|  |  |
| --- | --- |
| тов_знак_прав | МИНОБРНАУКИ РОССИИ  федеральное государственное автономное образовательное учреждение  высшего образования  «Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)»  **(СПбГЭТУ «ЛЭТИ»)** |

Кафедра САУ

ОТЧЕТ

по лабораторно-практической работе № 5

По курсу: «Техническое зрение»

Тема: Границы и контуры.

|  |  |  |
| --- | --- | --- |
| Студенты гр. 6491 |  | Зверев Г. Ю.  Михайленко Д. М. |
| Преподаватель |  | Моклева К. А. |

Санкт-Петербург

2020 г.

Цель работы: изучить способы выделения границ на изображении, поиск

контуров на границах и получения информации об объектах на основе контуров.

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.

Задание 2.

*Задание 2.1.*

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

Как отличается количество контуров? Почему?

*Задание 2.2.*

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.

Дополнительное задание №1

Для выполнения этого задания вам понадобится файл 5\_1.png. Для этого изображения все контуры на нечетном уровне вложенности нарисуйте красным цветом, все контуры на четном – синим цветом. На рисунке ниже приведен принцип нумерации уровней вложенности контуров.

Дополнительное задание №2

Для выполнения этого задания вам понадобится файл 5\_2.png. Для этого изображения нарисуйте все треугольники красным цветом, квадраты – синим, круги – зеленым. Используйте информацию о контурах и дополнительные операции над контурами.

ВАЖНО!!! В комментариях находится необходимая информация о том, какие аргументы, как и почему использованы внутри функций библиотеки OpenCV. Многострочные комментарии расположены перед использованием функций. Некоторые аргументы являются аргументами по умолчанию, а это значит, что их можно опускать (не задавать).

Задание №1

**Код программы:**

from cv2 import cv2

from matplotlib import pyplot as plt

""" Задание 1:

Изображения с четкими границами: автомобиль('vaz.png'), пЫНЯ('Putin.png'), логотип OpenCV('open-logo.png');

Изображения с нечеткими границами: лес('forest.png') и хамелеон('chameleon.png')"""

# img = cv2.imread('vaz.png', cv2.IMREAD\_REDUCED\_GRAYSCALE\_2)

# img = cv2.imread('Putin.png', cv2.IMREAD\_REDUCED\_GRAYSCALE\_4)

img = cv2.imread('open-logo.png', cv2.IMREAD\_GRAYSCALE)

# img = cv2.imread('forest.png', cv2.IMREAD\_REDUCED\_COLOR\_4)

# img = cv2.imread('chameleon.png', cv2.IMREAD\_REDUCED\_COLOR\_2)

"""Чем порядок производной больше, тем границы на изображении более размыты, появляется шум на изображении;

Порядок производной должен быть СТРОГО меньше размера ядра Собеля(ksize). По умолчанию ksize = 3;

Оператор Собеля 'ksize' вычисляет градиент яркости изображения в каждой точке, участки с большой величиной градиента (в основном, грани) будут видны как белые линии;

Параметр ddepth - глубина выходного изображения. Возьмем 'cv2.CV\_8U', что означает создание 8bit unsigned numpy array"""

sobel\_x = cv2.Sobel(src = img, ddepth = cv2.CV\_8U, dx = 1, dy = 0, ksize = 3)

sobel\_y = cv2.Sobel(img, cv2.CV\_8U, 0, 1)

sobel\_xy = cv2.Sobel(img, cv2.CV\_8U, 1, 1)

'''Оператор Лапласа идеально подходит для изображений с четкими границами'''

laplacian = cv2.Laplacian(img,cv2.CV\_8U, cv2.BORDER\_DEFAULT)

'''Перед использованием детектора границ Кенни рекомендуется выполнить размытие;

Отклонение от ядра по осям X,Y -> (x,y), где x,y целые нечетные числа'''

blur = cv2.GaussianBlur(img,(1,1), cv2.BORDER\_DEFAULT)

'''threshold1 - порог минимума, threshold2 - порог максимума;

apertureSize - размер для оператора Собеля. apertureSize = 3 по дефолту;

L2gradient если True, то градиент вычисляется точно, если False, то вычисляется по упрощенной формуле'''

canny = cv2.Canny(image = blur, threshold1 = 200, threshold2 = 225, apertureSize = 3, L2gradient = True)

"""!!!Matplotlib.pyplot.imshow() принимает параметр "cmap", который ИГНОРИРУЕТСЯ ТОЛЬКО для RGB изображений!!!

