



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

ИУ «Информатика и системы управления»

КАФЕДРА

ИУ-1 «Системы автоматического управления»

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К КУРСОВОЙ РАБОТЕ*

*на тему:*

---

*Исследование алгоритмов и методов  
детектирования, описания и извлечения  
конфигураций руки человека*

---

---

Студент ИУ1-71Б  
(Группа)

25/12/2021  
(Подпись, дата)

Д.И. Юдаков  
(И.О. Фамилия)

Руководитель курсовой работы

25/12/2021  
(Подпись, дата)

К.В. Парфентьев  
(И.О. Фамилия)

Консультант

25/12/2021  
(Подпись, дата)

К.В. Парфентьев  
(И.О. Фамилия)

2021 г.

**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)**

---

---

**УТВЕРЖДАЮ**

Заведующий кафедрой ИУ-1  
(Индекс)

К.А. Неусыпин  
(И.О. Фамилия)

« 11 » сентября 20 21 г.

**З А Д А Н И Е  
на выполнение курсовой работы**

по дисциплине Системотехника систем автоматизации и управления

Студент группы ИУ1-71Б

Юдаков Дмитрий Игоревич

(Фамилия, имя, отчество)

Тема КР Исследование алгоритмов и методов детектирования, описания  
и извлечения конфигураций руки человека

Направленность КР (учебная, исследовательская, практическая, производственная, др.)

учебная

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения КР: 25% к \_\_ н., 50% к \_\_ н., 75% к \_\_ н., 100% к \_\_ н.

**Задание** Исследовать основные и наиболее популярные алгоритмы и  
методы детектирования, описания и извлечения конфигураций руки человека.

---

---

---

**Оформление курсовой работы:**

Расчетно-пояснительная записка на 48 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

---

---

---

Дата выдачи задания « 11 » сентября 20 21 г.

**Руководитель КП** 11/09/2021 К.В. Парфентьев  
(Подпись, дата) (И.О. Фамилия)

**Студент** 11/09/2021 Д.И. Юдаков  
(Подпись, дата) (И.О. Фамилия)

# СОДЕРЖАНИЕ

СОДЕРЖАНИЕ . . . . .	3
ВВЕДЕНИЕ . . . . .	4
ПОСТАНОВКА ЗАДАЧИ . . . . .	5
1. МЕТОДЫ ДЕТЕКТИРОВАНИЯ КИСТИ ЧЕЛОВЕКА В СИСТЕМАХ ЧЕЛОВЕКО-МАШИННОГО ВЗАИМОДЕЙСТВИЯ . . . . .	6
1.1 Отделение фона изображения . . . . .	6
1.2 Метод распознавания кожи в HSV и YCbCr цветовых моделях . . . . .	9
1.3 Метод Оцу . . . . .	13
1.4 Mixture of Gaussians . . . . .	15
2. МЕТОДЫ ПОСТРОЕНИЯ КОНТУРА ОБЪЕКТА НА ИЗОБРАЖЕНИИ . . . . .	17
2.1 Выделение границ с помощью оператора Кэнни . . . . .	17
2.2 Топологический структурный анализ цифрового бинарного изображения с помощью отслеживания границ . . . . .	18
2.2.1 Формула площади Гаусса . . . . .	19
3. МЕТОДЫ НАХОЖДЕНИЯ КЛЮЧЕВЫХ ТОЧЕК НА КИСТИ . . . . .	21
3.1 Нахождение точек путём определения дефектов выпуклости . . . . .	21
3.1.1 Алгоритм Грэхема . . . . .	21
3.1.2 Алгоритм Джарвиса . . . . .	22
3.1.3 Алгоритм Киркпатрика . . . . .	24
3.1.4 Сравнение алгоритмов . . . . .	25
3.2 Локализация ключевых точек кисти руки на изображении на основе непрерывного скелета . . . . .	29
4. МЕТОДИКА И ОЦЕНКА ЕЁ ЭФФЕКТИВНОСТИ . . . . .	32
ЗАКЛЮЧЕНИЕ . . . . .	34
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ . . . . .	35
ПРИЛОЖЕНИЯ . . . . .	37
Приложение А . . . . .	37
Приложение Б . . . . .	40
Приложение В . . . . .	43
Приложение Г . . . . .	45
Приложение Д . . . . .	48

# ВВЕДЕНИЕ

Развитие технологий, продолжающееся уже более полувека, приводит к тому, что нижняя граница размеров процессоров уменьшается, в то время как их производительность продолжает увеличиваться. Для рядового пользователя результатом этого является разнообразие созданных интеллектуальных систем, используемых в повседневной жизни: от смартфонов до планшетов, от бытовой техники до домашних роботов. Камнем преткновения становится обмен информацией между компьютером и пользователем, расстёт потребность в исследовании новых способов, более естественных, чем ввод данных с клавиатуры, эмулирующих бытовое общение между людьми. Активно развивающимся направлением решения проблемы является управление компьютером с помощью голосовых команд. Тем не менее, несмотря на значительные успехи в области, этот способ применим далеко не во всех ситуациях. Другим исследуемым способом ввода данных является использование визуальных систем: передача информации посредством мимики и жестов. Для реализации последнего метода можно воспользоваться многими способами.

В данной работе идёт речь о детектирование кисти руки для дальнейшей её обработки. Детектирование позволяет находить кисть с определённой точностью, а в дальнейшем извлекать её конфигурацию и давать её описание, что позволяет не только задавать исполнение определённых команд по заранее определённым жестам, но и выполнять совершенно не характерные для обычной реализации данного метода действия.

В ходе работы исследуются различные методы и техники детектирования кисти руки человека. Важным требованием для реализации является минимизация времени задержки на обработку каждого кадра для комфортного использования в прикладных задачах.

# ПОСТАНОВКА ЗАДАЧИ

Поставленной **целью** является создание программы для детектирования кисти руки человека с помощью языка высокого уровня Python.

Для достижения цели необходимо решить ряд **задач**:

1. Сделать обзор теоретического материала по детектированию кисти руки человека и её последующего описания с помощью "особых" точек.
2. Найти и описать наиболее популярные методы обнаружения кисти человека и выделить из них самые эффективные.
3. Ознакомиться с методами извлечения конфигураций кисти человека.
4. Реализовать несколько алгоритмов данной задачи и провести их сравнение.

# 1. МЕТОДЫ ДЕТЕКТИРОВАНИЯ КИСТИ ЧЕЛОВЕКА В СИСТЕМАХ ЧЕЛОВЕКО-МАШИННОГО ВЗАИМОДЕЙСТВИЯ

Человеко-машинное взаимодействие (Human-computer interaction - HCI) - это междисциплинарное научное направление, изучающее взаимодействие между людьми и машинами. Предметом HCI является изучения, планирование и разработка методов взаимодействия человека с машиной, где в роли машины может выступать персональный компьютер, компьютерная система больших масштабов, система управления процессами и т.д. [6]. Под взаимодействием понимается любая коммуникация между человеком и машиной. Одним из методов HCI, получившим широкое распространение в последние годы, является взаимодействие, основанное на жестах человека [9, 13].

Задачу распознавания жестов руки можно разделить на подзадачи:

1. Отделение кисти руки от остальной части изображения.
2. Построение контура кисти.
3. Нахождение ключевых точек на кисти.
4. Классификация жеста исходя из статического или динамического расположения точек.

Первая подзадача, а именно детектирование кисти человека в кадре является ключевой, поскольку от качества её решения зависит качество выполнения остальных подзадач. Рассмотрим первую подзадачу, а именно детектирование кисти человека в кадре.

Существует множество решений этой подзадачи. Наиболее популярными из них являются отделение фона изображения, распознавание цвета кожи в кадре и метод Оцу. Подробно рассмотрим каждое из них.

## 1.1. Отделение фона изображения

В данном решении принимается, что в кадре движется только рука, а остальные части тела, включая фон, остаются неподвижными. Таким образом, если вначале инициализировать фон как  $bgr(x, y)$ , а новое изображение с жестом рассматривать как  $fgr(x, y)$ , то изолированный жест можно принять как разность между этими изображениями:

$$gst_i(x, y) = fgr_i(x, y) - bgr(x, y). \quad (1)$$

Полученный разностный жест переднего плана преобразуется в бинарное изображение, устанавливая соответствующий порог. Поскольку фон не является полностью статичным, например, если камера удерживается в руках оператора, то добавляется шумовая

часть. Чтобы получить изображение руки без шумов, этот метод сочетается с распознаванием кожи человека. Чтобы удалить этот шум, применяется анализ связанных компонентов, чтобы заполнить пустоты, если применяется заливка какой-либо области, а также для получения чётких краёв применяется морфологическая обработка.

Недостаток данного решения состоит в том, что довольно сложно отделить фон, даже если он задан, поскольку не ясно какие из пикселей изменились, а какие остались прежними из-за тени, смещения фокуса, изменения экспозиции и т.д. Даже если зафиксировать все параметры камеры, то тень так или иначе испортит качество решения.

Приведём два примера отделения кисти от фона изображения. На рис. 1 изображены два изображения, одно из которых является инициализированным фоном, а второе – кисть на этом фоне.

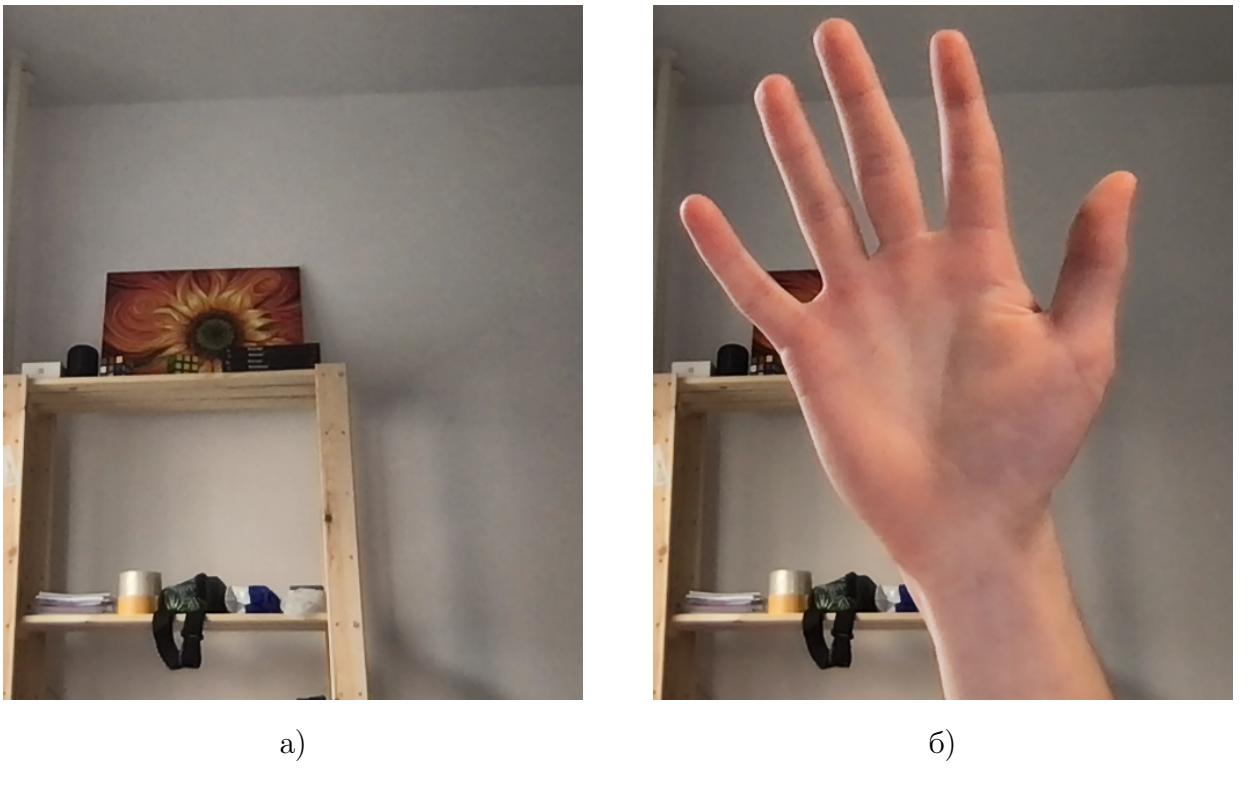


Рис. 1. Инициализированный фон (а) и изображение кисти на нём (б).

Для отделения кисти первым способом воспользуемся встроенной в библиотеку OpenCV функцией `absdiff()`.

