

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
имени М.В.Ломоносова  
Факультет вычислительной математики и кибернетики  
**Изучение и освоение методов анализа формы  
объектов на изображении**

Подготовил:  
студент 317 группы ВМК МГУ  
Долгушев Глеб Дмитриевич

Апрель 2025

## Содержание

<b>1 Введение</b>	<b>1</b>
<b>2 Постановка задачи</b>	<b>2</b>
<b>3 Описание данных</b>	<b>3</b>
<b>4 Описание реализованных алгоритмов</b>	<b>3</b>
4.1 Теоретическая справка . . . . .	3
4.2 Сегментация карт . . . . .	6
4.2.1 Бинаризация . . . . .	7
4.2.2 Определение границ карты . . . . .	8
4.3 Анализ рисунков на картах . . . . .	12
4.3.1 Выделение рисунков . . . . .	12
4.3.2 Анализ формы фигуры . . . . .	12
4.4 Эксперименты . . . . .	14
<b>5 Реализация программы</b>	<b>14</b>
5.1 Детали реализации . . . . .	14
5.2 Описание работы с интерфейсом . . . . .	15
<b>6 Выводы</b>	<b>16</b>
<b>7 Литература</b>	<b>20</b>

## 1 Введение

Цель данной работы заключается в разработке и реализации программы для эффективной сегментации объектов различных видов, подсчета данных объектов и дальнейшего анализа их формы по различным критериям.

Задачи:

1. Изучить различные алгоритмы, позволяющие определять форму объектов на изображении, сфокусировавшись на преобразованиях, помогающих в анализе многоугольников.
2. Реализовать функционал для загрузки и отображения изображений в графическом интерфейсе.
3. Разработать алгоритм первичной сегментации изображений с использованием точечных и пространственных преобразований, алгоритмов для определения формы объектов, морфологических операций и других необходимых методов обнаружения объектов на изображениях.
4. Реализовать алгоритм анализа сегментированных объектов, позволяющий провести выделения необходимых характеристик объекта, а именно:
  - (a) Определить, является ли объект многоугольником, или фигурой с гладкой границей
  - (b) В случае, если объект является многоугольником определить количество вершин
  - (c) В случае, если объект является многоугольником определить является ли он выпуклым
5. Обеспечить работу программы в интерактивном режиме с возможностью выбора изображения, решаемой задачи и визуализации необходимых промежуточных преобразований.
6. Провести тестирование программы на изображениях разного уровня сложности (Beginner, Intermediate, Expert) для проверки корректности сегментации и подсчёта объектов, сделать выводы по различным реализованным алгоритмам.

## 2 Постановка задачи

**Дано:**

Изображение (набор пикселей  $Im$ ) с объектами различных классов

**Найти:**

1. Маски (наборы пикселей  $Im_i$ ) для сегментации объектов (карты) а также сопоставить каждому изображению число - кол-во объектов на изображении.
2. Для каждой выделенной карты (иначе говоря набора пикселей  $Im_i$ ) выделить набор пикселей  $Fig_i$  соответствующий фигуре, нарисованной внутри карты. Каждой фигуре  $Fig_i$  сопоставить класс из списка [*Фигура с гладкой границей*, *Многоугольник*]. Для фигур второго класса также определить число  $V$  - кол-во вершин и метку  $Conv \in \{True, False\}$ , обозначающую, является ли многоугольник выпуклым.

### 3 Описание данных



Рис. 1: Пример исходного изображения 1



Рис. 2: Пример исходного изображения 2

Данные имеют формат изображений (.jpg). Представляют собой фотографии карт из неизвестной настольной игры на фонах различных видов. Фон и факт наложения карт друг на друга определяют сложность решения поставленной задачи. Примеры изображений приведены на Рис. 1 и Рис. 2. В дальнейшем будет описано решение для фотографий карт на однородном фоне.

### 4 Описание реализованных алгоритмов

#### 4.1 Теоритическая справка

В процессе создания данного алгоритма были использованы следующие алгоритмы и преобразования:

1. **Преобразование Хафа.** Используется для обнаружения геометрических примитивов.

В исходном пространстве изображения линия описывается уравнением:

$$\rho = x \cos \theta + y \sin \theta,$$

где  $\rho$  — расстояние от начала координат до линии,  $\theta$  — угол между нормалью к линии и осью  $OX$ .

Преобразование Хафа отображает каждую точку  $(x_i, y_i)$  изображения в кривую в параметрическом пространстве  $(\rho, \theta)$ :

$$\rho = x_i \cos \theta + y_i \sin \theta \quad \forall \theta \in [0, \pi).$$

Точки пересечения кривых в параметрическом пространстве соответствуют линиям в исходном изображении.

Задается *аккумуляторное пространство* - дискретизированное пространство  $(\rho, \theta)$  представляется в виде матрицы-аккумулятора  $A[\rho][\theta]$ , где каждый элемент накапливает "голоса" точек изображения. Для каждого ребра  $(x_i, y_i)$ :

$$A[\rho_k][\theta_k] \leftarrow A[\rho_k][\theta_k] + 1,$$

где  $\rho_k = \text{round}(x_i \cos \theta_k + y_i \sin \theta_k)$ ,  $\theta_k$  — дискретные величины (дискретизация задается параметрами).