6 изображений на 1-ой фигуре выглядят так, что невозможно найти разницу между ними.

Пример вывода я оставлю, а в дальнейшем буду пользоваться выводом на экран с помощью библиотеки openCV: cv2.imshow()

plt.subplot(3, 2, 1),plt.imshow(img, cmap = 'gray')

plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(3, 2, 2), plt.imshow(sobel\_x, cmap = 'gray')

plt.title('Sobel\_x'), plt.xticks([]), plt.yticks([])

plt.subplot(3, 2, 3), plt.imshow(sobel\_y, cmap = 'gray')

plt.title('Sobel\_y'), plt.xticks([]), plt.yticks([])

plt.subplot(3, 2, 4), plt.imshow(sobel\_xy, cmap = 'gray')

plt.title('Sobel\_xy'), plt.xticks([]), plt.yticks([])

plt.subplot(3, 2, 5), plt.imshow(laplacian, cmap = 'gray')

plt.title('Laplacian'), plt.xticks([]), plt.yticks([])

plt.subplot(3, 2, 6), plt.imshow(canny, cmap = 'gray')

plt.title('Canny'), plt.xticks([]), plt.yticks([])

plt.show()"""

cv2.imshow('Original', img)

cv2.imshow('Sobel\_x', sobel\_x)

cv2.imshow('Sobel\_y', sobel\_y)

cv2.imshow('Sobel\_xy', sobel\_xy)

cv2.imshow('Laplacian', laplacian)

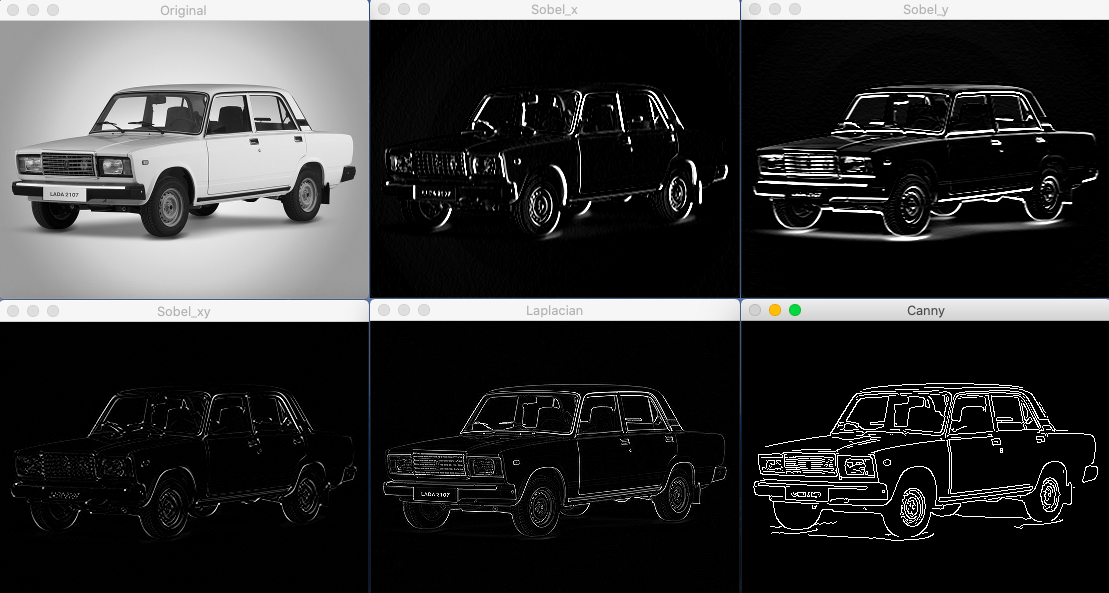
cv2.imshow('Canny', canny)

