МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра САУ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Техническое зрение»

ТЕМА: Границы и контуры.

Студент гр. 6492 _____ Огурецкий Д.В.

Преподаватель _____ Моклева К.А.

Санкт-Петербург

2020

Лабораторная работа №5

Цель: изучить способы выделения границ на изображении, поиск контуров на границах и получения информации об объектах на основе контуров.

Ход работы.

Поиск границ:

Выбираем изображение с четкими границами

Оператор Собеля — это дискретный дифференциальный оператор, вычисляющий приближение градиента яркости изображения. Оператор вычисляет градиент яркости изображения в каждой точке.

Так выглядит ядро оператора Собеля для x 3x3. Для y транспонируем.

```
-1 0 1
-2 0 2
-1 0 1
```

Описание параметров оператора Собеля cv2.Sobel :

ddepth – глубина результирующего изображения. Она может быть следующей: CV_8U, CV_16U, CV_16S, CV_32F или CV_64F. чтобы избежать переполнения при расчете целевое изображение должно быть 16-битным (CV_16U) при 8-битном исходном изображении. Для преобразования получившегося изображения в 8-битное можно использовать cv2.convertScaleAbs()

- xorder – порядок производной по оси Ox. (0,1 или 2)
- yorder – порядок производной по оси Oy.

xorder и yorder не могут равнятся 0 одновременно!

$$\texttt{dst} = \frac{\partial^{\texttt{xorder}+\texttt{yorder}} \texttt{src}}{\partial x^{\texttt{xorder}} \partial y^{\texttt{yorder}}}$$

ksize – размер расширенного ядра оператора Собеля. Принимает одно из значений 1, 3, 5 или 7. Если равно 1, то матрица имеет вид 3x1 для y и 1x3 для x [-1 0 1] (она была получена с помощью cv2.getDerivKernels)

scale – опциональный параметр, который задает коэффициент масштабирования для вычисляемых значений производных. По умолчанию масштабирование не применяется.

delta – опциональный параметр смещения интенсивности, добавляется перед сохранением результата в выходную матрицу.

dst=scale*src+delta

borderType – параметр, определяющий метод дополнения границы.

Таблица 1 значения параметра

Значение	Описание
BORDER_CONSTANT	Дополнить одинаковыми пикселями (черная рамка)
BORDER_WRAP	Дополнить пикселями с противоположного края
BORDER_REPLICATE	Копировать граничный пиксель
BORDER_REFLECT	Дополнить отраженными пикселями
BORDER_REFLECT_101	Дополнить отраженными пикселями, не копируя граничный
BORDER_DEFAULT	BORDER_REFLECT_101

Эксперимент со значением этого параметра будет показан далее.

Выберем картинку с четкими границами.

