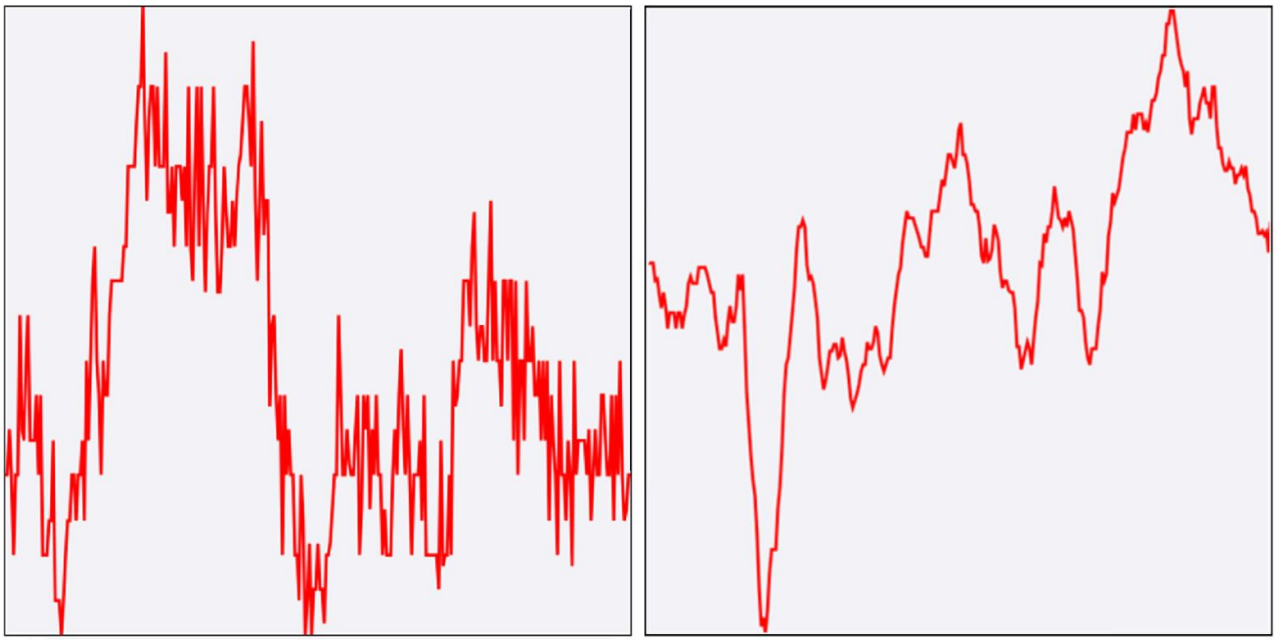


Рисунок 6 - Применение сглаживания для графика сигнала

4.2 Алгоритм нормализации сигнала

Алгоритм нормализации сигнала используется для приведения сигнала к определенным стандартным условиям, в нашем случае к диапазону значений от 0 до 1. Алгоритм заключается в следующем:

1. Нахождение минимального и максимального значения в массиве сигналов

2. Выражаем формулу для конкретной задачи (чтобы график имел значения от 0 до 1). В нашем случае (signal - minSignal) / (maxSignal - minSignal), где signal – значение сигнала, minSignal – минимальное значения сигнала, maxSignal – максимальное значение сигнала.
3. Применяем найденную формулу к каждому значению сигнала, чтобы нормализовать его.

```
function normalizeSignal(signals) {
    const minSignal = Math.min(...signals)
    const maxSignal = Math.max(...signals)
    return signals.map((signal) => (signal - minSignal) / (maxSignal - minSignal))
}
```

Листинг 4 - Функция, реализующая алгоритм нормализации

Нормализация помогает сделать сигнал более унифицированным для обработки и анализа.

4.3. Повышение частоты дискретизации графика

Повышение частоты дискретизации графика заключается в том, что имеющиеся отрезки разбиваются на более мелкие. Тем самым мы можем получить более детализированный график. Для вычисления значений в новых промежуточных точках можем воспользоваться алгоритмом кусочно-линейной интерполяции.