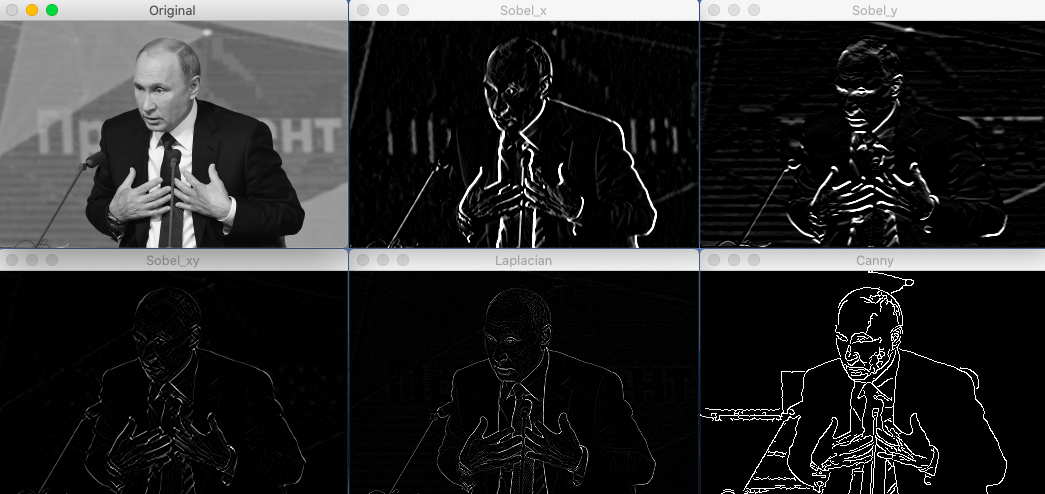
cv2.waitKey(0)

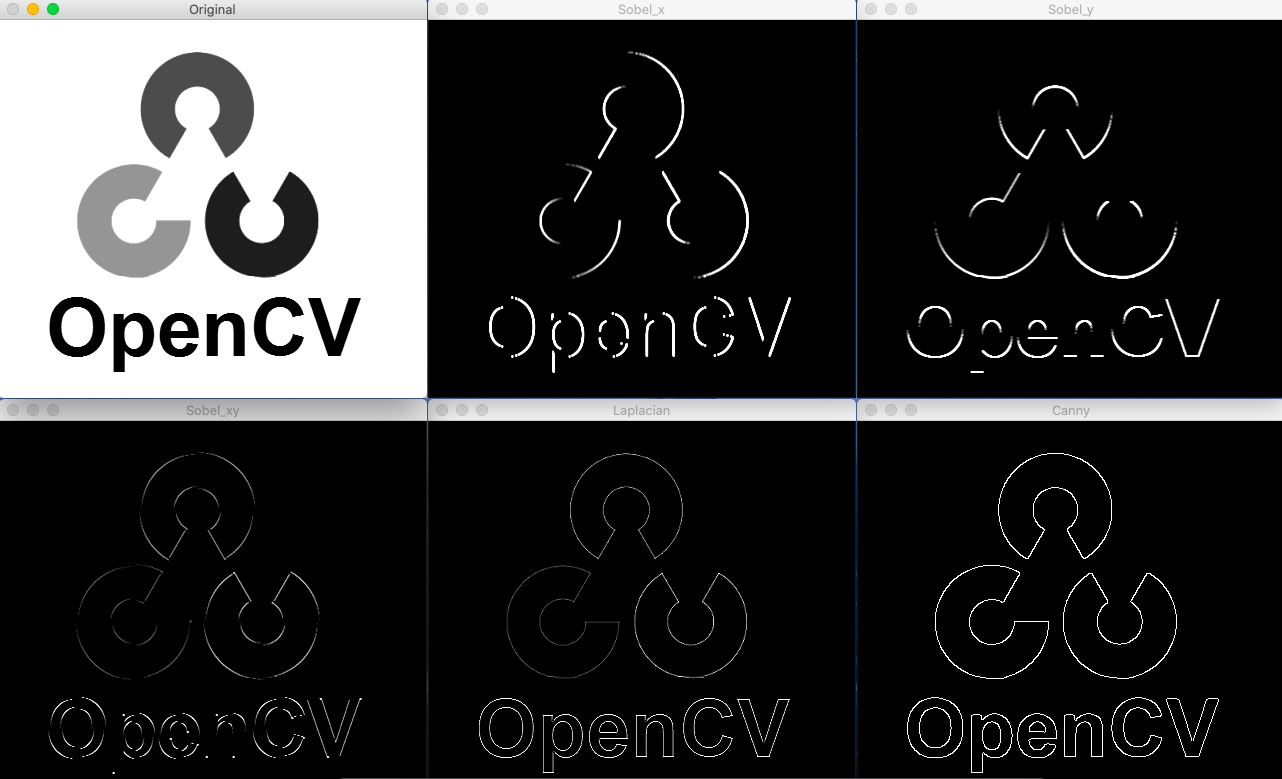
cv2.destroyAllWindows()