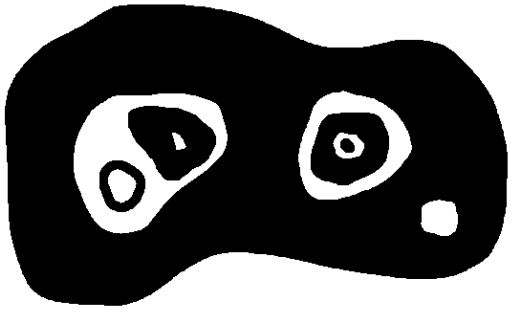


Рисунок 2 — корабль в сером

Рисунок 3 — Оператор Собеля при градиенте по x 1 порядка

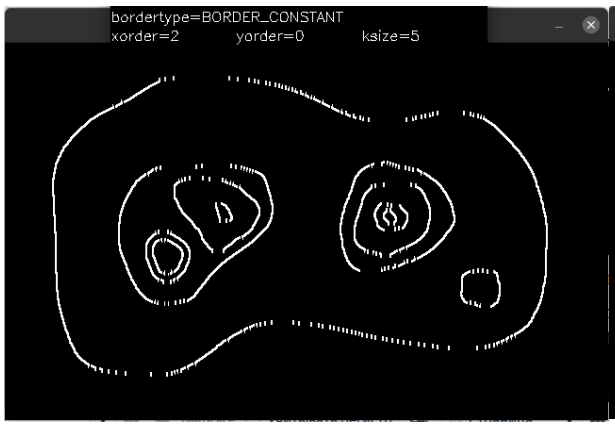


Рисунок 3 — Оператор Собеля при градиенте по x 2 порядка

Рисунок 3 — Оператор Собеля при градиенте по y 1 порядка

Рисунок 3 — Оператор Собеля при градиенте по y 2 порядка

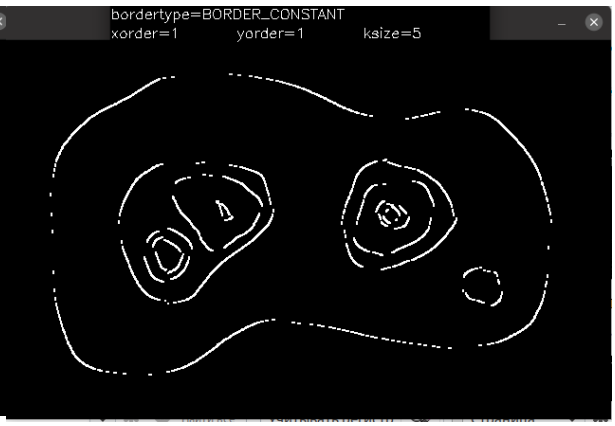


Рисунок 3 — Оператор Собеля при градиенте по x и y 1 порядка

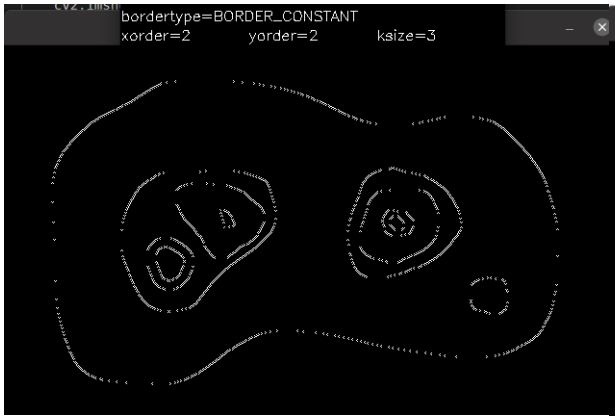


Рисунок 3 — Оператор Собеля при градиенте по x и y 2 порядка

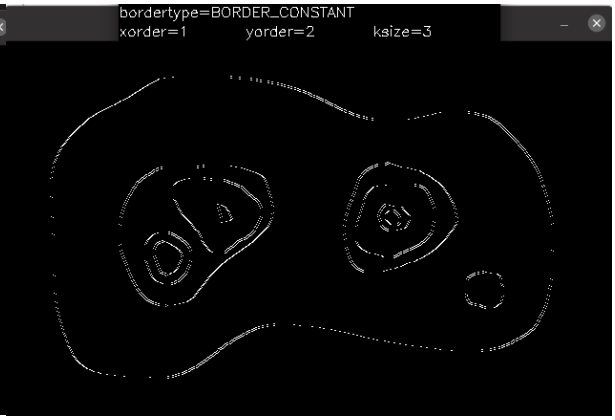


Рисунок 3 — Оператор Собеля при градиенте по x 1 порядка,а по y 2 порядка

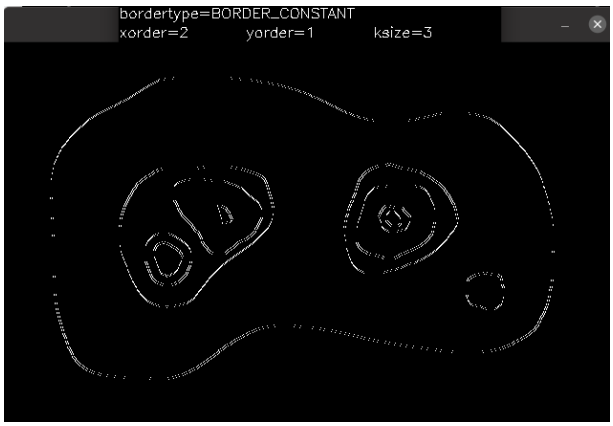


Рисунок 3 — Оператор Собеля при градиенте по x 2 порядка,а по y 1 порядка

По результатам исследования функции Собеля определено:
Параметр bordertype влияет только на рамку картинки.
При увеличении порядка производной появляется дополнительная граница.
Размер ядра влияет на толщину линии, но не влияет на количество обнаруженных границ.
Наибольшей четкости и непрерывности границ получается получить при градиенте только по X или только по У. При использовании градиента и по X и по У границы прерывистые. Наиболее четкие границы при использовании градиента по У 2 порядка.

Оператор Лапласа позволяет вычислить т.н. лапласиан изображения — суммирование производных второго порядка.OpenCV содержит для этого функцию cvLaplace()получает лапласиан изображения фактически, это оператор собеля с xorder = yorder = 2
Фактически имеем такое ядро.

0	1	0
1	-4	1
0	1	0

Рисунок 4 — Оператор Лапласса

Данный метод несмотря на схожесть с оператором Собеля обнаруживает все контуры. Ksize влияет на толщину контура.

Шаги детектора:

— Убрать шум и лишние детали из изображения

— Рассчитать градиент изображения

— Сделать края тонкими (edge thinning)

— Связать края в контура (edge linking)

Детектор использует фильтр на основе первой производной от гауссианы. Так как он восприимчив к шумам, лучше не применять данный метод на необработанных изображения. Сначала, исходные изображения нужно свернуть с гауссовым фильтром.

edges = cv2.Canny(image=img, threshold1=t1, threshold2=t2,

apertureSize=3, L2gradient=False)

image — одноканальное изображение для обработки (градации серого)
threshold1 — порог минимума
threshold2 — порог максимума
Если градиент пикселя выше верхнего порога, то пиксель принимается в качестве ребра
Если значение градиента пикселя находится ниже нижнего порога, то оно отклоняется.
Если градиент пикселя находится между двумя порогами, то он будет принят только в том случае, если он подключен к пикселю, который находится выше верхнего порога.
apertureSize – размер ядра для оператора Собеля

L2gradient=True – вычислять градиент точно

L2gradient=False – вычислять быстрее, с использованием нормы L1

Сначала применяется Фильтр Гаусса:

cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])

ksize — размер Гауссова ядра. ksize.width и ksize.height могут отличаться, но они оба должны быть положительными и нечетным.
sigmaX — стандартное отклонение Гауссова ядра в направлении X.

sigmaY — стандартное отклонение Гауссова ядра в Y направлении; если sigmaY равен нулю, то устанавливается равным sigmaX, если оба сигмы нули, они вычисляются из ksize.width и ksize.height, соответственно; для тогоб чтобы полностью контролировать результат, независимо от возможных будущих модификаций, рекомендуется указать все ksize, sigmaX и sigmaY.

ksize = (3, 3), sigmaX = 0, sigmaY = 0

borderType — пиксельный метод экстраполяции.

Рисунок 5 — После размытия Гаусса

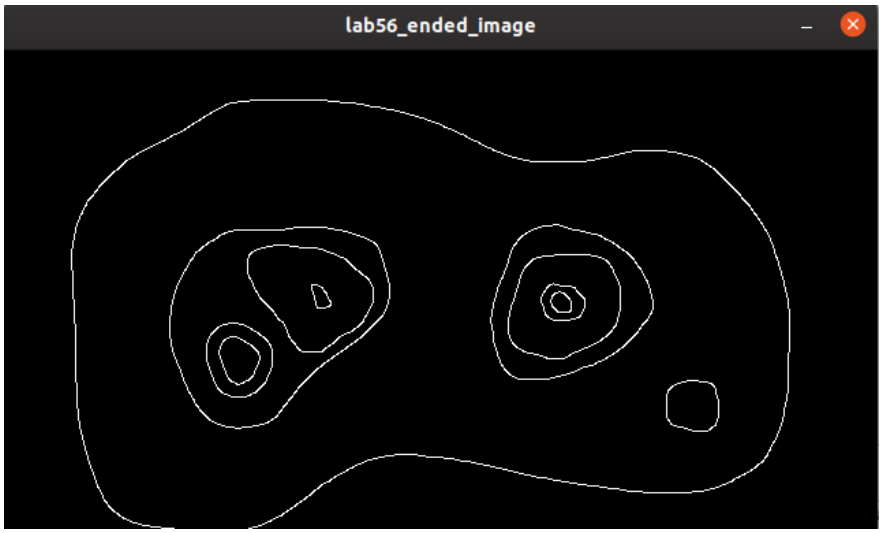


Рисунок 5 — Детектор Кенни , размеря дра равно 3

Рисунок 5 — Детектор Кенни , размеря дра равно 7

threshold1,threshold2, L2gradient не влияют на обнаружение границ
apertureSize – влияет тлько при установке размера ядра равного 7.

