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. 6491 _____ Бузи Дарья

Преподаватель _____ Моклева К.А.

Санкт-Петербург

2020

Лабораторная работа 5

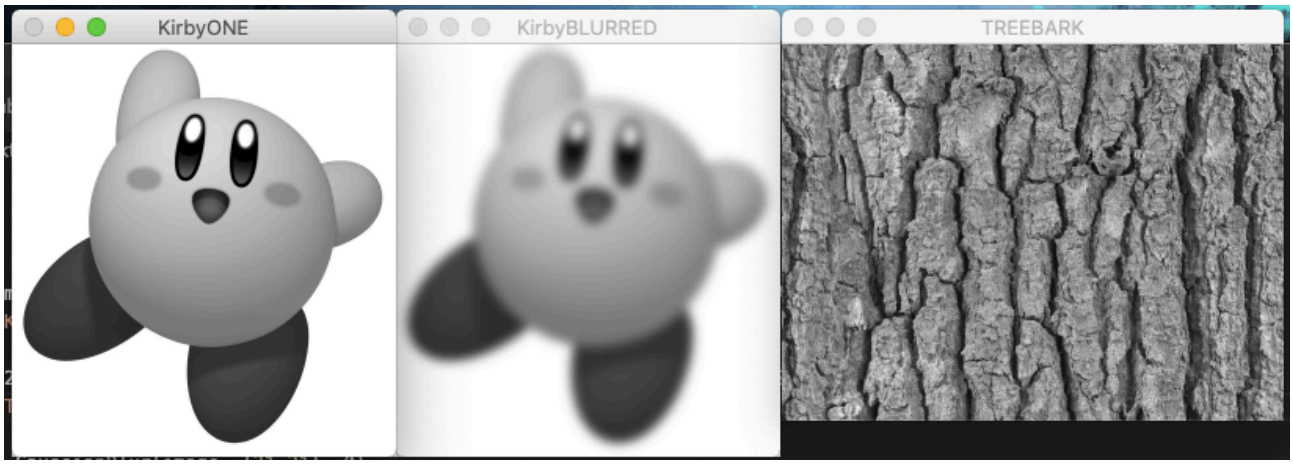
Границы и контуры

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

Ход работы:

Задание 1. Исследуйте все известные вам способы поиска границ на изображении. Для этого выберите несколько изображений, содержащих как четко отделимые от фона границы, так и нечеткие границы, почти сливающиеся с фоном. При применении методов аргументируйте выбор значений, передаваемых в качестве параметров методов.

Для задания выбираем такие картинки :