**Результат:**

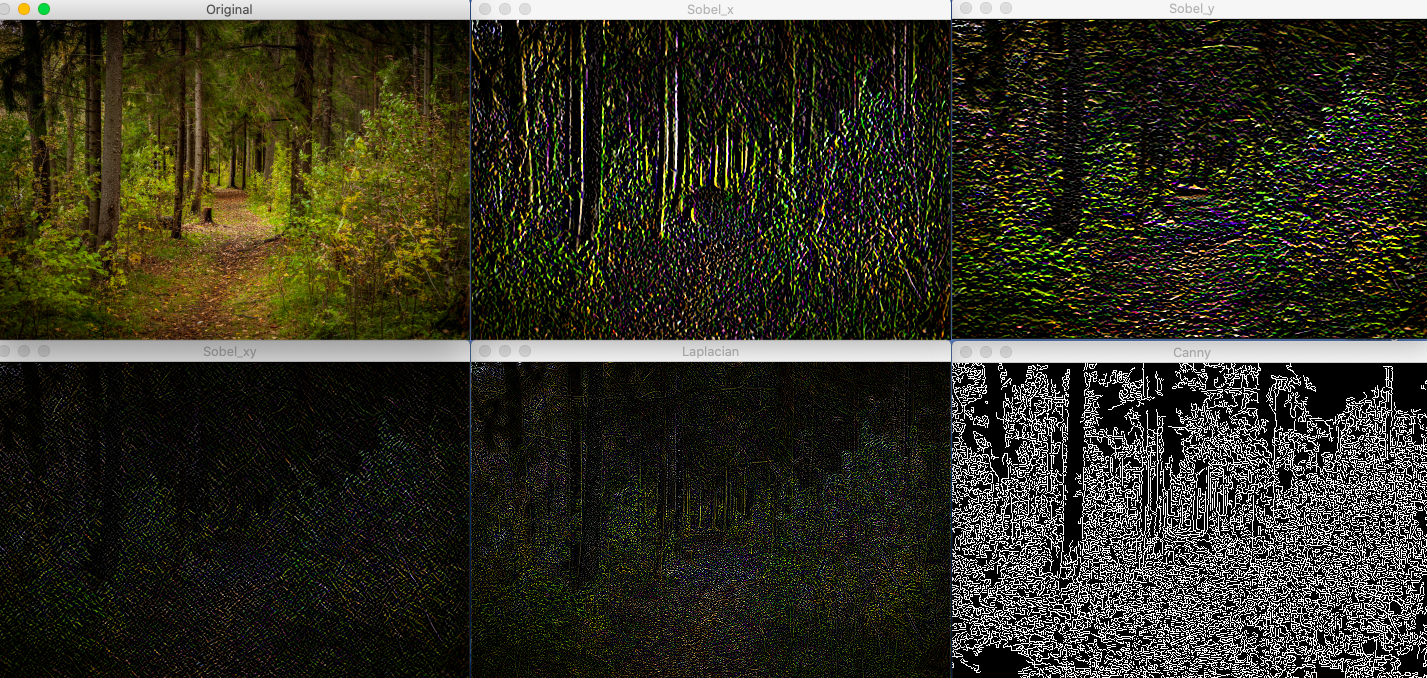
Изображения с четко отделимыми от фона границами:

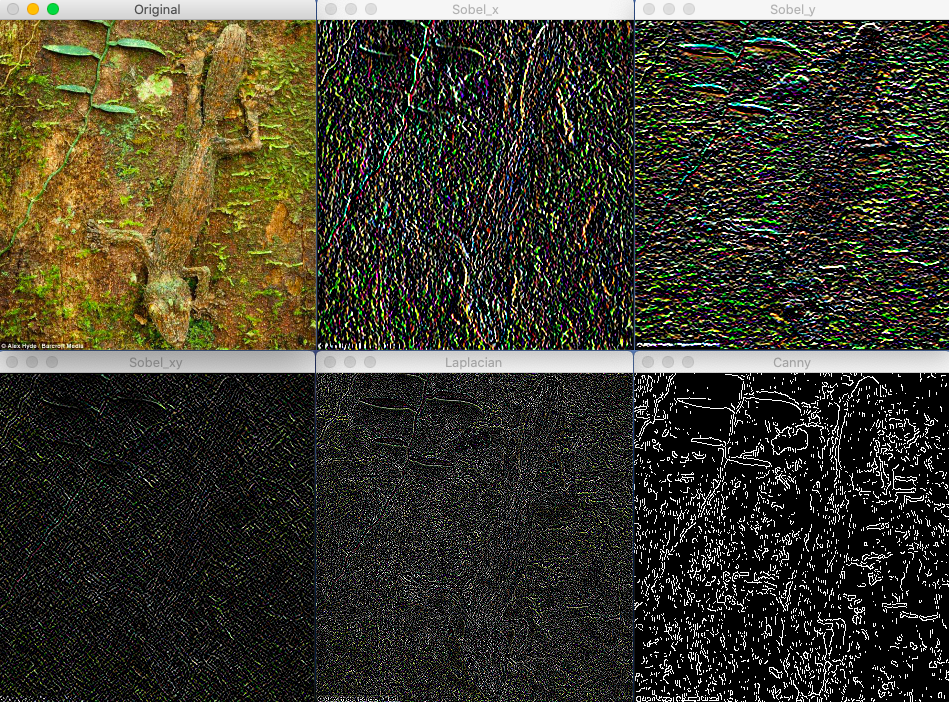






Изображения с нечеткими границами, почти сливающимися с фоном:





*Дополнительная информация об алгоритме Кенни:*

Пороги (threshold) - это произвольные значения, используемые алгоритмом Кэнни для принятия решения по каждому из пикселей на заключительном этапе своей работы. Порогов всего два: верхний и нижний.

Если значение магнитуды градиента каждого отдельного пикселя оказывается больше верхнего порога, то такой пиксель считается границей. Если значение магнитуды градиента оказывается меньше нижнего порога, то такой пиксель отбрасывается (более не учитывается).

Если же значение магнитуды градиента пикселя оказывается между порогами, то в этой ситуации в различных реализациях алгоритма проводят либо отдельную проверку, ориентируясь на значения и направления градиента у смежных пикселей, либо, как это имеет место быть в OpenCV, помечают его в качестве границы при условии, что рассматриваемый пиксель соседствует с пикселем, имеющим значение магнитуды градиента больше верхнего порога.

Размер ядра (kernel size или aperture size) - это размер квадратных матриц свёртки (convolution), которые используются для вычисления градиента на вертикальном и горизонтальном направлениях. В случае с OpenCV допустимыми значениями размера ядра могут быть только "3", "5" или "7". Эту функцию выполняет оператор Собеля, который прозрачно вызывается в функции Canny().

Свёртка в отношении изображений - это по сути операция вычисления нового значения для каждого пикселя на основе значений других пикселей, являющихся ему соседями. В свёртке, помимо самого изображения, участвует отдельная матрица, зачастую называемая окном или фильтром. Обычно используют квадратную, с равным, но при этом нечётным количеством строк и колонок. Нечётность нужна для того, чтобы в общем случае получить центральный элемент матрицы или, иначе, так называемый якорь:

( )( )( )

( )(x)( )

( )( )( )

В процессе свёртки на каждый пиксель исходного изображения как бы накладывается якорь, тем самым накрывая фильтром и все соседние у текущего пиксели. Затем вычисляется сумма произведений значений пикселей на значения соответствующих элементов фильтра, что и будет являться новым значением пикселя.

Задание №2.1

**Код программы:**

from cv2 import cv2

img = cv2.imread('open-logo.png')

img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

ret, thresh = cv2.threshold(img\_gray, 150, 255, cv2.THRESH\_BINARY)

"""Contours это список всех контуров изображения, каждый контур это Numpy array, где (x,y) координаты граничных точек объекта"""

contours1, hierarchy = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)

print('Количество контуров функции threshold():' + str(len(contours1)))

blur = cv2.GaussianBlur(img\_gray,(1,1), cv2.BORDER\_DEFAULT)

canny = cv2.Canny(image = blur, threshold1 = 200, threshold2 = 225, apertureSize = 3, L2gradient = True)

contours2, hierarchy = cv2.findContours(canny, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)

print('Количество контуров функции Canny():' + str(len(contours2)) + '\n')

# Выделим все контуры цветом на исходном изображении

cv2.drawContours(img, contours2, -1, (25, 150, 230), 2)

cv2.imshow('Orig', img)

cv2.imshow('Image gray', img\_gray)