Алгоритм состоит из следующих этапов:

- (a) Применение детектора краёв (например, Canny) для получения бинарного изображения границ  $E(x, y)$ .
- (b) Для каждой точки  $(x_i, y_i) \in E$  вычисляются все возможные  $(\rho, \theta)$  и обновляется аккумулятор  $A$ .
- (c) Локальные максимумы в  $A$  соответствуют параметрам  $\{\rho_j, \theta_j\}$ , которые будут выданы как обнаруженные линии. Порог  $T$  задаёт минимальное количество голосов для учёта кандидата.
- (d) Подавление немаксимумов в окрестности  $\Delta\rho \times \Delta\theta$  для исключения дубликатов.

**2. Алгоритм Рамера - Дугласа - Пекера.** Метод упрощения полигональных цепей. Алгоритм рекурсивно аппроксимирует исходную кривую, минимизируя количество точек при сохранении геометрической формы в пределах заданной погрешности.

Начнем с постановки задачи: пусть задана полигональная цепь  $P = \{p_1, p_2, \dots, p_N\}$ , где  $p_i = (x_i, y_i)$ . Требуется построить упрощенную цепь  $Q = \{q_1, q_2, \dots, q_M\}$ , такую что:

- $M \leq N$ ,
- Максимальное расстояние между  $P$  и  $Q$  не превышает  $\varepsilon$ ,

Алгоритм состоит из следующих шагов:

- (a) Выбирается начальная  $p_{\text{start}} = p_1$  и конечная  $p_{\text{end}} = p_N$  точки цепи  $P$  и добавляются в результат  $Q$ .
- (b) Выбирается точка с максимальным отклонением. Для всех промежуточных точек  $p_i$  ( $i = 2, \dots, N-1$ ) вычисляется расстояние  $d_i$  до прямой  $L$ , проведенной через  $p_{\text{start}}$  и  $p_{\text{end}}$ :

$$d_i = \frac{|(y_{\text{end}} - y_{\text{start}})x_i - (x_{\text{end}} - x_{\text{start}})y_i + x_{\text{end}}y_{\text{start}} - y_{\text{end}}x_{\text{start}}|}{\sqrt{(y_{\text{end}} - y_{\text{start}})^2 + (x_{\text{end}} - x_{\text{start}})^2}}.$$

Искомая точка —  $p_{\text{max}}$  с наибольшим  $d_{\text{max}} = \max(d_i)$ .

- (c) Далее происходит рекурсивное разделение до выполнения критерия останова. Если  $d_{\text{max}} > \varepsilon$  то  $P$  делится на два подотрезка:  $P_{\text{left}} = [p_{\text{start}}, p_{\text{max}}]$ ,  $P_{\text{right}} = [p_{\text{max}}, p_{\text{end}}]$  которые также обрабатываются. Если  $d_{\text{max}} \leq \varepsilon$ , то тбрасываются все промежуточные точки между  $p_{\text{start}}$  и  $p_{\text{end}}$ .

3. **Бинаризация изображения.** Преобразование изображения в черно-белый формат, при котором каждый пиксель изображения получает значение 0 или 255. Формально ее действие на изображение можно записать следующим образом:

$$I_{bin}(x, y) = \begin{cases} 255, & I(x, y) \in [left\_bound, right\_bound] \\ 0, & \text{иначе} \end{cases}$$

4. **Дилатация.** Увеличивает светлые области, помогает приблизить компоненту связности к ее выпуклой оболочке (замацывает внутренности). При применении дилатации каждый пиксель изображения, который находится в пределах выбранного ядра, становится белым, если хотя бы один из его соседей белый.

В случае дилатации преобразование можно записать так:

$$I_{dil}(x, y) = \max (\{I(x', y') \mid (x', y') \in N(x, y)\})$$

Здесь  $N(x, y)$  — область соседей пикселя  $(x, y)$ .

Пример ядра для дилатации:

$$S = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

5. **Эрозия.** Уменьшает размеры объектов, удаляет мелкие шумы. При применении эрозии каждый пиксель изображения, который находится в пределах выбранного ядра, становится черным, если хотя бы один из его соседей черный:

$$I_{erod}(x, y) = \min (\{I(x', y') \mid (x', y') \in N(x, y)\})$$

Здесь  $N(x, y)$  — область соседей пикселя  $(x, y)$ .

Пример ядра для эрозии:

$$S = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

6. **Фильтр Собеля.** Используется для выделения границ изображения. Вычисляет градиент интенсивности изображения с помощью свертки с двумя ядрами — горизонтальным и вертикальным.

Горизонтальное ядро Собеля:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Вертикальное ядро Собеля:

$$S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Градиент изображения вычисляется как:

$$G(x, y) = \sqrt{(S_x * I(x, y))^2 + (S_y * I(x, y))^2}$$

где  $*$  — операция свертки.

