Лабораторная работа №5

# Геометрические преобразования изображений

*Цели*

*Научитесь применять различные геометрические преобразования к изображениям, такие как перевод, вращение, аффинное преобразование и т. д.*

*Вы увидите следующие функции: cv2.getPerspectiveTransform*

## Преобразования изображений

OpenCV предоставляет две функции преобразования, **cv2.warpAffine** и **cv2.warpPerspective**, с помощью которых вы можете иметь все виды преобразований. cv2.warpAffine принимает матрицу преобразования 2x3, в то время как cv2.warperspective принимает матрицу преобразования 3x3 в качестве входных данных.

## Масштабирование

Масштабирование-это просто изменение размера изображения. OpenCV поставляется с функцией **cv2.resize()** для этой цели. Размер изображения можно задать вручную или задать коэффициент масштабирования. Используются различные методы интерполяции. Предпочтительными методами интерполяции являются cv2.INTER\_AREA для сжатия и cv2.INTER\_CUBIC (slow) & cv2.INTER\_LINEAR для масштабирования. По умолчанию используется метод интерполяции cv2.INTER\_LINEAR для всех целей изменения размера. Вы можете изменить размер входного изображения одним из следующих способов:

**import cv2**

**import numpy as np**

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

**#res = cv2.resize(img,None,fx=2, fy=2, interpolation = cv2.INTER\_CUBIC)**

**#OR**

**height, width = img.shape[:2]**

**res = cv2.resize(img,(2\*width, 2\*height), interpolation = cv2.INTER\_CUBIC)**

**cv2.namedWindow("res", cv2.WINDOW\_NORMAL)**

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

## Сдвиг

Сдвиг - это смещение местоположения объекта. Если вы знаете сдвиг в направлении (x,y), пусть это будет (t\_x,t\_y), вы можете создать матрицу преобразования \textbf{M} следующим образом:

M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y  \end{bmatrix}

Вы можете взять make it в массив Numpy типа np.float32 и передать его в функцию cv2.warpAffine (). См. ниже пример для сдвига (100,50). Третьим аргументом функции cv2.warpAffine() является размер выходного изображения, который должен быть в виде (ширина, высота). Помните, что ширина = количество столбцов, а высота = количество строк:

**import cv2**

**import numpy as np**

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

**rows, cols = img.shape[:2]**

**# задаем матрицу преобразования**

**M = np.float32([[1,0,100],[0,1,50]])**

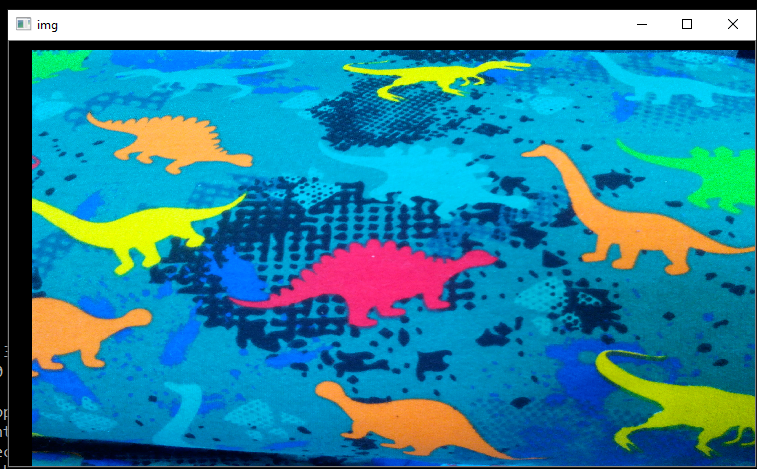
**dst = cv2.warpAffine(img,M,(cols,rows))**

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

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



## Вращение

Поворот изображения на угол ϑ достигается матрицей преобразования вида:

M = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta   \end{bmatrix}

Но OpenCV обеспечивает масштабированное вращение с регулируемым центром вращения, так что вы можете вращаться в любом месте, которое вы предпочитаете. Модифицированная матрица преобразования задается формулой

\begin{bmatrix} \alpha &  \beta & (1- \alpha )  \cdot center.x -  \beta \cdot center.y \\ - \beta &  \alpha &  \beta \cdot center.x + (1- \alpha )  \cdot center.y \end{bmatrix},

где

\begin{array}{l} \alpha =  scale \cdot \cos \theta , \\ \beta =  scale \cdot \sin \theta \end{array}

Чтобы найти эту матрицу преобразования, OpenCV предоставляет функцию **cv2.getRotationMatrix2D**. Проверьте приведенный ниже пример, который поворачивает изображение на 37 градусов относительно центра без какого-либо масштабирования.