Второй способ будет заключаться в следующем. Пусть даны два пикселя

$$p_1 = (r_1, g_1, b_1) \quad p_2 = (r_2, g_2, b_2), \quad (2)$$

тогда различие между ними будем определять как

$$D = \sqrt{(r_2 - r_1)^2 + (g_2 - g_1)^2 + (b_2 - b_1)^2}. \quad (3)$$

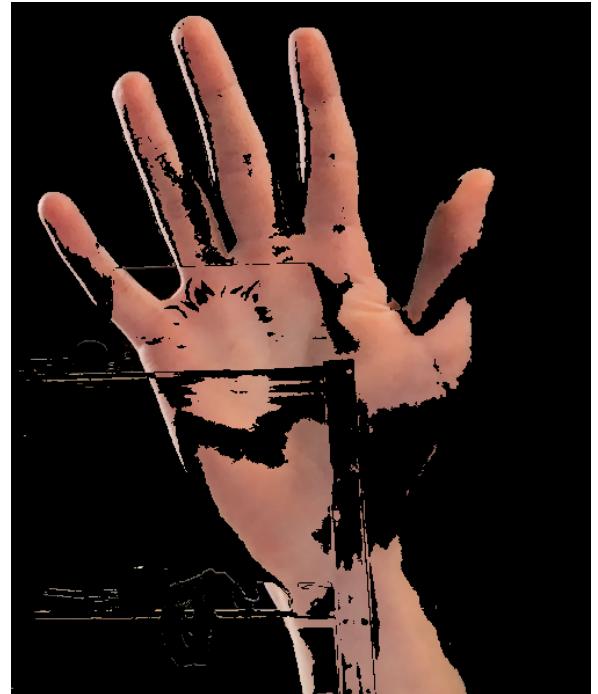
Затем создаём битовую маску  $M$ , в которой значения определяются как

$$M_{(i,j)} = \begin{cases} 1, & D > Threshold \\ 0, & D \leq Threshold \end{cases} \quad (4)$$

Результаты работы первого и второго метода представлены на рис. 2.



а)



б)

Рис. 2. Результат первого метода (а) и второго (б).

Как можно видеть, оба метода не совсем точно отделяют нужный объект от фона, поэтому рассмотрим метод распознавания кожи в HSV и YCbCr цветовых моделях.

## 1.2. Метод распознавания кожи в HSV и YCbCr цветовых моделях

Для того, чтобы детектировать цвет кожи на изображениях очень часто применяются различные цветовые модели, а именно HSV и YCbCr.

**HSV** (*Hue, Saturation, Value*) или **HSB** (*Hue, Saturation, Brightness*) – цветовая модель, в которой координатами цвета являются:

1. **Hue** – цветовой тон, (например, красный, зелёный или сине-голубой). Варьируется в пределах 0-360°, однако иногда приводится к диапазону 0-100 или 0-1.
2. **Saturation** – насыщенность. Варьируется в пределах 0-100 или 0-1. Чем больше этот параметр, тем "чище" цвет, поэтому иногда называют *чистотой цвета*. А чем ближе этот параметр к нулю, тем ближе цвет к нейтральному серому.
3. **Value** или **Brightness** – яркость. Также задаётся в пределах 0-100 или 0-1.

**YCbCr** – семейство цветовых пространств, которые используются для передачи цветных изображений в компонентном видео и цифровой фотографии. Является частью рекомендации МСЭ-R BT.601 при разработке стандарта видео Всемирной цифровой организации и фактически является масштабированной и смешённой копией YUV.

**Y** – компонента яркости, **Cb** – синий компонент цветности, **Cr** – красный компонент цветности.

Пример изображения в HSV изображён на рисунке 3.



a)



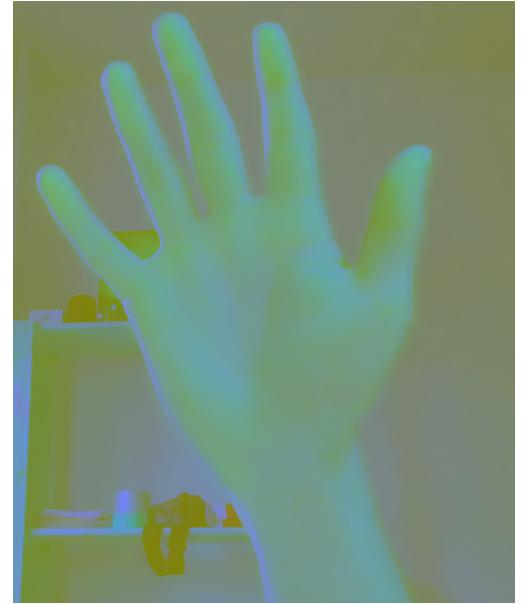
б)

Рис. 3. Изображение в RGB (а) и HSV (б).

Пример изображения в  $YCbCr$  изображен на рисунке 4.



a)



б)

Рис. 4. Изображение в RGB (а) и  $YCbCr$  (б).

Для того чтобы отделить кожу от остальной части изображения, используют определённые диапазоны составляющих цветовых моделей, которые находятся эмпирически или итеративно. В первом случае путём проб и ошибок подбираются значения составляющих. Можно заметить, что при данном подходе кожа будет иметь разные цветовые диапазоны при разном освещении. Покажем это. Зададим диапазон для изображения в HSV цветовой модели как

$$0 \leq H \leq 200,$$

$$15 \leq S \leq 255,$$

$$80 \leq V \leq 255,$$

и получим результат, представленный на рис. 5.

Можно видеть, что на первом изображении фон отделился практически идеально, но на втором видны фрагменты фона.

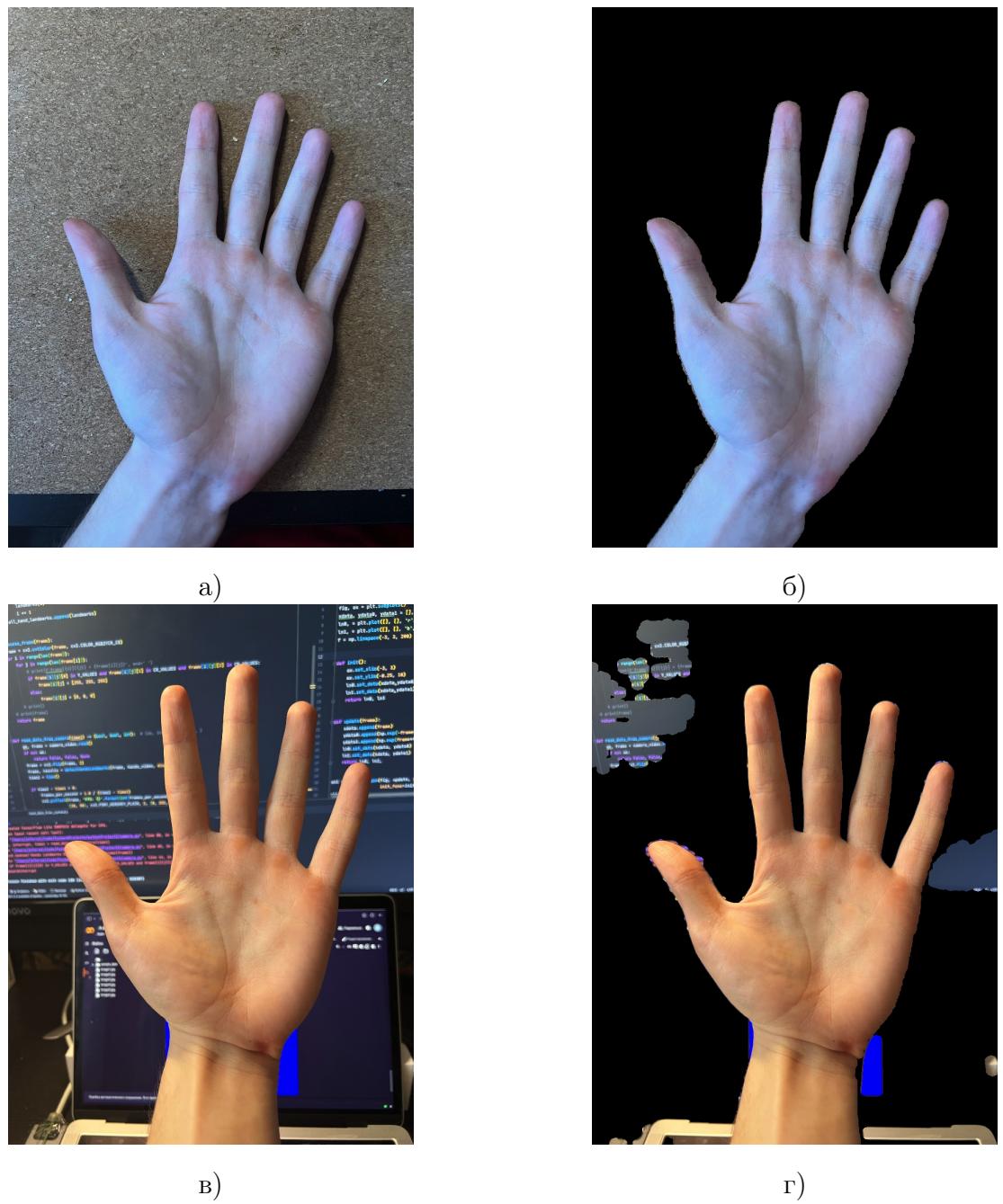


Рис. 5. Изображения с удалённым фоном в HSV.

Для цветовой модели YCbCr в статье [5] предложили два варианта диапазона компонент (5) и (6):

$$\begin{aligned}
 80 < Y &\leq 255, \\
 85 < C_b &< 135, \\
 135 < C_r &< 180
 \end{aligned} \tag{5}$$

$$Y \in \mathbb{V},$$

$$77 \leq C_b \leq 127, \quad (6)$$

$$133 \leq C_r \leq 173$$

Сравнение двух вариантов представлено на рисунке 6.

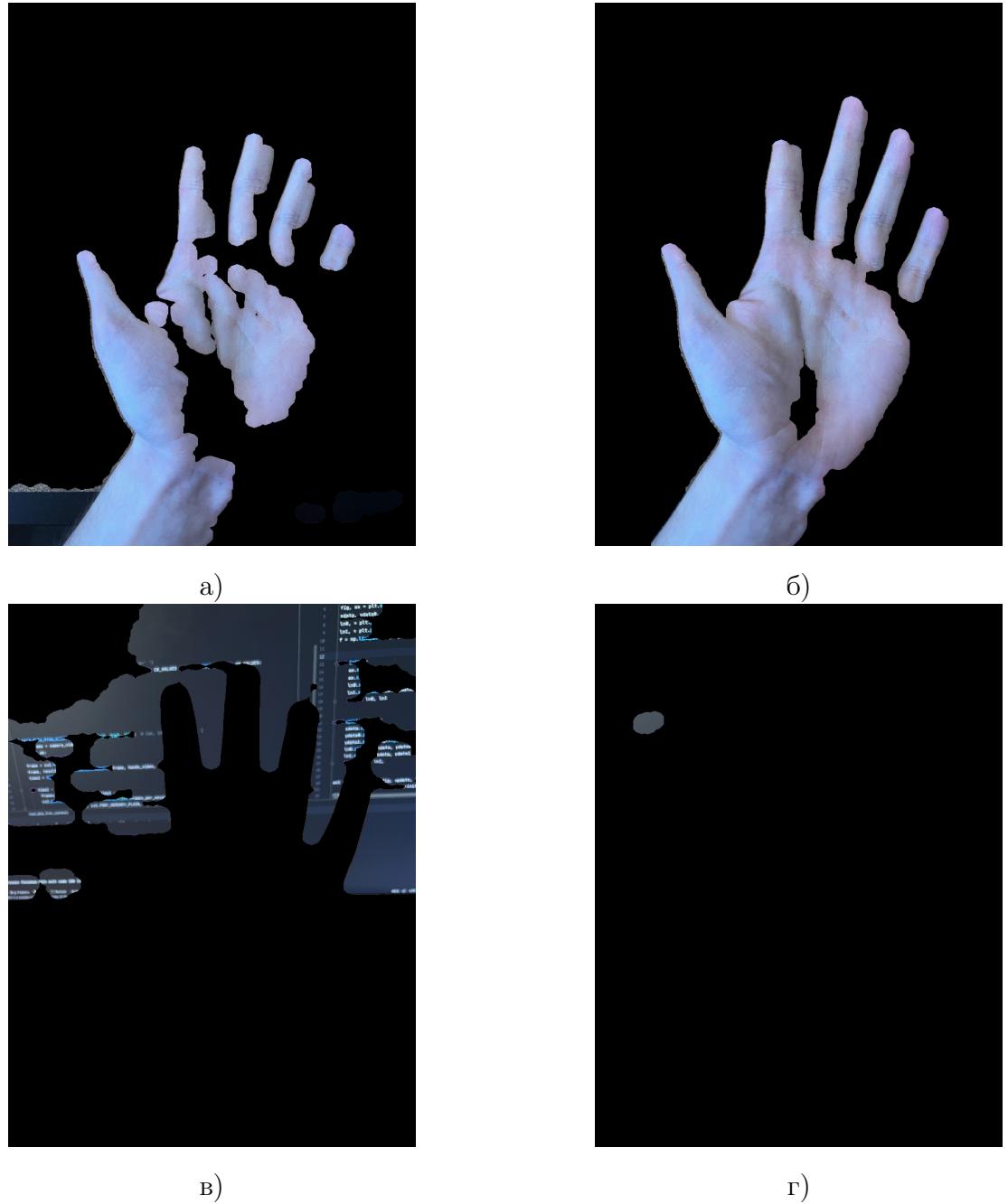


Рис. 6. Изображения с удалённым фоном в YCbCr.