Изображение с нечеткими границами

Рисунок 6 — Исходное изображение

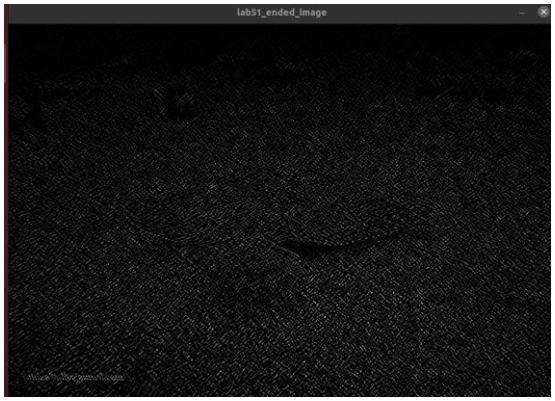


Рисунок 3 — Оператор Собеля

Рисунок 4 — Оператор Лапласса

Рисунок 5 — После размытия Гаусса

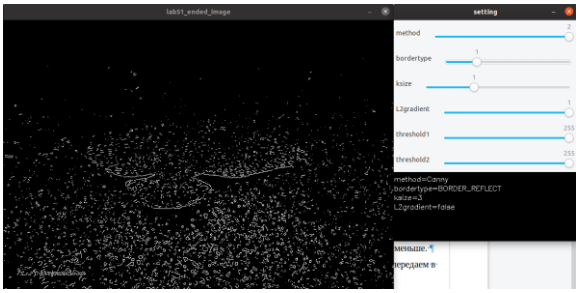


Рисунок 10 — Детектор Кенни

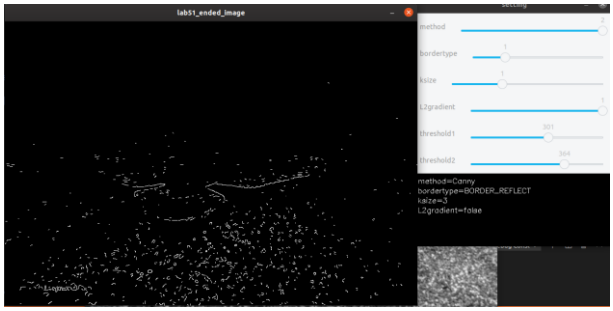


Рисунок 10 — Детектор Кенни

L2gradient практически не влияет на обнаружение границ. Однако при упрощенном расчете градиента «лишних» границ становится меньше. Границы данной рыбы определяются только по нижней части рыбы. Верхняя не определяется. Но по нижней части можно уже определить что за рыба и далее дорисовать границу рыбы, зная общий вид рыбы.

Вывод по границам:

У детектора Кенни по-другому чем больше нижний и верхний порог, тем меньше будет границ, так как больше пикселей будет отсеиваться. Тоже самое и с верхним порогом. При высоком пороговом значении границы будут видны только при большой разницы в яркости между пикселями.
Ksize = 3 - это размер ядра Собеля (по умолчанию). Чем он больш, тем больше границ детектируется.
При использовании детектора Кенни параметр L2gradient задан как True, чтобы градиент яркости вычислялся точнее.

Поиск контуров

findContours(кадр, режим_группировки, метод_упаковки)

Проверим функцию поиска границ findContours на бинарном изображении, полученном с помощью threshold, с помощью детектора границ Кенни.

кадр — должным образом подготовленная для анализа картинка. Это должно быть 8-битное изображение. Поиск контуров использует для работы монохромное изображение, так что все пиксели картинки с ненулевым цветом будут интерпретироваться как 1, а все нулевые останутся нулями. На уроке про поиск цветных объектов была точно такая же ситуация.

режим_группировки — один из четырех режимов группировки найденных контуров:

❒RETR_LIST — выдаёт все контуры без группировки;

❒RETR_EXTERNAL — выдаёт только крайние внешние контуры. Например, если в кадре будет пончик, то функция вернет его внешнюю границу без дырки.

❒RETR_CCOMP — группирует контуры в двухуровневую иерархию. На верхнем уровне — внешние контуры объекта. На втором уровне — контуры отверстий, если таковые имеются. Все остальные контуры попадают на верхний уровень.

❒RETR_TREE — группирует контуры в многоуровневую иерархию.

метод_упаковки — один из трёх методов упаковки контуров:

❒CHAIN_APPROX_NONE — упаковка отсутствует и все контуры хранятся в виде отрезков, состоящих из двух пикселей.

хранит абсолютно все точки контура. То есть любые 2 последующие точки (x1,y1) и (x2,y2) контура будут либо горизонтальными, либо вертикальными, либо диагональными соседями

❒CHAIN_APPROX_SIMPLE — склеивает все горизонтальные, вертикальные и диагональные контуры.

сжимает горизонтальные, вертикальные и диагональные сегменты и оставляет только их конечные точки. Например, прямоугольный контур справа вверху кодируется 4 точками.

❒CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS — применяет к контурам метод упаковки (аппроксимации) Teh-Chin.

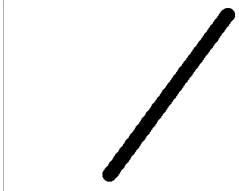


Рис 1 исходное изображение

Рис 2 поиск контуров на бинарном изображении, полученном с помощью функции threshold()

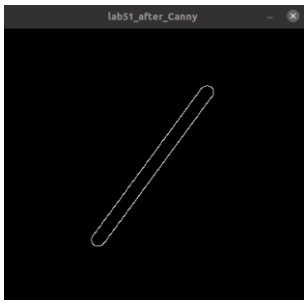


Рис 3 поиск контуров на бинарном изображении, полученном детектором границ Кенни

Количество контуров на бинарном изображении, полученном с помощью функции threshold().

Вывод из консоли:

contours count=2 with threshold;
with mode in findContours=RETR_LIST and method_type in findContours=CHAIN_APPROX_NONE.

