Лабораторная работа №1. Изучение и освоение методов обработки и сегментации изображений.

Петренко Дарья, 317 группа

8 апреля 2019

Содержание

Постановка задачи

Для решения была выбрана задача класса Intermediate.

Задание предполагает работу с изображениями фишек игрового набора Тантрикс. Каждая фишка представляет из себя правильный шестиугольник черного цвета, на который нанесено по одной линии каждого из трех цветов: красного, синего и желтого. Каждая линия представляет из себя короткую дугу большой кривизны, длинную дугу малой кривизны или прямолинейный сегмент. Форма и взаимное расположение цветных линий однозначно определяют номер фишки. Всего в игровом наборе 10 фишек, пронумерованных от 1 до 10. Изображение пронумерованных фишек из набора содержится в описании задания.

Задание состоит из двух частей. В первой по входному файлу типа Single_*.bmp необходимо определить номер изображенной фишки.

Входные данные второй части состоят из файла типа Group_*.bmp, требуется определить расположение и номера всех фишек в кадре.

Описание данных

Для отладки решения к заданию прилагается набор изображений в формате bmp, которые делятся на три группы в зависимости от количества и взаимного расположения фишек. Файл типа Single_*.bmp содержит изображение одиночной фишки, файл типа Group_*.bmp - нескольких несоприкасающихся фишек. Наиболее сложными считаются изображения группы соприкасающихся фишек (файл Path_*.bmp), они необходимы только для выполнения задания класса Expert и в данной работе использованы не были.

Описание метода решения

Для решения первой и второй части задания использовался один и тот же метод, так как задачи, предлагаемые для решения, аналогичны, различия заключаются только в типе входного файла и требованиях к формату выходных данных.

Изображение считывается в цветном и черно-белом вариантах. Черно-белое используется для обработки, а цветное - для вывода на экран промежуточной разметки и итогового результата (в случае, если решается второе задание).

На первом шаге алгоритма черно-белое изоражение бинаризуется: пикселям, значение которых меньше заданного порога, присваивается значение 0, а остальным 1. В полученном бинарном изображении выделяются компоненты связности, и для каждой из компонент строится выпуклая оболочка. Чтобы сгладить неровности на краях получившихся фигур, к полученной матрице применяется операция эрозии.

Следующим шагом алгоритма является распознавание контуров элементов полученного бинарного изображения. Для каждой компоненты связности изображения получаем массив координат точек ее контура.

В цикле обрабатываем полученные массивы координат, чтобы понять, какие из них могут соответствовать контурам фишки. Для начала убираем из рассмотрения фигуры, периметр которых меньше заданного значения (слишком маленькие, чтобы быть фишками). Границы оставшихся аппроксимируем с использованием функции, реализующей алгоритм Дугласа-Пеккера, на выходе получаем массив угловых точек фигуры. Считаем фигуру правильным шестиугольником, а следовательно, и фишкой, только если она имеет 6 вершин, а длина каждого ребра отличается от $1/6$ периметра не больше, чем на заданное маленькое значение.

На следующем шаге алгоритма для каждой из фишек определим, через какую пару ребер проходит каждая из цветных линий. Для каждого из трех цветов введем список, в который будем вносить индексы ребер, через которые, по нашему предположению, проходит линия соответствующего цвета.

Построим вспомогательный массив col_sample с образцами компонент bgr каждого из цветов, встречающегося в фишках: красного, синего, желтого, черного. Чтобы найти точку, лежащую на цветной полосе вблизи каждого из ребер, будем в цикле перебирать точки из окрестности центра ребра, немного смещаясь к одной из вершин или к центру фишки (вспомогательные коэффициенты для реализации такого перебора также хранятся в отдельном массиве). На каждой итерации будем находить компоненту массива col_sample, наиболее близкую к цвету рассматриваемой точки по евклидовой метрике (сравниваются трехмерные векторы компонент bgr). Выходим из цикла, если найденная компонента не соответствует черному цвету, и считаем цвет этой компоненты цветом линии, проходящей через ребро.

Если обход по точкам из окрестности завершен, а цвета всех рассмотренных точек отнесены алгоритмом к черному, то также выходим из цикла и считаем, что через ребро проходит синяя линия. Было сделано именно такое предположение, так как синий цвет наиболее близок к черному; более того, допущенные ошибки будут исправлены на следующем шаге алгоритма.