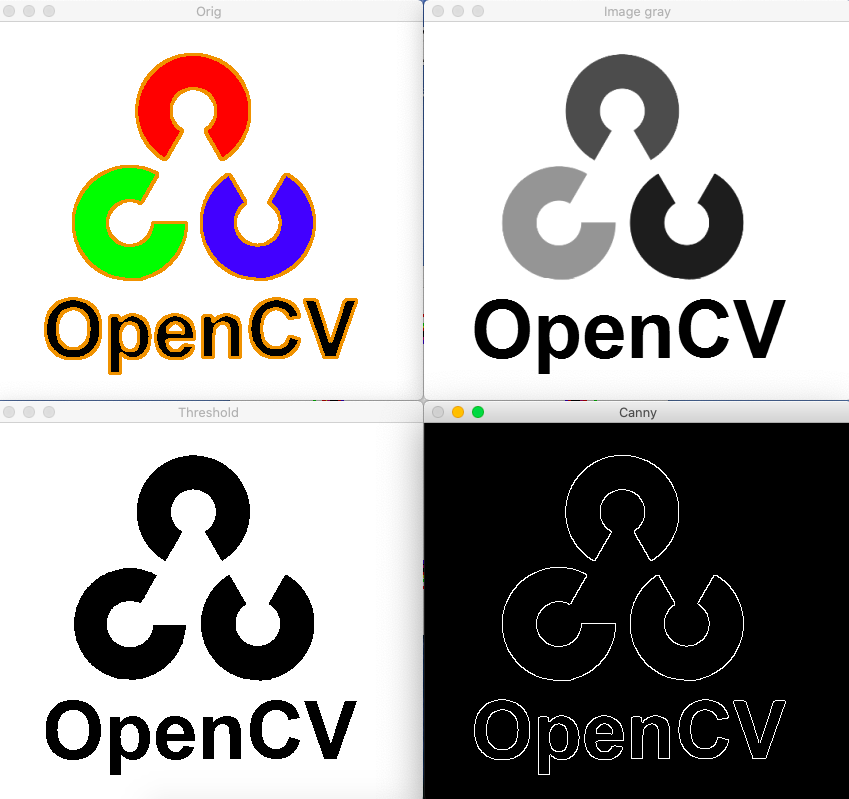
cv2.imshow('Threshold', thresh)

cv2.imshow('Canny', canny)

**Результат:**

Количество контуров функции threshold(): 13

Количество контуров функции Canny(): 23



Причина, по которой мы получаем разное количество контуров, заключается в том, что мы применяем детектор края Кенни к нашему изображению. Это дает результат «белых линий» - краев на изображении. Получаются "отдельные" линии, которые также являются контурами.

Задание №2.2

**Код программмы:**

from cv2 import cv2

from math import pi

circle = cv2.imread('circle.png')

circle\_gray = cv2.cvtColor(circle, cv2.COLOR\_BGR2GRAY)

\_, thresh\_circle = cv2.threshold(circle\_gray, 150, 255, cv2.THRESH\_BINARY)

contours, hierarchy = cv2.findContours(thresh\_circle, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)

print('Количество контуров: '+ str(len(contours)) + '\n')

cv2.drawContours(circle, contours, 1, (28, 255, 249), 2)

cv2.drawContours(circle, contours, 2, (255, 0, 0), 2)

outside = contours[1]

inside = contours[2]

'''cv2.contourArea() – вычисление площади.

The function computes a contour area. Similarly, to moments, the area is computed using the Green formula.

Thus, the returned area and the number of non-zero pixels.

Also, the function will most certainly give a wrong result for contours with self-intersections.'''

print('Периметр контура внешней окружности: ' + str(cv2.arcLength(outside, True)))

print('Площадь контура внешней окружности: ' + str(cv2.contourArea(outside)))

# Получим координаты внешнего ограничивающего прямоугольника:

x, y, w, h = cv2.boundingRect(outside)

# Нарисуем внешний огр.прямоугольник:

cv2.rectangle(circle, (x, y), (x + w, y + h), (0, 255, 0), 2)

print('Площадь внешнего ограничивающего прямоугольника: '+ str(w\*h))

# Получим координаты внешней описанной окружности:

(x,y),radius = cv2.minEnclosingCircle(outside)

center = (int(x),int(y))

radius = int(radius)

# Нарисуем внешнюю ограничивающую окружность:

cv2.circle(circle,center,radius,(0, 0, 255), thickness = 2)

print('Площадь внешней ограничивающей окружности: ' + str(pi\*(radius\*\*2)) + '\n')

print('Периметр контура внутренней окружности: ' + str(cv2.arcLength(inside, True)))

print('Площадь контура внутренней окружности: ' + str(cv2.contourArea(inside)))