Алгоритм кусочно-линейной интерполяции графика используется для аппроксимации непрерывной функции на отрезке при помощи линейных функций, т.е. алгоритм позволяет "сшивать" линейные отрезки, чтобы аппроксимировать исходную функцию. Алгоритм можно описать следующим образом:

1. Входными данными является набор точек (x, y), упорядоченный по x, описывающий график сигналов.
2. Для каждых двух соседних точек (x_1, y_1) и (x_2, y_2) можем найти значение графика в точке x, которая находится на промежутке $[x_1, x_2]$, используя уравнение прямой по двум точкам:

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1}.$$

4.4. Алгоритм удаления тренда

Тренд – основная тенденция изменения графика или, другими словами, длительные монотонные изменения. Так как на графике необходимо будет найти периодические колебании, то, чтобы они стали более выраженными, стоит избавиться от тренда.

Алгоритм заключается в том, что мы применяем высокое сглаживание и тем самым избавляемся от высокочастотных и среднечастотных колебаний в графике. После этого необходимо вычесть результат из исходного графика.

4.5. Алгоритм центрирования графика сигнала

Прежде чем начать работать с сигналом, также рекомендуется выполнить центрирование графика, чтобы его центр масс находился в нуле. Алгоритм крайне прост и заключается в следующем:

1. Вычисление центра масс графика. Для этого возьмем среднее арифметическое всех элементов.
2. Из каждого значения исходного графика вычитаем вычисленные ранее цент масс.

5. Определение графика частот

5.1. Дискретное преобразование Фурье

На практике, для вычисления значений периодограмм используют дискретное преобразование Фурье (DFT), которое определяется следующим образом:

$$X_j = DFT_j\{x_k\}_{k=0}^{N-1} = \sum_{k=0}^{N-1} x_k e^{-i\frac{2\pi}{N}kj} \;\; j = \overline{0, N-1}.$$

Тогда формулы для периодограммы и взаимной периодограммы будут выглядеть следующим образом:

$$D_j = \frac{1}{N^2}\left|DFT_j\{x_k\}_{k=0}^{N-1}\right|^2, j = \overline{0, N/2}$$

$$D_{xy}[j] = \frac{1}{N^2}\left[\overline{DFT_j\{x_k\}_{k=0}^{N-1}}\right]\left[DFT_j\{y_k\}_{k=0}^{N-1}\right], j = \overline{0, N/2}.$$

5.2. Быстрое преобразование Фурье

Для вычисления дискретного преобразования Фурье требуется время $O(N^2)$, что делает его использование нежелательным для программной реализации. Поэтому обычно его заменяют на быстрое преобразование Фурье (FFT).

Существует несколько алгоритмов быстрого преобразования Фурье. Однако зачастую, говорят именно об алгоритме Кули-Тьюки. В общем случае он позволяет сократить время выполнения до $O\big(N(p_1 + \cdots + p_n)\big)$, где $N = p_1 \ldots p_n$. Рассмотрим схему этого алгоритма.

Запишем дискретное преобразование Фурье следующим образом:

$$X(k) = \sum_{j=0}^{N-1} x(j)\omega^{kj}, \qquad k = \overline{0, N-1}, \qquad \omega = e^{-i\frac{2\pi}{N}}$$

Первым делом, предполагаем, что $N = N_1 N_2$ для некоторых натуральных N_1 и N_2, а индексы j и k выражаем следующим образом:

$$j = j_1 + N_1 j_2, \qquad j_1 = \overline{0, N_1 - 1}, \qquad j_2 = \overline{0, N_2 - 1},$$
$$k = k_2 + N_2 k_1, \qquad k_1 = \overline{0, N_1 - 1}, \qquad k_2 = \overline{0, N_2 - 1}.$$

Тогда формула быстрого преобразования приобретет вид:

$$X(k_2 + N_2 k_1) = \sum_{j_1=0}^{N_1-1} \sum_{j_2=0}^{N_2-1} x(j_1 + N_1 j_2) \omega^{(k_2+N_2 k_1)(j_1+N_1 j_2)}.$$

Далее преобразуем входной и выходной векторы в двумерные массивы:

$$x'[j_1, j_2] = x[j_1 + N_1 j_2,$$
$$X'[k_1, k_2] = X[k_2 + N_2 k_1].$$

Тогда, учитывая, что $\omega^{N_1 N_2 k_1 j_2} = 1$:

$$X'[k_1, k_2] = \sum_{j_1=0}^{N_1-1} \beta^{j_1 k_1} \left(\omega^{j_1 k_2} \sum_{j_2=0}^{N_2-1} x'[j_1, j_2] \alpha^{j_2 k_2} \right),$$

где $\alpha = \omega^{N_1}, \beta = \omega^{N_2}$.