Количество контуров на бинарном изображении, полученном детектором границ Кенни:

```
method=Gauss>>canny
bordertype=BORDER_REFLECT
ksize=3
L2gradient=true
method_type in findContours=CHAIN_APPROX_NONE
mode in findContours=RETR_LIST
```

Вывод из консоли:

contours count=2 with Gauss>>canny;

with mode in findContours=RETR_LIST and method_type in findContours=CHAIN_APPROX_NONE.

Собственно говоря на такой простой картинке контуров всего 2.

Исследуем более сложную картинку, например ту же рыбу.

Рис 1 исходное изображение

Рис 2 поиск контуров на бинарном изображении, полученном с помощью функции threshold()

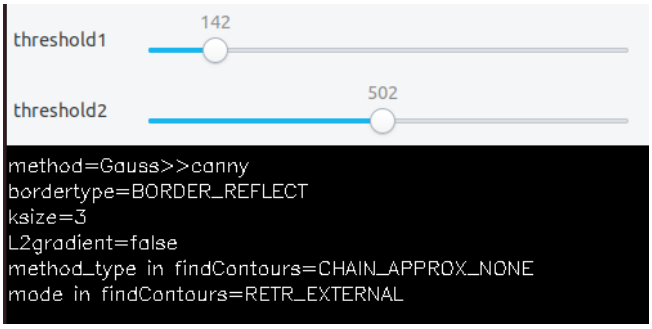
Рис 3 поиск контуров на бинарном изображении, полученном детектором границ Кенни

Количество контуров на бинарном изображении, полученном с помощью функции threshold().

Вывод из консоли:

contours count=3078 with threshold;
with mode in findContours=RETR_LIST and method_type in findContours=CHAIN_APPROX_NONE.

Количество контуров на бинарном изображении, полученном детектором границ Кенни:



Вывод из консоли:

contours count=77 with Gauss>>canny;

with mode in findContours=RETR_LIST and method_type in findContours=CHAIN_APPROX_NONE.

Исследуем влияние метода упаковки на количество точек контуров для данного изображения

Рис 2 бинарное изображение, полученном детектором границ Кенни

Вывод из консоли:

contours count=90

points count of countur[0]:25

with Gauss>>canny;

with mode in findContours=RETR_LIST and method_type in findContours=CHAIN_APPROX_NONE.

contours count=90

points count of countur[0]:15

with Gauss>>canny;

with mode in findContours=RETR_LIST and method_type in findContours=CHAIN_APPROX_SIMPLE.
contours count=90
points count of countur[0]:7
with Gauss>>canny;
with mode in findContours=RETR_LIST and method_type in findContours=CHAIN_APPROX_TC89_L1.
contours count=90
points count of countur[0]:8
with Gauss>>canny;
with mode in findContours=RETR_LIST and method_type in findContours=CHAIN_APPROX_TC89_KCOS.

Видно, что больше всего точек при методе CHAIN_APPROX_NONE чего и следовало ожидать, в остальных методах количество точек меньше. Из исследования можно сделать вывод, что метод упаковки не влияет на количество контуров, он влияет только на количество точек.

Вывод: Контуров у изображения, полученного с помощью детектора Кенни меньше, чем у изображения threshold, так как детектор Кенни выделил меньше границ благодаря гибкой настройке порогов. При простых изображениях как отрезок, количество контуров получается одинаковым при обоих способах выделения границ.

Окружность:

Найдем для окружности внутренний и внешний контур. Вычислим их длину и площадь.

Нужно использовать `cv2.drawContours`

drawContours(кадр, контуры, индекс, цвет[, толщина[, тип_линии)

кадр — кадр, поверх которого мы будем отрисовывать

контуры; контуры — те самые контуры, найденные функцией findContours; индекс — индекс контура, который следует отобразить. -1 — если нужно отобразить все контуры; цвет — цвет контура; толщина — толщина линии контура; тип_линии — тип соединения точек вектора:LINE_4 4-connected line

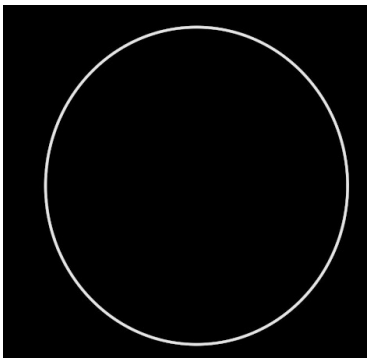


Рисунок 13 — Исходное изображение окружности

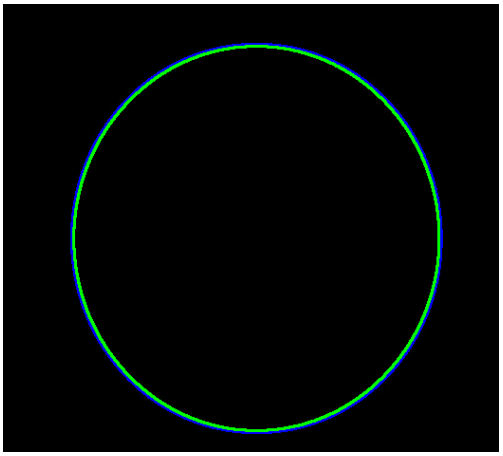


Рисунок 14 — Окружность с выделенным внешним и внутренним контуром

Результаты:

Contours Rings = 2

P inside = 1309.9524418115616

S inside = 122448.5

P out = 1325.0235097408295

S out = 125414.0

Длина и площадь внешнего контура больше, так как внешний контур больше внутреннего и , соответственно, у него больше количество писелей.

Находим ограничивающие прямоугольник и окружность для внешнего и внутреннего контура.