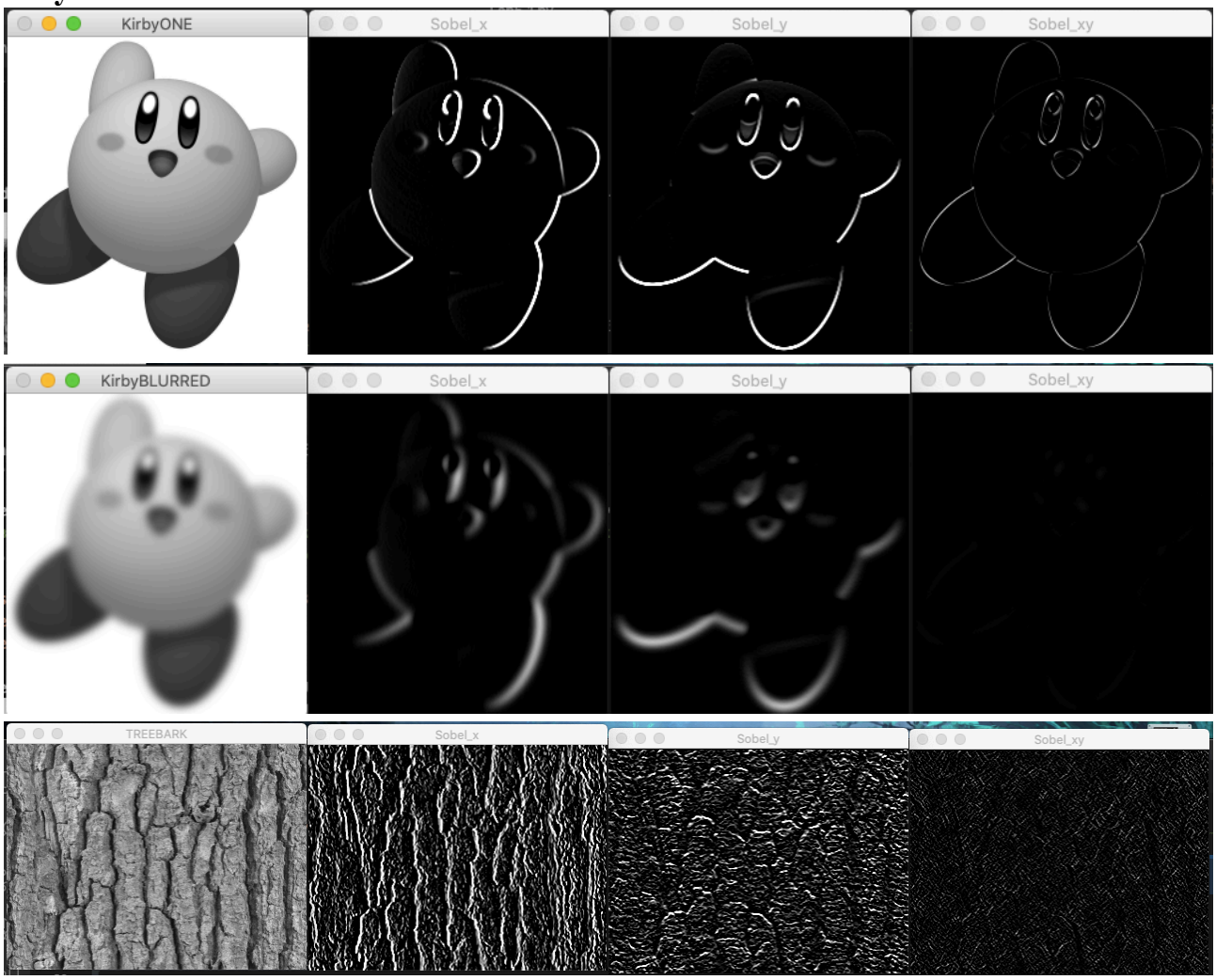
1) **Оператор Собеля**

Код (пример одной картинки):

```
1.  import cv2
2.  import numpy
3.
4.  treebark = cv2.imread('/Users/dariabusi/Desktop/
    tree.jpg',cv2.IMREAD_REDUCED_GRAYSCALE_4) #считываем с директивы
    изображение, возвращается массив с данными изображения в серых тонах,
    размер изображения уменьшен в 4 раза
5.  cv2.imshow ('TREEBARK', treebark)#вывод изображения на экран
6.
7.  """Оператор Собеля: дискретный дифференциальный оператор, вычисляющий
    приближение градиента яркости изображения.
8.  Градиент яркости вычисляется в каждом пикселе, участки с большим значением
    градиента будут белыми. Формат функции cv2.Sobel():
9.  grad_x = cv2.Sobel(
10. img, # исходное изображение
11. ddepth, # глубина полученного изображения, в этом случае cv2.CV_8U - 8bit
    unsigned numpy array
12. xorder, # порядок производной (x)
13. yorder, # порядок производной (y)
```

```
ksize=3, # размер ядра (равен 3 по умолчанию)
)

"""

sobel_x = cv2.Sobel(treebark, cv2.CV_8U, 1, 0, 3)
sobel_y = cv2.Sobel(treebark, cv2.CV_8U, 0, 1)
sobel_xy = cv2.Sobel(treebark, cv2.CV_8U, 1, 1)

cv2.imshow('Sobel_x', sobel_x)#вывод результатов на экран: производные яркости изображения по горизонтали
cv2.imshow('Sobel_y', sobel_y)#производные яркости изображения по вертикали
cv2.imshow('Sobel_xy', sobel_xy)#производные яркости изображения по горизонтали и вертикали

cv2.waitKey(0)
```

Результаты:



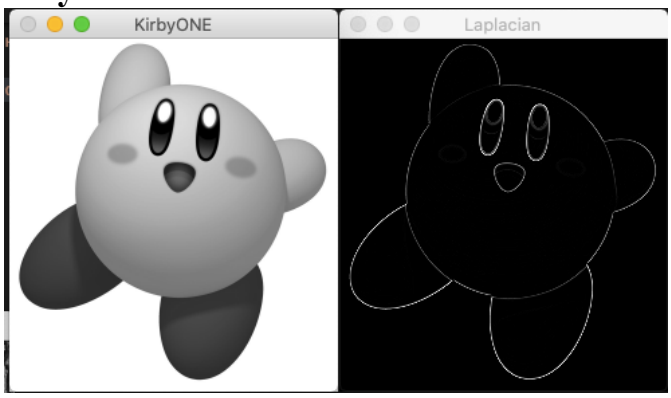
Рассуждение: оператор Собеля лучше использовать для картинок с четкими границами, но границы нечеткие.

