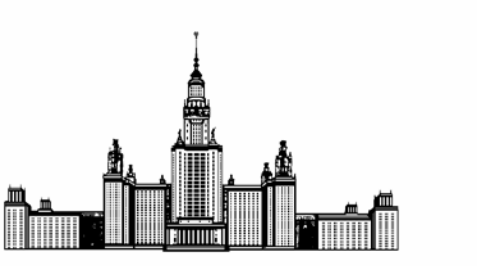
Московский государственный университет
имени М. В. Ломоносова



Факультет Вычислительной Математики и Кибернетики

Кафедра Математических Методов Прогнозирования

Практическое задание по курсу «Обработка и распознавание изображений»

ЛАБОРАТОРНАЯ РАБОТА № 1

«Изучение и освоение методов обработки и сегментации изображений»

Выполнил студент 3 курса
317 группы:

Серов Сергей Сергеевич

Преподаватель:

Местецкий Леонид Моисеевич

Москва, 2017

1. Постановка задачи

В задании требуется разработать и реализовать программу для работы с изображениями фишек игрового набора Тримино.

Программа должна обеспечить:

- ввод и отображение на экране изображений в формате BMP;
- сегментацию изображений на основе точечных и пространственных преобразований;
- поиск фишек на картинках;
- классификацию фишек на картинках.

Для отладки и обучения алгоритма к заданию прилагаются 9 изображений различной сложности. Мы будем использовать изображения класса **Expert** для обучения и демонстрации работы программы. На этих изображениях фишки расположены на пестром фоне (есть изображения ягод клубники на скатерти, которая является фоном) с неоднородным освещением. Примеры таких изображений представлены ниже.

Pict_3_1.bmp

Pict_4_2.bmp

В задание входят две задачи на изображениях разной сложности:

1. Определить положение фишек на изображении;

2. Определить маркировку фишек на изображении.

Выход программы – текстовый файл, в котором каждая запись описывает положение и код одной фишки в следующем формате:

N – количество фишек на картинке,

X, Y; m1, m2, m3;

Здесь (X, Y) - координаты центра фишки на изображении (X - номер столбца, Y - номер строки), m1, m2, m3 – код фишки – количество точек в углах треугольника.

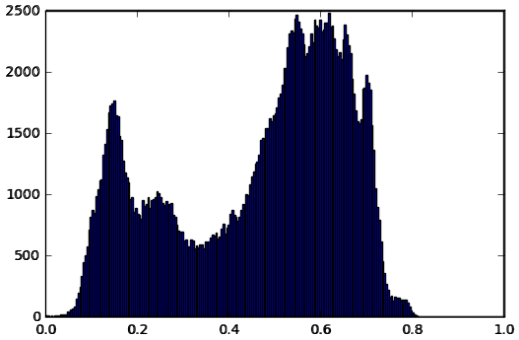
Считается, что положение фишки определилось верно, если отклонение от истинного центра составляет не более 60 пикселов.

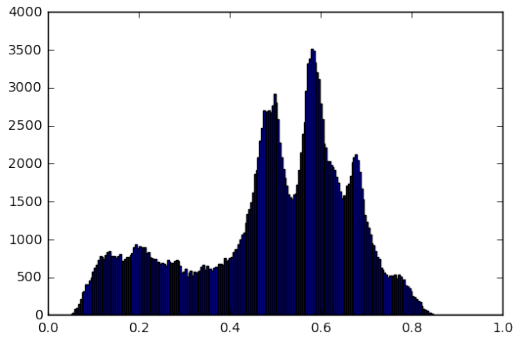
2. Метод решения

Опишем разработанный метод решения задачи поэтапно. Выберем два фрагмента входных изображений класса Expert, на которых будем демонстрировать результаты работы каждого этапа.