Легко видеть, что оба диапазона работают не идеально, но первый показал себя намного лучше.

Таким образом, можно сделать вывод, что для детектирования цвета кожи лучше брать изображение в цветовой модели HSV.

Рассмотрим следующий метод отделения фона изображения, а именно метод Оцу.

### 1.3. Метод Оцу

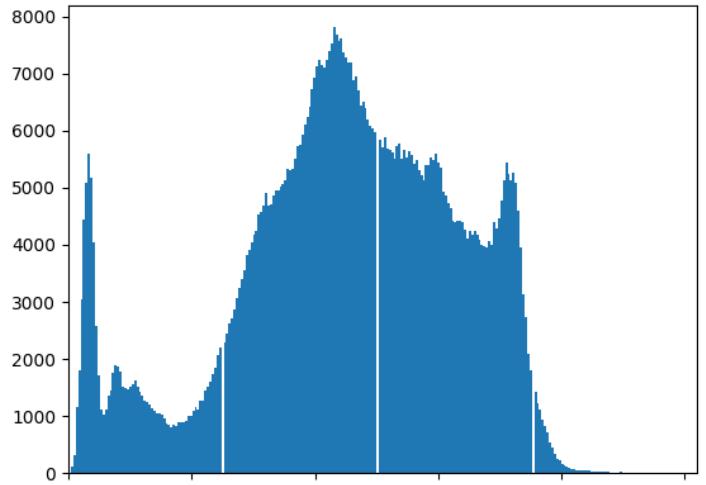
В 1979 году Нобуюки Оцу опубликовал статью [12] метода порогового разделения, основываясь на гистограмме серых цветов изображения.

Метод Оцу – это алгоритм вычисления порога бинаризации для полутонового изображения, используемый в области компьютерного распознавания образов и обработки изображений для получения чёрно-белых изображений. Алгоритм позволяет разделить пиксели двух классов ("полезные" и "фоновые"), рассчитывая такой порог, чтобы внутреклассовая дисперсия была минимальной. Для того чтобы упростить алгоритм, используется гистограмма монохромного изображения.

Диаграммы для изображений изображены на рис. 7. Как видно из рисунков, на данных изображениях довольно проблематично отделить фон от изображения какой-то единственной пороговой величиной, поэтому и алгоритм Оцу на таких изображениях сработает не очень хорошо, как это показано на рис. 8. Реализация данного алгоритма и построение гистограмм изображений находятся в приложении А.



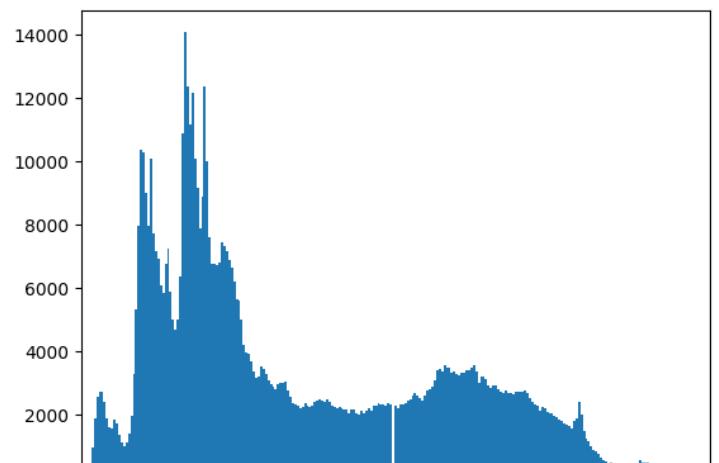
a)



б)



в)



г)

Рис. 7. Изображение 1 и 2 и их гистограммы.

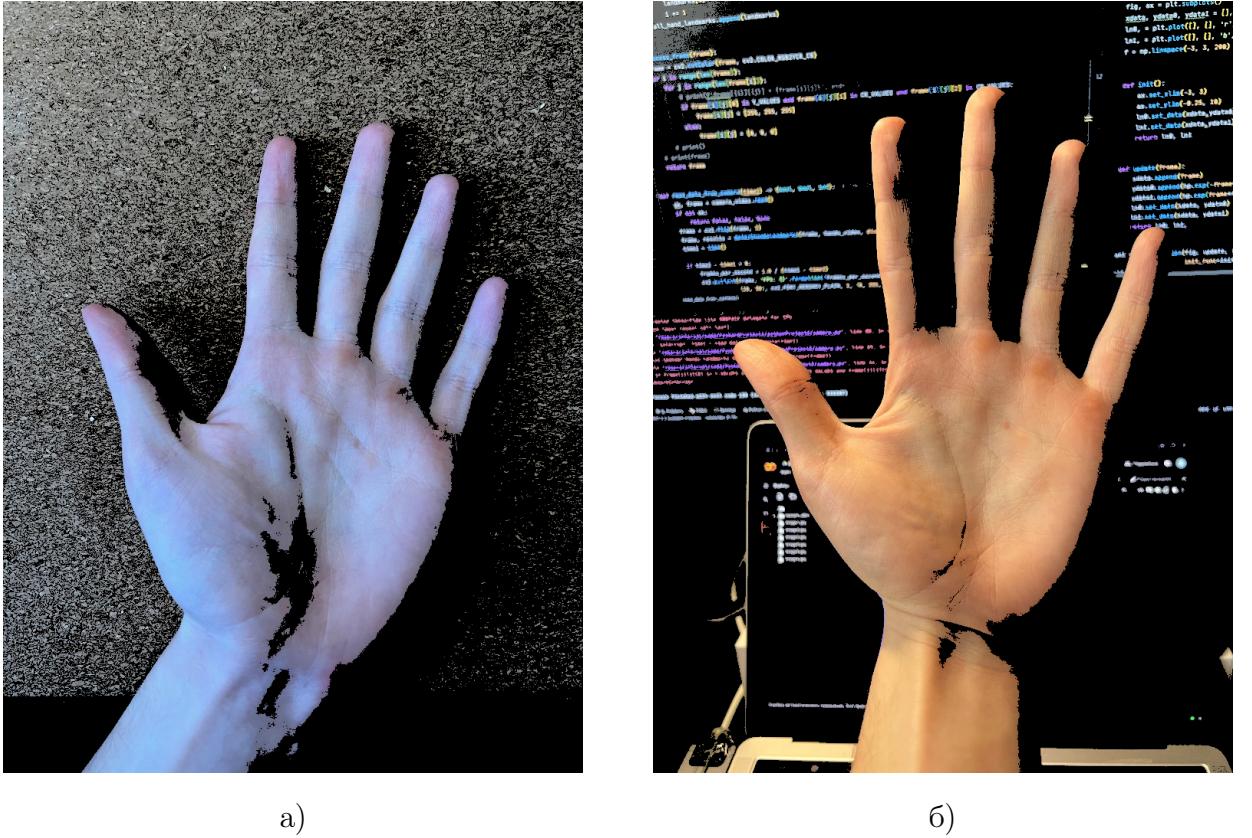


Рис. 8. Результат работы программы с алгоритмом Оцу

#### 1.4. Mixture of Gaussians

Стандартным подходом к построению модели фона, использующимся для многих прикладных задач, является смесь гауссовых распределений (Mixture of Gaussians, MoG) [10, 17]. Чаще всего для каждого пикселя текущего кадра с номером  $n$  строится функция плотности вероятности  $P_n = P(\nu_n(p))$ , и MoG используется именно этот подход. Предполагается, что для каждого пикселя текущего изображения она может быть представлена смесью нормальных распределений, где  $G$  – их число в смеси.

Для инициализации гауссиан для каждого пикселя чаще всего применяют либо EM-алгоритм (Expectation-maximization algorithm), либо k-means, что достаточно затратно в вычислительном плане. Число входящих в смесь распределений  $G$  обычно принимают равным от 3 до 5. Также существует подход, позволяющий автоматически подбирать необходимое количество гауссиан [17].

В библиотеке OpenCV данный метод реализуется с помощью встроенной функции `createBackgroundSubtractorMOG2()`. Данная функция обучается на изображениях каждого кадра и лучше всего работает в режиме последовательности, то есть на видео. Результат работы метода MoG представлен на рисунке 9.



Рис. 9. Результат отделения фона от изображения с помощью метода MoG.

В коде видно, что сначала происходит обучение на первых изображениях фона, а затем добавляется новый кадр, на котором стирается фон.

Рассмотрев наиболее популярные методы отделения кисти человека от фона изображения, можно сделать вывод, что некоторые из методов работают наилучшим образом при специфических условиях, поэтому необходимо их дополнительное исследование.

## 2. МЕТОДЫ ПОСТРОЕНИЯ КОНТУРА ОБЪЕКТА НА ИЗОБРАЖЕНИИ

Отслеживание границ – один из основных методов обработки оцифрованных двоичных изображений. Он производит последовательность координат или цепных кодов от границ между связным компонентом.

### 2.1. Выделение границ с помощью оператора Кэнни

Оператор Кэнни (детектор границ Кэнни или алгоритм Кэнни) – оператор обнаружения границ изображения. Кэнни изучил математическую проблему получения фильтра, оптимального по критериям выделения, локализации и минимизации нескольких откликов одного края. Он показал, что искомый фильтр является суммой четырёх экспонент. Он также показал, что этот фильтр может быть хорошо приближен первой производной Гауссианы. Кэнни ввёл понятие *подавление немаксимумов* (Non-Maximum Suppression), которое означает, что пикселями границ объявляются пиксели, в которых достигается локальный максимум градиента в направлении вектора градиента.

Целью Кэнни было разработать оптимальный алгоритм обнаружения границ, удовлетворяющий трём критериям:

- хорошее обнаружение (повышение отношения сигнал/шум);
- хорошая локализация (правильное определение положения границы);
- единственный отклик на одну границу.

Из этих критериев затем строится целевая функция стоимости ошибок, минимизацией которой находится "оптимальный" линейный оператор для свёртки с изображением.

Первым этапом алгоритма является *сглаживание*, то есть размытие изображения для удаления шума. Оператор Кэнни использует фильтр, который может быть хорошо приближен к первой производной гауссианы. Для  $\sigma = 1.4$ :

$$B = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \cdot A. \quad (7)$$

Следующим этапом является *поиск градиентов*. Границы отмечаются там, где градиент изображения приобретает максимальное значение. Они могут иметь различное направление, поэтому алгоритм Кэнни использует четыре фильтра для обнаружения гори-

зонтальных, вертикальных и диагональных ребер в размытом изображении.

$$G = \sqrt{G_x^2 + G_y^2}, \Theta = \arctan\left(\frac{G_y}{G_x}\right). \quad (8)$$

Угол направления вектора градиента при этом округляется и может принимать такие значения:  $0^\circ$ ,  $45^\circ$ ,  $90^\circ$ ,  $135^\circ$ .

Затем происходит *подавление немаксимумов*, когда только локальные максимумы отмечаются как границы, *двойная пороговая фильтрация* – потенциальные границы определяются порогами и *трассировка области неоднозначности*, когда итоговые границы определяются путём подавления всех краёв, не связанных с определёнными (сильными) границами.

Перед применением детектора обычно преобразуют изображение в оттенки серого, чтобы уменьшить вычислительные затраты.

Пример работы оператора Кэнни в цветовых моделях RBG, GRAY, HSV и YCbCr показан на рис. 10.

Можно видеть, что алгоритм Кэнни довольно неплохо позволяет определить границы объекта. Но для задачи извлечения конфигурации кисти необходимо извлекать особые точки изображения, а этот метод лишь выделяет границы на изображении, не определяя сам контур. В связи с этим необходимо рассмотреть топологический структурный анализ цифрового бинарного изображения с помощью отслеживания границ.

## 2.2. Топологический структурный анализ цифрового бинарного изображения с помощью отслеживания границ

Этот метод был разработан Сатоши Сузуки и Кейчи Эйбом в 1985 году [14]. Алгоритм предполагает нахождение контуров с учетом вложенности, то есть способен определить, когда в контур одного объекта вложен другой. Реализация данного алгоритма лежит в основе функции `findContours()` в библиотеке OpenCV, предназначеннай для исследования и решения задач, связанных с компьютерным зрением.

Для отрисовки контуров, полученных с помощью данной функции, можно воспользоваться функцией `drawContours()`.

Поскольку функция `findContours()` находит все контуры на изображении, то нужно среди них выбрать один единственный. Пусть кисть занимает наибольшее пространство на изображении и, соответственно, имеет наибольший контур. Среди всех контуров будем отбирать тот, *площадь* которого имеет наибольшее значение. Площадь контура будем находить с помощью *формулы площади Гаусса*.