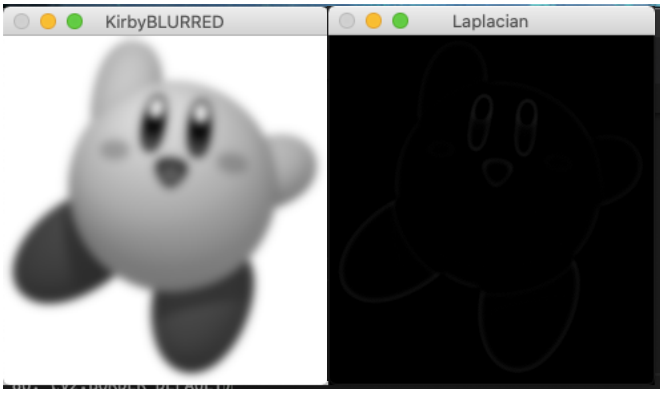
2) Оператор Лапласа

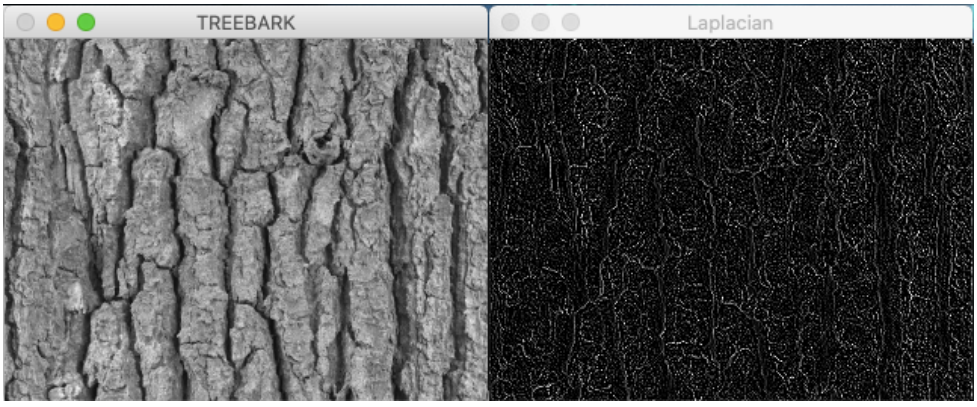
Код (пример одной картинки):

```
1. import cv2
2. import numpy
3.
4. treebark = cv2.imread('/Users/dariabusi/Desktop/
   tree.jpg',cv2.IMREAD_REDUCED_GRAYSCALE_4) #считываем с
   директивы изображение, возвращается массив с данными
   изображения в серых тонах, размер изображения уменьшен в 4
   раза
5. cv2.imshow ('TREEBARK', treebark)#вывод изображения на экран
6.
7. """ Оператор Лапласа: позволяет суммировать производные
   второго порядка. Формат функции:
8. dst = cv.Laplacian(
9. src_gray, # исходное изображение
10. ddepth, # глубина x2
11. ksize=3, # размер ядра
12. scale=1, # масштабирующий коэфф.
13. delta=0, # смещение
14. bType # тип границы"""
15.
16. laplas = cv2.Laplacian(treebark, cv2.CV_8U,
   cv2.BORDER_DEFAULT)
17. cv2.imshow('Laplacian', laplas)
18.
19. cv2.waitKey(0)
```

Результаты:







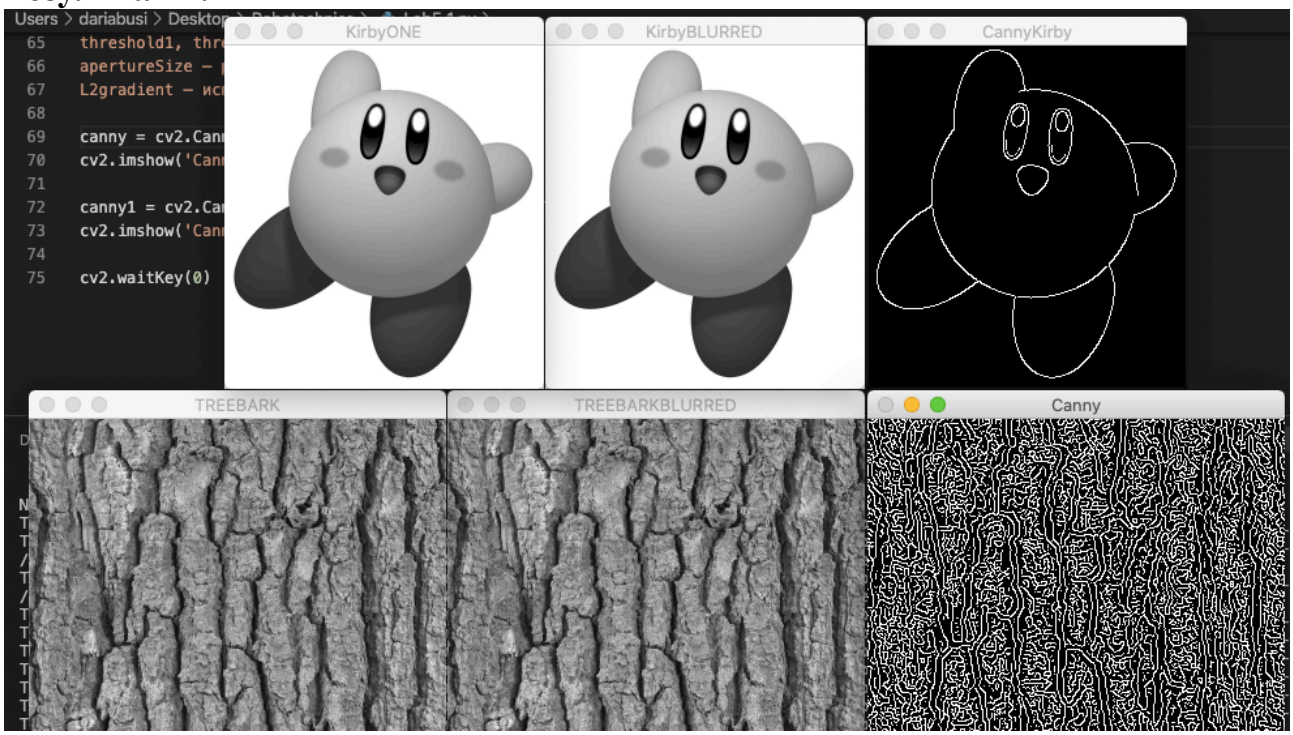
Рассуждение: оператор Лапласа работает лучше с четкими границами

3) Детектор границ Кенни
Код (пример одной картинки):

```
import cv2
import numpy


image = cv2.imread('/Users/dariabusi/Desktop/kirby.jpg',cv2.IMREAD_REDUCED_GRAYSCALE_4)
cv2.imshow ('KirbyONE', image)

treebark = cv2.imread('/Users/dariabusi/Desktop/tree.jpg',cv2.IMREAD_REDUCED_GRAYSCALE_4) #считываем с директивы изображение,
#возвращается массив с данными изображения в серых тонах, размер изображения уменьшен в 4 раза
cv2.imshow ('TREEBARK', treebark)#вывод изображения на экран
""" Детектор границ Кенни: метод выделения границ изображения. Первый шаг: убрать шум и лишние детали из изображения,
это делается размытием, где (x,y) - отклонение от ядра  ksize = 3 по осям x и y"""

blurred = cv2.GaussianBlur(image, (1,1), 0)
cv2.imshow ('KirbyBLURRED', blurred)

treeblur = cv2.GaussianBlur(treebark, (1,1), cv2.BORDER_DEFAULT)
cv2.imshow ('TREEBARKBLURRED', treeblur)

"""Формат функции cv2.Canny:
edges = cv2.Canny(image=img, threshold1=t1, threshold2=t2, apertureSize=3, L2gradient=False)
image – исходное изображение
threshold1, threshold2 – нижний и верхний порог
apertureSize – размер ядра Собеля (равен 3)
L2gradient – использование нормы L2 """

canny = cv2.Canny( blurred, 122, 225,)
cv2.imshow('CannyKirby', canny)

canny1 = cv2.Canny( treeblur, 122, 225, None, 7, True)
cv2.imshow('Canny', canny1)

cv2.waitKey(0)
```

Рассуждение: детектор границ Кенни является одним из лучших детекторов. С ним возможно работать с любыми типами границ.

