Лабораторная работа №4

# Градиенты Изображения

*Цель:*

*В этой главе мы научимся:*

*Найти градиенты изображения, края и т. д*

*Мы увидим следующие функции : cv2.Sobel(), cv2.Scharr(), cv2.Laplacian() и т. д*

*Теория*

*OpenCV предоставляет три типа градиентных фильтров или фильтров высоких частот: Sobel, Scharr и Laplacian.* Мы увидим каждого из них.

**Производные Собеля и Шарра**

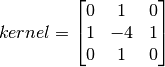
Операторы Собеля-это совместная операция гауссовского сглаживания плюс дифференцирования, поэтому она более устойчива к шуму. Вы можете указать направление производных, которые будут взяты, вертикальное или горизонтальное (по аргументам, порядку и порядку соответственно). Вы также можете указать размер ядра по размеру аргумента. Если ksize = -1, то используется фильтр Шарра 3x3, который дает лучшие результаты, чем фильтр Собеля 3x3. RTFM для используемых ядер.

**Производные Лапласа**

Он вычисляет лапласиан изображения, заданного соотношением,

\Delta src = \frac{\partial ^2{src}}{\partial x^2} + \frac{\partial ^2{src}}{\partial y^2}

где каждая производная находится с использованием производных Собеля. Если размер k = 1, то для фильтрации используется следующее ядро:



Ниже код показывает все операторы в одной диаграмме. Все ядра имеют размер 5х5. Глубина выходного изображения передается -1, чтобы получить результат в типе **np.uint8.**