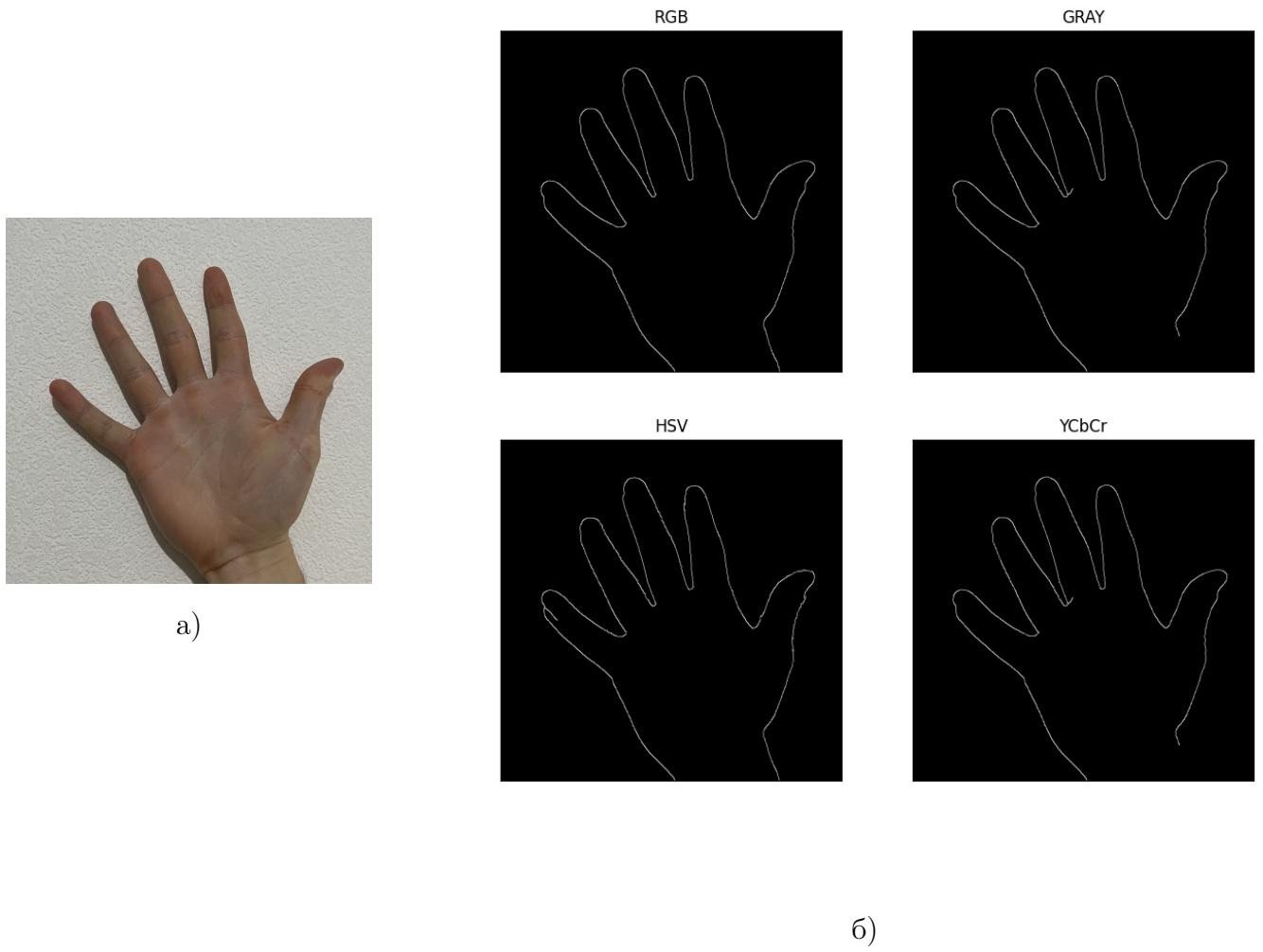


Рис. 10. Пример работы оператора Кэнни.

### 2.2.1. Формула площади Гаусса

Формула площади Гаусса (формула землемера или формула шнурования или алгоритм шнурования) — формула определения площади простого многоугольника, вершины которого заданы декартовыми координатами на плоскости. В формуле векторным произведением координат и сложением определяется площадь области, охватывающей многоугольник, а затем из нее вычитается площадь окружающего многоугольника, что дает площадь многоугольника внутри.

Формула может быть представлена следующим выражением:

$$S = \frac{1}{2} \left| \sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n \right| = \\ = \frac{1}{2} |x_1 y_2 + x_2 y_3 + \cdots + x_{n-1} y_n + x_n y_1 - x_2 y_1 - x_3 y_2 - \cdots - x_n y_{n-1} - x_1 y_n| , \quad (9)$$

где

$S$  — площадь многоугольника,

$n$  – количество сторон многоугольника,

$(x_i, y_i), i = \overline{1, n}$  – координаты вершин многоугольника.

Другие представления этой же формулы:

$$\begin{aligned} S &= \frac{1}{2} \left| \sum_{i=1}^n x_i(y_{i+1} - y_{i-1}) \right| = \frac{1}{2} \left| \sum_{i=1}^n y_i(x_{i+1} - x_{i-1}) \right| = \\ &= \frac{1}{2} \left| \sum_{i=1}^n x_i y_{i+1} - x_{i+1} y_i \right| = \frac{1}{2} \left| \sum_{i=1}^n \det \begin{pmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{pmatrix} \right|, \end{aligned} \quad (10)$$

где

$$x_{n+1} = x_1, \quad x_0 = x_n,$$

$$y_{n+1} = y_1, \quad y_0 = y_n.$$

Рассмотрим пример использования данных функций. Сравним результат поиска контуров с предварительной обработкой изображения, заключающейся в отделении объекта от фона, и без обработки (только с переводом изображения в черно-белый формат) и получим результаты, представленные на рис. 11.

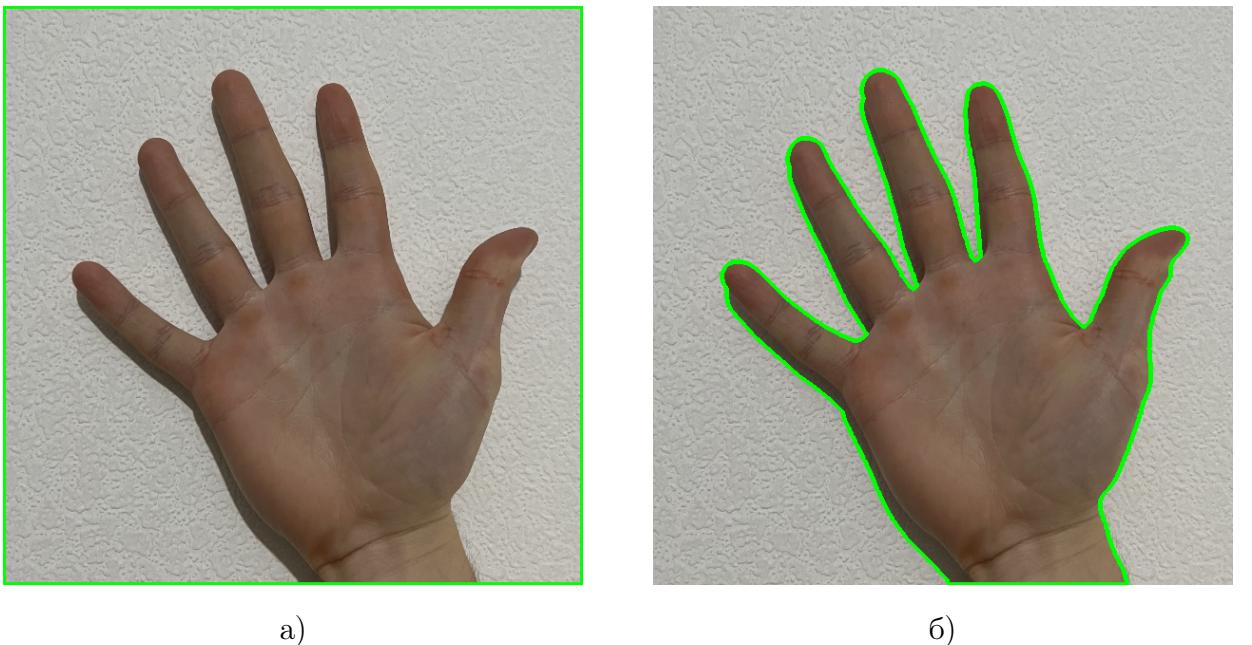


Рис. 11. Пример поиска контура на изображении без предварительной обработки (а) и с предварительной обработкой с помощью метода Оцу (б).

Можно видеть, что пороговая бинаризация изображения методом Оцу позволила с высокой точностью определить истинное расположение контура ладони. Это означает, что для дальнейшего исследования контуров необходима предобработка изображения.

### 3. МЕТОДЫ НАХОЖДЕНИЯ КЛЮЧЕВЫХ ТОЧЕК НА КИСТИ

Особую сложность представляет задача определения положения ключевых точек на кисти человека. Для этой нетривиальной задачи было предложено несколько методов её решения. Рассмотрим каждый из них.

#### 3.1. Нахождение точек путём определения дефектов выпуклости

В предыдущем разделе был рассмотрен метод построения контура кисти с помощью топологического структурного анализа цифрового бинарного изображения с помощью отслеживания границ. Этот метод хорош тем, что позволяет позволять определить точные координаты точек контура. С помощью этих точек существует возможность построить выпуклую оболочку или их *наименьшее множество*.

Выпуклой оболочкой множества  $X$  называется наименьшее выпуклое множество, содержащее  $X$ . «Наименьшее множество» здесь означает наименьший элемент по отношению к вложению множеств, то есть такое выпуклое множество, содержащее данную фигуру, что оно содержится в любом другом выпуклом множестве, содержащем данную фигуру.

Рассмотрим основные алгоритмы построения выпуклой оболочки.

##### 3.1.1. Алгоритм Грэхема

Алгоритм Грэхема [7] — алгоритм построения выпуклой оболочки в двумерном пространстве. В этом алгоритме задача о выпуклой оболочке решается с помощью стека, сформированного из точек-кандидатов. Все точки входного множества заносятся в стек, а потом точки, не являющиеся вершинами выпуклой оболочки, со временем удаляются из него. По завершении работы алгоритма в стеке остаются только вершины оболочки в порядке их обхода против часовой стрелки.

Временная сложность алгоритма =  $O(n \log n)$ .

**Описание алгоритма:**

Пусть точки  $p = [p_0, \dots, p_{n-1}]$  — входной массив точек.

1. Найти самую "нижнюю" точку в массиве (ту, в которой наименьшая среди всех координата  $y$ ). Если таких точек несколько, то среди них выбрать точку с наименьшей координатой  $x$ . Найденная точка  $P_0$  является первой точкой выпуклой оболочки;

2. Перебрать остальные  $n - 1$  точек и отсортировать их по полярному углу относительно  $P_0$  в направлении против часовой стрелки. Если полярный угол нескольких точек одинаков, то выбрать ближайшую к  $P_0$ ;
3. После сортировки, проверить, есть ли точки с одинаковым полярным углом. Если да, то удалить все эти точки, кроме ближайшей к  $P_0$ . Положить размер нового массива равным  $m$ ;
4. Если  $m < 3$ , то алгоритм прерывается. Выпуклую оболочку определить невозможно;
5. Создать пустой стек  $S$  и положить в него первые три точки нового массива  $p_0, p_1, p_2$ ;
6. Для каждой из оставшихся  $m - 3$  точек:
  - 6.1. Удаляем точки из стека пока ориентация трёх следующих точек не против часовой стрелки (они не совершают левый поворот): точка наверху стека, точка следующая после неё и текущая точка  $p_i$ ;
  - 6.2. Добавляем точку  $p_i$  в  $S$ ;
7. Стек  $S$  содержит точки выпуклой оболочки.

Реализация алгоритма Грэхема на языке Python находится в приложении Б. Протестировав алгоритм на случайном наборе точек, получаем результат, представленный на рисунке 12.

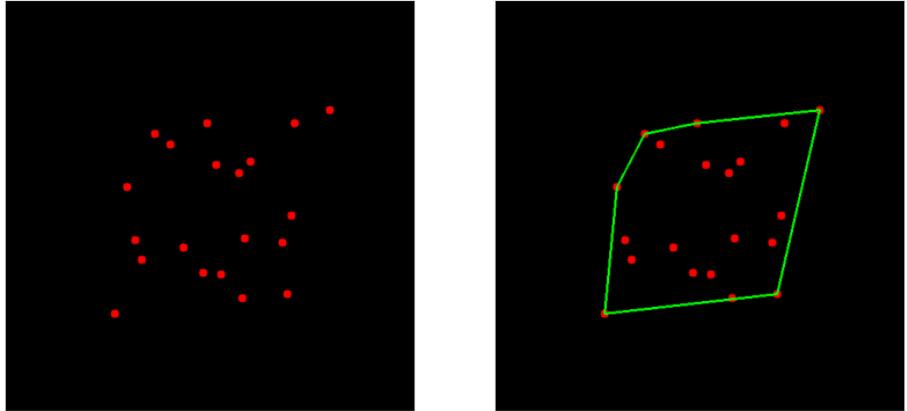


Рис. 12. Сгенерированные точки (слева) и их выпуклая оболочка, найденная с помощью алгоритма Грэхема (справа).