**import cv2**

**import numpy as np**

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

**rows, cols = img.shape[:2]**

**M = cv2.getRotationMatrix2D((cols/2,rows/2),37,1)**

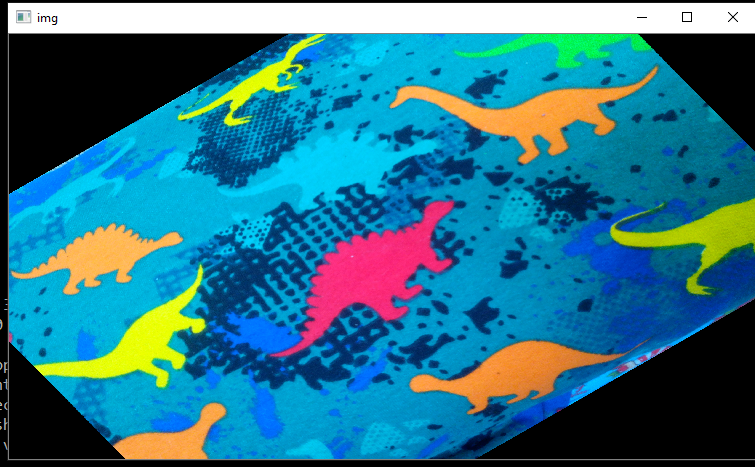
**dst = cv2.warpAffine(img,M,(cols,rows))**

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

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



## Аффинное преобразование

При аффинном преобразовании все параллельные линии в исходном изображении будут по-прежнему параллельны в выходном изображении. Чтобы найти матрицу преобразования, нам нужны три точки из входного изображения и их соответствующие местоположения в выходном изображении. Затем cv2.getAffineTransform создаст матрицу 2x3, которая будет передана в cv2.warpAffine.

**Import cv2**

**import numpy as np**

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

**rows, cols = img.shape[:2]**

**pts1 = np.float32([[50,50],[200,50],[50,200]])**

**pts2 = np.float32([[10,100],[200,50],[100,250]])**

**M = cv2.getAffineTransform(pts1,pts2)**

**dst = cv2.warpAffine(img,M,(cols,rows))**

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

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

## Перспективное преобразование

Для перспективного преобразования вам понадобится матрица преобразования 3x3. Прямые линии останутся прямыми даже после трансформации. Чтобы найти эту матрицу преобразования, вам нужно 4 точки на входном изображении и соответствующие точки на выходном изображении. Среди этих 4 точек 3 из них не должны быть коллинеарными. Тогда матрицу преобразования можно найти с помощью функции cv2.getPerspectiveTransform. Затем примените cv2.warpPerspective с этой матрицей преобразования 3x3.