**import cv2**

**import numpy as np**

**img = cv2.imread('dHWei9jpqrM.jpg',1)**

**laplacian = cv2.Laplacian(img,cv2.CV\_64F)**

**sobelx = cv2.Sobel(img,cv2.CV\_64F,1,0,ksize=5)**

**sobely = cv2.Sobel(img,cv2.CV\_64F,0,1,ksize=5)**

**cv2.namedWindow('img', cv2.WINDOW\_NORMAL)**

**cv2.imshow('img',img)**

**cv2.namedWindow('laplacian', cv2.WINDOW\_NORMAL)**

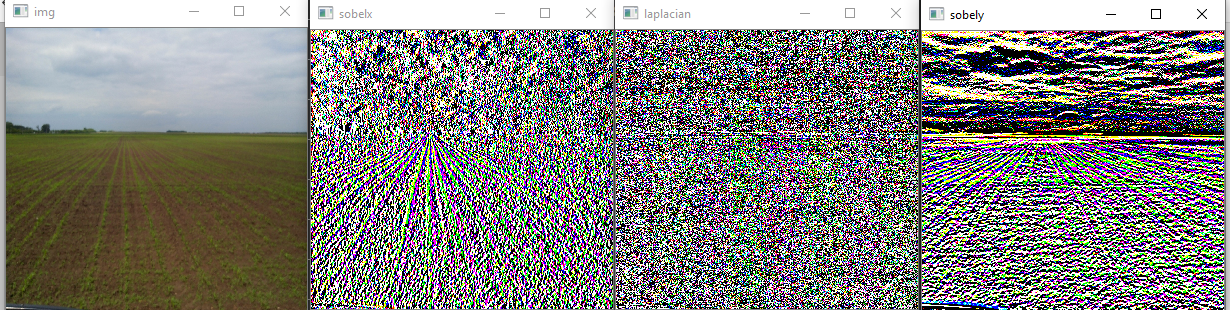
**cv2.imshow('laplacian',laplacian)**

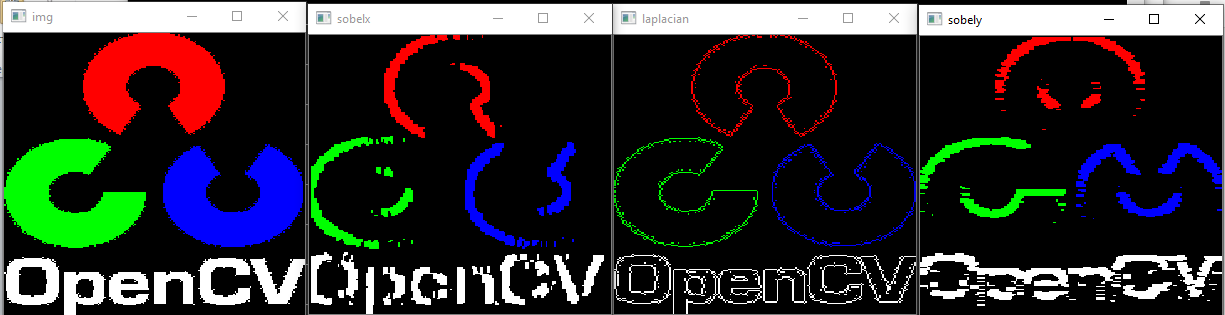
**cv2.namedWindow('sobelx', cv2.WINDOW\_NORMAL)**

**cv2.imshow('sobelx',sobelx)**

**cv2.namedWindow('sobely', cv2.WINDOW\_NORMAL)**

**cv2.imshow('sobely',sobely)**





В нашем последнем примере тип выходных данных-cv2.CV\_8U или np.uint8. Но тут есть небольшая проблема. Переход от черного к белому принимается за положительный наклон (он имеет положительное значение), а переход от белого к Черному принимается за отрицательный наклон (он имеет отрицательное значение). Поэтому при преобразовании данных в np.uint8 все отрицательные наклоны делаются равными нулю. Проще говоря, вы упускаете эту грань.

Если вы хотите обнаружить оба ребра, лучше всего сохранить выходной тип данных в некоторых более высоких формах, таких как cv2.CV\_16S, cv2.CV\_64F и т. д., взять его абсолютное значение и затем преобразовать обратно в cv2.CV\_8U. Ниже код демонстрирует эту процедуру для горизонтального фильтра Собеля и разницу в результатах.

**import cv2**

**import numpy as np**

**img = cv2.imread('opencv-logo-white.png',1)**

**sobelx8u = cv2.Sobel(img,cv2.CV\_8U,1,0,ksize=5)**

**# Output dtype = cv2.CV\_64F. Then take its absolute and convert to cv2.CV\_8U**

**sobelx64f = cv2.Sobel(img,cv2.CV\_64F,1,0,ksize=5)**

**abs\_sobel64f = np.absolute(sobelx64f)**

**sobel\_8u = np.uint8(abs\_sobel64f)**

**cv2.namedWindow('sobelx8u', cv2.WINDOW\_NORMAL)**

**cv2.imshow('sobelx8u',sobelx8u)**

**cv2.namedWindow('sobelx64f', cv2.WINDOW\_NORMAL)**

**cv2.imshow('sobelx64f',sobelx64f)**

**cv2.namedWindow('abs\_sobel64f', cv2.WINDOW\_NORMAL)**

**cv2.imshow('abs\_sobel64f',abs\_sobel64f)**

**cv2.namedWindow('sobel\_8u', cv2.WINDOW\_NORMAL)**

**cv2.imshow('sobel\_8u',sobel\_8u)**

**cv2.waitKey(0)**



# Выделение границ

*Цель*

*В этой главе мы познакомимся с концепцей обнаружения границ (Canny Edge Detection)*

*Функции OpenCV для этого : cv2.Canny()*

**Теория**

Canny Edge Detection-это популярный алгоритм обнаружения краев. Он был разработан Джоном Ф. Кэнни в 1986 году. Это многоступенчатый алгоритм, и мы будем проходить через каждый этап.

**1 - шумоподавление**

Поскольку обнаружение краев чувствительно к шуму в изображении, первым шагом является удаление шума в изображении с помощью гауссовского фильтра 5x5. Мы уже видели это в предыдущих главах.