### 3.1.2. Алгоритм Джарвиса

Алгоритм Джарвиса [8] (или алгоритм обхода Джарвиса, или алгоритм заворачивания подарка) определяет последовательность элементов множества, образующих выпук-

лую оболочку для этого множества. Метод можно представить как обтягивание верёвкой множества вбитых в доску гвоздей. Алгоритм работает за время  $O(nh)$ , где  $n$  – общее число точек на плоскости, а  $h$  – число точек в выпуклой оболочке. В худшем случае –  $O(n^2)$ , когда все точки попадают в выпуклую оболочку.

### Описание алгоритма:

Пусть дано множество точек  $P = \{p_1, p_2, \dots, p_n\}$ . В качестве начальной берётся самая левая нижняя точка  $p_1$  (её можно найти за  $O(n)$  обычным проходом по всем точкам), она точно является вершиной выпуклой оболочки. Следующей точкой ( $p_2$ ) берём такую точку, которая имеет наименьший положительный полярный угол относительно точки  $p_1$  как начала координат. После этого для каждой точки  $p_i$  ( $2 < i \leq |P|$ ) против часовой стрелки ищется такая точка  $p_{i+1}$ , путём нахождения за  $O(n)$  среди оставшихся точек (+ самая левая нижняя), в которой будет образовываться наибольший угол между прямыми  $p_{i-1}p_i$  и  $p_ip_{i+1}$ . Она и будет следующей вершиной выпуклой оболочки. Сам угол при этом не ищется, а ищется только его косинус через скалярное произведение между лучами  $p_{i-1}p_i$  и  $p_ip_{i+1}$ , где  $p_i$  – последний найденный минимум,  $p_{i-1}$  – предыдущий минимум, а  $p'_{i+1}$  – претендент на следующий минимум. Новым минимумом будет та точка, в которой косинус будет принимать наименьшее значение (чем меньше косинус, тем больше его угол). Нахождение вершин выпуклой оболочки продолжается до тех пор, пока  $p_{i+1} \neq p_1$ . В тот момент, когда следующая точка в выпуклой оболочке совпала с первой, алгоритм останавливается — выпуклая оболочка построена (рис. 13).

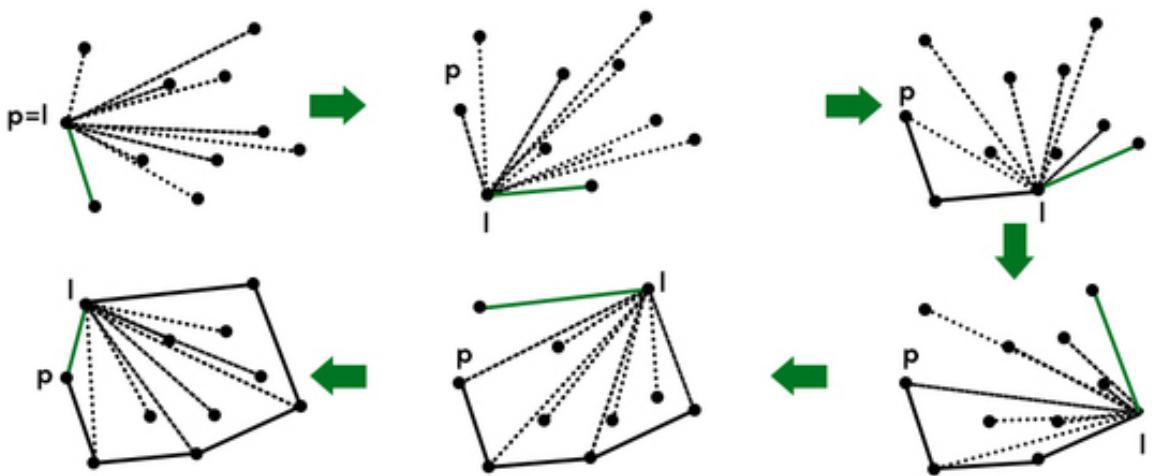


Рис. 13. Пример работы алгоритма Джарвиса.

Реализация алгоритма Джарвиса на языке Python находится в приложении В. Также протестируем алгоритм Джарвиса на случайном наборе точек и получим результат, представленный на рисунке 14.

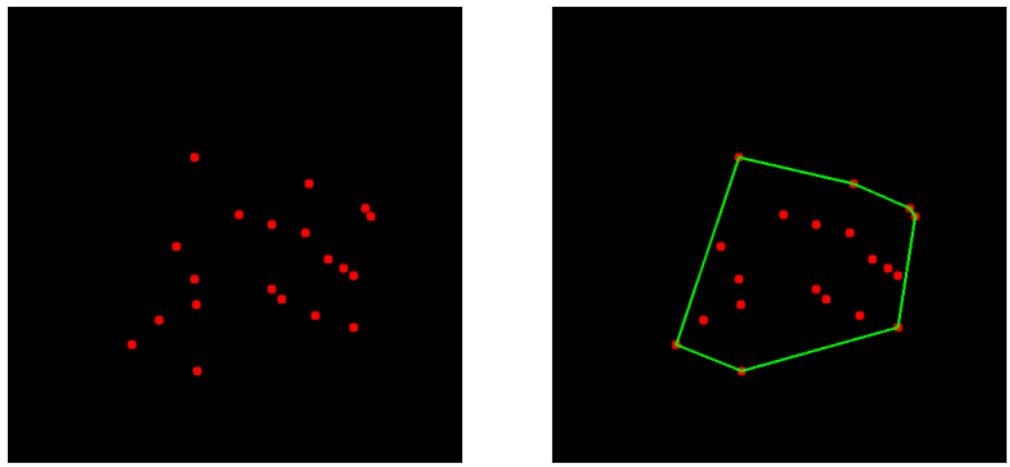


Рис. 14. Сгенерированные точки (слева) и их выпуклая оболочка, найденная с помощью алгоритма Джарвиса (справа).

Можно видеть, что алгоритм отработал верно, но у него есть один существенный недостаток – он намного медленнее по сравнению с алгоритмом Грэхема.

### 3.1.3. Алгоритм Киркпатрика

Алгоритм Киркпатрика [11] заключается в построении выпуклой оболочки методом "разделяй и властвуй".

#### Описание алгоритма:

Дано множество  $S$ , состоящее из  $N$  точек.

1. Если  $N \leq N_0$  ( $N_0$  – некоторое небольшое целое число), то построить выпуклую оболочку одним из известных методов и остановиться, иначе перейти к шагу 2.
2. Разобьём исходное множество  $S$  произвольным образом на два примерно равных по мощности подмножества  $S_1$  и  $S_2$  (пусть  $S_1$  содержит  $N/2$  точек, а  $S_2$  содержит  $N - N/2$  точек).
3. Рекурсивно находим выпуклые оболочки каждого из подмножеств  $S_1$  и  $S_2$ .
4. Строим выпуклую оболочку исходного множества как выпуклую оболочку объединения двух выпуклых многоугольников  $CH(S_1)$  и  $CH(S_2)$ .

Поскольку  $CH(S) = CH(S_1 \cup S_2) = CH(CH(S_1) \cup CH(S_2))$ , сложность этого алгоритма является решением рекурсивного соотношения  $T(N) \leq 2T(N/2) + f(N)$ , где  $f(N)$  – время построения выпуклой оболочки объединения двух выпуклых многоугольников, каждый из

которых имеет около  $N/2$  вершин. Таким образом, временная сложность этого алгоритма равна  $O(N \log N)$ .

Реализация алгоритма Киркпатрика на языке Python находится в приложении Г. Результат работы показан на рисунке 15.

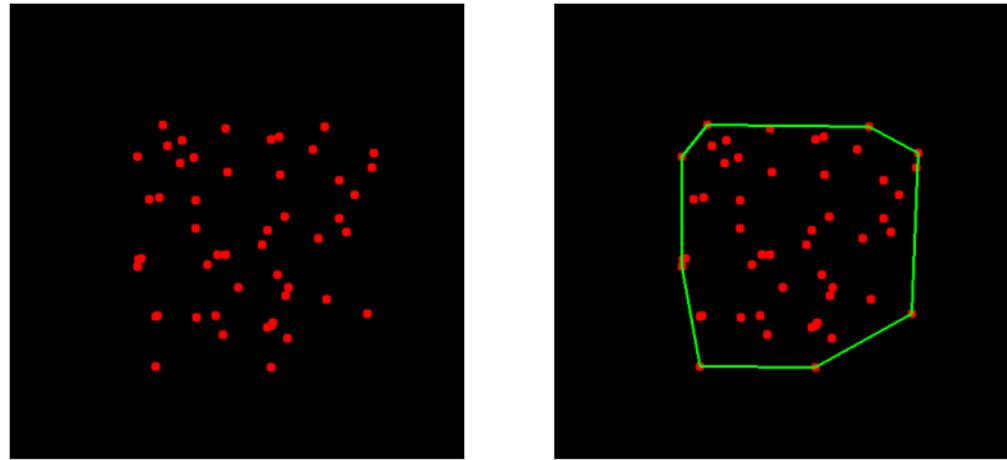


Рис. 15. Сгенерированные точки (слева) и их выпуклая оболочка, найденная с помощью алгоритма Киркпатрика (справа).

### 3.1.4. Сравнение алгоритмов

Для того чтобы выбрать наиболее быстрый алгоритм, проведём их сравнение по времени работы на 1000 тестовых случайно сгенерированных данных.

Результаты работы сведены в таблицу 1.

Алгоритм	Максимальное время, мс	Минимальное время, мс	Среднее время, мс
Грэхема	5317	581	2193.0
Джарвиса	187204	5979	66401.0
Киркпатрика	23102	2008	6944.0

Таблица 1. Сравнение алгоритмов построения выпуклой оболочки.

Можно видеть, что самым эффективным в данном случае оказался алгоритм Грэхема, именно его и будем использовать.

Алгоритм Грэхема уже реализован в библиотеке OpenCV в виде функции `convexHull()`.

Результат построения контура и выпуклой оболочки кисти руки изображён на рисунке 16.

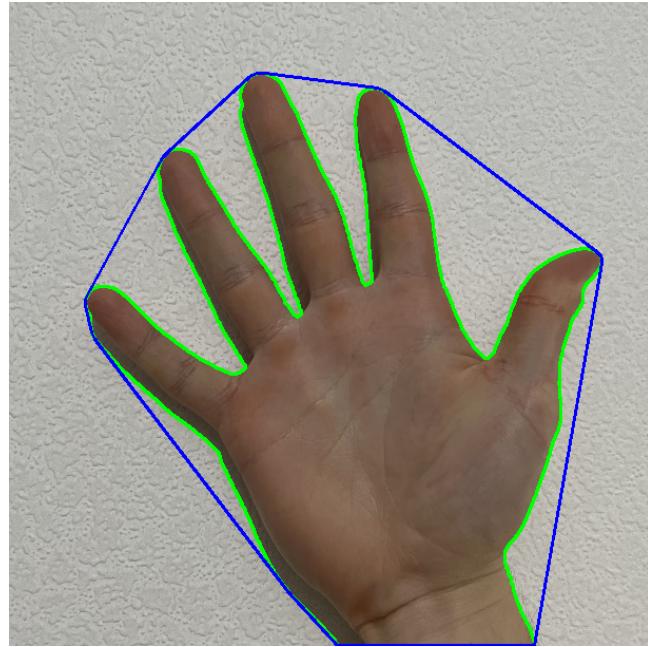


Рис. 16. Выделение контура и выпуклой его оболочки на кисти.

Для определения необходимых ключевых точек на кисти необходимо найти *дефекты выпуклости* контура и выпуклой оболочки.

Дефект выпуклости в данном случае – это максимальное расстояние между выпуклой оболочкой и контуром (рис. 17).

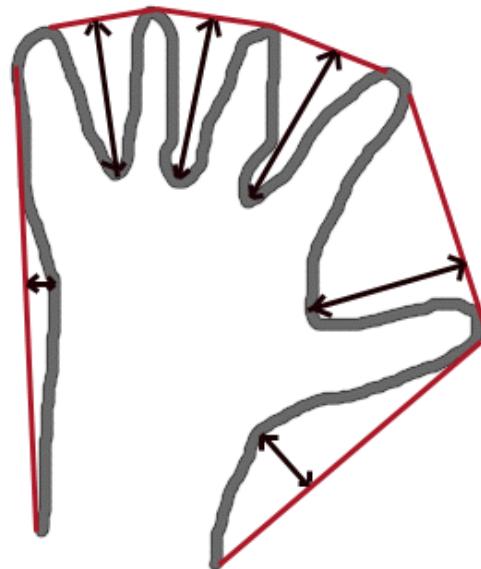


Рис. 17. Дефекты выпуклости.