Рассмотрим списки, в которые мы вносили индексы ребер. Если в каждом списке по два элемента, то считаем, что все цвета распознаны верно, и переходим к следующему шагу. В противном случае в цикле исправляем допущенные на предыдущем шаге ошибки. На каждой итерации работаем с двумя списками, в одном из которых больше двух элементов, а в другом меньше. Для каждого ребра из большего списка восстанавливаем координаты точки, в которой рассматривался цвет, и сравниваем его с образцом цвета, соответствующего меньшему списку (как и раньше, находим евклидово расстояние между векторами).

Перемещаем ребро, цвет точки которого больше остальных похож на образец, из большего списка в меньший. Выходим из цикла, когда во всех списках станет по два элемента.

На текущем этапе алгоритма для каждого из трех цветов имеем список с номерами двух ребер шестиугольника, через которые проходит линия этого цвета. Осталось по этим данным определить номер фишки. Сначала для каждого из цветов найдем количество ребер шестиугольника, лежащего между вершинами этого цвета (выбираем порядок обхода, для которого это число будет наименьшим). Найденное значение однозначно определяет форму сегмента: 1 - короткая дуга, 2 - длинная дуга, 3 - прямая линия.

Воспользуемся изображением десяти фишек набора, приведенным в описании задания, и составим матрицу соответствия номеров фишек и форм сегментов трех цветов. Информация о форме позволяет однозначно идентифицировать почти все фишки, кроме двух пар: 1 и 10, 7 и 8. Обработаем эти случаи отдельно.

Чтобы отличить фишку с номером 1 от фишки с номером 10, воспользуемся следующим фактом: если совершить обход ребер шестиугольника против часовой стрелки, начиная с любого из двух ребер, через которые проходит желтая линия, то красный сегмент будет пересечен раньше синего в случае работы с фишкой №1 и позже синего в случае работы с фишкой №10. Выполним преобразование номеров ребер, соответствующее повороту шестиугольника, так, чтобы номер 0 имело одно из желтых ребер. Теперь достаточно сравнить меньшие из номеров ребер в парах, соответствующих красному и синему цветам, чтобы определить номер фишки. Аналогично действуем и в случае фишек 7 и 8, вместо желтой дуги сделав началом отсчета синюю.

Когда номер каждой фишки определен, нанесем эту информация на копию исходного изображения и выведем его на экран, если выполняется задание №2. В случае выполнения задания №1 просто напечатаем найденный номер.

Описание программной реализации

Программный код написан на языке Python с использованием библиотек skimage, cv2.

Для считывания изображения из файла использовалась функция imread из библиотеки cv2, а для вывода на экран - функция imshow из библиотеки matplotlib.pyplot (для корректного отображения необходимо предварительно перевести изображение из представления brg, используемого при работе с функциями cv2, в rgb).

Бинаризация черно-белого изображения производилась с использованием функции cv2.threshold. Компоненты связности в бинаризованном изображении выделялись при помощи функции label библиотеки skimage.measure. Функция возвращает количество найденных компонент, а также целочисленную матрицу, по размеру совпадающую с входной. Нулевые пиксели

входной матрицы остаются нулевыми и в выходной, а остальным присваиваются значения от 1 до числа компонент связности так, что элементы матрицы, лежащие в одной компоненте связности, имеют одно и то же значение, уникальное для каждой компоненты.

Для построения выпуклой оболочки использовалась функция convex_hull_image из библиотеки skimage.morphology, а для применения операции эрозии - функция erosion из той же библиотеки.

Распознавание контуров производилось с использованием функции cv2.findContours. Она реализует алгоритм, описанный в статье Сатоши Сузуки "Topological structurcal analysis of digitized binary images by border following" , и для каждого найденного контура возвращает вектор его точек. Этот вектор в дальнейшем подвергался аппроксимации с использованием функции cv2.approxPolyDP, реализующей алгоритм Дугласа-Пекера.

Для нанесения разметки на копию исходного изображения использовались функции circle, rectangle, drawContours, putText из библиотеки cv2.