Результаты:



Задание 2. Исследуйте работу функции findContours() на двух типах бинарных изображений:
1) бинарные изображения, полученные с помощью функции threshold()
2) бинарные изображения границ, полученные детектором границ Кенни

Код задания:

```
import cv2
import numpy


image = cv2.imread('/Users/dariabusi/Desktop/leti.jpg',cv2.IMREAD_GRAYSCALE)
cv2.imshow ('LETI_ORIGINAL', image)

new_threshold, img = cv2.threshold(image, 220, 255 , cv2.THRESH_BINARY)
cv2.imshow ('Thresh_Binary', img)

blurred = cv2.GaussianBlur(image, (1,1), 0)
canny = cv2.Canny( blurred, 122, 225,)
cv2.imshow('CannyKirby', canny)

"""функция findContours рассчитывает количество в бинарном изображении, где:
contours - список контуров изображения, и контура хранятся как матрицы Numpy;
hierarchy - вектор, количество элементов равен количеству контуров;
image - исходное изображение;
mode -  режим поиска контура;
method - метод аппроксимации контуров."""
contours, hierarchy = cv2.findContours(image=img, mode=cv2.RETR_TREE, method=cv2.CHAIN_APPROX_SIMPLE)
contours1, hierarchy = cv2.findContours(image=canny, mode=cv2.RETR_TREE, method=cv2.CHAIN_APPROX_SIMPLE)

```

```
24. print('Amount of contours when applying threshold: ',str(len(contours)))
25. print('Amount of contours when applying canny edge detection:
    ',str(len(contours1)))
26.
27. cv2.waitKey(0)
```

Результаты:


```
Amount of contours when applying threshold:  261
Amount of contours when applying canny edge detection:  284
```

Как отличается количество контуров? Почему?

В зависимости какой способ создания бинарного изображения применяется, количество контуров будет разным. В результате преобразования изображения с помощью threshold получаются черные линии, которые воспринимаются как контура, а детектор границ Кенни дает дополнительные белые линии-контура.

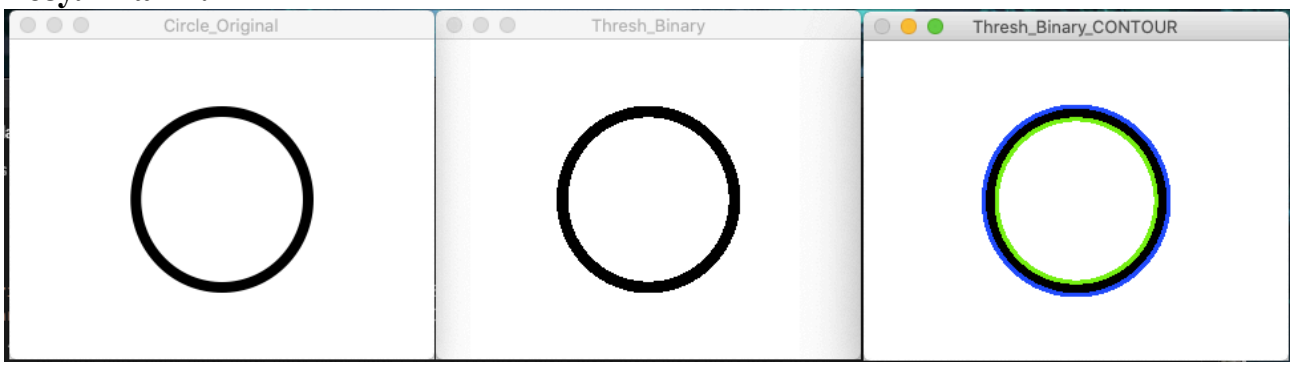
Возьмите изображение окружности с толщиной линии в несколько пикселей (вы можете самостоятельно нарисовать его, например, в paint). Вычислите контуры на этом изображении. Найдите один контур, который описывает окружность с внешней стороны линии, и один контур, который описывает окружность с внутренней стороны линии.
Для них вычислите длину, площадь. Почему значения отличаются таким образом? Для каждого контура вычислите ограничивающий прямоугольник и ограничивающую окружность. Сравните значения площадей ограничивающих фигур с площадями контуров. Прокомментируйте результат.