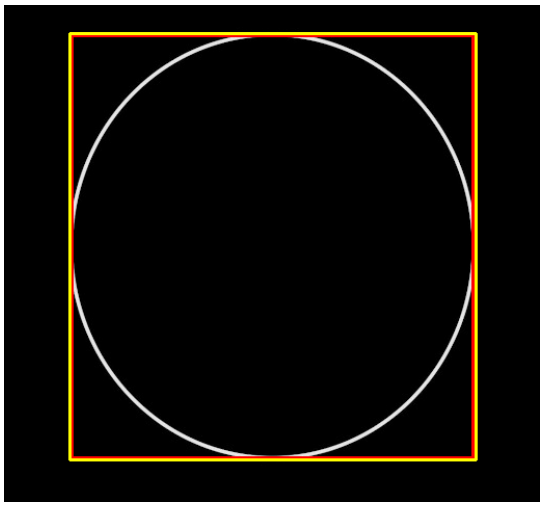


Рисунок 15 — Ограничивающие прямоугольники

S inside rectangle = 156716

S outside rectangle = 160310

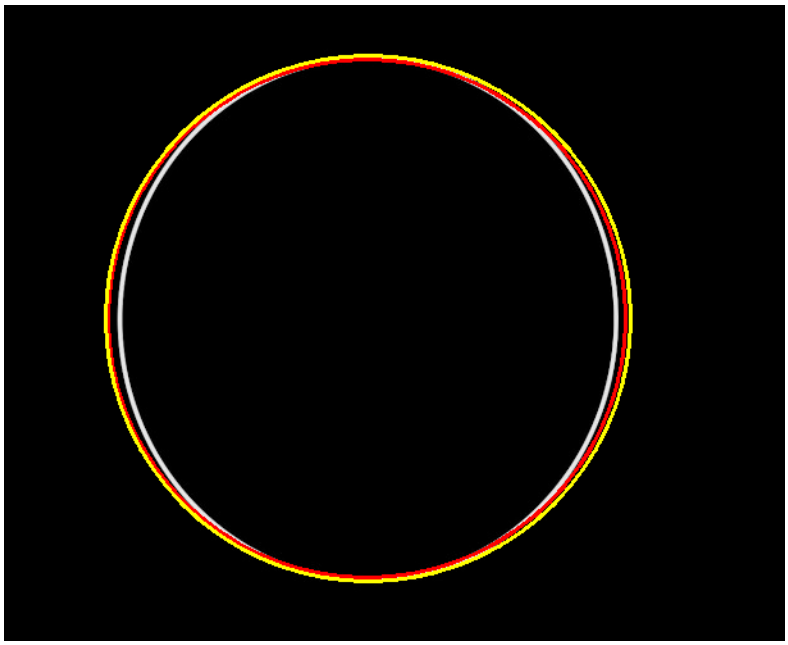


Рисунок 16 — Ограничивающие окружности

S inside circle = 128124.56000000001

S outside circle = 131958.5

Площади ограничивающих фигур больше площадей контуров, так как они больше в размере, чем контура.

Вывод: в лабораторной работе произведено обнаружение и отрисовка контуров на рисунках. С помощью методов Laplasse, Canny, Sobel можно выделить границы, при этом перед использованием Canny нужно произвести размытие методом Gauss. После этого применяется метод fincContours, который обнаруживает контура на рисунке. Далее можно нарисовать эти контура, используя методы drawcontours. Также можно считать длину контуров и площадь внутри замкнутых контуров.

Также в лабораторной работе исследовано дерево, которое составляется при вызове метода findContours.

Дополнительное задание №1

все контуры на нечетном уровне вложенности нарисуйте красным цветом, все контуры на четном - синим цветом.

После непродолжительных исследований было выяснено, что действительное количество контуров получается обнаруживать, используя метод фильтрации для выделения границ : threshold: THRESH_BINARY_INV

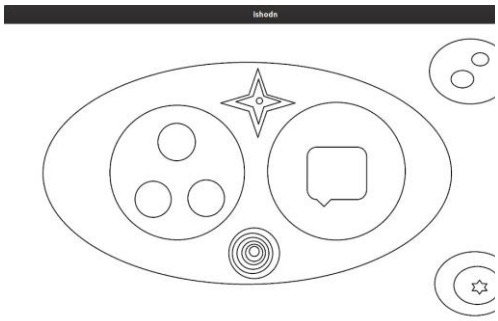


Рис 1 исходное изображение

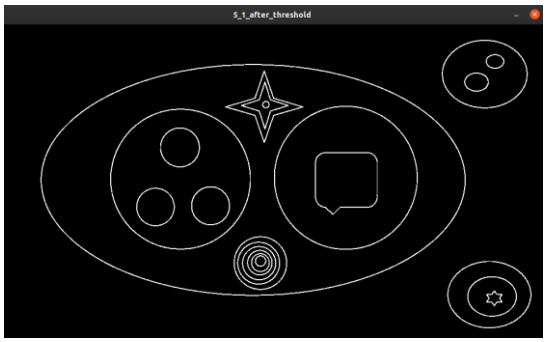
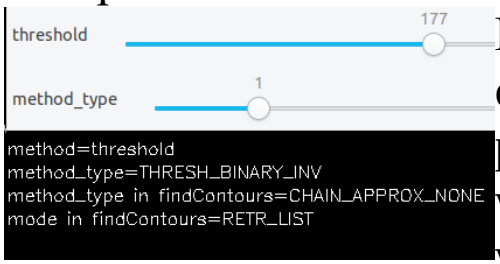


Рис 2 после обработки threshold



Параметры threshold

Результат количества контуров
contours count=44
points count of countur[0]:73
with threshold;
with mode in findContours=RETR_LIST and method_type in findContours=CHAIN_APPROX_NONE.

Количество контуров 44, но на самом деле их 22, так как поиск производится по внешним и внутренним.

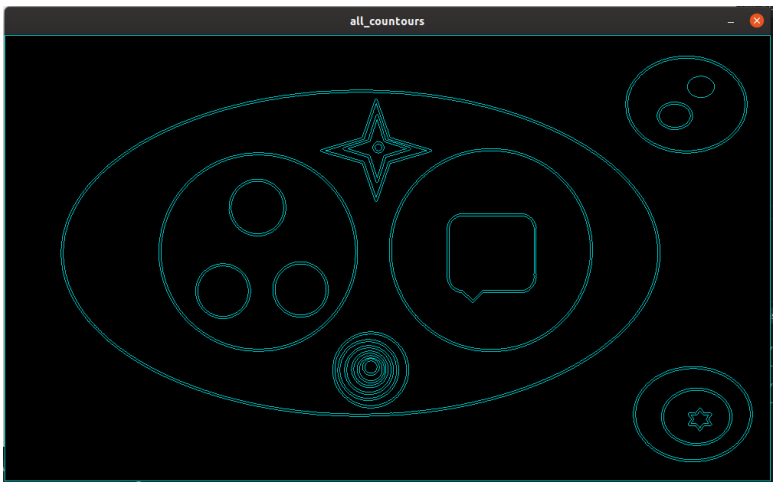
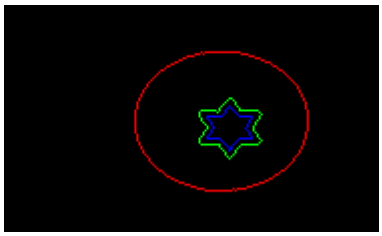