**2 - нахождение градиента интенсивности изображения**

Сглаженное изображение затем фильтруется ядром Собеля как в горизонтальном, так и в вертикальном направлении, чтобы получить первую производную в горизонтальном направлении (G\_x) и вертикальном направлении (G\_y). Из этих двух изображений мы можем найти градиент края и направление для каждого пикселя следующим образом:

Edge\_Gradient \; (G) = \sqrt{G_x^2 + G_y^2}

Angle \; (\theta) = \tan^{-1} \bigg(\frac{G_y}{G_x}\bigg)

Направление градиента всегда перпендикулярно краям. Он округляется до одного из четырех углов, представляющих вертикальное, горизонтальное и два диагональных направления.

**3 - Не-максимальное подавление**

После получения величины и направления градиента выполняется полное сканирование изображения, чтобы удалить все нежелательные пиксели, которые могут не составлять края. Для этого на каждом пикселе проверяется, является ли он локальным максимумом в своей окрестности в направлении градиента. Проверьте изображение ниже:

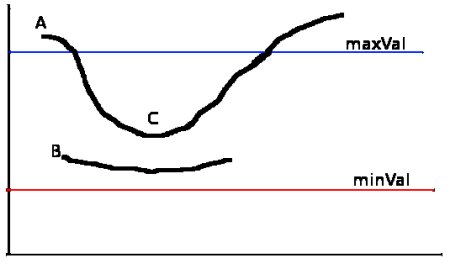


Точка А находится на краю ( в вертикальном направлении). Направление градиента перпендикулярно краю. Точки В и с находятся в градиентных направлениях. Поэтому точка а сверяется с точками В и С, чтобы увидеть, образует ли она локальный максимум. Если это так, то он рассматривается для следующей стадии, в противном случае он подавляется ( ставится на ноль).

Короче говоря, результат, который вы получаете, - это двоичное изображение с “тонкими краями”.

**4 - Пороговый гистерезис**

Эта стадия решает, какие из всех ребер действительно являются ребрами, а какие нет. Для этого нам нужны два пороговых значения, minVal и maxVal. Любые ребра с градиентом интенсивности больше, чем maxVal, обязательно будут ребрами, а те, что ниже minVal, обязательно будут не ребрами, поэтому отбрасываются. Те, кто находится между этими двумя порогами, классифицируются как ребра или не-ребра на основе их связности. Если они соединены с пикселями “уверенного края”, они считаются частью ребер. В противном случае они также отбрасываются:



Край а находится выше maxVal, поэтому считается “уверенным краем”. Хотя ребро С находится ниже maxVal, оно соединено с ребром а, так что оно также считается допустимым ребром, и мы получаем эту полную кривую. Но край B, хотя это и выше, минимальное-в той же области, как пограничного С, он не подключен к любой “точно-край”, так что отбрасывается. Поэтому очень важно, чтобы мы выбрали minVal и maxVal соответственно, чтобы получить правильный результат.

Этот этап также удаляет мелкие пиксельные шумы в предположении, что ребра-это длинные линии.

Итак, то, что мы, наконец, получаем, - это сильные края в изображении.

**Canny Edge Detection в OpenCV**

OpenCV помещает все вышесказанное в одну функцию cv2.Canny(). Посмотрим, как это использовать. Первый аргумент-это наш входной образ. Второй и третий аргументы-это наши minVal и maxVal соответственно. Третий аргумент-aperture\_size. Это размер ядра Собеля, используемого для поиска градиентов изображения. По умолчанию он равен 3. Последний аргумент-градиент L2, который задает уравнение для нахождения величины градиента. Если он True, то он использует уравнение, упомянутое выше, которое является более точным, в противном случае он использует эту функцию: Edge\_Gradient \; (G) = |G_x| + |G_y|.