В библиотеке OpenCV уже существует реализация расчётов дефектов выпуклости в виде функции `convexityDefects()`.

Результат расчёта дефектов выпуклости представлен на рисунке 18.

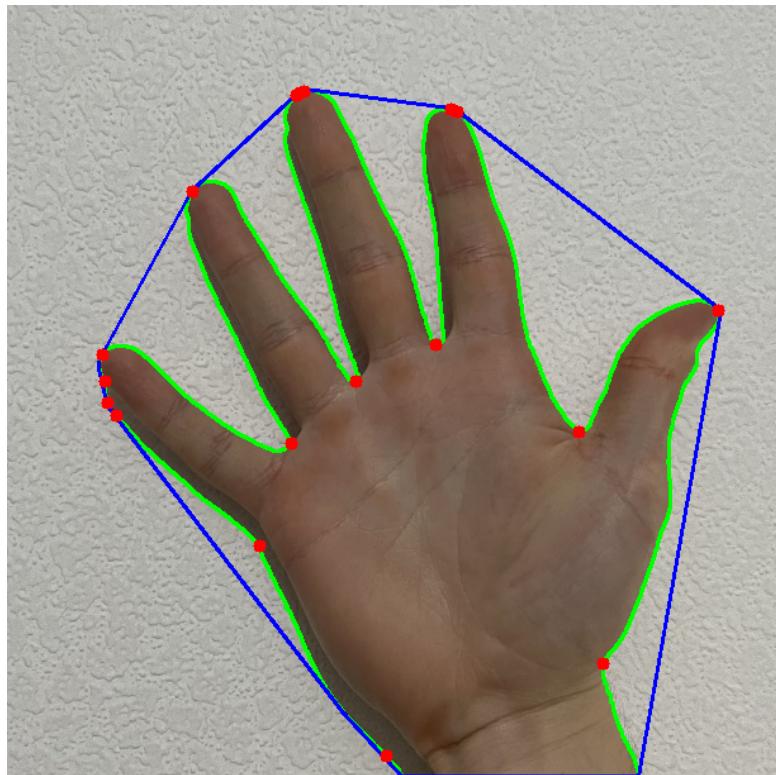


Рис. 18. Пример вычисления точек дефектов выпуклости (красные).

Можно видеть, что некоторые точки очень близко находятся друг к другу. Это связано с тем, что выпуклая оболочка касается контура несколько раз в почти одном и том же месте. Чтобы избавиться от этого напишем функцию очистки этих точек (приложение Д).

Функция работает следующим образом. Берётся первая попавшаяся точка дефекта выпуклости, затем перебираются оставшиеся, пока они попадают в круг радиусом  $\text{ALPHA}$  с центром в первой точке. Эти точки не попадают в итоговый результат. Затем берётся следующая точка и т.д. Если после прохождения всех точек их количество больше 7 (2 точки снизу руки + максимум 5 точек-“пальцев”), то  $\text{ALPHA}$  инкрементируется.

Результаты работы программы поиска дефектов выпуклости с очисткой “ненужных” точек выпуклости можно видеть на рисунке 19.

Для того чтобы окончательно найти все точки на руке необходимо найти центр масс контура.

*Центром масс* фигуры является арифметическое среднее всех точек фигуры. Пусть фигура состоит из  $n$  отдельных точек  $x_1, \dots, x_n$ , тогда центр масс определяется как

$$c = \frac{1}{n} \sum_{i=1}^n x_i.$$

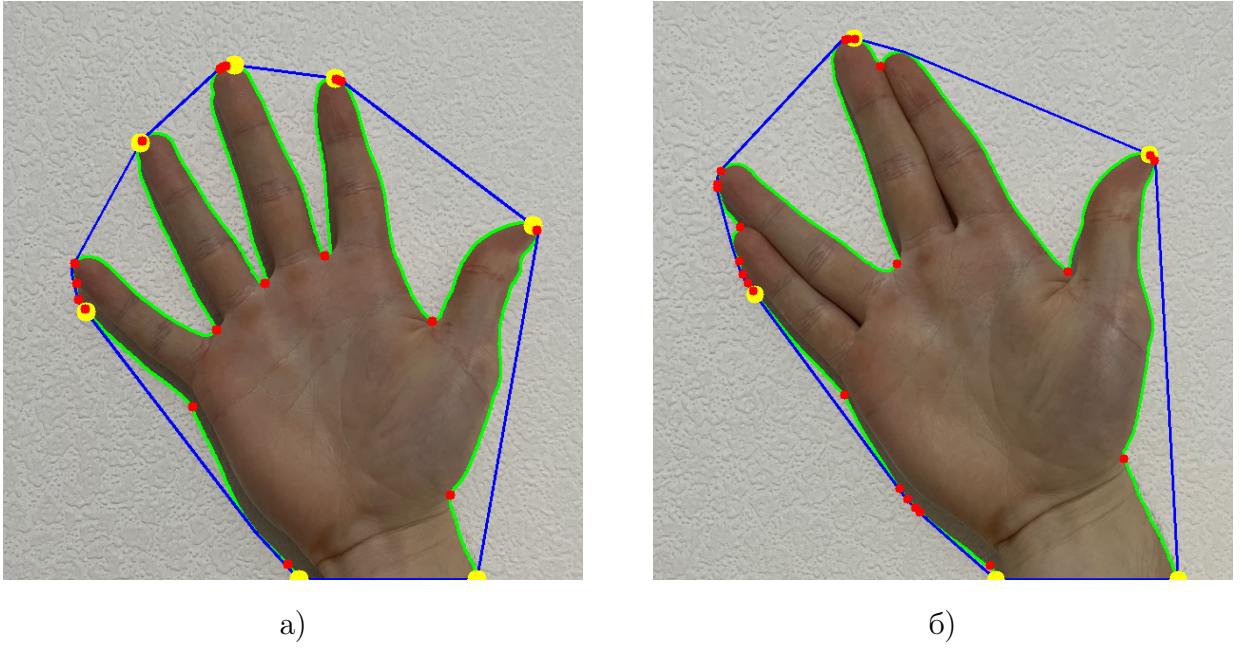


Рис. 19. Результат очистки дефектов выпуклости (оставшиеся точки отмечены на рисунках желтым цветом).

В контексте обработки изображения и компьютерного зрения каждая фигура состоит из пикселей и центром масс является взвешенное среднее всех пикселей, составляющих фигуру.

Центр масс контура можно найти с помощью момента. Момент изображения – это конкретное средневзвешенное значение интенсивности пикселей изображения. Как и во всех остальных науках, моменты характеризуют распределение материи относительно точки или оси. Формула для момента изображения выглядит следующим образом:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y), \quad (11)$$

где  $I(x, y)$  – интенсивность пикселя в точке  $(x, y)$ ,  $x, y$  относятся к строке и столбцу изображения.

Проблема формулы (11) состоит в том, что моменты чувствительны к позициям  $x$  и  $y$ . Если есть необходимость в определении моментов, независимых к месту расположения контура, то нужно использовать формулу *центральных моментов*:

$$M_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q I(x, y),$$

где  $\bar{x}$  и  $\bar{y}$  – средние значения  $x$  и  $y$  соответственно.

Координата центра масс изображения таким образом будет определяться как:

$$\begin{aligned} C_x &= \frac{M_{10}}{M_{00}}, \\ C_y &= \frac{M_{01}}{M_{00}}, \end{aligned} \quad (12)$$

где  $(C_x, C_y)$  – координата центра масс изображения.

В библиотеке OpenCV есть функция `moments()`, определяющая моменты изображения. Добавив в предыдущий код определение координаты центра масс и построение лучей, получим результат, отображённый на рисунке 20.

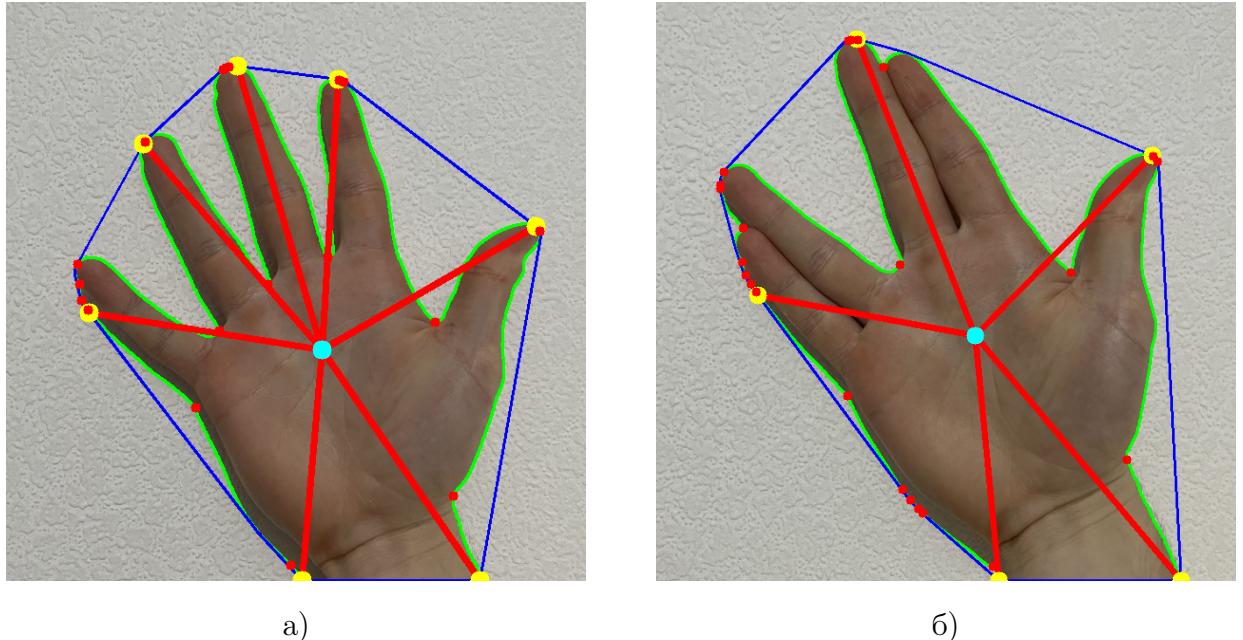


Рис. 20. Результат поиска ключевых точек руки.

### 3.2. Локализация ключевых точек кисти руки на изображении на основе непрерывного скелета

Для локализации ключевых точек в статье [3] предлагается использовать непрерывный скелет. Предполагается, что уже успешно был выполнен этап сегментации и имеется отсегментированное изображение с силуэтом кисти руки. На основе контурного представления силуэта жеста строится его скелет. Для определения скелета используется понятие *максимального пустого круга*.

**Определение 1.** Для многоугольной фигуры  $F$  максимальным пустым кругом называется всякий круг  $B$ , полностью содержащийся внутри фигуры  $F$ , такой, что любой другой круг  $B'$ , содержащийся внутри фигуры  $F$ , не содержит в себе  $B$ .

**Определение 2.** Скелетом многоугольной фигуры  $F$  является множество центров её максимальных пустых кругов.

Непрерывный скелет многоугольной фигуры является подмножеством диаграммы Воронова [1]. Совокупность общих линий всех пар несмежных ячеек диаграммы Воронова образуют ветви скелета [1]. На скелете определена радиальная функция  $R(x, y)$ , ставящая

в соответствие каждой точке скелета  $(x, y)$  значение радиуса максимального пустого круга с центром в этой точке.

В большинстве случаев скелет ладони имеет шумы в виде малозначимых ветвей, которые как правило, мешают дальнейшему анализу. Для удаления шумовых ветвей используется дополнительная обработка, называемая "стрижкой" [1]. Процесс "стрижки" заключается в удалении ветвей, граничащих с контурами силуэта руки.

Существующие эффективные алгоритмы позволяют выполнять построение скелета за время  $O(N \log N)$ , где  $N$  – число вершин в многоугольнике [2]. В связи с тем, что скорость построения скелета напрямую зависит от количества углов многоугольной фигуры, то для ускорения построения скелета можно применить аппроксимацию этой фигуры [4].

Демонстрация процесса построения скелета представлена на рисунке 21. На основе непрерывного скелета и радиальной функции  $R(x, y)$  можно с большой точностью вычислить координаты кончиков пальцев и координаты центра ладони. Каждый палец может принимать два условных состояния: сжатый в кулак или разжатый. Все ветви скелета, соответствующие пальцу, оканчиваются вершиной степени 1. Ветвь пальца можно разделить на две части: палец и пясть.