Код задания:

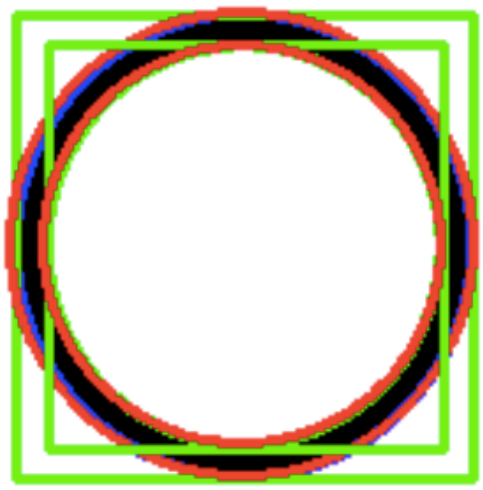
```
import cv2
import numpy
from math import pi

image1=cv2.imread('/Users/dariabusi/Desktop/testsub.jpg',cv2.IMREAD_REDUCED_COLOR_2)
image = cv2.imread('/Users/dariabusi/Desktop/testsub.jpg',cv2.IMREAD_REDUCED_GRAYSCALE_2)
cv2.imshow ('Circle_Original', image)

new_threshold, img = cv2.threshold(image, 220, 255, cv2.THRESH_BINARY)
cv2.imshow ('Thresh_Binary', img)

contours, hierarchy = cv2.findContours(image=img, mode=cv2.RETR_TREE, method=cv2.CHAIN_APPROX_SIMPLE)

print('Amount of contours: ',str(len(contours)))

"""функция cv2.drawContours позволяет выделять контура определенным цветом определенной толщины, где:
cv2.drawContours(
image=image1 - исходное изображение для выделения конntуров,
contours=contours - все контура изображения,
contourIdx=1 - номер индекса определенного контура,
color=(0, 0, 0) - индексы цвета BRG,
thickness=3 - толщина выделяемого контура"""
cv2.drawContours (image1, contours, 1, (255, 0, 0), 2)
cv2.drawContours (image1, contours, 2, (0, 255, 0), 2)
cv2.imshow ('Thresh_Binary_CONTOUR', image1)

"""функция arcLength расчитывает периметер дуги(контура), где:
perimeter = cv2.arcLength(cnt,True)
cnt - контур
True - указывает замкнутость дуги"""
print('perimeter inner circle: ' ,str(cv2.arcLength(contours[1], True)))
"""функция contourArea считает площать указанного контура"""
print('area inner circle: ' ,str(cv2.contourArea(contours[1])))

"""функция boundingRect выдает ограничивающий прямоугольник с координатами верхней левой точки угла прямоугольника,
шириной w и высотой h. по полученным результатам строим его с помощью rectangle"""
x, y, w, h = cv2.boundingRect(contours[1])
cv2.rectangle(image1, (x, y), (x + w, y + h), (0, 255, 0), 2)
print('area of the outer bounding rectangle: ', str(w*h))

"""minEnclosingCircle выдает координаты внешней описанной окружности с центром x,y и радиусом radius """
(x,y),radius = cv2.minEnclosingCircle(contours[1])
center = (int(x),int(y))
radius = int(radius)
cv2.circle(image1,center,radius,(0, 0, 255), thickness = 2)
print('area of outer enclosing circle: ' ,str(pi*(radius**2)))

print('perimeter outer circle: ' + str(cv2.arcLength(contours[2], True)))
print('area outer circle: ' + str(cv2.contourArea(contours[2])))

x, y, w, h = cv2.boundingRect(contours[2])
cv2.rectangle(image1, (x, y), (x + w, y + h), (0, 255, 0), 2)

print('area of inner bounding rectangle: '+ str(w*h))
(x,y),radius = cv2.minEnclosingCircle(contours[2])
center = (int(x),int(y))
radius = int(radius)
```

```
58. cv2.circle(image1,center,radius,(0, 0, 255), thickness = 2)
59. print('area of inner enclosing circle: ' , str(pi*(radius**2)))
60. cv2.waitKey(0)
```

Результаты:




```
Amount of contours:  3
perimeter inner circle:  463.93102049827576
area inner circle:  15482.0
area of the outer bounding rectangle:  19880
area of outer enclosing circle:  15836.768566746146
perimeter outer circle: 402.81832122802734
area outer circle: 11549.0
area of inner bounding rectangle: 15004
area of inner enclosing circle:  11689.86626400762
```



Рассуждение: ограничивающие контура начальной черной окружности имеют меньшую площадь, чем “enclosing circles”. Они имеют больший периметр и занимают больше пикселей.