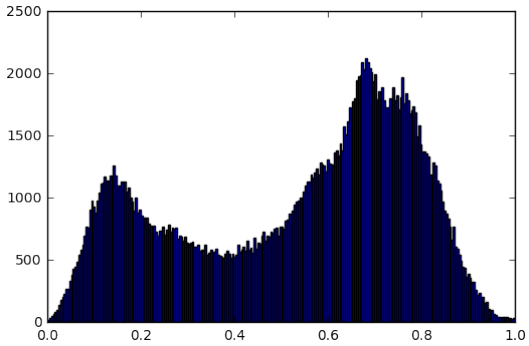
1. Эквализация гистограмм

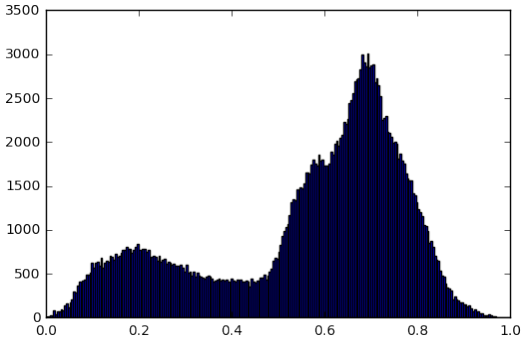
На первом этапе решения построим гистограммы имеющихся изображений.



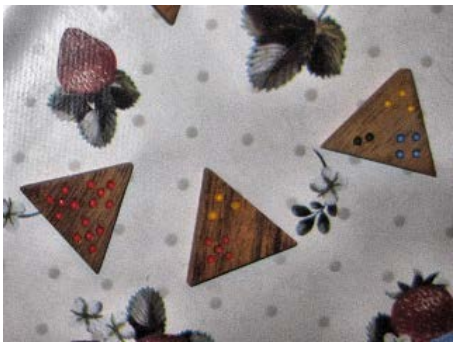


Заметим, что существуют отрезки значений яркости пикселей, на которые почти не приходится пикселей этих изображений. Соответственно, рястянем гистограмму на весь диапазон [0;1] с помощью функции **equalize_adapthist**.



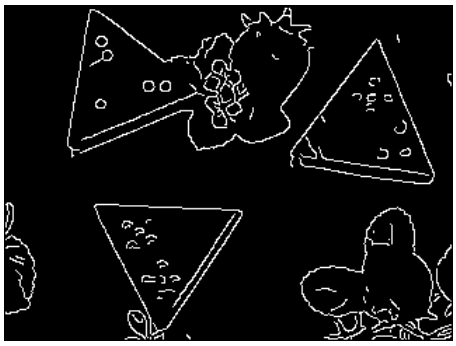


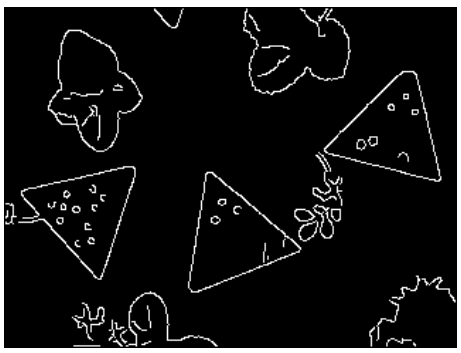
Приведем полученные изображения с эквализированными гистограммами.



2. Выделение границ

На втором этапе решения выделим границы на все трех каналах цветовой модели RGB с помощью детектора границ Кэнни, реализованного в scikit-image в функции **feature.canny**. Зададим в качестве параметров подобранные верхний и нижний пороги для отсечения сильных границ и последующего отсечения немаксимумов из слабых границ внутри этой функции. Приведем результат второго этапа.

