Отчет по лабораторной работе №1

Обработка и распознавание изображений

Никифорова Мария Дмитриевна,
317 группа

Москва
Март 2023

Содержание

1 Поставновка задачи

Требовалось разработать и реализовать программу для работы с изображениями фишек игрового набора Тантрикс, производящую сегментацию и классификацию фишек по их номеру(уровень Intermediate).

2 Описание данных

На вход алгоритму распознавания подается изображение одной фишки или группы из несоприкасающихся фишек на однородном или немного зашумленном фоне. Также на избражении могут быть тени и блики от освещения. Изображения имеют расширение .bmp .

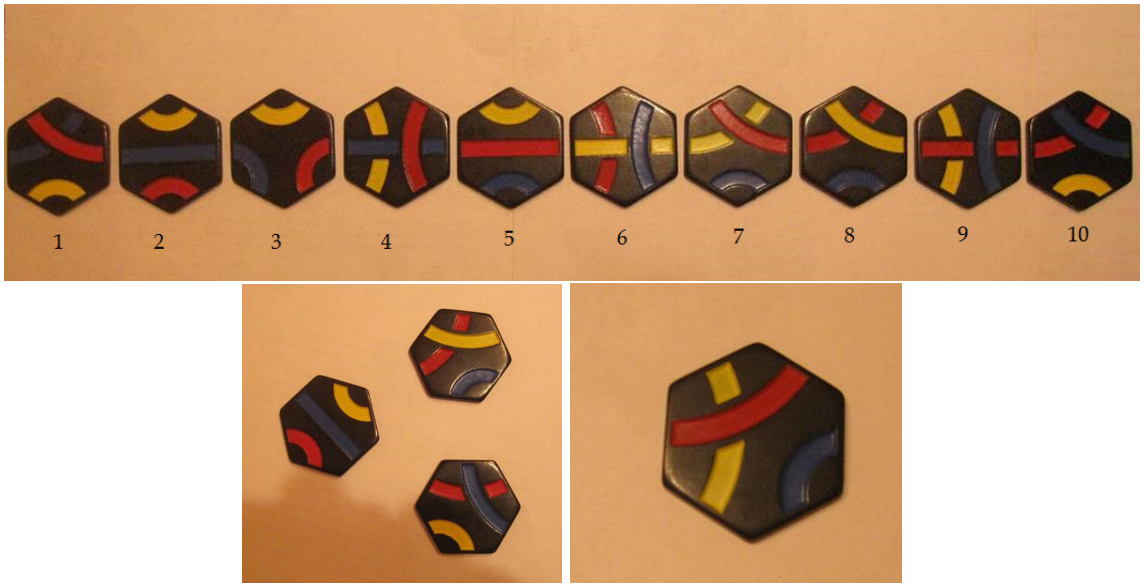

Рис. 1: Примеры изображений для распознавания

3 Описание метода решения

1) Сегментация
Сначала изображение переводилось в grayscale-формат, затем преобразовывалось по формуле $New\_Im(x, y) = 2 * Im(x, y) + 30$ для повышения контрастности, после этого проводилась бинаризация по прогу 250, в ходе которой фишки выделялись на общем фоне. Также при бинаризации выделялись сегменты линий желтого цвета, но, так как они находятся внутри границы фишки, на дальнейший поиск границы шестиугольника это не появлило.