**7. Оператор Кэнни.** Также используется для выделения границ изображения. Состоит из пяти отдельных шагов:

- (a) Размытие изображения для удаления шума. Применяется гауссово размытие с ядром заданного размера.

$$G(x, y) = \frac{1}{2\pi\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- (b) Поиск градиентов при помощи описанного выше оператора Собеля
- (c) Подавление не-максимумов. Только локальные максимумы отмечаются как границы.
- (d) Двойная пороговая фильтрация. Назначается два порога:  $T_{low}$  и  $T_{high}$ . Пиксели  $G > T_{high}$  считаются сильными границами, пиксели с  $G < T_{low}$  отбрасываются, остальные считаются слабыми границами.
- (e) Трассировка области неоднозначности. Сохраняются только слабые пиксели, которые соединены с сильными границами через цепочку соседних слабых пикселей. Используется поиск связных компонент.

8.

## 4.2 Сегментация карт

Для понимания тех или иных шагов в описании алгоритма необходимо в кратце описать весь пайплайн работы с изображением:

1. Проводим бинаризацию изображения для сегментации компонент связности на изображении, которые представляют собой отдельные карты или группы накладывающихся друг на друга карт.
2. При помощи преобразования Хафа выделяем границы найденных компонент связности, как набор прямых линий.
3. Группируем полученные линии так, чтобы по ним можно было восстановить искомый объект (карту)

Подробное описание всех шагов далее.

#### 4.2.1 Бинаризация

Целью бинаризации, как было сказано выше, является выделение компонент связности на изображении, которые представляют собой отдельные карты или группы накладывающихся друг на друга карт.

Для поставленной задачи и имеющихся данных при бинаризации были поставлены следующие цели:

1. **Акцент на выделении краев объектов.**

Очень часто центр карты на изображении является засвеченным, что объясняется их глянцевой поверхностью. Данный факт затруднил бы сегментацию всей карты при помощи одной лишь бинаризации. Для алгоритма, который будет подробно описан далее основным будет сходство границ выделенного объекта с прямоугольником, следовательно, важно выделить корректно именно крайние части карт.

2. **Фильтрация шумовых объектов.**

На самих картах содержатся не связанные с задачей рисунки, обозначим их *шумовыми*. Некорректная обработка *шумовых* объектов приводит к сильным нарушениям в форме границ сегментируемого участка изображения и разбиению одной карты на несколько компонент связности, поэтому допускается выделение несответствующих задаче объектов (например, темные части фона) в пользу избавления от описанных *шумовых* рисунков. Пример на изображении Рис. 3

Для подбора порога бинаризации была построена гистограмма распределения интенсивности по каждому каналу на данных с однородным фоном Рис. 4, а также проведены перебор возможных порогов от (100, 125, 155) до (165, 150, 150) с шагом 5.



Рис. 3: Результаты бинаризации с порогом, который не адаптирован для отсеивания шумовых рисунков

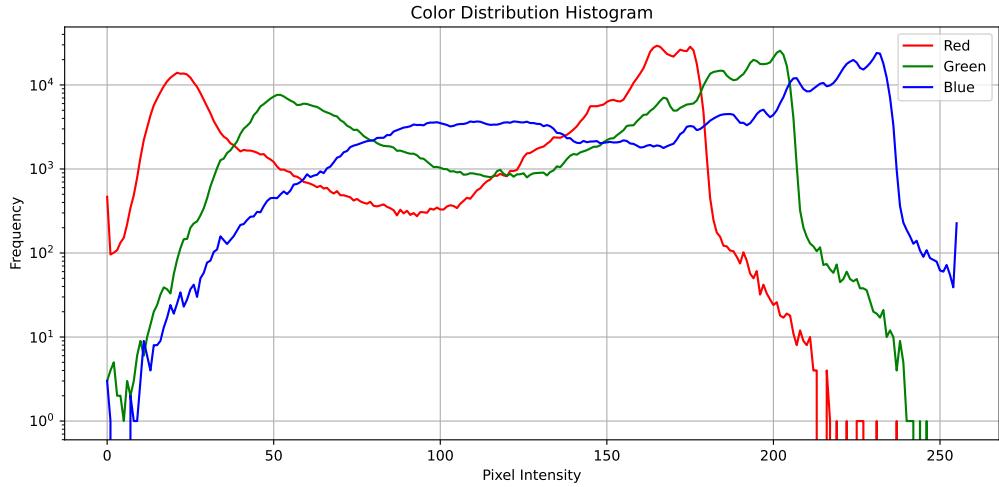


Рис. 4: Результаты анализа распределения цветов на рассматриваемых данных



Рис. 5: Результаты бинаризации с порогом, который адаптирован для отсеивания шумовых рисунков

Пример выделения компонент связности с порогом, подобранным в соответствии с постановкой задачи приведен на Рис. 5. Как можно видеть, кнутренностя карт действительно часто не захвачены, кроме того есть область, явно не представляющая собой набор карт - она будет легко классифицирована как шумовая далее. Стоит обратить внимание на границы компонент связности - проблема *шумовых* рисунков решена, карты больше не разбиваются на несколько компонент связности, границы четкие.

#### 4.2.2 Определение границ карты

Как было сказано выше, первый шаг заключается в выделении границ найденных компонент связности при помощи преобразования Хафа. Формализуем задачу для каждой компоненты связности в отдельности.

Пусть нам дана бинарная растровая маска компоненты связности  $I$  - мн-во пикселей со значениями  $\in \{0, 255\}$ . Необходимо найти такое множество прямых линий  $L$ , которое описывает границу данной маски  $I$ .  $L = \{(\rho_i, \theta_i) | i = \overline{1, n}\}$ , то есть прямые линии заданы уже в векторном формате.