**import cv2**

**import numpy as np**

**img = cv2.imread('perspective1.jpg')**

**rows, cols = img.shape[:2]**

**pts1 = np.float32([[27,34],[182,27],[13,190],[191,191]])**

**pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])**

**M = cv2.getPerspectiveTransform(pts1,pts2)**

**dst = cv2.warpPerspective(img,M,(300,300))**

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

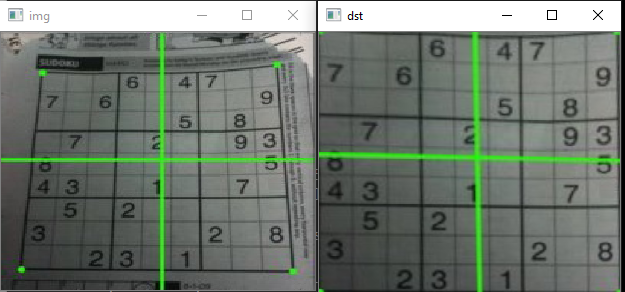
**cv2.imshow('img',img)**

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

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

**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)**

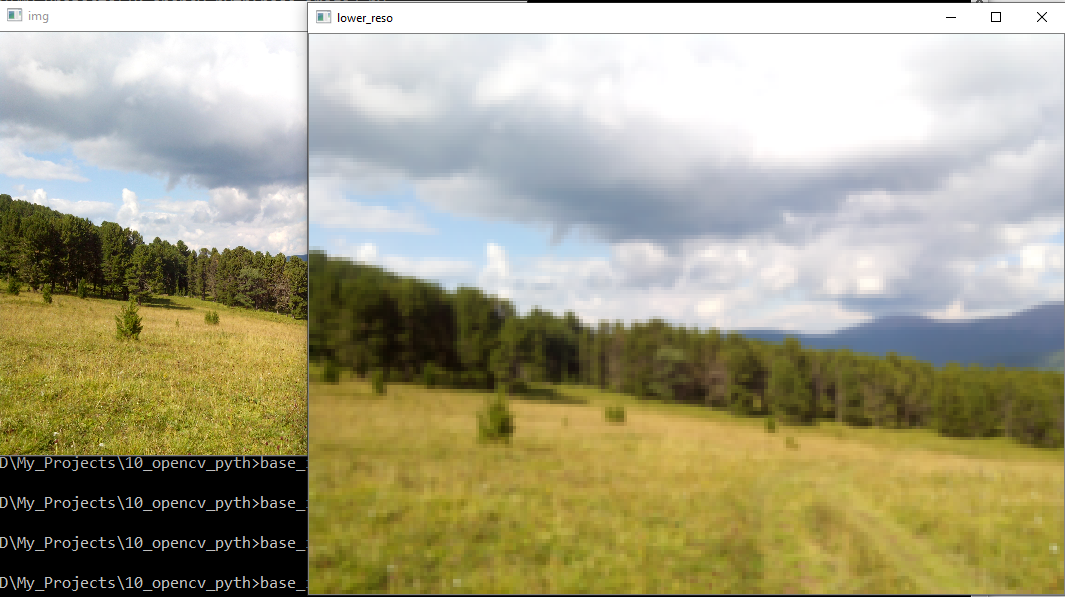
**cv2.imshow('img',img)**

**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.

# Анализ изображений – Гистограммы

*Цель*

*Научитесь этому*

*Находите гистограммы, используя как функции OpenCV, так и Numpy*

*Построение гистограмм с использованием функций OpenCV и Matplotlib*

*Вы увидите следующие функции : cv2.calcHist(), np.histogram() и т. д.*

Теория

Так что же такое гистограмма ? Вы можете рассматривать гистограмму как график или график, который дает вам общее представление о распределении интенсивности изображения. Это график со значениями пикселей (в диапазоне от 0 до 255, не всегда) по оси X и соответствующим количеством пикселей в изображении по оси Y.

Это просто другой способ понимания образа. Глядя на гистограмму изображения, вы получаете интуитивное представление о контрасте, яркости, распределении интенсивности и т. д. (или любых других цветовых характеристиках). Почти все инструменты обработки изображений сегодня предоставляют функции на гистограмме.

Вы можете увидеть изображение и его гистограмму. (Помните, что эта гистограмма рисуется для изображения в оттенках серого, а не для цветного изображения). Левая область гистограммы показывает количество более темных пикселей в изображении, а правая область-количество более ярких пикселей. Из гистограммы вы можете видеть, что темная область больше, чем более яркая область, а количество средних тонов (значения пикселей в среднем диапазоне, скажем, около 127) очень меньше.

## Нахождение гистограмм

Теперь у нас есть представление о том, что такое гистограмма, мы можем посмотреть, как это найти. И OpenCV, и Numpy поставляются со встроенной функцией для этого. Прежде чем использовать эти функции, нам нужно понять некоторые термины, связанные с гистограммами.

BINS. Приведенная выше гистограмма показывает количество пикселей для каждого значения пикселя, то есть от 0 до 255. т. е. вам нужно 256 значений, чтобы показать приведенную выше гистограмму. Но подумайте, что делать, если вам нужно найти не количество пикселей для всех значений пикселей отдельно, а количество пикселей в интервале значений пикселей? скажем, например, вам нужно найти количество пикселей, лежащих между 0 и 15, затем от 16 до 31, ..., от 240 до 255. Для представления гистограммы вам потребуется всего 16 значений. И это то, что показано в примере, приведенном в учебниках OpenCV по гистограммам.

Поэтому вы просто разбиваете всю гистограмму на 16 частей, и значение каждой части - это сумма всех пикселей в ней. При этом каждая подчасть называется “бин” (BINS). В первом случае, количество ячеек, в которых 256 (один для каждого пикселя), а во втором случае, это всего лишь 16. Бункеры представлен термин histSize в OpenCV документы.

DIMS : это количество параметров, для которых мы собираем данные. В этом случае мы собираем данные только об одном-значении интенсивности. Итак, вот он 1.

RANGE: это диапазон значений интенсивности, которые вы хотите измерить. В норме она равна [0,256], то есть всем значениям интенсивности.

**1. Расчет гистограммы в OpenCV**

Итак, теперь мы используем функцию **cv2.calcHist()** для поиска гистограммы. Давайте ознакомимся с функцией и ее параметрами :

**cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])**

images: это исходное изображение типа uint8 или float32. она должна быть дана в квадратных скобках, то есть “[img]”.

channels: он также дан в квадратных скобках. Это индекс канала, для которого мы рассчитываем гистограмму. Например, если входное значение-изображение в оттенках серого, то его значение равно [0]. Для цветного изображения вы можете передать [0], [1] или [2] для вычисления гистограммы синего,зеленого или красного канала соответственно.

mask: Маска. Чтобы найти гистограмму полного изображения, она задается как “нет”. Но если вы хотите найти гистограмму определенной области изображения, вы должны создать для этого изображение маски и дать его в качестве маски. (Я покажу пример позже.)

histSize : это представляет собой количество наших ячеек. Нужно дать в квадратных скобках. Для полномасштабного анализа мы переходим к [256].

ranges: это наш диапазон. Обычно это [0,256].