# Получим координаты внутреннего ограничивающего прямоугольника:

x, y, w, h = cv2.boundingRect(inside)

# Нарисуем внутренний огр.прямоугольник:

cv2.rectangle(circle, (x, y), (x + w, y + h), (133, 21, 199),2)

print('Площадь внутреннего ограничивающего прямоугольника: '+ str(w\*h))

# Получим координаты внутренней ограничивающей окружности:

(x,y),radius = cv2.minEnclosingCircle(inside)

center = (int(x),int(y))

radius = int(radius)

cv2.circle(circle,center,radius,(0,69,255),2)

print('Площадь внутренней ограничивающей окружности: ' + str(pi\*(radius\*\*2)) + '\n')

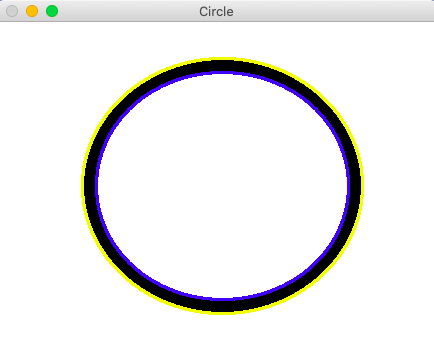
cv2.imshow('Circle', circle)

cv2.waitKey(0)

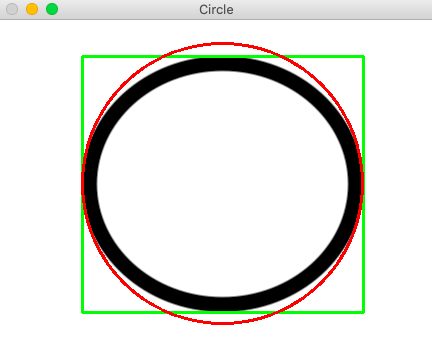
cv2.destroyAllWindows()

**Результат:**

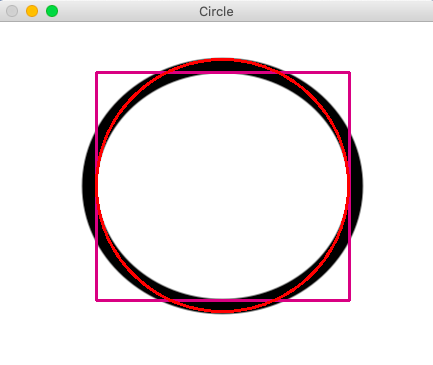
Внутренний и внешний контуры окружности:



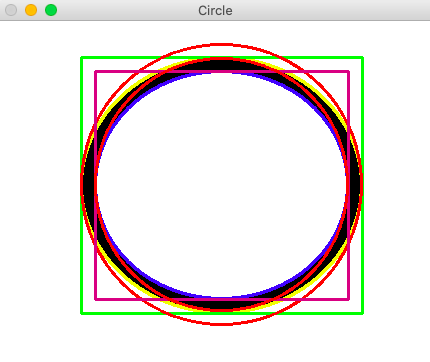
Описанная окружность и описанный прямоугольник внешнего контура:

****

Описанная окружность и описанный прямоугольник внутреннего контура:

****

Итог:

****

Периметр контура внешней окружности: 892.891478061676

Площадь контура внешней окружности: 56984.0

Площадь внешнего ограничивающего прямоугольника: 73014

Площадь внешней ограничивающей окружности: 62458.00354601868

Периметр контура внутренней окружности: 788.3229365348816

Площадь контура внутренней окружности: 44154.0

Площадь внутреннего ограничивающего прямоугольника: 56726

Площадь внутренней ограничивающей окружности: 49087.385212340516

Значения получены в пикселях. Площади фигур, которые ограничивают контуры, больше, потому что они больше пикселей занимают на изображении.