Данная задача будет выполняться следующим образом:

1. При помощи алгоритма, предложенного Suzuki и Abe для выделения контуров находится множество пикселей  $C$  - контур множества  $I$
2. Применяется преобразование Хафа для поиска на изображении  $C$  прямых

линий.

Результаты представлены на Рис. 6.

В случае, если в обрабатываемой компоненте связности есть только одна карта - задача является решенной, далее предоставим алгоритм разделения полученного множества  $L$  на группы  $G_k$ , где каждое  $G_k$  содержит границы только одной карты.

Для решения данной задачи введем следующую эвристiku. Так как все данные фотографии сделаны практически параллельно поверхности стола и на довольно большом расстоянии от него, то будем считать что каждая карта имеет форму *прямоугольника*. Кроме того, из тех же соображений предположим, что карты на каждой картинке имеют приблизительно одинаковый размер. Таким образом, множество  $G_k$  должно содержать границы только одного *прямоугольника*.



Рис. 6: Выделенные контуры по описанному алгоритму. На первом изображении - после алгоритма Suzuki и Abe, на втором - после преобразования Хафа

Итак, опишем алгоритм решения поставленной задачи. Предлагаемый метод основан на анализе углового сходства линий и их взаимной ориентации (параллельность/ортогональность) с адаптивным управлением параметрами группировки. Алгоритм включает следующие этапы:

#### Первичная группировка линий.

- Входные данные - набор линий  $L = \{(\rho_i, \theta_i) | i = \overline{1, n}\}$ , где каждая линия  $l_i$  задается параметрами  $(\rho_i, \theta_i)$  в полярных координатах.
- Критерий объединения - линия  $l_j$  добавляется в существующую группу  $G_k$ , если выполняется одно из условий:
  1.  $|\theta_j - \mu_k| < \varepsilon$ , где  $\mu_k$  — средний угол группы  $G_k$ ,  $\varepsilon$  — порог углового отклонения, допускающий, что карты не являются строгими прямоугольниками. То есть линия параллельна представлению группы (за представление берется линия  $(\rho, \mu_k)$ ,  $\rho$  - любое).

2.  $\left| |\theta_j - \mu_k| - \frac{\pi}{2} \right| < \varepsilon$ . То есть линия параллельна представлению группы.
3. Обновление статистики группы - Средний угол  $\mu_k$  пересчитывается как среднее по параллельным линиям в группе:

$$\mu_k^{(new)} = \frac{\sum_{(\rho, \theta) \in G_k} \theta [|\theta - \mu_k| < \varepsilon]}{\sum_{(\rho, \theta) \in G_k} [|\theta - \mu_k| < \varepsilon]}$$

**Рекурсивное уточнение групп.** Для каждой группы  $G_k$  выполняется проверка условий:

1. Количество параллельных представлению группы линий ( $|\theta_i - \mu_k| < \varepsilon$ ) > 2.
2. Количество ортогональных представлению группы линий ( $\left| |\theta_i - \mu_k| - \frac{\pi}{2} \right| < \varepsilon$ ) > 2.
3. Общее число линий в группе  $|G_k| > 4$ .

При выполнении любого условия происходит рекурсивный вызов обработки группы  $G_k$  с уменьшенным порогом  $\varepsilon' = \alpha \cdot \varepsilon$  ( $\alpha \in (0, 1)$  - подбираемый гиперпараметр) и глубиной рекурсии  $r' = r - 1$ . После чего найденные подгруппы добавляются в множество  $L$ , а разбиваемая повторно группа удаляется.

**Постобработка на корневом уровне.** Группы с  $|G_k| < 2$  линий повторно кластеризуются с изначальным порогом углового отклонения  $\varepsilon$ .

Для дальнейшего упоминания еще раз перечислим подбираемые гиперпараметры алгоритма:  $\varepsilon$  — погрешность измерения углов,  $\alpha$  — коэффициент уменьшения  $\varepsilon$  на каждом уровне рекурсии,  $r$  — глубина рекурсии.

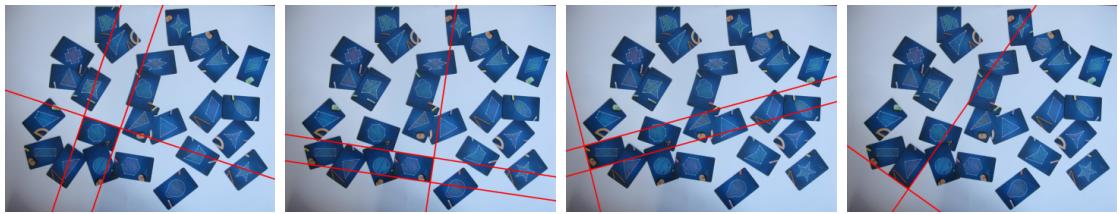


Рис. 7: Результаты группировки контуров

Обоснуем необходимость столь сложной конструкции приведенного алгоритма. Основная мотивация применения рекурсивной структуры заключается в наличии карт, находящихся в одной группе и лежащих под практически идеальным прямым углом, а также наличием карт, форма которых искажена из-за

ракурса фотографии. Последовательно применяя сначала более грубую группировку, а затем более точную алгоритм на первом этапе выделяет все карты, форма которых немного искажена, а на втором более точно выделяет карты, лежащие под похожими углами.