**import cv2**

**import numpy as np**

**img = cv2.imread('logo.jpg',1)**

**edges = cv2.Canny(img,10,180,5)**

**cv2.namedWindow('img', cv2.WINDOW\_NORMAL)**

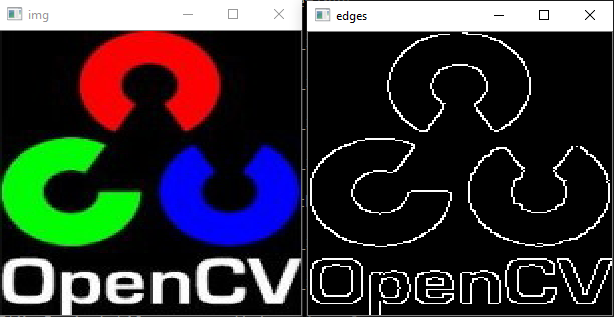
**cv2.imshow('img',img)**

**cv2.namedWindow('edges', cv2.WINDOW\_NORMAL)**

**cv2.imshow('edges',edges)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



# Пирамиды изображений

*Цель*

*В этой главе, мы узнаем о пирамидах изображений*

*Рассмотрим функции: cv2.pyrUp(), cv2.pyrDown()*

**Теория**

Обычно мы работали с изображением постоянного размера. Но в некоторых случаях нам приходится работать с изображениями разного разрешения одного и того же изображения. Например, при поиске чего-то на изображении, например лица, мы не уверены, какого размера объект будет присутствовать на изображении. В этом случае нам нужно будет создать набор изображений с различным разрешением и искать объект во всех изображениях. Эти наборы изображений с разным разрешением называются пирамидами изображений (потому что, когда они хранятся в стопке с самым большим изображением внизу и самым маленьким изображением вверху, они выглядят как пирамида).

Существует два вида пирамид изображений. 1) пирамида Гаусса и 2) пирамиды Лапласа.

Более высокий уровень (низкое разрешение) в Гауссовой пирамиде формируется путем удаления последовательных строк и столбцов в изображении более низкого уровня (более высокое разрешение). Затем каждый пиксель на более высоком уровне формируется вкладом от 5 пикселей на нижележащем уровне с гауссовыми весами. Таким образом, изображение M \* N становится изображением M/2 \* N/2. Таким образом, площадь уменьшается до одной четвертой от первоначальной площади. Это называется Октава. Та же картина продолжается, когда мы поднимаемся вверх в пирамиде (т. е. разрешение уменьшается). Точно так же при расширении площадь становится 4 раза на каждом уровне. Мы можем найти Гауссовы пирамиды с помощью **cv2.pyrDown()** и cv2.функции **pyrUp()**.

Грубо говоря - cv2.pyrDown() – понижение разрешения изображения вполовину, а pyrUp() – повешение разрешения.