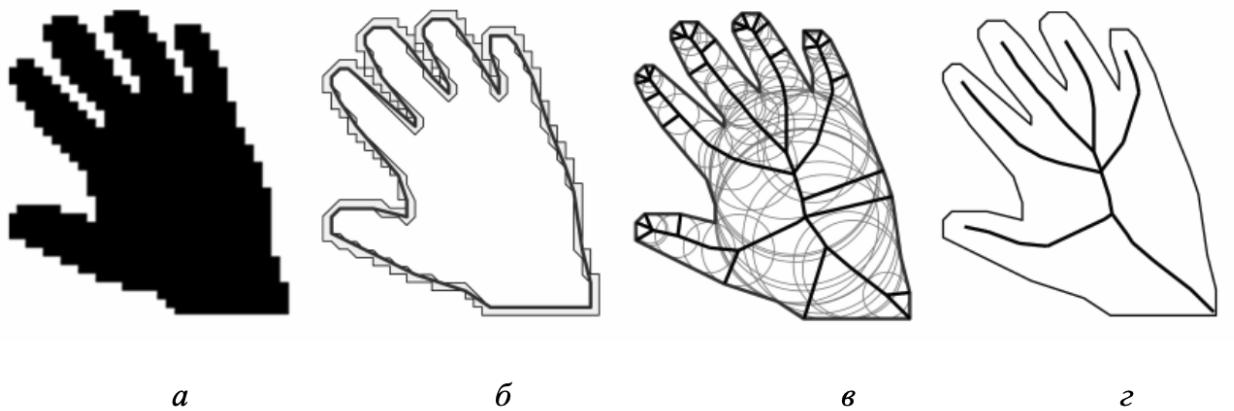


Рис. 21. Процесс построения скелета: *а* – исходное изображение; *б* – аппроксимированное изображение; *в* – скелет многоугольника; *г* – скелет после стрижки

Для классификации ветвей пальцев используется набор эвристических правил:

1. Ветвь пальца лежит на графе между вершинами со степенями 1 и 3.
2. Радиальная функция ветви на вершине степени 1 увеличивается более чем в 2,5 раза по сравнению с вершиной степени 3.
3. Радиальная функция начинает резко расти, то есть частные производные  $R'$  больше заданного порога.

Первая точка на ветви, где производная радиальной функции превышает заданный порог, является точкой конца пальца. Центром ладони будем считать точку, лежащую

на скелете ладони, радиальная функция которой принимает максимальное значение. На рисунке 22 демонстрируется результат вычисления ключевых точек на изображении.

Для распознавания простого, ограниченного набора жестов достаточно составить набор эвристических правил, основанных на следующих данных: количество пальцев, их длина, количество циклов в графе и их габариты. В более сложных случаях набора эвристических правил мало, и для распознавания жестов применяются дескрипторы формы кисти руки, состоящие из определённых инвариантных признаков.

Достоинством метода распознавания жестов на основе непрерывного скелета является его быстродействие, высокая точность локализации особых точек и, как следствие, высокий результат распознавания.

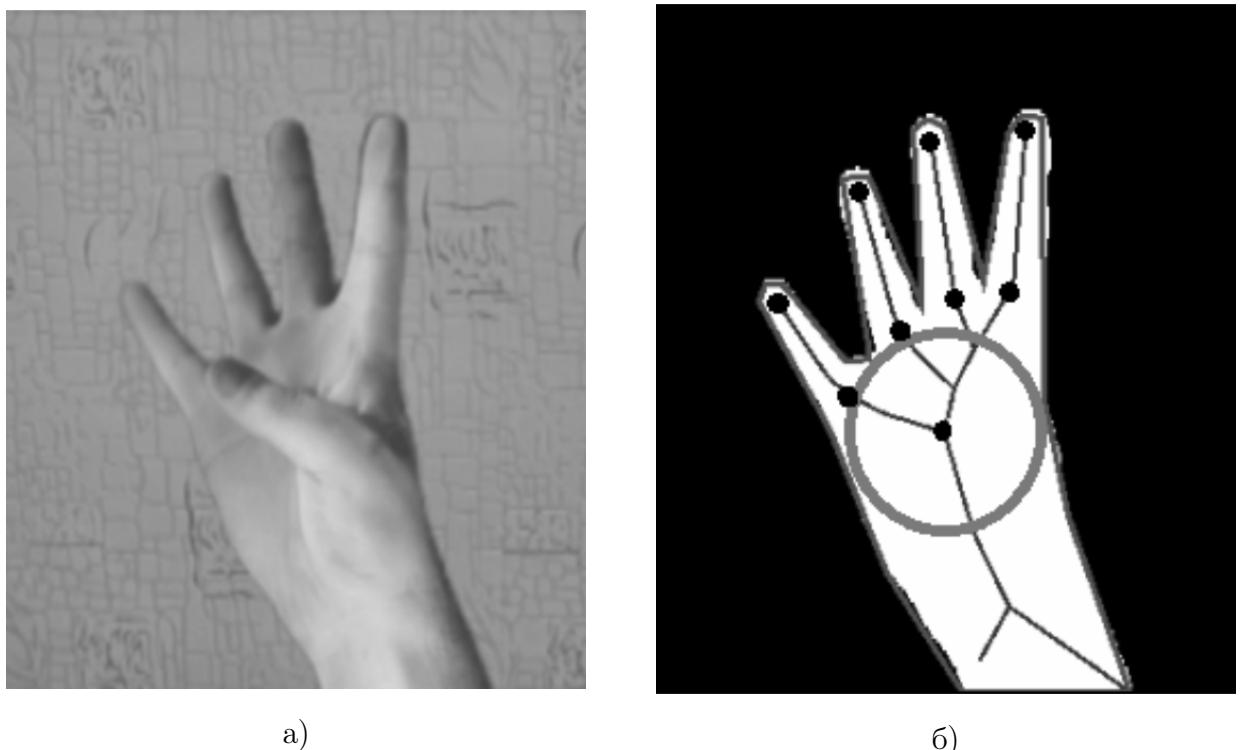


Рис. 22. Определение ключевых точек: *a* – исходное изображение; *б* – скелет и ключевые точки на нём.

## 4. МЕТОДИКА И ОЦЕНКА ЕЁ ЭФФЕКТИВНОСТИ

Основными критериями выбора методики являются *гибкость* использования, *скорость* и *точность* работы. В связи с этим в качестве итогового алгоритма детектирования кисти и извлечения её конфигурации был взят следующий:

1. Входное изображение разбивается на два множества "фон" и "передняя часть" с помощью метода MoG. Ранее было указано, что данный метод обладает свойством *обучения*, что позволяет его использовать в системах реального времени без жёсткого задания вида фона изображения. В свою очередь данный факт приводит к *гибкому* использованию данного алгоритма.
2. "Передняя часть" изображения кисти переводится в бинарное изображение, искусственно размывается и подвергается обработке методом Оцу, удаляя неточности, возникшие в результате первого шага. На данном этапе принимается, что кисть полностью отделилась от фона.
3. На изображении кисти производится поиск контура с помощью топологического структурного анализа цифрового бинарного изображения с помощью отслеживания границ. Если найденных контуров несколько, то выбирается контур с наибольшей площадью, найденной по формуле площади Гаусса. На данном этапе контур кисти считается найденным.
4. Выполняется поиск точки центра масс контура с помощью формулы (12). Данная точка является центром ладони кисти.
5. Строится выпуклая оболочка точек контура кисти с помощью алгоритма Грэхема.
6. На основе выпуклой оболочки, найденной на шаге 4, и контура, найденного на шаге 3, находятся дефекты выпуклости, которые являются точками пальцев кисти.
7. Точка центра ладони кисти и точки, найденные на шаге 6, составляют конфигурацию кисти, а именно её расположение в кадре и количество выгнутых пальцев.

Проведём тестирование данной методики. Входные данные – видеофрагмент длительностью 20 секунд с частотой кадров 30 кадров в секунду, то есть  $20 \cdot 30 = 600$  кадров. Выходные данные – последовательность изображений в виде кадров из видеофрагмента с результатами работы.

При просмотре результатов, что при резких перемещениях кисти руки в кадре алгоритм не надолго "сбивается" вследствие переобучения метода MoG, из-за этого на некоторых кадрах ладонь была распознана не точно и, соответственно остальные шаги выполнены неверно. Численные результаты представлены в таблице 2.

Таким образом, точность детектирования равна  $\frac{473}{600} = 0.79$ , а точность извлечения конфигурации  $\frac{358}{600} = 0.597$ .

Всего изображений	600
Кисть верно распознана	473
Конфигурация кисти верно определена	358

Таблица 2. Результаты работы алгоритма.

Разбор причин неверного определения конфигурации кисти показал, что при жесте "V" с двумя пальцами, безымянный палец и мизинец слегка выходят за границу ладони, вследствие чего данные пальцы были указаны как вытянутые и итоговое количество пальцев равно 4, что является неверным. Можно сделать вывод, что данная методика отлично работает при несложных движениях кисти, таких как сжатие в кулак, вытягивание пальцев по одному, либо по несколько, если пальцы расположены поодаль друг от друга.

# ЗАКЛЮЧЕНИЕ

Детектирование рук и распознавание жестов являются одними из наиболее популярных прикладных задач компьютерного зрения, но до сих пор в их решении имеются проблемы с получением точного результата из-за сложности снижения шума.

Во-первых, для решения задачи детектирования кисти были рассмотрены различные подходы, а именно детектирование на основе цветов в разных цветовых моделях (YCbCr и HSV), разделения пикселей изображения на две группы – "фон" и "передняя (полезная) часть и рекурсивные методы построения модели фона. В результате исследования было получено, что наиболее эффективным способом в статическом изображении является детектирование в цветовой модели HSV, а при наблюдении в динамике, то есть на видеофрагменте, то наилучшие результаты показал метод MoG.

Во-вторых, для построения контура кисти на изображении были исследованы два метода: выделение с помощью оператора Кэнни и топологический структурный анализ цифрового бинарного изображения с помощью отслеживания границ. Последний позволяет находить точные координаты контура, поэтому является наилучшим среди выбранных.

В-третьих, для определения конфигурации кисти, было решено использовать её описание в "ключевых" точках. Их поиск был произведён с помощью определения дефектов выпуклости алгоритмами Грэхема, Джарвиса и Киркпатрика. Сравнение данных алгоритмов производилось по времени, поскольку именно оно является решающим в детектировании кисти в реальном времени, при одинаковой точности. Алгоритм Грэхема показал наилучшие результаты с разницей более чем в 3 раза по сравнению с алгоритмом Киркпатрика и более чем в 25 раз по сравнению с алгоритмом Джарвиса. Без дефектов выпуклости поиск "ключевых" точек производился с помощью метода локализации на основе непрерывного скелета, также показавший неплохие результаты.

В-четвёртых, на основе исследований была разработана методика детектирования и извлечения конфигурации кисти, показавшая неплохие результаты при несложных движениях кисти человека.

В дальнейшем есть возможность использования полученной конфигурации кисти для применения в большом множестве задач, включая распознавание жестов или управление какими-либо автоматическими системами.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Местецкий Л. М. Непрерывная морфология бинарных изображений: фигуры, скелеты, циркуляры. М. : Физматлит, 2009.
2. Местецкий Л. М., Рейер И. Непрерывное скелетное представление изображения с контролируемой точностью // International Conference Graphicon. М., 2003. С. 51–54.
3. Носов А. В. Локализация ключевых точек кисти руки на изображении на основе непрерывного скелета. Математические методы моделирования, управления и анализа данных, с. 77-79, 2015.
4. Носов А. В. Алгоритм распознавания жестов рук на основе скелетной модели кисти руки // Вестник СибГАУ. 2014. Вып. 2(54). С. 62–67.
5. Basilio, Jorge & Torres, Gualberto & Sanchez-Perez, Gabriel & Toscano, Karina & Perez-Meana, Hector. Explicit image detection using YCbCr space color model as skin detection, 2011. - p. 123-128.
6. Dix A., Finlay J., Abowd G.D., Beale R. Human-Computer Interaction. - Third Edition, Pearson Education Limited: 2004. - p. 857
7. Graham R. L. An efficient algorithm for determining the convex hull of a finite planar set // Info. Pro. Lett. – 1972. – Т. 1. – С. 132-133.
8. Jarvis R.A. On the identification of the convex hull of a finite set of points in the plane // Information Processing Letters, Volume 2, Issue 1, 1973, p. 18-21.
9. Jiangqin W., Wen G. A FAst Sign Word Recognition Method for Chinese Sign Language // In Proceedings of the Third International Conference on Advances in Multimodal Interfaces (ICMI '00). - Springer-Verlag, London, 2000. - p.599-606
10. Kaewtrakulpong P., Bowden R.. An improved adaptive background mixture model for real-time tracking with shadow detection // Video-Based Surveillance Systems, pp. 135-144. Springer, 2002.
11. Kirkpatrick, David G.; Seidel, Raimund (1986). "The ultimate planar convex hull algorithm?". SIAM Journal on Computing. 15 (1): 287–299.
12. N. Otsu. A threshold selection method from gray-level histograms (англ.) // IEEE Trans. Sys., Man., Cyber. : journal. — 1979. — Vol. 9. — p. 62–66.