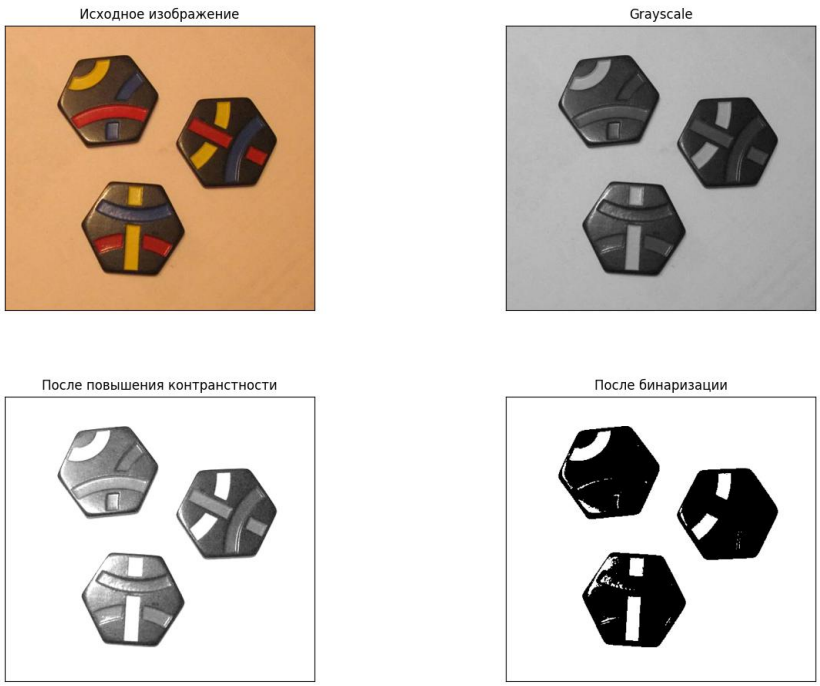

Рис. 2: Пример последовательного применения преобразований

После этого на бинаризованном изображении осуществлялся поиск контуров. Далее производился перебор найденных контуров и отбор тех из них, которые являются контурами фишек. Во избежание неправильного поиска границ фишки, связанного с бликами, которые после бинаризации имеют такой же цвет, как и фон, бралась выпуклая оболочка от контуров. Для отобранных конкуров фишек осуществлялась аппроксимация контура до шестиугольника. Для построенных шестиугольников высчитывались их центры, взятые как средние значения координат углов. Также для удобства строились ограничивающие прямоугольники для каждой фишки.

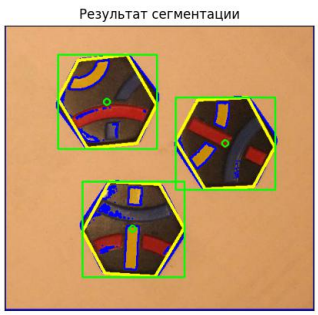

Рис. 3: Синим выделены найденные контуры, желтым - стороны многоульника, зеленым - центры шестиугольников и bounding boxes

2) Классификация

Классификация фишки производилась на основе последовательности расположения цветов сегментов линий, прилежащих к сторонам шестиугольника и взятых в направлении против часовой стрелки.

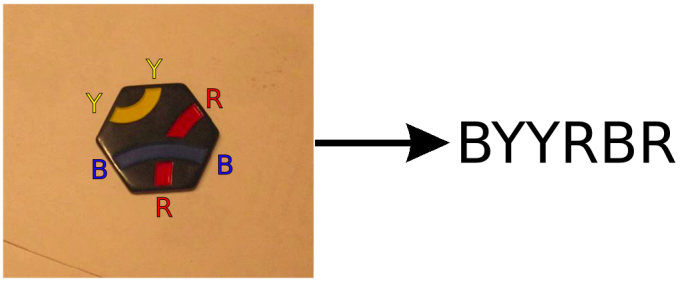

Рис. 4: Пример генерации последовательности цветов по изображению фишки

Осуществлялся перебор всех найденных многоугольников, и для каждого такого многоугольника строилась своя последовательность.

Для определения цвета линии, прилежащей к стороне, создавались маски каждого цвета, и выбирался тот цвет, пересечение которого с маской середины стороны максимально.

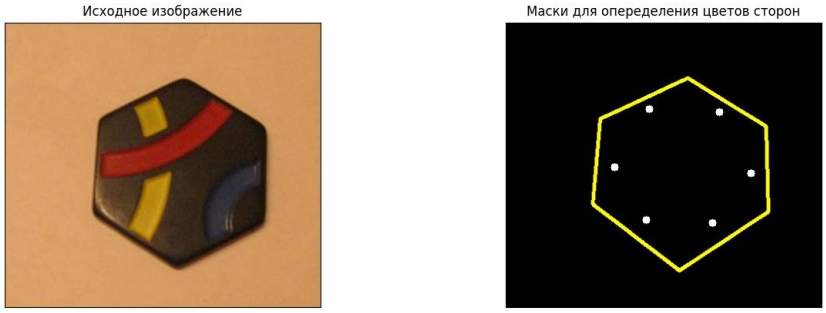

Рис. 5: Пример генерации масок для определения цветов линий, прилежащих к стороне

Маски цветов:

- *Синий*

  Для поиска сегментов линий синего цвета изображение фишки переводилось в цветовое пространство HSV, в котором выбирались те пиксели, которые лежат в диапазоне от (74, 0, 0) до (183, 75, 255). Также из исходного изображения выделялся канал синего цвета, элементы которого преобразовывались по формуле $New\_Im(x, y) = 4 * Im(x, y) + 10$, а затем бинаризовывались с порогом, равным среднему значению после преобразования. После этого на бинаризованном изображении выделялись связаные компоненты, и в итоговую маску для синего цвета попадали те компоненты, пересечение которых с маской из HSV имели прощадь большую, чем 30 процентов от всей площади маски в HSV. Такой подход обусловлен тем, что в HSV синий цвет выделяется не полностью, однако в этом цветовом пространстве он более стабилен. В RGB же синий выделяется полностью, но помимо цвета линии в маску также попадает шум.