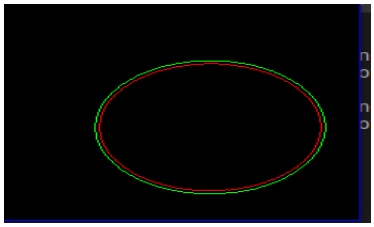
Рис 3 Все контура

Здесь необходимо дать определение списку hierarchy: иерархия — информация о топологии контуров. Каждый элемент иерархии представляет собой сборку из четырех индексов, которая соответствует контуру[i]:

- ❒иерархия[i][0] — индекс следующего контура на текущем слое;
- ❒иерархия[i][1] — индекс предыдущего контура на текущем слое:
- ❒иерархия[i][2] — индекс первого контура на вложенном слое;
- ❒иерархия[i][3] — индекс родительского контура.

Перед выполнением необходимо проверить последовательность следования контуров . Отобразим контура, последовательность определяется цветом. Индексы контуров такие: Синий — 0, красный — 1, зеленый -2.





mode in findContours=RETR_LIST mode in findContours=RETR_TREE

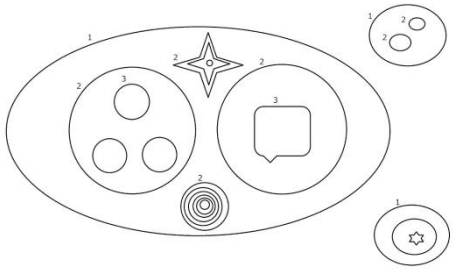
Так выглядит иерархия контуров. Это массив hierarchy при

```
[[[-1 -1  1 -1]
  [ 7 -1  2  0]
  [-1 -1  3  1]
  [-1 -1  4  2]
  [-1 -1  5  3]
  [-1 -1  6  4]
  [-1 -1 -1  5]
  [39  1  8  0]
  [-1 -1  9  7]
  [21 -1 10  8]
  [-1 -1 11  9]
  [-1 -1 12 10]]
```

по данному массиву можно сказать, что значение -1 в 4 столбце значит, что родительский контур отсутствует и узел является корнем
значение -1 в 3 столбце означает , что узел является листом
значение -1 в 2 столбце означает, что предыдущий узел
на данном уровне отсутствует,то есть данный узел является самым левым
значение -1 в 2 столбце означает, что предыдущий узел на данном уровне н
а данном уровне отсутствует,то есть данный узел является самым правым
если и в первом и втором столбце стоит -1, то узел является одним на данном уровне

```
[-1 -1 13 11]
[-1 -1 14 12]
[-1 -1 15 13]
[-1 -1 16 14]
[-1 -1 17 15]
[-1 -1 18 16]
[-1 -1 19 17]
[-1 -1 20 18]
[-1 -1 -1 19]
[29  9 22  8]
[-1 -1 23 21]
[25 -1 24 22]
[-1 -1 -1 23]
[27 23 26 22]
[-1 -1 -1 25]
[-1 25 28 22]
[-1 -1 -1 27]
[33 21 30  8]
[-1 -1 31 29]
[-1 -1 32 30]
[-1 -1 -1 31]
[-1 29 34  8]
[-1 -1 35 33]
[-1 -1 36 34]
[-1 -1 37 35]
[-1 -1 38 36]
[-1 -1 -1 37]
[-1  7 40  0]
[-1 -1 41 39]
[43 -1 42 40]
[-1 -1 -1 41]
[-1 41 44 40]
[-1 -1 -1 43]]]
```

В соответствии с требуемой нумерацией уровней вложенности программа должна выглядеть так. Стоит отметить здесь, что внешний контур рамки рисунка отсутствует.



Для решения данной задачи используется рекурсия, то есть вызов функции самой себя. Здесь используется обход дерева в прямом порядке, как наиболее простой в реализации и понимании.

Остается вопрос, какие контура окрашивать: внешние или внутренние. Для того, чтобы поменять необходимо изменить в строчке 28 кода node на dc.