**import cv2**

**import numpy as np**

**img = cv2.imread('IMG\_20190819\_155021.jpg',1)**

**#edges = cv2.Canny(img,10,100,5)**

**lower\_reso = cv2.pyrDown(img)**

**lower\_reso = cv2.pyrDown(lower\_reso)**

**lower\_reso = cv2.pyrDown(lower\_reso)**

**lower\_reso = cv2.pyrDown(lower\_reso)**

**cv2.namedWindow('img', cv2.WINDOW\_NORMAL)**

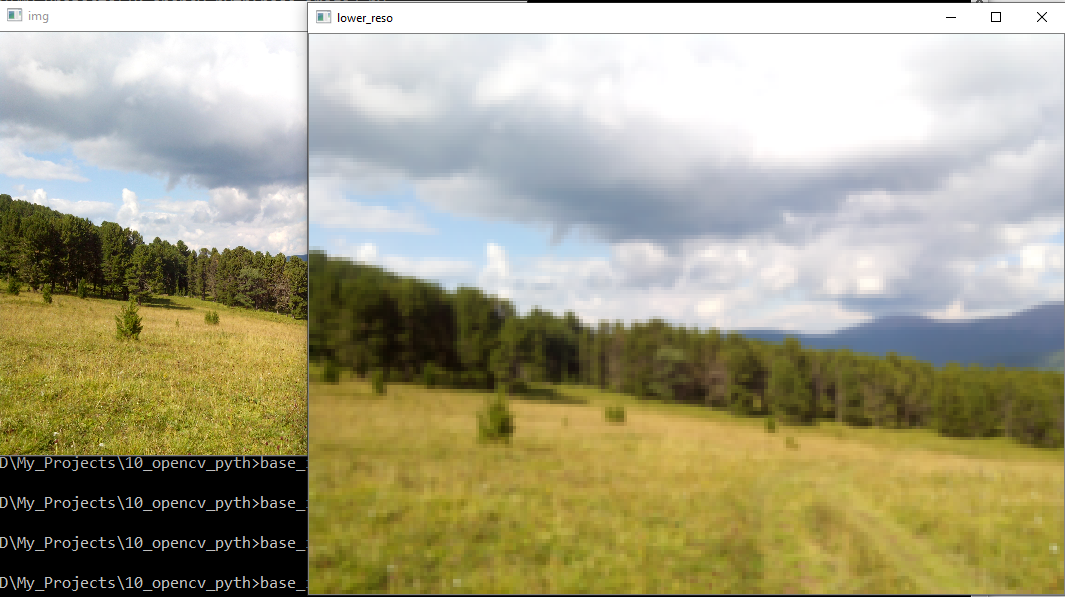
**cv2.imshow('img',img)**

**cv2.namedWindow('lower\_reso', cv2.WINDOW\_NORMAL)**

**cv2.imshow('lower\_reso',lower\_reso)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



После нескольких этапов повышения разрешения

**up\_reso = cv2.pyrUp(img)**

**up\_reso = cv2.pyrUp(up\_reso)**

**up\_reso = cv2.pyrUp(up\_reso)**



Лапласианские пирамиды образованы из гауссовых пирамид. Для этого нет исключительной функции. Изображения пирамид Лапласа похожи только на изображения краев. Большинство его элементов-нули. Они используются при сжатии изображений. Уровень в пирамиде Лапласа образуется разницей между этим уровнем в Гауссовой пирамиде и расширенной версией его верхнего уровня в Гауссовой пирамиде.

Одно из применений пирамид-смешивание изображений. Например, при сшивании изображений вам нужно будет сложить два изображения вместе, но это может выглядеть не очень хорошо из-за разрывов между изображениями. В этом случае смешивание изображений с пирамидами дает вам плавное смешивание, не оставляя большого количества данных на изображениях.

# Контуры в OpenCV

## Контуры….начала….

*Цель*

*Поймите, что такое контуры.*

*Научитесь находить контуры, рисовать контуры и т. д*

*Вы изучите функции : cv2.findContours(), cv2.drawContours()*

**Что такое контуры?**

Контуры можно объяснить просто как кривую, соединяющую все непрерывные точки (вдоль границы), имеющие одинаковый цвет или интенсивность. Контуры являются полезным инструментом для анализа формы и обнаружения и распознавания объектов.

Для большей точности используйте двоичные изображения. Поэтому, прежде чем найти контуры, примените пороговое или осторожное обнаружение краев.

Функция **cv2.findContours()** изменяет исходное изображение. Поэтому, если вы хотите получить исходное изображение даже после нахождения контуров, уже сохраните его в некоторых других переменных.

В OpenCV поиск контуров подобен поиску белого объекта на черном фоне. Поэтому помните, что объект, который нужно найти, должен быть белым, а фон-черным.

В функции cv2.findContours() есть три аргумента: первый-исходное изображение, второй - режим поиска контура, третий - метод аппроксимации контура. И он выводит изображение, контуры и иерархию. Контуры - это список Python всех контуров на изображении. Каждый отдельный контур представляет собой числовой массив координат (x,y) граничных точек объекта.

О втором и третьем аргументах, а также об иерархии мы поговорим подробнее позже. До тех пор значения, заданные им в примере кода, будут нормально работать для всех изображений.