Рис. 6: Пример генерации маски синего цвета

- *Желтый*

  Желтый цвет хорошо выделяется в канале зеленого цвета, поэтому проводилась бинаризация изображения из этого канала с порогом, равным среднему значению по каналу, увеличенному на 30, а после, для удаления шумов, применялось открытие с ядром размером (5, 5), состоящим из единиц.

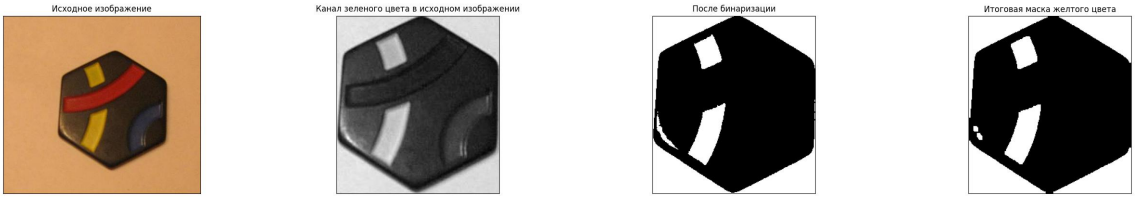

Рис. 7: Пример генерации маски желтого цвета

- *Красный*

  В канале красного цвета помимо самого красного цвета также хорошо выделяется желтый. Поэтому применялось вычитание из канала красного цвета канала зеленого, результат вычитания бинаризовывался по порогу, равному среднему значению, умноженному на 1.8, затем так же применялось открытие.

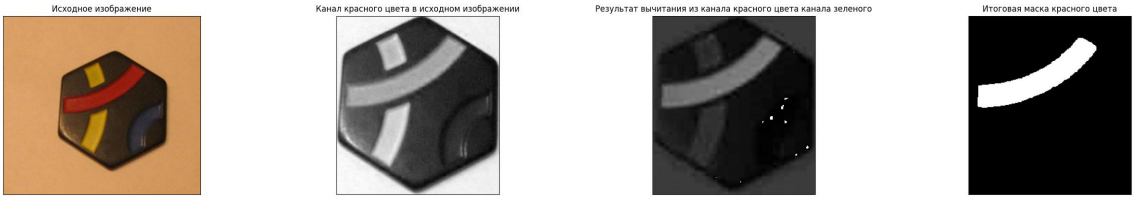

Рис. 8: Пример генерации маски красного цвета

Сопоставление последовательности цветов сторон
Так как фишки могут быть повернуты произвольным образом, то сгенерированная последовательность отличается от эталонной последовательности на величину циклического сдвига. Осуществлялся перебор эталонных последовательностей, и для каждой такой последовательности осуществлялись циклические сдвиги на величину от 0 до 5. Результат сдвига сравнивался с эталонной последовательностью, вычислялось, в скольких символах они отличаются, и в качестве ответа брался тот индекс, у которого эта величина минимальная при каком-то сдвиге.

4 Описание программной реализации

Программа написана на языке Python и оформлена в виде Jupyter-notebook. Использовались библиотеки Numpy, OpenCV, Matplotlib.

Вывод изображений, промежуточных результатов и ответа производился при помощи функций из библиотеки matplotlib.pyplot. Также для решения проблемы, связанной с тем, что изображение, открытое при помощи функции cv2.imread() имеет каналы BGR, а функция matplotlib.pyplot.imshow() принимает на вход изображения с каналами RGB, была реализована функция convert(), которая принимает на вход изображение с каналами BGR и возвращает это же изображение с каналами RGB.

Далее приведен список используемых функций и описание того, для чего они применялись. Для перевода изображения в оттенки серерого и формат HSV: cv2.cvtColor(img, cv.COLOR_BGR2GRAY), cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

Для бинаризации: cv2.threshold(gray_, threshold, max_value, cv2.THRESH_BINARY), в которой задается изображение в оттенках серого, порог бинаризации и максимальное значение.

Для повышения контрастности: cv2.convertScaleAbs(img, alpha, beta), которая преобразовывает изображение в оттенках серого по формуле $New\_Im(x, y) = alpha * Im(x, y) + beta$

Для поиска контуров: cv2.findContours(img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

Для поиска выпуклой оболочки: cv2.convexHull(points)

Для аппроксимации контура: cv2.approxPolyDP(contour, eps, True)

Для вычисления среднего: np.mean(array)

Для построения ограничивающего прямоугольника: cv2.boundingRect(hexagon)

Для взятия диапазона HSV: cv2.inRange(hsv, hsv_min, hsv_max)

Для поиска связных компонент: cv2.connectedComponents(img, connectivity, cv2.CV_32S)

Для реализации открытия: cv2.morphologyEx(yet, cv2.MORPH_OPEN, kernel).

5 Эксперименты

На всех выданных тестовых изображениях предложенный алгоритм показал верный результат.