При этом получившиеся преобразования меньшей длинны также можно вычислять с помощью этого алгоритма.

Кроме того, часто принимают, что $N = 2^p$, где $p \in \mathbb{N}$. Тогда либо $N_1 = 2$, либо $N_2 = 2$. В этом случае, скорость выполнения алгоритма будет равна $O(2\mathrm{N} \log_2 N)$.

5.3. Преобразование графика сигналов в график частот

Для получения графика частот было применено быстрое преобразование Фурье, описанное ранее. Для этого воспользуемся библиотекой FFT.js[2].

График частот выглядит следующим образом. Ось абсцисс представляет собой ось частот. Каждой частоте соответствует единственное значение – интенсивность данной частоты. Чем выше это значение, тем большее влияние оказывает колебания такой частоты на исходный график. Таким образом, найдя на частотном графике пики (точки экстремума), мы можем определить частоту с максимальным влиянием. Это и будет искомый ЧСС.

Также важно заметить, что от количества входных данных зависит шаг частотной шкалы выходного графика, то есть его точность. Нам необходимо, чтобы этот шаг был как минимум равен одному удару в минуту или $\frac{1}{60}$ Гц. Зная, что этот шаг можно рассчитать по формуле:

$$\Delta x = \frac{1}{2Nd},$$

где d – шаг временной шкалы исходного графика в секундах, N – число элементов. Исходя из того, что мы применяем алгоритм повышения частоты дискретизации графика, $d = 0{,}001$, можем вычислить N. $N = 3000$. Поэтому будем расширять массив сигналов до этого размера, заполняя его нулями в конце.

Так как ЧСС человека в среднем колеблется от 50 до 140 ударов в минуту, ограничим вывод графика константами minFrequency (минимальная частота) и maxFrequency (максимальная частота) равными $\frac{40}{60}$ Гц и $\frac{150}{60}$ Гц соответственно. Таким образом, на рисунке 7 представлен график частот при стабильном сигнале в состоянии покоя.

[2] Исходный код библиотеки доступен по адресу: https://github.com/indutny/fft.js/