Далее по выделенным группам  $G_k$  нам необходимо определить прямоугольник, ограничивающий конкретную карту. Для решения этой задачи предлагается следующий алгоритм.

Для каждой группы, в которой есть хотя бы две линии, необходимо взять точку их пересечения. Предположим, что нам известны средние размеры карты на изображении (о том, как они находились, будет сказано далее), тогда существует 8 возможных положений карты относительно точки пересечения 2-х ее границ (Рис. 8). Предлагается сравнить площади пересечения каждого из возможных вариантов с маской компоненты связности  $I$  и выбрать тот, для которого эта площадь максимальна.

Даже в такой вариации, при использовании лишь двух пересечений прямых алгоритм будет хорошо справляться с выделением карт. Но было бы неправильным не использовать всю доступную информацию. Поэтому были реализованы следующие модификации:

1. В случае, если найдены все 4 стороны прямоугольника, никакой дополнительной обработки не требуется, задача решена. Кроме того, данные группы (такие что  $|G_k| = 4$ ) можно обрабатывать первыми и использовать для обновления размеров карт (прямоугольников), которые нужны для реализации описанного выше алгоритма.

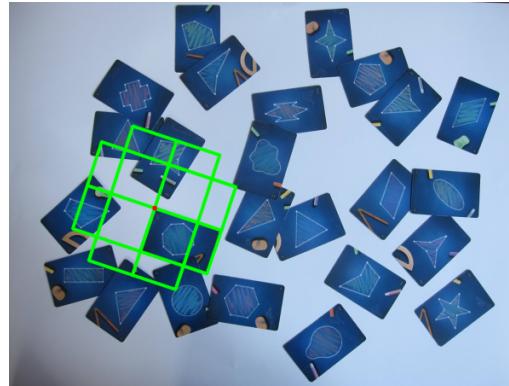


Рис. 8: Возможные положения карты, при том что известны две ее стороны, пересекающиеся в отмеченной точке

2. В случае, когда найдены 3 стороны прямоугольника, требуется перебрать не 8, а всего 2 возможных положения карты.
3. Заметим, что некоторые шумовые области (края изображений), которые выделялись на этапе бинаризации попадут в группу  $|G_k| < 2$ , и границы

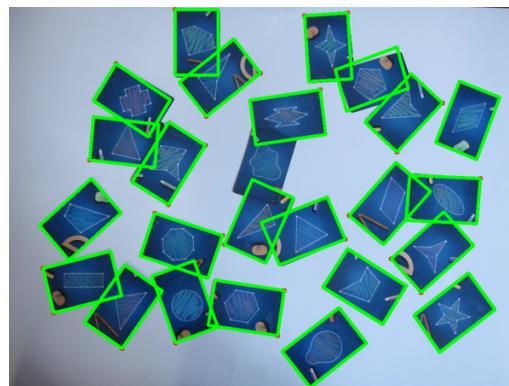


Рис. 9: Результат применения сконструированного алгоритма

для них не будут построены. Таким образом, подобные "лишние" области будут отсеены.

Пример работы описанного пайплайна виден на Рис. 9.

### 4.3 Анализ рисунков на картах

Опишем постановку задачи. После реализации первой части алгоритма была получена возможность выделить для каждой карты бинарную маску  $M_i, i = \overline{1, K}$ , где  $K$  - количество карт на картинке. Таким образом, необходимо, имея маску карты  $M$  и исходное изображения, выделить на нем фигуру, нарисованную в центре карты, и определить ее форму.

#### 4.3.1 Выделение рисунков

Итак, необходимо отделить фигуру в центре карты, для этого были проведены эксперименты с различными операторами (примеры для оператора Кэнни и адаптивной бинаризации на Рис. 11 и Рис. 12 соответственно). Огромным преимуществом выбранной постановки задачи является возможность довольно хорошо чистить выделяемые операторами границы карт, благодаря чему появляется возможность выбора адаптивной бинаризации как основного алгоритма. Сравнения Кэнни и адаптивной бинаризации представлены в Таблице 1.

	Плюсы	Минусы
Canny	Устойчивость к шуму, четкие границы	Низкая полнота выделения фигур, объединение границ <i>шумовых</i> и целевых объектов
Adaptive Binarization	Не смешивает границы <i>шумовых</i> и целевых объектов лучше полнота поиска границ	Неустойчивость к шуму, толстые границы,

Таблица 1: Сравнительный анализ различных методов бинаризации фигур на изображении

После применения бинаризации на изображении (она применяется по отдельности к максированному при помощи  $M_i$  изображению) необходимо найти контуры при помощи алгоритма Suzuki и Abe, среди найденных контуров выбирается тот, который имеет наибольшую площадь и сожержит центр выделенной карты. Полученные контуры для каждой карты образуют множество  $Cnt = \{cnt_i | i = \overline{1, K}\}$

#### 4.3.2 Анализ формы фигуры

Имея контуры всех фигур на изображении не составит труда определить оставшиеся характеристики, а именно: является ли контур многоугольником

или фигурой с гладкой границей, какое количество вершин имеет многоугольник (если контур им оказался) и является ли он выпуклым.