13. Sanna A., Lamberti F., Paravati G., Henao R., Eduardo A., Manuri F. A Kinect-Based Natural Interface for Quadrotor Control // Intelligent Technologies for Interactive Entertainment, Volume 78. Springer Berlin Heidelberg, 2012. - p. 48-56
14. Satoshi S., Keiich A. 1985. Topological Structural Analysis of Digitized Binary Images by Border Following. Computer vision, graphics, and image processing, 30.
15. Teh C-H and Chin Roland T. On the detection of dominant points on digital curves. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 11(8):859–872, 1989.
16. Van Droogenbroeck M., Barnich O.. Vibe: A disruptive method for background subtraction. // In T. Bouwmans, F. Porikli, B. Hoferlin, A. Vacavant, editors, Background Modeling and Foreground Detection for Video Surveillance, chapter 7. Chapman and Hall/CRC, pages 7.1-7.23, July 2014.
17. Zivkovic Z., Heijden F. Efficient adaptive density estimation per image pixel for the task of background subtraction // Pattern recognition letters, Vol. 27(7), pp. 773-780, 2006.

# ПРИЛОЖЕНИЯ

## Приложение А. Код реализации алгоритма Оцу и построения изображений.

---

```
1 import numpy as np
2 import cv2
3 import matplotlib.pyplot as plt
4
5 INPUT_TO_OUTPUT = {'image3.jpg':
6                     {'histogram': 'histogram_for_3.png',
7                      'output': 'otsu_exmpl_for_3.png'},
8                     'image7.jpg':
9                     {'histogram': 'histogram_for_7.png',
10                    'output': 'otsu_exmpl_for_7.png'}}
11
12 def get_min_max(image_flatten: np.ndarray) -> tuple:
13     min_pix, max_pix = image_flatten[0], image_flatten[0]
14     for pixel in image_flatten:
15         if pixel < min_pix:
16             min_pix = pixel
17         if pixel > max_pix:
18             max_pix = pixel
19     return min_pix, max_pix
20
21 def create_histogram(image_flatten: np.ndarray,
22                      size: int, min_pix: int) -> np.array:
23     hist = np.zeros(size)
24     for pixel in image_flatten:
25         hist[pixel - min_pix] += 1
26     return hist
27
28 def otsu_threshold(image: np.ndarray):
29     img_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
30     img_gray_flatten = img_gray.flatten()
31     min_pixel, max_pixel = get_min_max(img_gray_flatten)
32     size = max_pixel - min_pixel + 1
33     histogram = create_histogram(img_gray_flatten, size, min_pixel)
```

```

31     m = 0
32     n = 0
33     for i in range(size):
34         m += i * histogram[i]
35         n += histogram[i]
36     max_sigma = -1
37     threshold = 0
38     alpha1 = 0
39     beta1 = 0
40     for i in range(size - 1):
41         alpha1 += i * histogram[i]
42         beta1 += histogram[i]
43         w1 = float(beta1) / n # Вероятность класса 1
44         w2 = 1 - w1
45         a = float(alpha1) / beta1 - float(m - alpha1) / (n - beta1)
46         sigma = w1 * w2 * a * a
47         if sigma > max_sigma:
48             max_sigma = sigma
49             threshold = i
50     threshold += min_pixel
51     return threshold
52 def save_histogram(image, fname: str):
53     image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY).flatten()
54     plt.hist(image, 255)
55     plt.xlim([0, 255])
56     plt.savefig(fname)
57     plt.show()
58 if __name__ == '__main__':
59     for fname, outputs in INPUT_TO_OUTPUT.items():
60         image = cv2.imread(fname)
61         save_histogram(image, outputs['histogram'])
62         thresh = otsu_threshold(image)
63         print(f'Threshold = {thresh}')
64         ret, img_thresh = cv2.threshold(cv2.cvtColor(image, cv2.COLOR_BGR2GRAY),
65                                         thresh, 255, cv2.THRESH_BINARY)
66         imask = img_thresh == 255

```

```
67     canvas = np.zeros_like(image, np.uint8)
68     canvas[imask] = image[imask]
69     cv2.imwrite(outputs['output'], canvas)
```

---

## Приложение Б. Код реализации алгоритма Грэхема.

---

```
1  from functools import cmp_to_key
2
3  class Point:
4      def __init__(self, x=None, y=None):
5          self.x = x
6          self.y = y
7
8      # Рассстояние между двумя точками
9      def dist_sq(p1, p2):
10         return ((p1.x - p2.x) * (p1.x - p2.x) +
11                 (p1.y - p2.y) * (p1.y - p2.y))
12
13     # Определение ориентации трёх точек
14     # (перекрестное произведение векторов pq и qr)
15     def orientation(p, q, r):
16         val = ((q.y - p.y) * (r.x - q.x) -
17                (q.x - p.x) * (r.y - q.y))
18
19         if val == 0:
20             return 0    # коллинеарны
21
22         elif val > 0:
23             return 1    # по часовой
24
25         else:
26             return 2    # против часовой
27
28     # Сравнение двух точек для сортировки
29     def compare(p1, p2):
30         global p0
31         o = orientation(p0, p1, p2)
32
33         if o == 0:
34             return -1 if dist_sq(p0, p2) >= dist_sq(p0, p1) else 1
35
36         else:
37             return -1 if o == 2 else 1
```

```

34  def convex_hull(input_points: list) -> list:
35      # Конвертируем в класс Point
36      points = [Point(point[0], point[1]) for point in input_points]
37      n = len(input_points)
38      # Находим минимальную точку
39      min_y = points[0].y
40      min_i = 0
41      for i in range(1, n):
42          y = points[i].y
43          if ((y < min_y) or
44              (min_y == y and points[i].x < points[min_i].x)):
45              min_y = points[i].y
46              min_i = i
47
48      points[0], points[min_i] = points[min_i], points[0]
49
50      # Сортируем массив
51      global p0
52      p0 = points[0]
53      points = sorted(points, key=cmp_to_key(compare))
54
55      m = 1    # Начальный размер нового массива
56      # Удаляем лишние точки
57      for i in range(1, n):
58          while ((i < n - 1) and
59                  (orientation(p0, points[i], points[i + 1]) == 0)):
60              i += 1
61          points[m] = points[i]
62          m += 1
63      if m < 3:
64          return None
65
66      S = [points[0], points[1], points[2]]
67      for i in range(3, m):
68          while ((len(S) > 1) and
69                  (orientation(S[-2], S[-1], points[i]) != 2)):

```

```
70         S.pop()
71         S.append(points[i])
72
73     return [(p.x, p.y) for p in S]
```

---

## Приложение В. Код реализации алгоритма Джарвиса.

---

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      # поиск самой левой нижней точки
7  def left_index(points):
8      minn = 0
9      for i in range(1, len(points)):
10         if points[i].x < points[minn].x or \
11             (points[i].x == points[minn].x and
12              points[i].y > points[minn].y):
13             minn = i
14
15     return minn
16
17     # аналогично orientation в алгоритме Грэхема
18  def orientation(p, q, r):
19      val = (q.y - p.y) * (r.x - q.x) - \
20            (q.x - p.x) * (r.y - q.y)
21
22      if val == 0:
23          return 0
24
25      elif val > 0:
26          return 1
27
28      return -1
29
30  def convex_hull(init_points):
31      n = len(init_points)
32
33      if n < 3:
34
35          return
36
37      points = [Point(p[0], p[1]) for p in init_points]
38      l = left_index(points)
39
40      hull = []
41
42      p = l
```

```
34     q = 0
35
36     while True:
37         hull.append(p)
38         q = (p + 1) % n
39         for i in range(n):
40             if (orientation(points[p],
41                             points[i], points[q]) == 2):
42                 q = i
43
44             p = q
45             if p == l:
46                 break
47
48     return [(points[i].x, points[i].y) for i in hull]
```

---

## Приложение Г. Код реализации алгоритма Киркпатрика.

---

```
1  from collections import namedtuple
2  from random import randint
3
4  Point = namedtuple('Point', 'x y')
5
6  def flipped(points):
7      return [Point(-point.x, -point.y) for point in points]
8
9  def quickselect(ls, index, lo=0, hi=None, depth=0):
10     if hi is None:
11         hi = len(ls) - 1
12     if lo == hi:
13         return ls[lo]
14     if len(ls) == 0:
15         return 0
16     pivot = randint(lo, hi)
17     ls = list(ls)
18     ls[lo], ls[pivot] = ls[pivot], ls[lo]
19     cur = lo
20     for run in range(lo+1, hi+1):
21         if ls[run] < ls[lo]:
22             cur += 1
23             ls[cur], ls[run] = ls[run], ls[cur]
24     ls[cur], ls[lo] = ls[lo], ls[cur]
25     if index < cur:
26         return quickselect(ls, index, lo, cur-1, depth+1)
27     elif index > cur:
28         return quickselect(ls, index, cur+1, hi, depth+1)
29     else:
30         return ls[cur]
31
32 def bridge(points, vertical_line):
33     candidates = set()
```

```

34     if len(points) == 2:
35         return tuple(sorted(points))
36     pairs = []
37     modify_s = set(points)
38     while len(modify_s) >= 2:
39         pairs += [tuple(sorted([modify_s.pop(), modify_s.pop()]))]
40     if len(modify_s) == 1:
41         candidates.add(modify_s.pop())
42     slopes = []
43     for pi, pj in pairs[:]:
44         if pi.x == pj.x:
45             pairs.remove((pi, pj))
46             candidates.add(pi if pi.y > pj.y else pj)
47         else:
48             slopes += [(pi.y-pj.y)/(pi.x-pj.x)]
49     median_index = len(slopes)//2 - (1 if len(slopes) % 2 == 0 else 0)
50     median_slope = quickselect(slopes, median_index)
51     small = {pairs[i] for i, slope in enumerate(slopes) if slope < median_slope}
52     equal = {pairs[i] for i, slope in enumerate(slopes) if slope == median_slope}
53     large = {pairs[i] for i, slope in enumerate(slopes) if slope > median_slope}
54     max_slope = max(point.y-median_slope*point.x for point in points)
55     max_set = [point for point in points if \
56                 point.y-median_slope*point.x == max_slope]
57     left = min(max_set)
58     right = max(max_set)
59     if left.x <= vertical_line < right.x:
60         return left, right
61     if right.x <= vertical_line:
62         candidates |= {point for _, point in large | equal}
63         candidates |= {point for pair in small for point in pair}
64     if left.x > vertical_line:
65         candidates |= {point for point, _ in small | equal}
66         candidates |= {point for pair in large for point in pair}
67     return bridge(candidates, vertical_line)
68
69 def connect(lower, upper, points):

```

```

70     if lower == upper:
71         return [lower]
72     max_left = quickselect(points, len(points)//2-1)
73     min_right = quickselect(points, len(points)//2)
74     left, right = bridge(points, (max_left.x + min_right.x)/2)
75     points_left = {left} | {point for point in points if point.x < left.x}
76     points_right = {right} | {point for point in points if point.x > right.x}
77     return connect(lower, left, points_left) + connect(right, upper, points_right)
78
79 def upper_hull(points):
80     lower = min(points)
81     lower = max({point for point in points if point.x == lower.x})
82     upper = max(points)
83     points = {lower, upper} | {p for p in points if lower.x < p.x < upper.x}
84     return connect(lower, upper, points)
85
86 def convex_hull(init_points):
87     points = [Point(p[0], p[1]) for p in init_points]
88     upper = upper_hull(points)
89     lower = flipped(upper_hull(flipped(points)))
90     if upper[-1] == lower[0]:
91         del upper[-1]
92     if upper[0] == lower[-1]:
93         del lower[-1]
94     return upper + lower

```

---

## Приложение Д. Код функции очистки лишних точек дефектов выпуклости.

---

```
1 import math
2 import numpy as np
3
4 def squeeze(array):
5     array = array[0] if len(array) == 1 else array
6     return [p[0] for p in array] if len(array[0]) == 1 else array
7
8 # "очистка" выпуклой оболочки до не больше 7 точек
9 def clear_convex_hull(hull_index, contour):
10    distance_point = lambda p1, p2: \
11        math.sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)
12    hull_index, contour = squeeze(hull_index), squeeze(contour)
13    ALPHA = 10
14    while True:
15        boundary = len(hull_index) - 1
16        clean_hull, i = [], 0
17        while i < boundary:
18            clean_hull.append(hull_index[i])
19            while i < boundary and \
20                distance_point(contour[hull_index[i]],
21                               contour[hull_index[i + 1]]) < ALPHA:
22                i += 1
23            i += 1
24            if len(clean_hull) > 7:
25                ALPHA += 1
26            else:
27                break
28    return np.array(clean_hull[:-1]) if \
29        distance_point(contour[clean_hull[0]],
30                     contour[clean_hull[-1]]) < ALPHA else np.array(clean_hull)
```

---