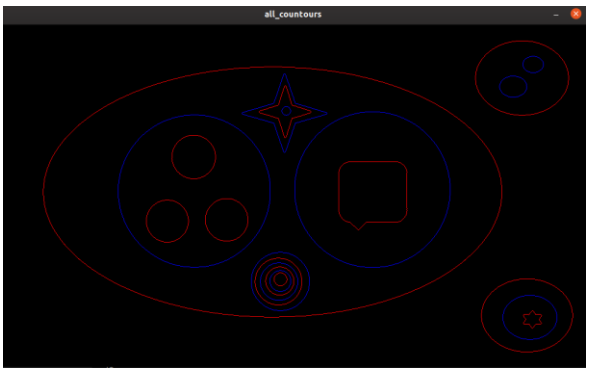


рис 1 С окрашиванием внешних контуров

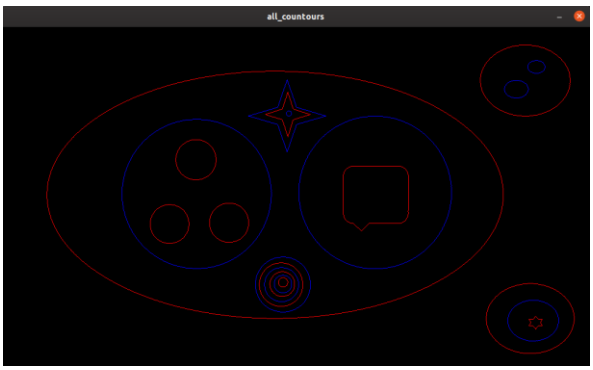


рис 2 с окрашиванием внутренних контуров

Код

```
import cv2
import numpy as np
# читаем исходное изображение с четкими границами
# поиск границ оператором Собеля
name_file="5_1"
img1 = cv2.imread(name_file+'.jpg', cv2.IMREAD_GRAYSCALE)
shape = img1.shape
title_window = name_file + '_ended_image'
cv2.imshow("ishodn",img1)

threshold_rings, img1 = cv2.threshold(img1, 177, 255, cv2.THRESH_BINARY)
img,contours, hierarchy = cv2.findContours(img1, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
img = np.empty((shape[0], shape[1], 3 ),np.uint8)

EMPTY = -1
def draw_contour(node,color):
    if color:
        cv2.drawContours(img, contours, node, (0, 0, 255), 1)
    else:
        cv2.drawContours(img, contours, node, (255, 0, 0), 1)

#color индикатор цвета 0-синий 1-красный
#node индекс следующего контура
def get_node(node,color):
    dc,uc = hierarchy[0][node][2],hierarchy[0][node][3]
    if uc!=EMPTY: #проверяем, что узел не является корнем,
    #так как корень окрашивать не нужно, это рамка рисунка
        draw_contour(node,color) #чтобы окрасить внутренний поменять node на dc
        #нужно перейти к следующиему контуру, как будто мы его окрасили
        #следующий контур будет сыном к текущему всегда
        dc=hierarchy[0][dc][2]
    if dc != EMPTY:
        #у узла есть дети
        child_node = dc
        rc = hierarchy[0][child_node][0]
        while 1: #пока не пройдемся по всем сыновьям узла node
            get_node(child_node,not color)#вызываем, но окрашиваем в противоположный цвет
            if rc!=EMPTY: # есть ли следующий сын?
```

```
                #переход к следующему сыну узла первоначального node
                child_node = rc
                rc = hierarchy[0][child_node][0]
            else:
                break
#окраска по нашим правилам
#начинаем с первого контура,но так как первый это рамка рисунка
#то окрашивать его не надо
#но ставим, как будто мы окрашиваем его в синий
get_node(0,0)
cv2.imshow('all_countours',img)
while 1:
    if cv2.waitKey(100)==27: #esc
        break
```

Дополнительное задание №2

Для этого изображения нарисуйте все треугольники красным цветом, квадраты -синим, круги - зеленым. Используйте информацию о контурах и дополнительные операции над контурами.

ContourArea используется для вычисления площади замкнутого контура , поэтому второй аргумент равен true

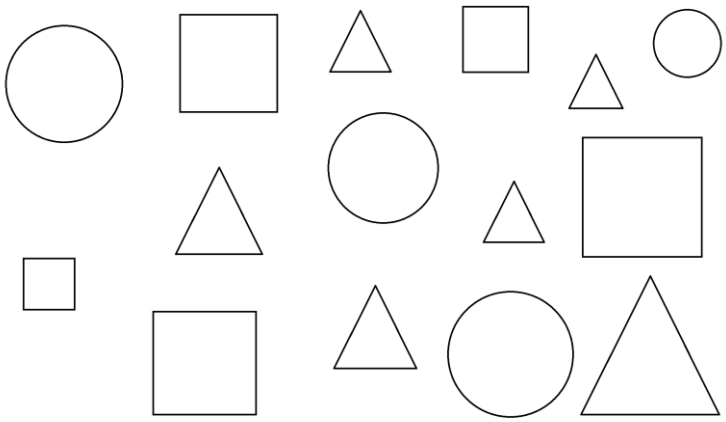


рис 1 исходная картинка

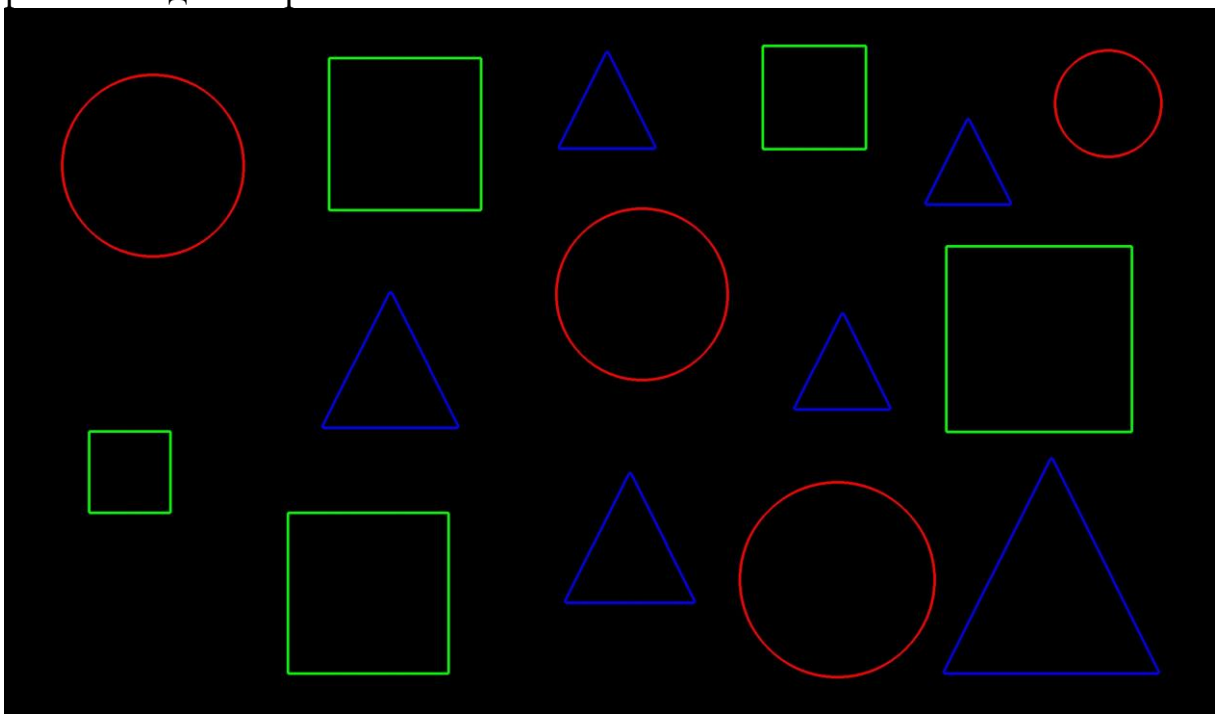


рис 2 Подсвеченные фигуры

Алгоритм использует сравнение площадей, фигур.