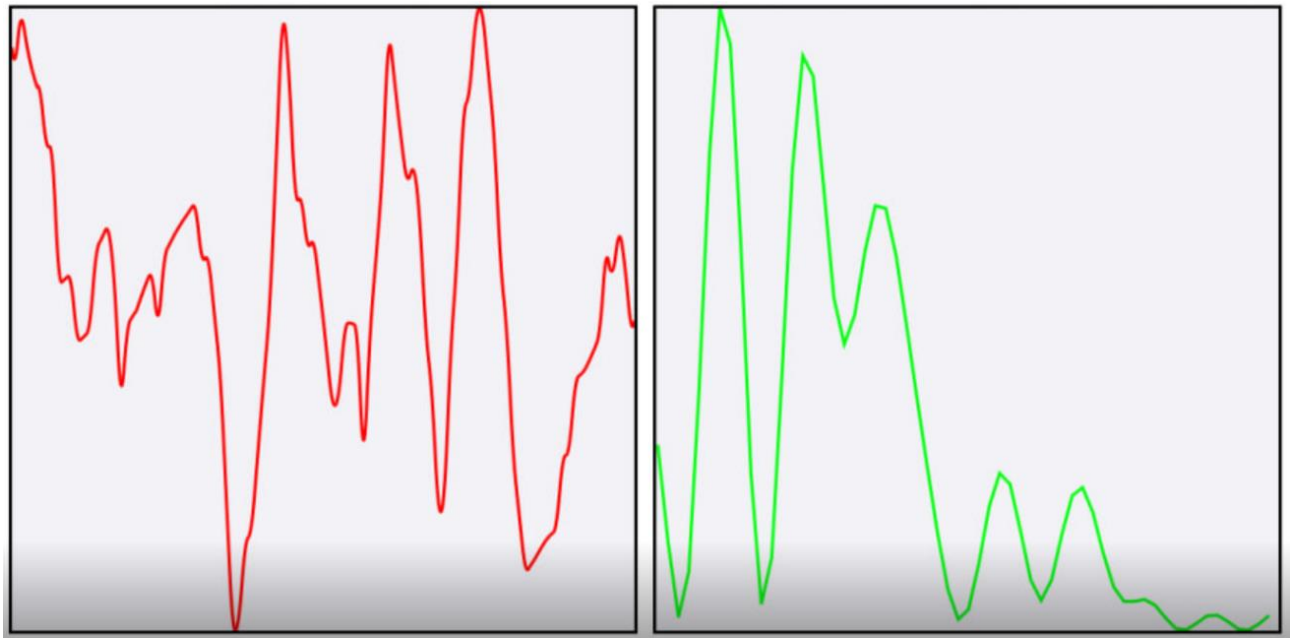


Рисунок 7 - График сигнала и частот

5.4 Определение ЧСС

Частотный график, полученный нами ранее, может давать небольшие погрешности, поэтому недостаточно просто выводить значение частоты, имеющей максимальную интенсивность. В этом случае пульс будет нестабильным и постоянно меняющимся из-за того, что внешние факторы могут легко повлиять на исходных график сигнала.

Для этого мы будем брать усредненное значение определенного ЧСС за некий промежуток времени. Чем это промежуток больше, тем меньшее влияние будут иметь внешние факторы и тем стабильнее будет определяться пульс.

6. Тестирование

6.1. Тестирование на сгенерированных данных

Тестирование программы проводилось в несколько этапов. Перед непосредственно подачей на вход данных сигналов в режиме реального времени, необходимо было убедиться в правильности работы алгоритма.

Для этого необходимо сгенерировать определенный набор данных с заранее известной нам частотой колебания. В качестве таких данных возьмем следующую функцию:

$$y = 150 \sin\left(\frac{\pi}{250} x\right)$$

Функция была специально подобрана таким образом, чтобы в отметке 1000 миллисекунд было ровно количество колебаний, в нашем случае 2 колебания. То есть, она имеет частоту 2Гц или 120 ударов в минуту.

График определенной функции выше представлен на рисунке 8.

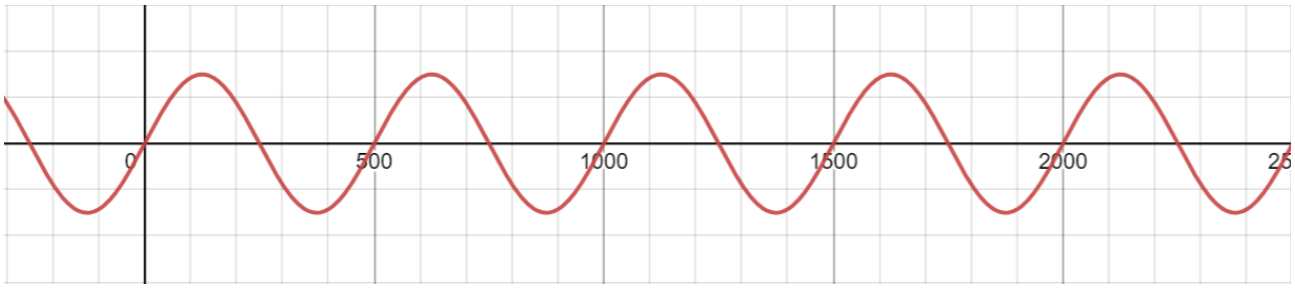


Рисунок 8 - График синусоиды

График частот ограничен значениями 60 и 130 ударов в минуту. То есть, он расположен между значениями 1 Гц и ~2.16 Гц. Проведя данный тест, получили следующие результаты, изображенные на рисунке 9.