Эксперименты

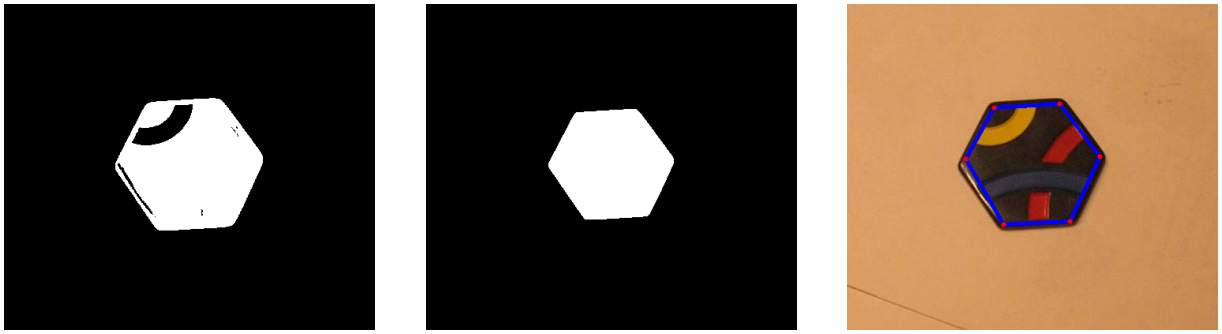
Проведем серию экспериментов на данных из предоставленой для отладки выборки.

Задание №1

В этой части задания на вход подается файл Single_*.bmp, а выходными данными является напечатанный в тесктовом виде номер изображенной фишки. Ниже приведены результаты работы программы для нескольких файлов: промежуточные изображения и итоговый результат.

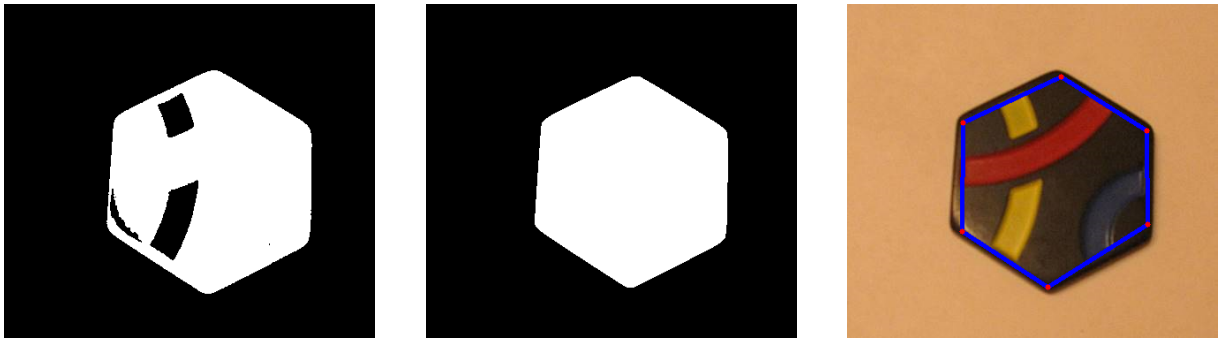
Промежуточные изображения для каждого файла соответствуют следующим этапам работы программы:
1) получено после бинаризации;
2) получено после выделения компонент связности, построения выпуклой оболочки и применения операции эрозии;
3) на исходном изображении отмечены углы и ребра найденных шестиугольников.