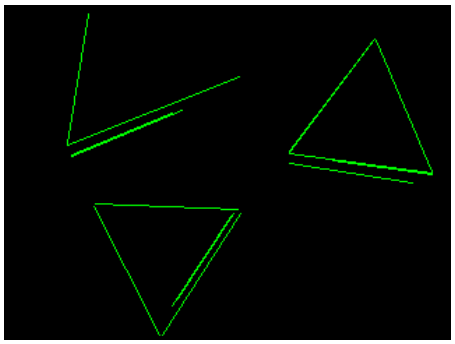


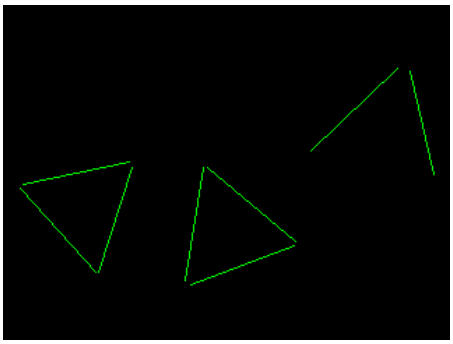


3. Выделение прямых линий

На третьем этапе используем преобразование Хаффа для детектирования прямых линий на изображениях, полученных на втором этапе. Воспользуемся реализацией вероятностного преобразования Хаффа в

библиотеке scikit-image – **transform.probabilistic_hough_line**. Зададим в качестве параметров минимальную длину искомых линий и максимальное число пикселей, на которое граница может отличаться от идеальной прямой. Результат приведем на нижеследующих рисунках.





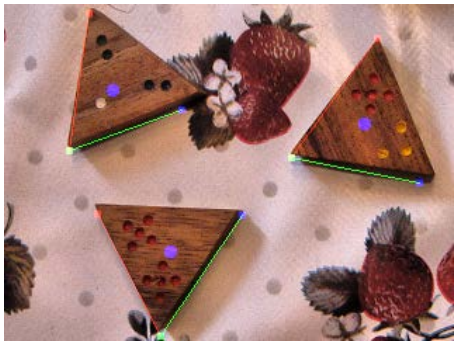
4. Поиск треугольников среди линий

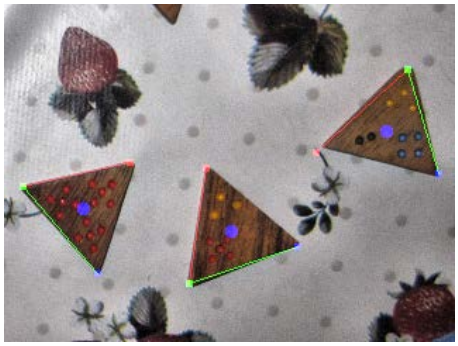
На четвертом этапе будем отбирать среди всех найденных линий те, которые, по нашему предположению, образуют нужные нам треугольники. Будем рассматривать все пары выделенных линий.

Выберем некоторые отличительные признаки этих пар:

- длина линий лежит в некоторых подобранных пределах;
- строго один из концов одной линии лежит достаточно близко к одному из концов другой линии;
- косинус угла между линиями составляет 0.5±0.1.

Будем предположительно считать треугольниками все пары прямых. После этого удалим треугольники, у которых евклидово расстояние между вершинами двух найденных нами углов меньше заданного порога и треугольники, в которых косинус был близок к 0.5 в случае тупого угла. После этого вычислим координаты центра каждого треугольника как покоординатное среднее арифметическое его трех вершин и удалим близко лежащие центры треугольников, заменив их одним с усредненными вершинами и центром. На картинках ниже линиями и маленькими квадратами обозначены стороны и углы треугольников, а синими кругами – их центры.