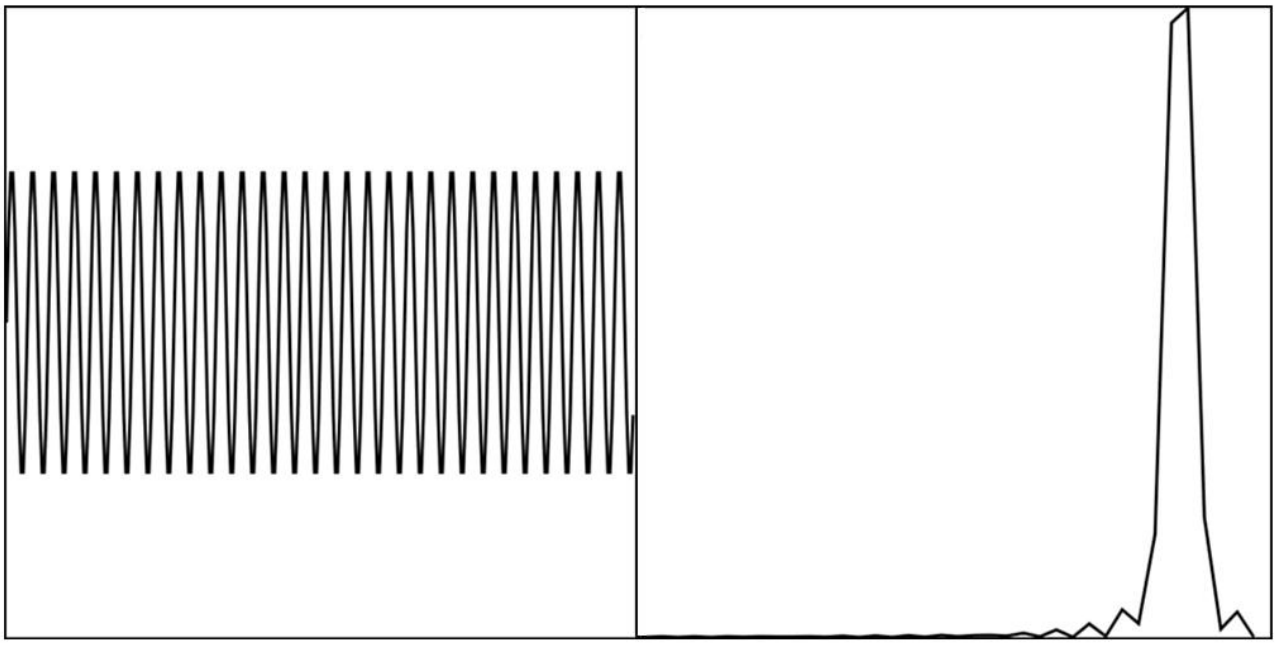


Рисунок 9 - График сигнала и частоты

6.2. Окно настроек

Для более удобного тестирования различных данных было принято решение дополнительно реализовать окно настроек. Для этого часть параметров была вынесена в отдельный конфигурационный файл, а на страницу была добавлена специальная форма. При изменении значений в форме, меняются также соответствующие параметры, а графики после сохранения изменения принимают новый вид. Форма изображена на рисунке 10.