Для начала предлагается проверить фигуру на гладкость. Наиболее подходящим оказывается преобразование Хафа. Попробуем найти на контуре прямые линии, если нашли - то фигура является многоугольником, если нет - имеет гладкую границу.

Далее, для негладких фигур предлагается использовать описанный в теоретической главе алгоритм Рамера-Дугласа-Пекера, который позволяет аппроксимировать найденные объекты многоугольниками. Результаты применения данного подхода представлены на Рис. 10.

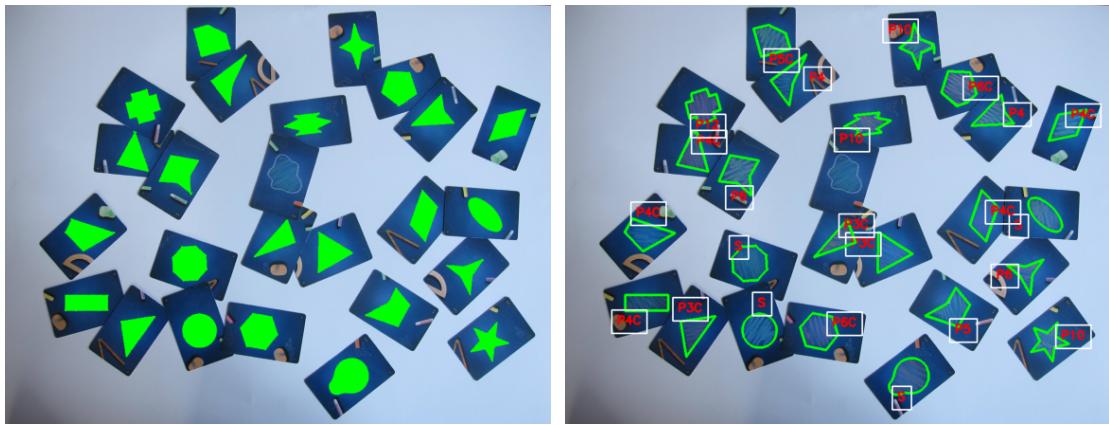


Рис. 10: Результаты применения алгоритмов для сегментации фигур внутри карт и определения их характеристик

Как можно заметить, результат не идеален, хоть и близок к таковому. Объясняется некорректная работа алгоритма двумя факторами:

- 1. Маленькое разрешение изображения.** При обработке фигур с большим количеством фигур алгоритм Хафа работает не лучшим образом, так как "линии" в растровом изображении представляются не совсем точно. Требуется сильно занижать параметр порога  $T$ , описанный выше, из-за чего начинают выеляться прямые на границе гладких фигур. Иначе говоря, стороны некоторых фигур настолько малы, что их сложно отличить от гладкой границы в виду особенностей работы алгоритмов.
- 2. Наложения карт.** Несмотря на то, что наложения карт почти не перекрывают сами фигуры, и человек способен при небольшом перекрытии определить истинную форму фигуры, алгоритм Рамера-Дугласа-Пекера часто работает иначе и добавляет углы найденной фигуре. Данный фактор сложно считать за ошибку, так как он обусловлен постановкой задачи (работать необходимо с формой объектов на изображении).

## 4.4 Эксперименты

При реализации метода были проведены следующие эксперименты (в силу того, что алгоритм работает отдельно с компонентами связности, выделенными на первых этапах, предоставить визуализацию затруднительно):

1. Перебор порогов бинаризации для выделения компонент связности.
2. Перебор порогов для классификации выделенных компонент связности
3. Перебор операторов для выделения внутренних фигур (Рис. 12 Рис. 11)
4. Перебор гиперпараметров для алгоритмов Хафа и Рамера - Дугласа - Пекера



Рис. 11: Выделение контуров внутренних фигур с использованием оператора Кэнни



Рис. 12: Выделение контуров внутренних фигур с использованием адаптивной пороговой бинаризации

## 5 Реализация программы

### 5.1 Детали реализации

Для написания пользовательского интерфейса была использована библиотека `streamlit`, для работы с изображениями и использования различных линейных и пространственных преобразований также использовались модули: `PIL`, `NumPy`, `OpenCV`, для визуализации использовался `matplotlib`.

Проект имеет следующую структуру:

- `detection_app.py` [код для реализации пользовательского интерфейса]
- `models.py` [реализованные алгоритмы для решения задачи]
- `samples` [папка доступных без загрузки изображений для анализа]

Для запуска приложения необходимо перейти в отправленную папку `code`, создать новое виртуальное окружение:

```
python -m venv .app-venv
```

Установить необходимые зависимости:

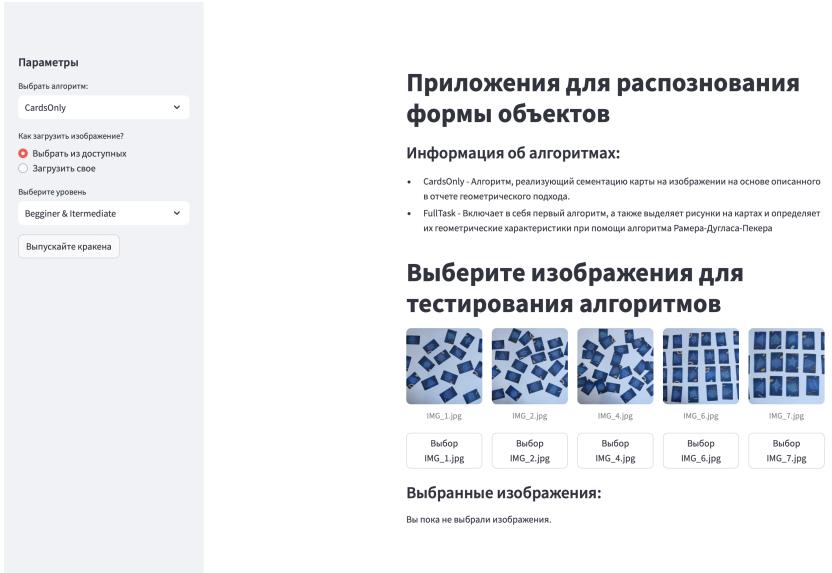


Рис. 13: Главная страница в приложении

```
pip install -r requirements.txt
```

И запустить приложение командой:

```
streamlit run detection_app.py
```

После его приложение будет доступно в браузере по адресу <http://localhost:8501>.

Разработка и тестирование приложения производились на версии `python:3.12.8`

## 5.2 Описание работы с интерфейсом

При открытии приложения пользователь попадает на главную страницу (Рис. 13), на которой описаны все реализованные алгоритмы и отображается интерфейс для работы с изображениями.

Далее можно выбрать пункт **Загрузить свое** для загрузки изображения из файла, или по средствам взаимодействия с кнопками **Выбор** и **Убрать** выбрать одно или несколько изображений из списка, перед выбором из списка изображений справа, в левом меню можно выбрать уровень изображений для дальнейшей работы (иллюстрация на Рис. 14).

После выбора / загрузки изображений запустить выбранный в левом меню алгоритм можно при помощи кнопки **Выпускайте кракена** (иллюстрация на Рис. 15).

Если пользователь загрузил хотя бы одно изображение, то запуск алгоритма произойдет успешно. Анализ изображений потребует некоторое время (особенно если их выбрано несколько), а об успешном запуске будет свидетельствовать индикатор **RUNNING...** в правом верхнем углу (иллюстрация на Рис. 16).

Результаты работы алгоритма будут показаны на отдельной странице (Рис. 17). Если пользователь выбрал несколько изображений, то для переключения между результатами для каждого из них стоит использовать кнопки **Предыдущий результат** и **Следующий результат**. Для возврата к главной странице служит кнопка **Назад на главную**.



Рис. 14: Выбор изображений из списка

На Рис. 18 показано, как отображаются результаты для задания Intermediate. Перемещаясь на странице можно получить более подробный отчет о последовательно применяющихся преобразованиях, на Рис. 18 показаны результаты в более удобной форме, на Рис. 19 продемонстрировано всплывающее окно, в котором в деталях отражен процесс работы алгоритма с разными компонентами связности.

## 6 Выводы

В ходе работы были изучены различные алгоритмы, позволяющие определять форму объектов на изображении, при этом основное внимание было уделено преобразованиям, показывающим себя наилучшим образом в задачах анализа многоугольников, исследованы возможности их комбинирования для выполнения поставленных задач сегментации изображений и выделения характеристик фигур на изображении.

На основе изученного материала были построены два алгоритмы для сегментации карт (с учетом того, что они могут накладываться друг на друга) и для анализа формы центральной фигуры на карте.

Были проанализированы возможности каждого алгоритма и проведены необходимые корректировки гиперпараметров для достижения наилучших результатов.

Было спроектировано и разработано удобное приложение для демонстрации реализованных алгоритмов на различных изображениях.

Подытожим качество работы предложенных алгоритмов:

- Выделение карт** Хорошая сегментация и подсчет объектов на изображениях с однородным фоном, за исключением одного, на котором найденный