Файл Single_1.bmp



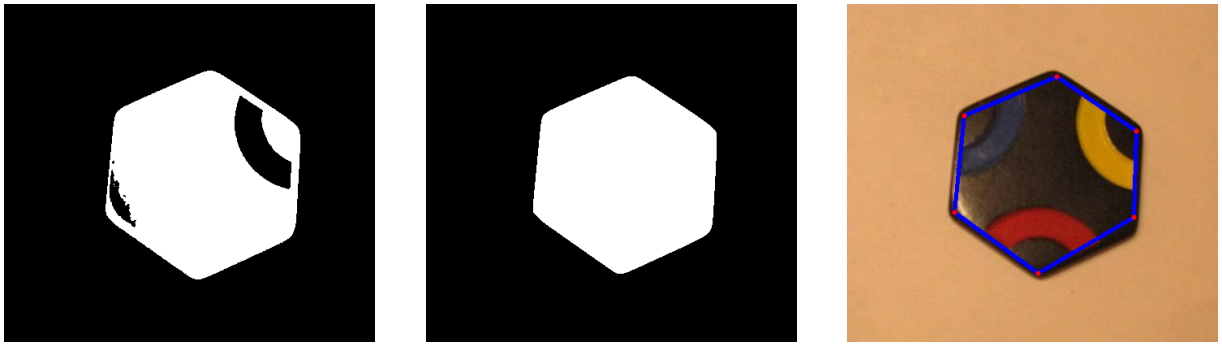
Вывод программы: фишка №10

Файл Single_2.bmp



Вывод программы: фишка №7

Файл Single_3.bmp

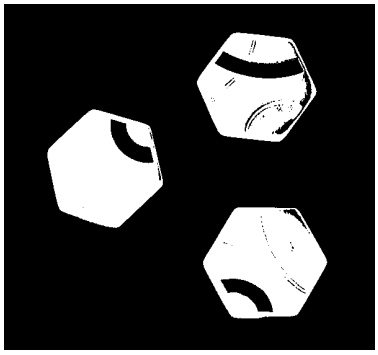
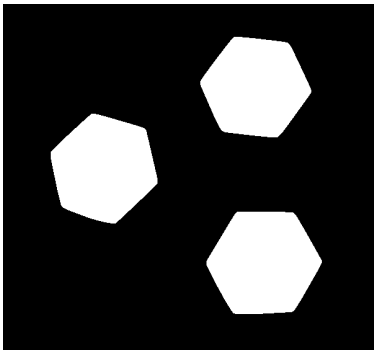
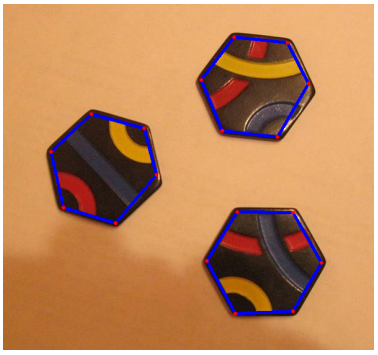


Вывод программы: фишка №3

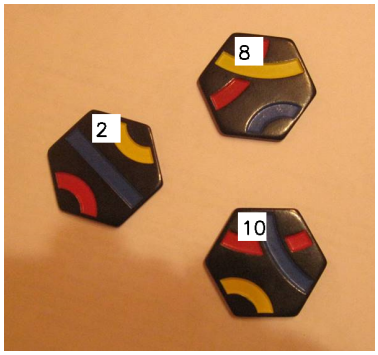
Задание №2

В задании №2 на вход подается файл Group_*.bmp, выходными данными является копия этого файла с нанесенными на нее номерами фишек. Для каждого из рассмотренных файлов приведем результаты промежуточных шагов, аналогичных первой части задания, а также итоговое изображение.

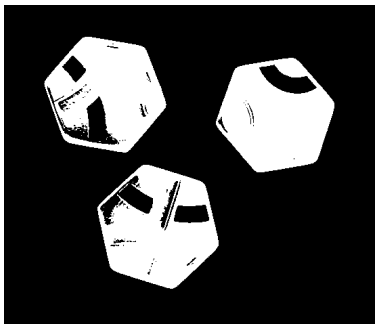
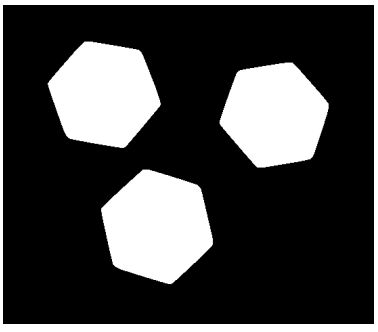
Файл Group_1.bmp

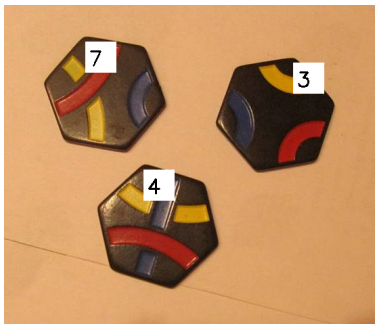
Вывод программы:



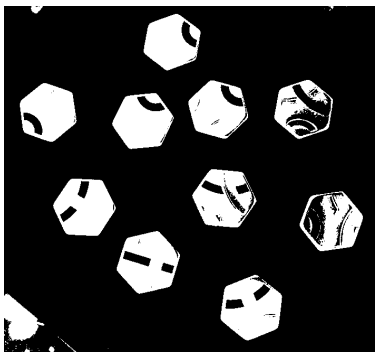
Файл Group_2.bmp

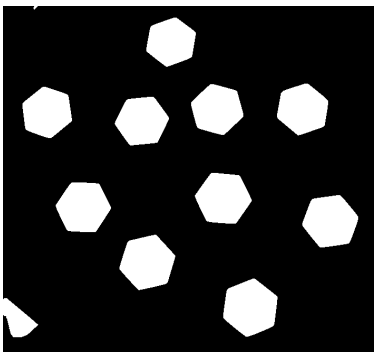



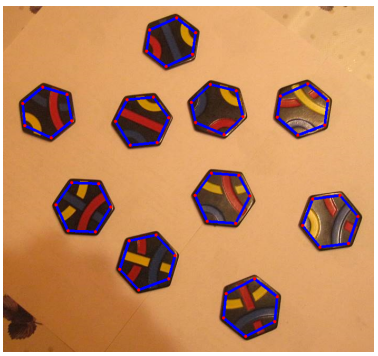

Вывод программы:



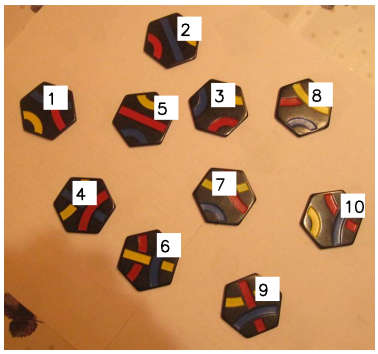
Файл Group_4.bmp





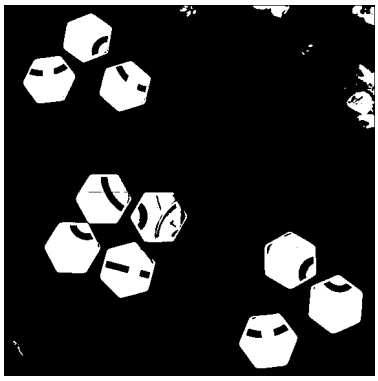
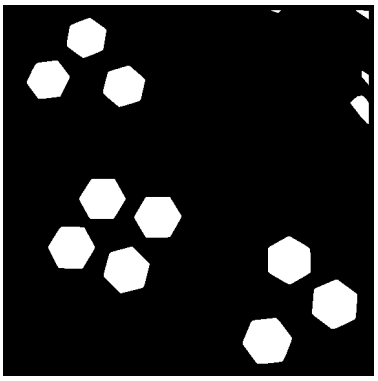
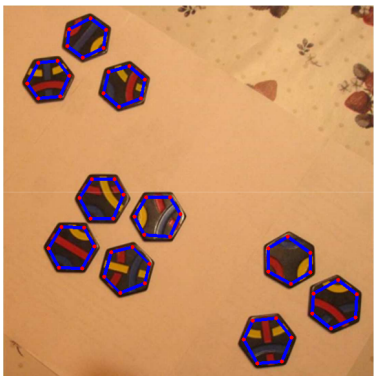


Вывод программы:

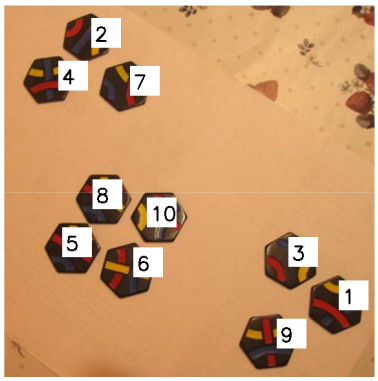


Файл Group_10.bmp

Файл представляет из себя скриншот из описания задания.

Вывод программы:



Вывод

Как видно из приведенных результатов экспериментов, все фишки были безошибочно найдены, а их номера корректно определены. Задание можно считать успешно выполненным.