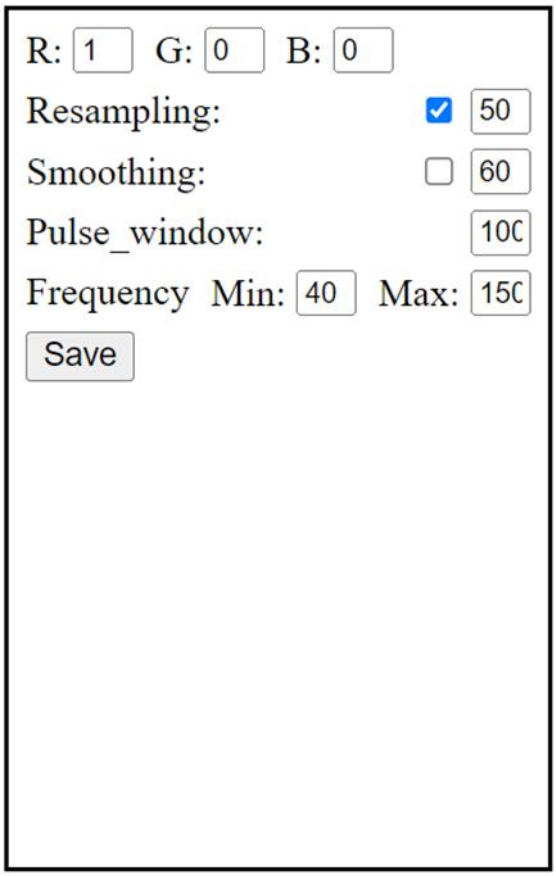


Рисунок 10 - Форма для задания параметров

Параметры, которые можно динамически изменять:

- Коэффициенты при RGB компонентах, для вычисления сигнала
- Флаг использования повышения частоты дискретизации графика и параметр, отвечающий за количество разбиений.
- Флаг использования сглаживания графика и параметр, отвечающий за степень сглаживания
- Окно усреднения пульса
- Минимальная и максимальная частота на графике

Проведя исследовательскую работу, были выявлены наиболее оптимальные значения этих параметров, конкретно в моей реализации программы:

- Коэффициенты для сигнала: R = 0.7, G = 0.3, B = 0

- Использования повышения частоты дискретизации графика до 1000Гц, то есть берем значения в каждую миллисекунду
- Сглаживание с окном в 60 значений
- Усреднений пульса с окном в 100 значений
- Минимальная и максимальная частота не оказывают влияние на тестирование. Мы его задаем исключительно для более удобного отображения значений на графике. Оптимально использовать интервал от 50 до 150 ударов в минуту, по описанной ранее причине.

Помимо этого, были также параметры, которые задавались строго, но имели такое же важное значение в реализации программы. Например, для выявления тренда использовалось сглаживание с окном в 1 секунду.

6.3. Тестирование в состоянии покоя

Первое тестирование программы в режиме реального времени проводилось на человеке в состоянии покоя. В таком состоянии нормальный пульс равняется 60 ударов в минуту, если не учитывать индивидуальные особенности, заболевания и прочее.

Объекту необходимо максимально избежать влияния внешних факторов во время тестирования: не двигать видеокамеру, не менять освещение в комнате, более того, желательно использовать дополнительный источник света, направленный на лицо испытуемого, а также, по возможности, не двигаться самому объекту. При соблюдении этих условий, сигнал получится максимально чистый, с минимальны количеством шума. Также необходимо учитывать, что на сигнал влияет качество видео, записываемого камерой.

По результатам данного тестирования (рисунок 11), программа показала пульс 69 ударов в минуту. Измерив пульс самостоятельно был получен результат 67 ударов в минуту, что соответствует результатам тестирования с небольшой погрешностью.