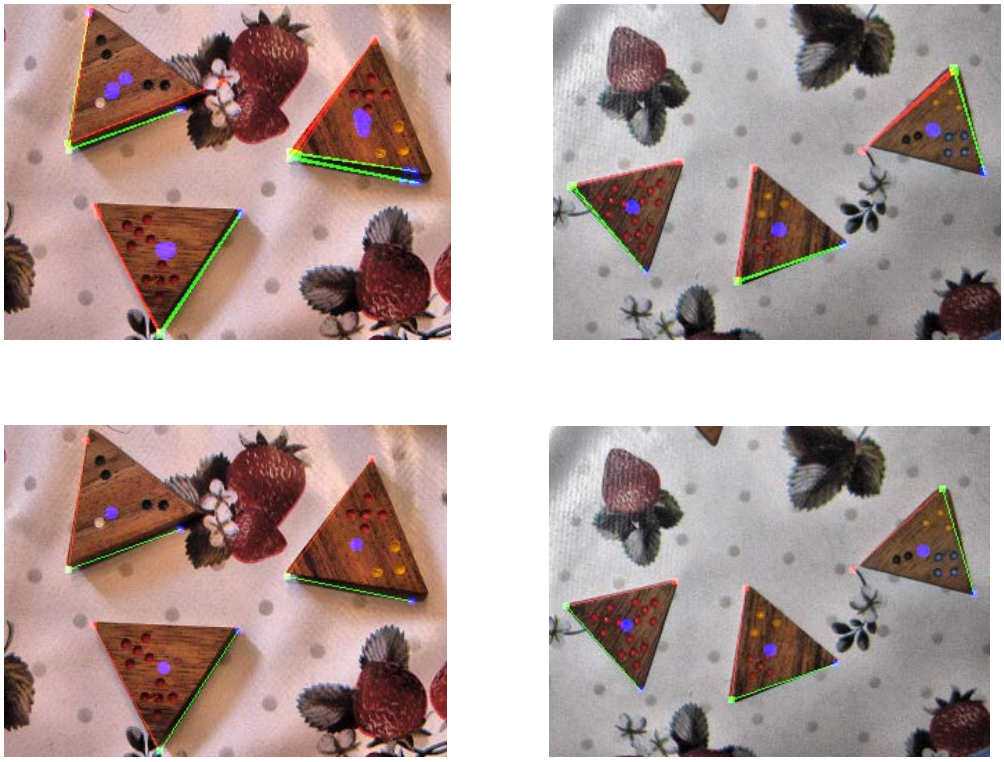




5. **Многократный перезапуск алгоритма нахождения линий и поиска треугольников**

На пятом этапе многократно повторим запуски преобразования Хаффа, выделяющего линии и алгоритма отбора треугольников из найденных линий. Удалим из полученного множества те треугольники, для которых минимальное евклидово расстояние от вершин до вершин других треугольников, меньше заданного порога. Каждый раз будем усреднять координаты центра.

Продемонстрируем сначала результат многократного запуска вышеуказанных алгоритмов, а затем результат удаления лишних треугольников.



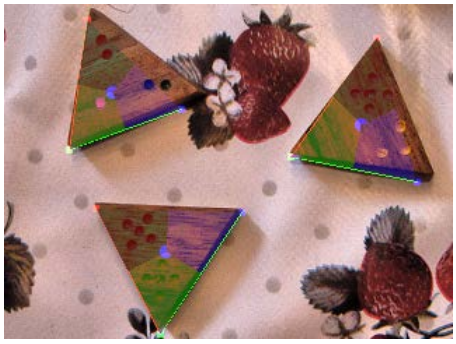
6. **Удаление некорректных треугольников по цветовым признакам**

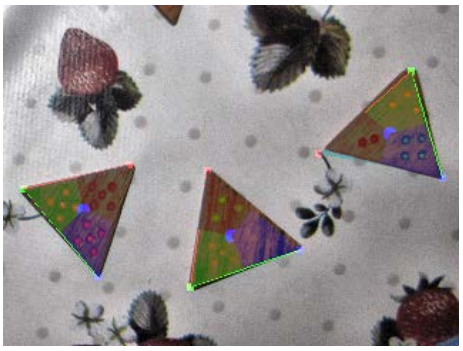
На шестом этапе заметим, что иногда встречаются ситуации, когда после всех пройденных этапов треугольник может находиться в ягоде клубники на скатерти, которая является фоном. Для удаления таких эффектов предлагается для центров всех полученных треугольников вычислить значение дисперсии величины красного канала по 100 точкам, разбросанным в квадрате, описанном около данного центра, с заданной длиной стороны. Если значение дисперсии больше некоторого предельного значения, то треугольник удаляется.