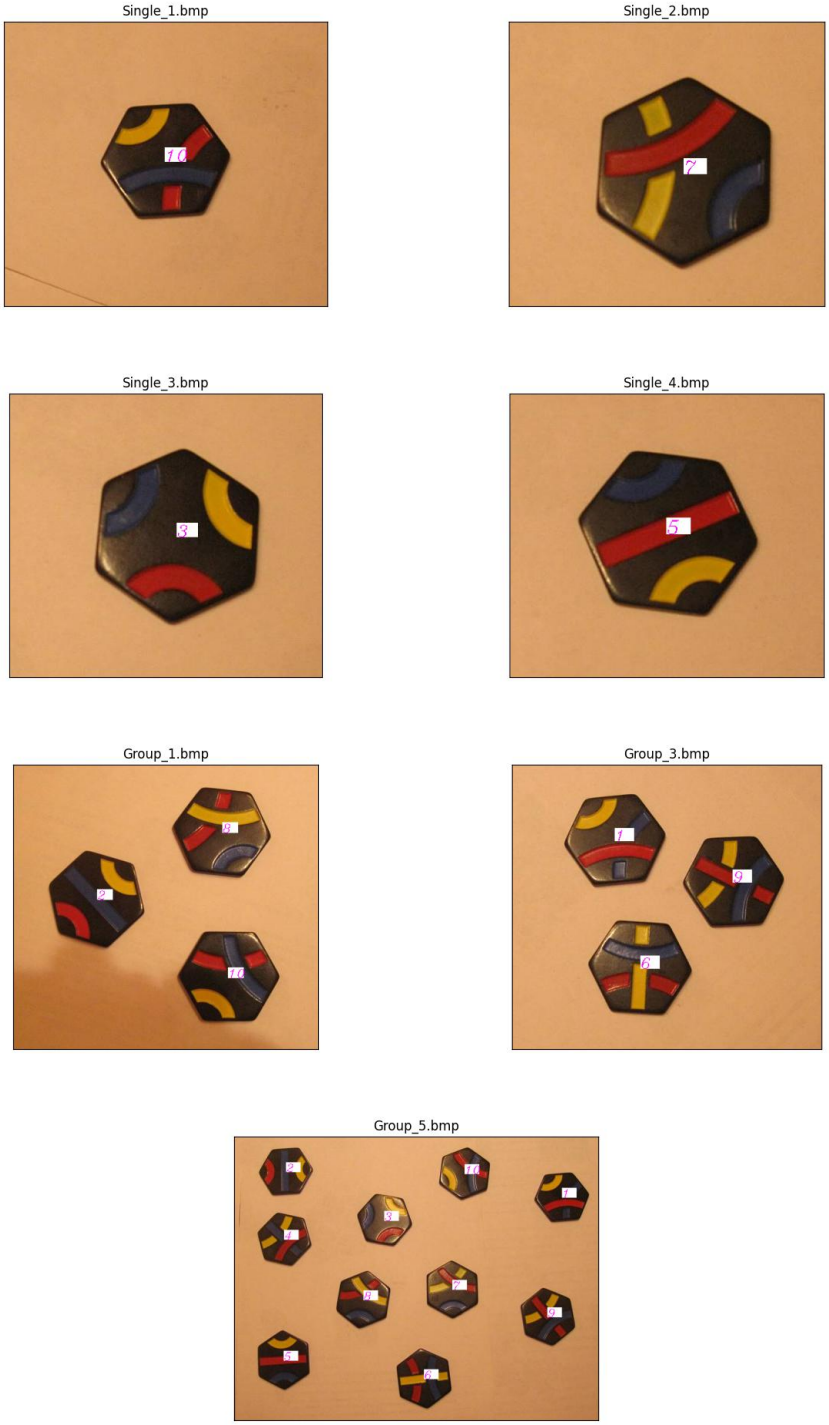

Рис. 9: Результат применения алгоритма к тестовым изображениям

В ходе поиска решений проводилось множество различных экспериментов. Например, для сглаживания границ фишек и нивелирования влияния бликов после повышения контрастности изображения в оттенках серого применялся фильтр Гаусса, но в таком случае существовал риск того, что близкорасположенные фишки могут "слипнуться".

6 Выводы

Предложенный алгоритм показал хорошие результаты на тестовых изображениях, однако является неустойчивым, так как определение цветов базируется на эмпирических наблюдениях, и, вероятно, будет давать неверную классификацию, если качество цветов на фотографии будет плохим. Создание последовательности на основе цветов середины сторон шестиугольника является более устройчивой процедурой, так как даже если фотография будет сделана под углом, то расположение этих точек останется прежним. Однако здесь корректному выделению границ многоульника может помешать зашумленный или неконтрастный фон. Также алгоритм не умеет сообщать пользователю, что на вход подана неправильная картинка.