В OpenCV для поиска контуров имеется функцией findContours, которая имеет вид:

**findContours( кадр, режим\_группировки, метод\_упаковки [, контуры[, иерархия[, сдвиг]]])**

**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:

CV\_RETR\_LIST — выдаёт все контуры без группировки;

CV\_RETR\_EXTERNAL — выдаёт только крайние внешние контуры. Например, если в кадре будет пончик, то функция вернет его внешнюю границу без дырки.

CV\_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.

CV\_RETR\_TREE — группирует контуры в многоуровневую иерархию.

метод\_упаковки — один из трёх методов упаковки контуров:

CV\_CHAIN\_APPROX\_NONE — упаковка отсутствует и все контуры хранятся в виде отрезков, состоящих из двух пикселей.

CV\_CHAIN\_APPROX\_SIMPLE — склеивает все горизонтальные, вертикальные и диагональные контуры.

CV\_CHAIN\_APPROX\_TC89\_L1,CV\_CHAIN\_APPROX\_TC89\_KCOS — применяет к контурам метод упаковки (аппроксимации) Teh-Chin.

**контуры** — список всех найденных контуров, представленных в виде векторов;

**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.

**сдвиг** — величина смещения точек контура.

**Как нарисовать контуры?**

Для рисования контуров используется функция **cv2.drawContours**. Он также может быть использована для рисования любой фигуры, если у вас есть ее граничные точки. Его первый аргумент-исходное изображение, второй аргумент - контуры, которые должны быть переданы в виде списка Python, третий аргумент-индекс контуров (полезно при рисовании отдельного контура. Чтобы нарисовать все контуры, передайте -1), а остальные аргументы-цвет, толщина и т. д.

Чтобы нарисовать все контуры на изображении:

**img = cv2.drawContours(img, contours, -1, (0,255,0), 3)**

Чтобы нарисовать отдельный контур, скажем, 4-й контур:

**img = cv2.drawContours(img, contours, 3, (0,255,0), 3)**

Но в большинстве случаев нижеприведенный метод будет полезен:

**cnt = contours[4]**

**img = cv2.drawContours(img, [cnt], 0, (0,255,0), 3)**

Функция OpenCV для отображения контуров **drawContours**

Полученные с помощью функции findContours контуры хорошо бы каким-то образом нарисовать в кадре. Машине это не нужно, зато нам это поможет лучше понять как выглядят найденные алгоритмом контуры. Поможет в этом ещё одна полезная функция — drawContours.

**drawContours( кадр, контуры, индекс, цвет[, толщина[, тип\_линии[, иерархия[, макс\_слой[, сдвиг]]]]])**

**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;

**иерархия** — информация об иерархии контуров;

**макс\_слой** — индекс слоя, который следует отображать. Если параметр равен 0, то будет отображен только выбранный контур. Если параметр равен 1, то отобразится выбранный контур и все его дочерние контуры. Если параметр равен 2, то отобразится выбранный контур, все его дочерние и дочерние дочерних! И так далее.

**сдвиг** — величина смещения точек контура.

Пример выделения контуров.