7. **Разрезание треугольников на 3 четырехугольника**

На седьмом этапе разрежем треугольник на 3 четырехугольника, чтобы в дальнейшем определять количество точек около каждого из углов треугольника. Для этого, зная координаты центра и одного угла (который, как мы считаем, найден достаточно точно), опустим высоты на все 3 стороны

треугольника и вырежем четырехугольники, каждый из которых ограничен двумя высотами и частями двух смежных сторон. Эта процедура проиллюстрирована на нижеследующих картинках. Четырехугольники выделены разными цветами.





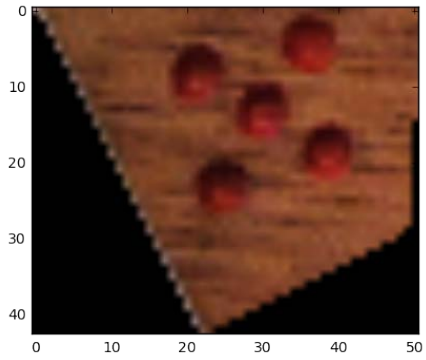
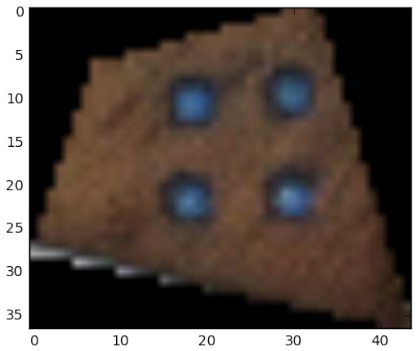
8. Подсчет количества точек в каждом четырехугольнике

Сделаем очень важное наблюдение. На всех обучающих картинках выполнено следующее: количество точек в углу треугольника однозначно определяется по цвету этих точек. Отсюда делаем основополагающий вывод: достаточно определить **цвет** одной точки в каждом углу треугольника, чтобы дать точный ответ о количестве точек в этих углах.

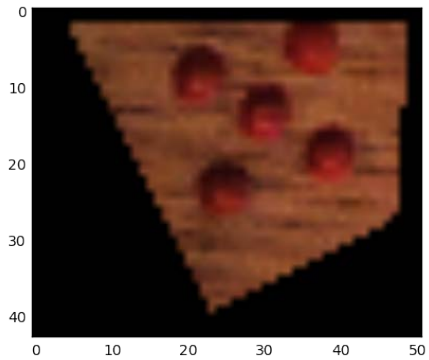
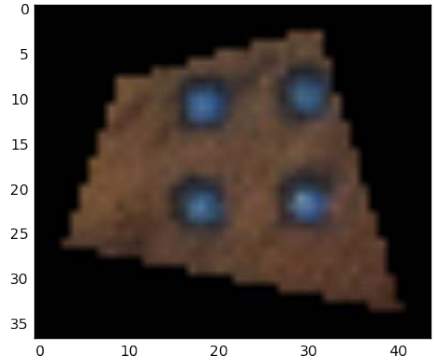
Сначала сгладим изображения всех четырехугольников фильтром Гаусса со значением sigma=0.2 (**skimage.filters.gaussian**). Перейдем в цветовую модель HSV и подадим на вход алгоритму поиска аномалий Isolation Forest (**sklearn.ensmble.IsolationForest**) значения этих трех каналов для всех точек каждого вырезанного четырехугольника. На выходе для каждой точки получим значение решающей функции, характеризующей аномальность этой точки. Вычислим порог бинаризации методом Отсу (**skimage.filters.threshold_otsu**), а затем применим к полученной бинарной картинке медианный фильтр с ядром размера 3*3 (**skimage.filters.median**) и морфологическую операцию закрытия с ядром 2*2 (**skimage.morphology.binary_closing**). На полученном бинарном изображении выделим связные компоненты (**skimage.measure.label**, **skimage.measure.regionprops**) и отберем любую из них, которая соответствует подобранным параметрам по площади (**region.area**), эквивалентному диаметру (**region.equivalent_diameter**) и эксцентриситету (**region.eccentricity**). Для этой компоненты вокруг ее центра (**region.centroid**) построим круг диаметра, равного эквивалентному диаметру и посчитаем медиану величин всех трех каналов в модели HSV. Далее, зная априорные центральные и наиболее часто встречающиеся параметры цветов точек (белого, черного / зеленого, желтого, синего и красного), сначала определим черный и белый цвета по значению каналов Value и Saturation, а затем найдем цвет, до которого евклидово расстояние по каналу Hue от