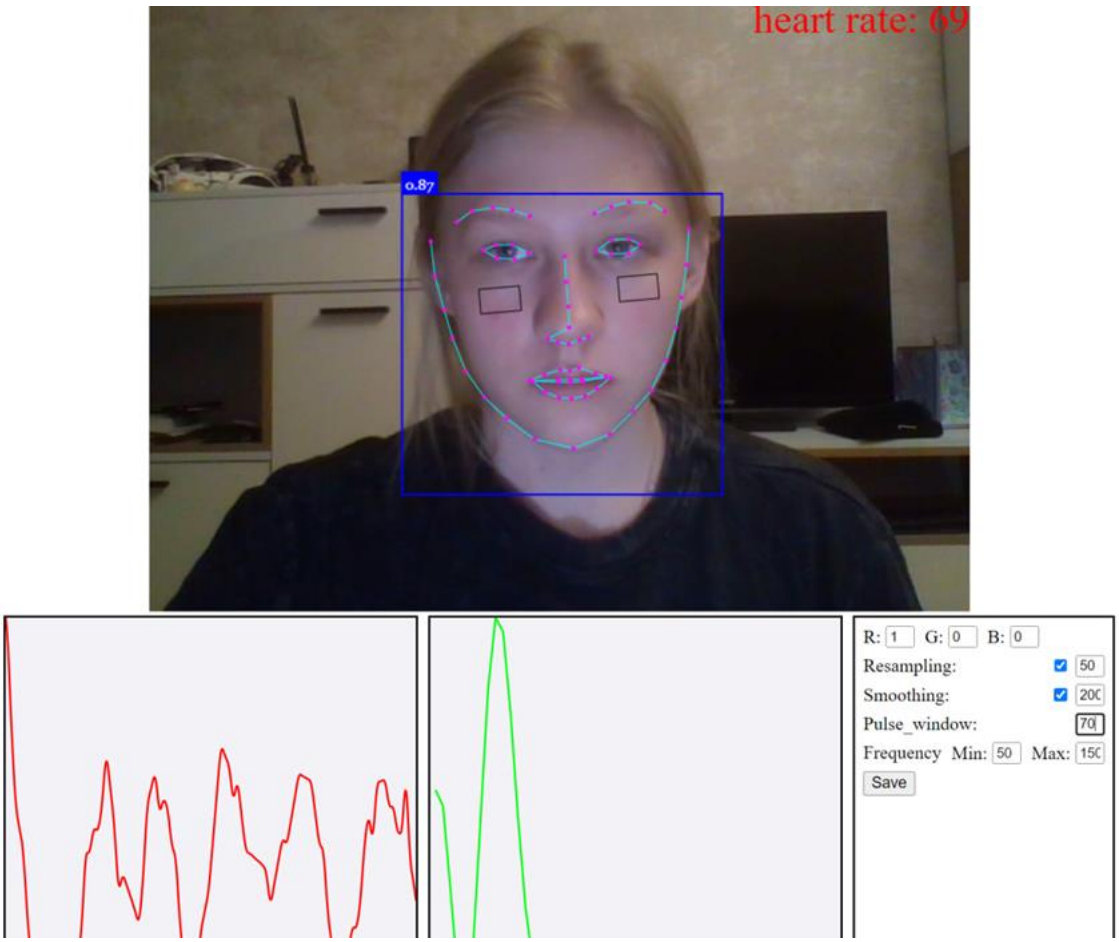


Рисунок 11 - Тестирование в состоянии покоя

6.4. Тестирование после физической активности

После небольшой физической нагрузки, результат тестирования (рисунок 12) составлял ЧСС в 105 ударов в минуту, что почти соответствует реальному ЧСС, который на самом деле был 98 ударов в минуту.