**import numpy as np**

**import cv2**

**image = cv2.imread('IMG\_20191109\_180417.jpg')**

**# избавляемся от фона**

**# Convert BGR to HSV**

**hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)**

**lower\_blue = np.array([50,200,20])**

**upper\_blue = np.array([150,255,255])**

**# Threshold the HSV image to get only blue colors**

**mask = cv2.inRange(hsv, lower\_blue, upper\_blue)**

**# формируем черный фон для отрисовки контуров на нем**

**nmask = cv2.bitwise\_not(mask)**

**fon=cv2.bitwise\_and(mask, nmask)**

**# убираем мелкие шумовые детали**

**kernel = np.ones((5,5),np.uint8)**

**opening = cv2.morphologyEx(nmask, cv2.MORPH\_OPEN, kernel)**

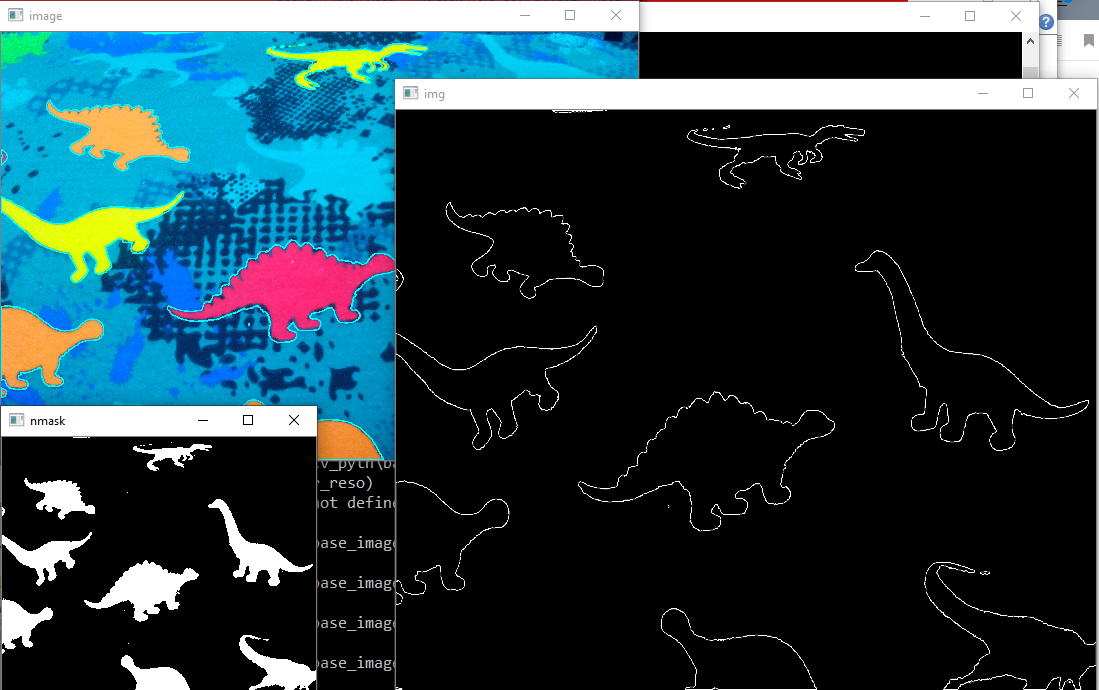
**# выделяем контуры**

**contours, hier = cv2.findContours(opening, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)**

**# отрисовываем контуры на черном фоне, и на исходном изображении**

**cv2.drawContours(fon,contours, -1, (255,0,0), 2, cv2.LINE\_AA, hier, 1 ) #W**

**cv2.drawContours(image,contours, -1, (255,255,0), 5, cv2.LINE\_AA, hier, 1 ) #W**



**Метод аппроксимации контуров**

Это третий аргумент в функции cv2.findContours. Что же оно обозначает на самом деле? Выше мы говорили, что контуры-это границы формы с одинаковой интенсивностью. Он хранит координаты (x,y) границы фигуры. Но хранит ли он все координаты ? Это определяется данным методом аппроксимации контуров.

Если вы передадите cv2.CHAIN\_APPROX\_NONE, то все граничные точки будут сохранены. Но действительно ли нам нужны все точки? Например, вы нашли контур прямой линии. Вам нужны все точки На линии, чтобы представить эту линию? Нет, нам нужны только две конечные точки этой линии. Это то, что делает cv2.CHAIN\_APPROX\_SIMPLE. Он удаляет все лишние точки и сжимает контур, тем самым экономя память.

Вопросы к лабораторной работе

1. привести подробное описание основных функций, рассмотренных в работе - функция, что делает, аргументы, параметры, возвращаемые результаты, типы данных и тп.
2. выполнить примеры, описанные в лабораторной - предоставить комментированный код, поэтапные результаты выполнения действий (исходные фото-, видео- данные для примеров берутся собственные, или из открытых источников или из базового набора библиотеки OpenCV)