посчитанной медианы до априорных значений минимально. Сопоставим цвет с количеством точек в этом углу и выведем ответ.

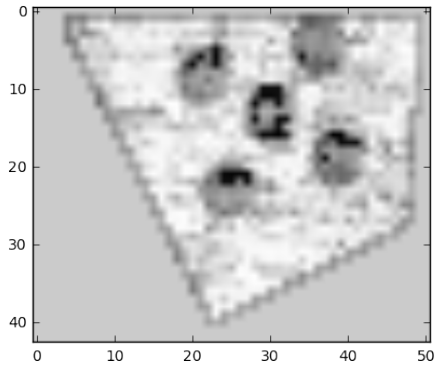
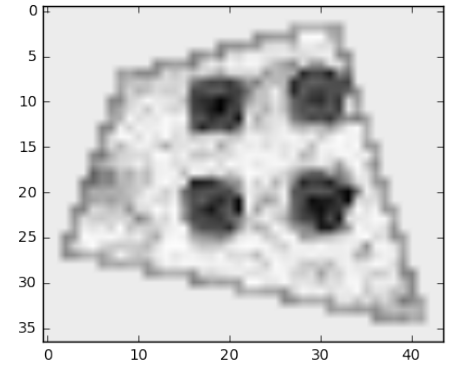
Нижеследующие картинки иллюстрируют описанный алгоритм.

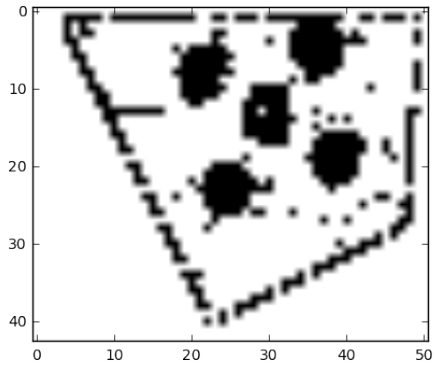
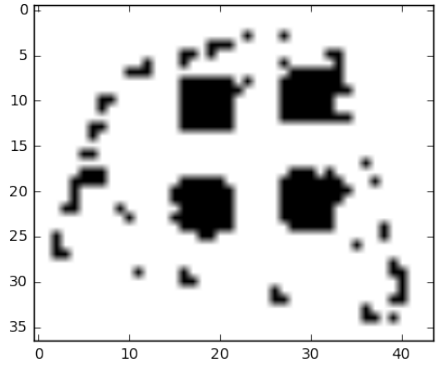
Исходные четырехугольники.