Код

```
1  import cv2
 2 import numpy as np
 3 import math as m
 4 name_file="5_2"
 5 img1 = cv2.imread(name_file+'.png', cv2.IMREAD_GRAYSCALE)
```

```
shape = img1.shape
threshold_rings, img1 = cv2.threshold(img1, 122, 255, cv2.THRESH_BINARY)
img,contours, hierarchy = cv2.findContours(img1, cv2.RETR_TREE, cv2.CHAIN_APPROX_S
img = np.empty((shape[0], shape[1], 3 ),np.uint8)
for con_ind in range(1,len(contours),2):
    [a,r] = cv2.minEnclosingCircle(contours[con_ind])
    S=m.pi*r*r
    [vv,ww,v,w]=cv2.boundingRect(contours[con_ind])
    P_kv = v*4
    MIST=800
    if cv2.contourArea(contours[con_ind],True)-S<MIST and cv2.contourArea(contours
        #рисуем окружность
        cv2.drawContours(img,contours,contourIdx=con_ind,color=(0,0,255),thickness
    elif cv2.arcLength(contours[con_ind],True) - P_kv <50 and cv2.arcLength(contou
        #рисуем квадрат
        cv2.drawContours(img, contours ,contourIdx=con_ind, color=(0 , 255, 0), th
    else:
        #рисуем треугольник
        cv2.drawContours(img, contours ,contourIdx=con_ind, color=(255 , 0, 0), th
cv2.imwrite("img.jpg",img)
```

Приложение 1 Код программы для первого пункта

```
import cv2
name_file="lab5_circle"
img1 = cv2.imread(name_file+'.jpg', cv2.IMREAD_GRAYSCALE)
cv2.imshow("ishodn",img1)

#1-ая часть задания
end_img = cv2.Sobel(img1, cv2.CV_16U , 1, 1 , 0, ksize = 3, scale = 1, delta = 0, borderType=1 )
cv2.imshow(name_file+'_after_Sobel' ,cv2.convertScaleAbs(end_img) )
end_img = cv2.Laplacian(img1, cv2.CV_16U , 0, ksize = 3, scale = 1, delta = 0, borderType=1)
cv2.imshow(name_file+'_after_Laplacian' ,cv2.convertScaleAbs(end_img) )
end_img = cv2.GaussianBlur(img1, ksize = (3, 3), sigmaX = 0, sigmaY = 0, borderType=1)
cv2.imshow(name_file+'_after_Gaus' ,end_img)
end_img = cv2.Canny(end_img, threshold1 = 100, threshold2 = 255, apertureSize = 3, L2gradient = 1)
cv2.imshow(name_file+'_after_Canny' ,end_img)

#2-ая часть задания
#после обработки детектором Канни
img,contours, hierarchy = cv2.findContours(cv2.convertScaleAbs(end_img), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
#вывод количества контуров найденных
print("contours count="+str(len(contours)))
print("points count of counturs[0]:"+str(len(contours[0])))
#после обработки тресхолдом
end_img = cv2.threshold(img1, 127 ,255,cv2.THRESH_BINARY)
cv2.imshow(name_file+'_after_threshold' ,cv2.convertScaleAbs(end_img[1]) )
img,contours, hierarchy = cv2.findContours(cv2.convertScaleAbs(end_img[1]),
cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
#вывод количества контуров найденных
print("contours count="+str(len(contours)))
print("points count of counturs[0]:"+str(len(contours[0])))

#3-я часть
threshold_rings, rings = cv2.threshold(img1, 170, 255, cv2.THRESH_BINARY)
img,contours, hierarchy = cv2.findContours(rings, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
outside = contours[0]
inside = contours[1]
print('Contours Rings =', len(contours))
# рисуем внутреннюю(зеленая) и внешнюю(синяя) окружность
imgc = cv2.imread(name_file+'.jpg')
rings = cv2.drawContours(imgc, contours, 0, (255, 0, 0), 2)
rings = cv2.drawContours(rings, contours, 1, (0, 255, 0), 2)
cv2.imshow('rings2', rings)
# Выводим значение длины окружности и ее площадь
print('P inside = ', cv2.arcLength(inside, True))
print('S inside = ', cv2.contourArea(inside))
print('P out = ', cv2.arcLength(outside, True))
print('S out = ', cv2.contourArea(outside))

x1, y1, w1, h1 = cv2.boundingRect(inside)
x2, y2, w2, h2 = cv2.boundingRect(outside)
# Рисуем ограничивающий внешний и внутренний прямоугольник
imgc = cv2.imread(name_file+'.jpg')
cv2.rectangle(imgc, (x1, y1), (x1+w1, y1+h1), (0, 0, 255), 2)
cv2.rectangle(imgc, (x2, y2), (x2+w2, y2+h2), (0, 255, 255), 2)
```

```
53  cv2.imshow('rectangle', imgc)
54  # Рассчитываем площадь ограничивающих прямоугольников
55  S_in_rectangle = w1*h1
56  S_out_rectangle = w2*h2
57  print('S inside rectangle = ', S_in_rectangle)
58  print('S outside rectangle = ', S_out_rectangle)
59
60  # ------------- ограничивающая окружность ----------------
61  # Находим координаты центра и радиус ограничивающих окружностей
62  (x1, y1), r1 = cv2.minEnclosingCircle(inside)
63  (x2, y2), r2 = cv2.minEnclosingCircle(outside)
64  # Преобразуем полученные значения в int
65  center1 = (int(x1), int(y1))
66  radius1 = int(r1)
67  center2 = (int(x2), int(y2))
68  radius2 = int(r2)
69  # Рисуем внешнюю(красный) и внутреннюю(желтый) ограничивающие окружности
70  rings = cv2.imread(name_file+'.jpg')
71  cv2.circle(rings, center1, radius1, (0, 0, 255), 2)
72  cv2.circle(rings, center2, radius2, (0, 255, 255), 2)
73  cv2.imshow('circle', rings)
74  cv2.waitKey(0)
75  # Вычисляем площадь ограничивающих окружностей
76  S_in_circle = 3.14*(radius1**2)
77  S_out_circle = 3.14*(radius2**2)
78  print('S inside circle = ', S_in_circle)
79  print('S outside circle = ', S_out_circle)
80
81  while 1:
82      if cv2.waitKey(100)==27: #esc
83          break
```