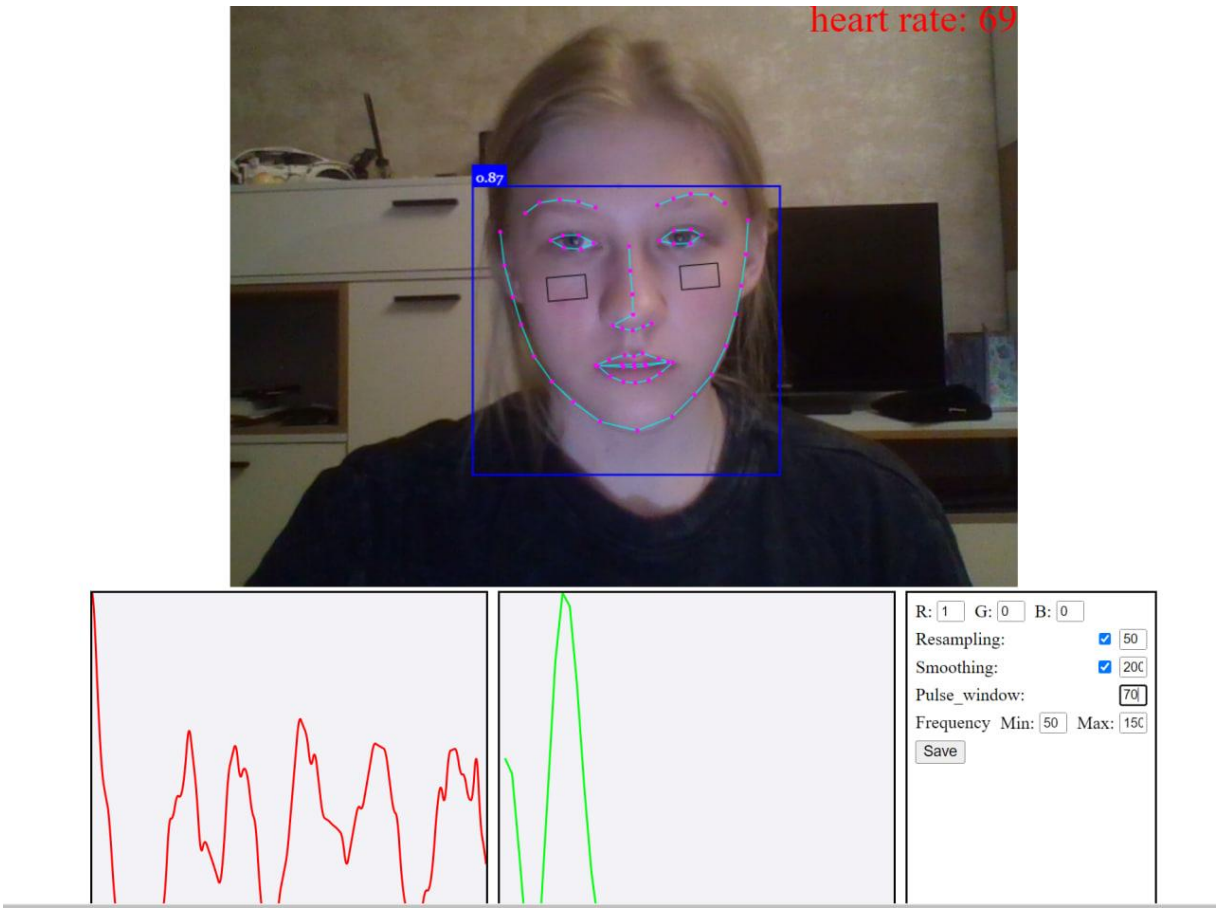


Рисунок 12 - Тестирование после физической активности

ЗАКЛЮЧЕНИЕ

В данной курсовой работе, был реализован алгоритм, который способен на основе минимальных изменений цветовых компонент области лица рассчитывать частоту сердечных сокращений. Данные фиксируются при помощи самой обычно видеокамеры.

Преимуществом данной программной реализации является ее доступность и простота в использовании, так как в современном мире почти каждый обладает камерой. Также на основе ряда тестов было выявлено, что алгоритм, действительно, способен определять пульс человека, хоть с и некой погрешностью, как мы могли заметить ранее.

Однако, все не так однозначно. Во время разработки и тестирования программы, были выявлены следующие проблемы.

Процесс определения лица и обработки сигнала занимают довольно большое время, особенно учитывая тот факт, что считывание происходит в режиме реального времени и входные данные постоянно изменяются. Из-за этого происходят задержки в получении кадров, что влияет на исходный сигнал и, следственно, на конечный результат. В теории, решением данной проблемы может послужить оптимизация алгоритмов. Также проблемой является воздействие внешних факторов, от которых, к сожалению, мы не сможем избавиться, но в наших силах их минимизировать.

Решив данные проблемы, мы получим отличный инструмент, который может применяться во многих сферах. Например, набирающим популярность дистанционным докторам, которым смогут проводить онлайн диагностику и определять ЧСС таким способом.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

[1] Мошкевич В.С. Фотоплетизмография (Аппаратура и методы исследования). М.: Медицина, 1970. — 208 с.

[2] Lee H. et al. Real-time realizable mobile imaging photoplethysmography //Scientific Reports. – 2022. – Т. 12. – №. 1. – С. 7141.

[3] Tohma A. et al. Evaluation of remote photoplethysmography measurement conditions toward telemedicine applications //Sensors. – 2021. – Т. 21. – №. 24. – С. 8357.

[4] Lee R. J., Sivakumar S., Lim K. H. Review on remote heart rate measurements using photoplethysmography //Multimedia Tools and Applications. – 2023. – С. 1-30.

[5] Cooley J.W. An algorithm for the machine calculation of complex Fourier series / J.W. Cooley, J.W. Tukey // Mathematics of Computation. – 1965. – Vol. 19. – P. 297—301.