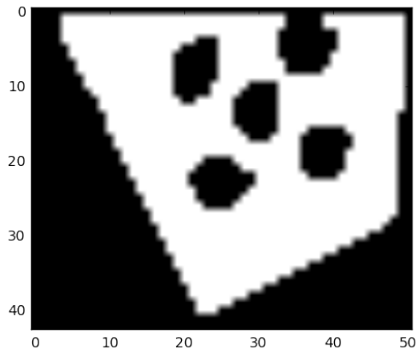
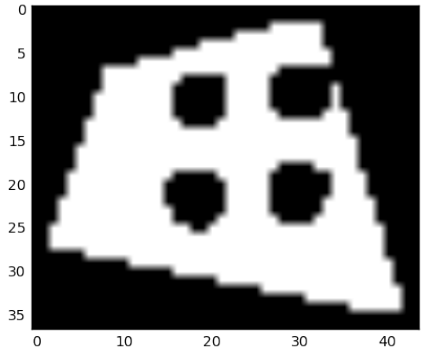
Применен фильтр Гаусса.

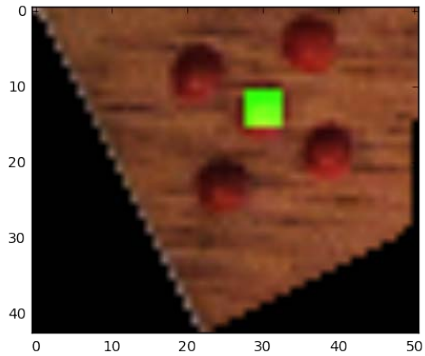
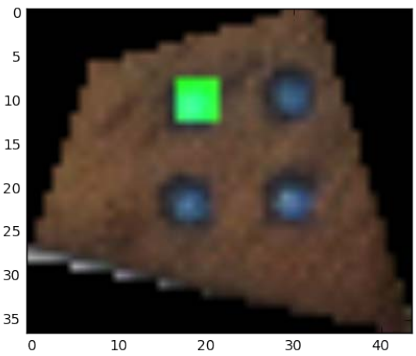
Выход алгоритма Isolation Forest.

Сделана бинаризация по Отсу.

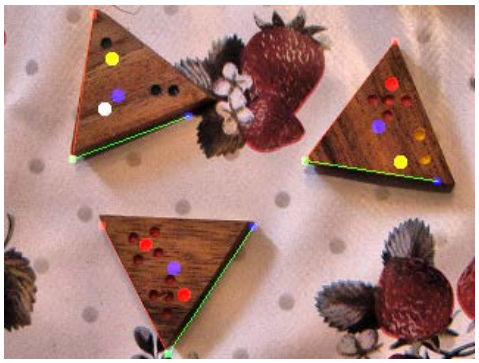
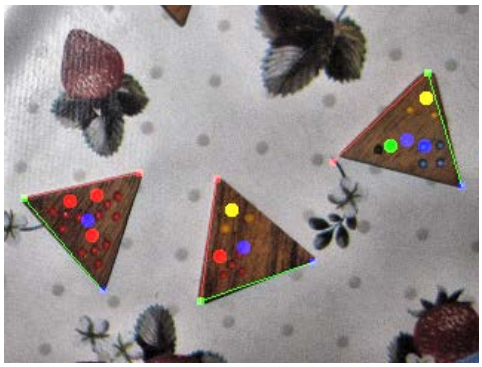



Применен медианный фильтр и операция закрытия.

Выделена одна любая точка на четырехугольнике.

Приведем итоговый результат для двух рассматриваемых фрагментов изображений. Здесь цвет точки кодирует количество точек в каждом углу.

3. Программная реализация

Программа реализована на языке Python 3.5 в среде IPython Notebook с использованием библиотек numpy, scipy, matplotlib, scikit-image и scikit-learn. К отчету прилагается файл формата .ipynb, содержащий код всех описанных ниже процедур.

Полное решение задачи уровня Expert может быть получено путем последовательного запуска всех ячеек в приложенном файле .ipynb после изменения пути к изображениям в одной из них.