Дополнительное задание №2

**Код программы:**

from cv2 import cv2

img = cv2.imread('5\_2.png')

img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

ret, thresh = cv2.threshold(img\_gray, 200, 255, cv2.THRESH\_BINARY\_INV)

contours, hierarchy = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)

'''epsilon отвечает за точность аппроксимации;

так как мы итерируем, то каждый обход(шаг) у нас будет только 1 контур -> contourIdx = 0;

contours = [approx] в [], т.к drawContours ожидает массив (список в случае Python) контуров, а не только один массив (который возвращается из approxPolyDP).'''

for contour in contours:

approx = cv2.approxPolyDP(curve = contour, epsilon = 0.01 \* cv2.arcLength(contour, True), closed = True)

if len(approx) == 3:

cv2.drawContours(image = img, contours = [approx], contourIdx = 0, color = (0, 0, 255), thickness = 3)

elif len(approx) == 4:

cv2.drawContours(img, [approx], 0, (255, 0, 0 ), 3)

else:

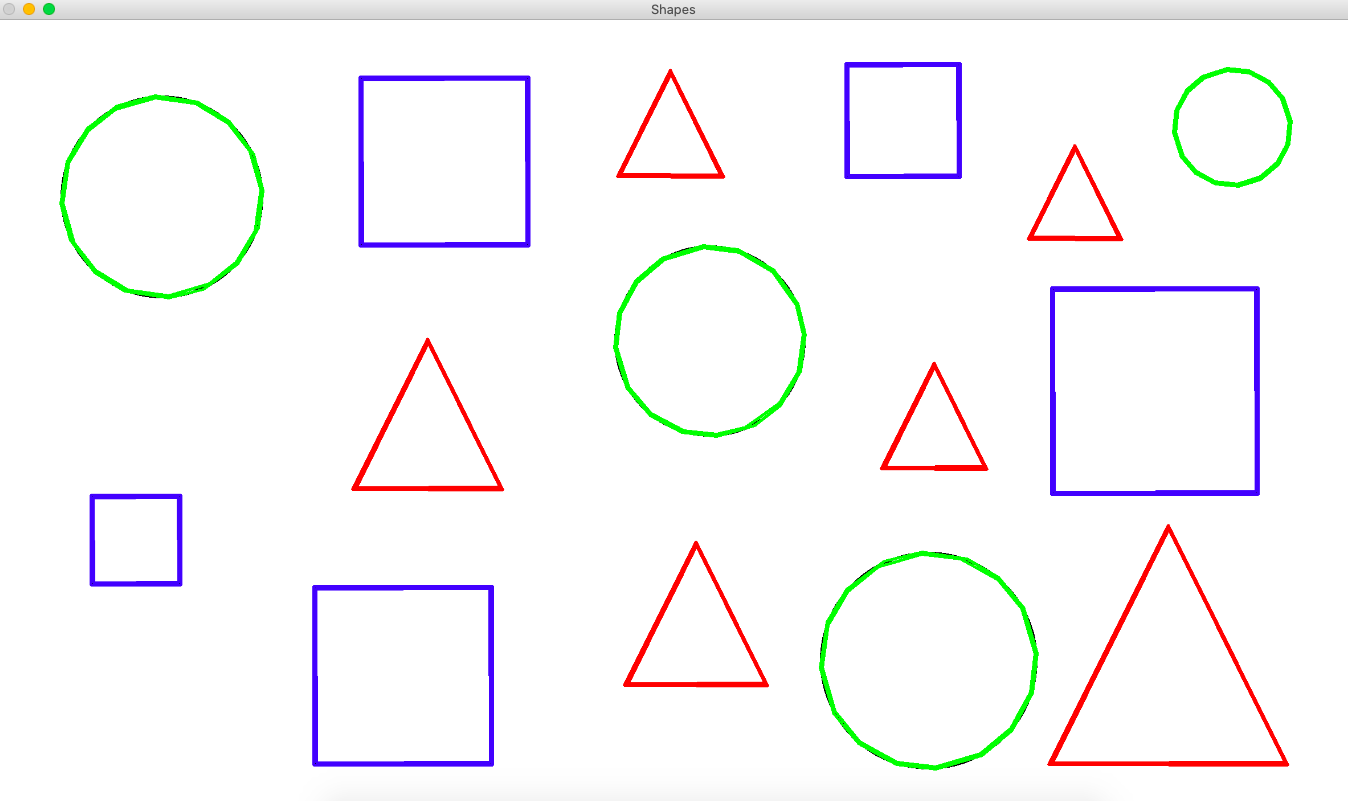
cv2.drawContours(img, [approx], 0, (0, 255, ), 3)

cv2.imshow('Shapes', img)

cv2.waitKey(0)

cv2.destroyAllWindows

**Результат:**



Вывод: в ходе лабораторной работы были изучены способы выделения границ на изображении, поиск контуров на границах и получения информации об объектах на основе контуров.