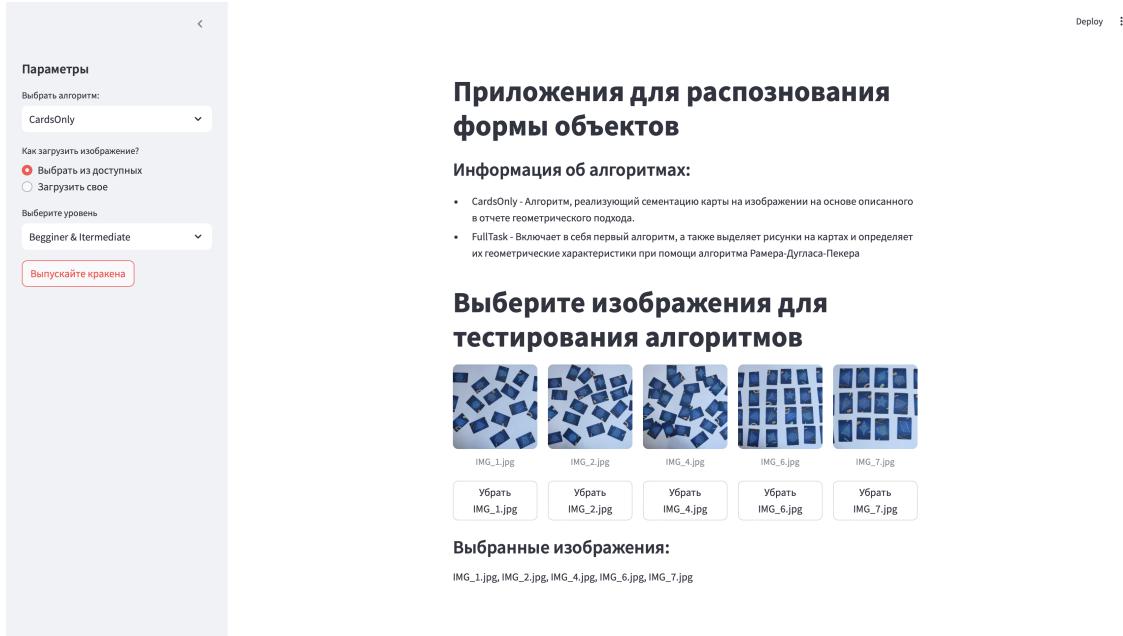


Рис. 15: Запуск алгоритма



Рис. 16: Индикатор успешного запуска

прямоугольник образуется двумя параллельными сторонами. Такой вариант позволяет посчитать карту, что приводит к корректному численному выводу алгоритма, но задача сегментации требует отдельного анализа, что привело бы к еще большему усложнению кода, поэтому было решено опустить этот случай. Очевидно, что несложным способом (например динамическим вычитанием из компоненты связности уже найденных карт) этот случай возможно обработать.

Deploy ⋮

## Результаты работы алгоритма

[← Предыдущий результат](#)    [Назад на главную](#)    [Следующий результат →](#)

**Начальное изображение**      **Результаты подсчета**

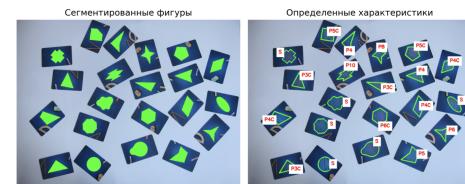


21

Найдено карт

Визуализация преобразований для алгоритма FullTask

**Промежуточные результаты**



[Показать меню](#)

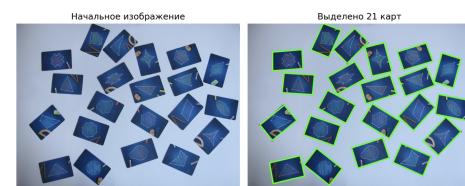
Рис. 17: Показ результатов на примере алгоритма, проводящего полный анализ изображения

Deploy ⋮

**Финальные результаты**



**Результаты поиска карт**



[Показать меню](#)

Рис. 18: Продолжение показа результатов

2. **Алгоритм анализа формы фигур.** Идеально сегментирует фигуры на всех изображениях на однотонном фоне, также со 100% точностью определяет выпуклость многоугольника. Основные препятствия корректного определения количества углов упомянуты выше: маленькое разрешение картинок (из-за чего дискретное представление сложно отличить от гладкой фигуры) и наличие небольшого наложения карт / шумов на централь-

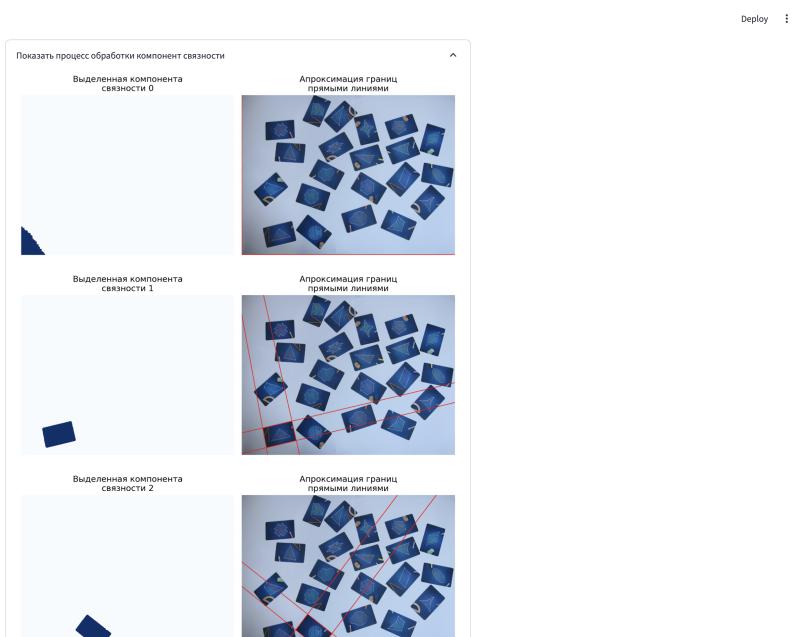


Рис. 19: Всплывающее окно для просмотра процесса подсчета карт

ную фигуру.

## **7 Литература**

### **Список литературы**

- [1] Гонзалес Р., Вудс Р. Цифровая обработка изображений. М., Техносфера, 2006.