Далее приведем прототипы всех используемых небиблиотечных функций.

def equalize_hist(filename):

Эта функция применяет к изображению по пути filename адаптивную эквализацию гистограммы и сохраняет результат в другую папку.

def Canny(img):

Эта функция применяет детектор границ Кэнни к каждому цветовому каналу заданного изображения и сохраняет результат в папку.

def Hough(canny):

В этой функции ко входному изображению применяется вероятностное преобразование Хаффа для выделения прямых линий на нем. Также в ней сохраняется демонстрация результата, а в качестве результаты выдается массив найденных отрезков.

def check_length(lengths, dim, line_num):

Эта функция проверяет, лежит ли длина найденного отрезка под номером line_num в измерении dim массива lengths в определенном диапазоне.

def check_closeness(lines, dim, line_num_1, line_num_2):

Эта функция проверяет, лежит ли хотя бы один конец первого заданного отрезка из массива lines рядом с хотя бы одним концом второго заданного отрезка.

def check_angle(lines, lengths, dim, line_num_1, line_num_2):

Эта функция проверяет, лежит ли косинус угла между двумя заданными отрезками в определенном диапазоне.

def delete_identical_triangles(triangles):

Эта функция удаляет из списка треугольников triangles все, кроме одного, из каждой группы тех, центры которых лежат ближе определенного расстояния.

def delete_obtuse_angles(triangles):

Эта функция удаляет из списка треугольников triangles те, центральный угол которых является тупым.

def find_centers(triangles):

Эта функция для каждого треугольника из списка triangles находит его центр как среднее арифметическое координат его вершин и возвращает список центров.

def check_close_centers(triangles, centers):

Эта функция удаляет из списка треугольников triangles все, кроме одного, из каждой группы таких, для которых их центры лежат ближе определенного расстояния.

def find_triangles(lines):

Эта функция среди всех найденных линий находит все треугольники и удаляет дублирующие друг друга треугольники и треугольники с тупым центральным углом.

def dist_between_points(point_1, point_2):

Эта функция вычисляет евклидово расстояние между двумя точками в двумерном пространстве.

def dist_between_triangles(triangle_1, triangle_2):

Эта функция вычисляет минимум евклидова расстояния между вершинами двух треугольников.

def check_same_triangles(triangles):

Эта функция удаляет из списка треугольников все лежащие близко, кроме одного экземпляра из каждой группы таких треугольников.

def check_colour_features(centers, img, triangles):

Эта функция удаляет из списка треугольников те, дисперсия интенсивности красного цвета в некоторой окрестности центра превышает определенное значение.

def build_triangles_on_centers_angles(triangles, centers):

Эта функция для каждого треугольника из списка строит равносторонний треугольник по его центральному углу и его центру и возвращает новые списки треугольников и центров.

def cut_triangles(triangles, centers):

Эта функция для каждого треугольника из списка возвращает координаты вершин трех четырехугольников, на которые он разрезается через центр и середины сторон.

def find_nearest_colour(mean_h, mean_s, mean_v):

Эта функция по заданным средним значениям каналов HSV-представления изображения определяет, к какому из возможных цветов этот цвет ближе всего.

def count_balls(polygons, img):

Эта функция для каждого изображения и координат четырехугольников на нем возвращает количество точек в нём и координату одной из них (если их не ноль).

def save_results(img, triangles, centers, polygons, balls_coords, balls_count, suffix):

Эта функция сохраняет в определенную папку изображение с выделенными на нем треугольниками и указанным цветом точек.

def save_to_file(filename, centers, balls_count):

Эта функция сохраняет в файл формата .txt результат работы программы в том виде, который требуется в задании.

4. Эксперименты

В заключении приведем результаты работы алгоритма на всех четырех известных изображениях класса Expert.

Здесь видно, что только на одном изображении не выделяется один треугольник Тримино. Что касается определения количества точек, то точность составляет примерно 80%.

5. Выводы

6. Литература

Гонзалес Р., Вудс Р. Цифровая обработка изображений. М., Техносфера, 2006.