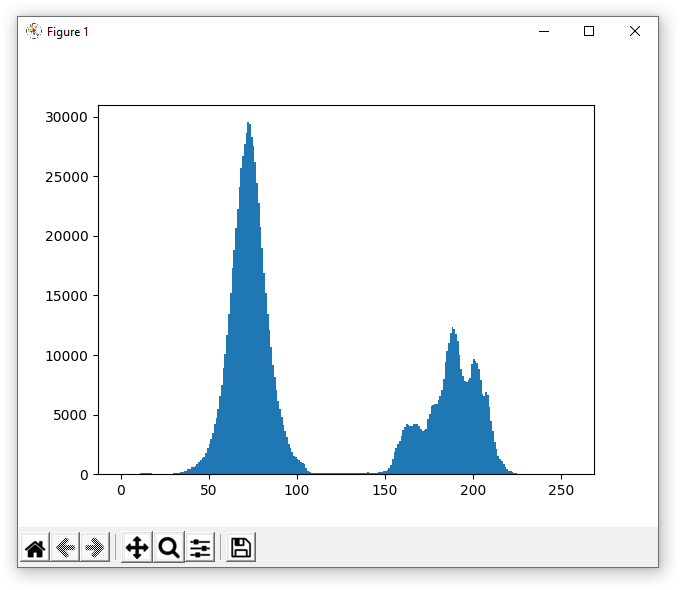
**import cv2**

**import numpy as np**

**from matplotlib import pyplot as plt**

**img = cv2.imread('image\_gray.jpg',0)**

**plt.hist(img.ravel(),256,[0,256]); plt.show()**



**2. вычисление гистограммы в Numpy**

Numpy также предоставляет вам функцию np.histogram(). Поэтому вместо того, чтобы calcHist() функция, вы можете попробовать ниже линии :

**hist,bins = np.histogram(img.ravel(),256,[0,256])**

это то же самое, что мы рассчитывали ранее. Но бункеры будут иметь 257 элементов, потому что Numpy вычисляет бункеры как 0-0.99, 1-1.99, 2-2.99 и т.д. Таким образом, окончательный диапазон будет 255-255.99. Чтобы представить это, они также добавляют 256 в конце бункеров. Но нам это не нужно. До 255 достаточно.

У Numpy есть еще одна функция, **np.bincount ()**, которая намного быстрее (примерно в 10 раз) np.histogram(). Так что для одномерных гистограмм вы можете лучше попробовать это. Не забудьте установить minlength = 256 в np.bincount. Например:

**hist = np.bincount(img.ravel(),minlength=256)**

Функция OpenCV работает быстрее (примерно в 40 раз), чем np.histogram(). Так что придерживайтесь функции OpenCV.

Пример отрисовки гистограммы для цветной Или вы можете использовать обычный график matplotlib, который был бы хорош для графика BGR. Для этого вам нужно сначала найти данные гистограммы.

**import cv2**

**import numpy as np**

**from matplotlib import pyplot as plt**

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

**color = ('b','g','r')**

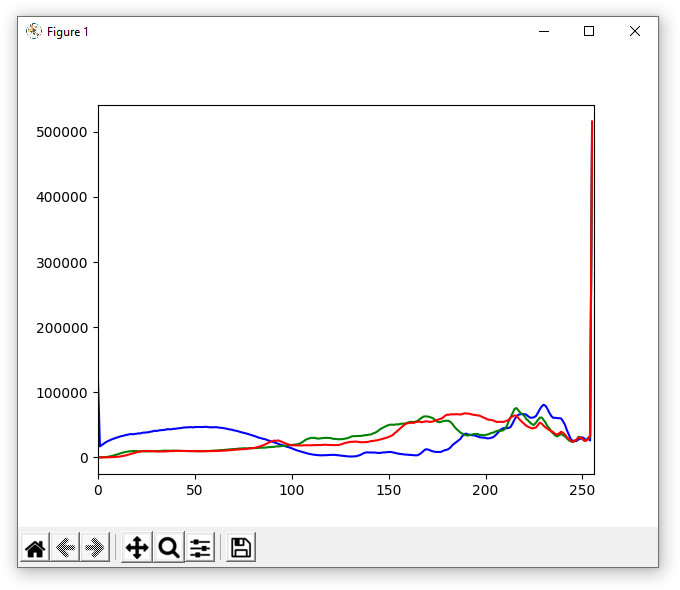
**for i,col in enumerate(color):**

**histr = cv2.calcHist([img],[i],None,[256],[0,256])**

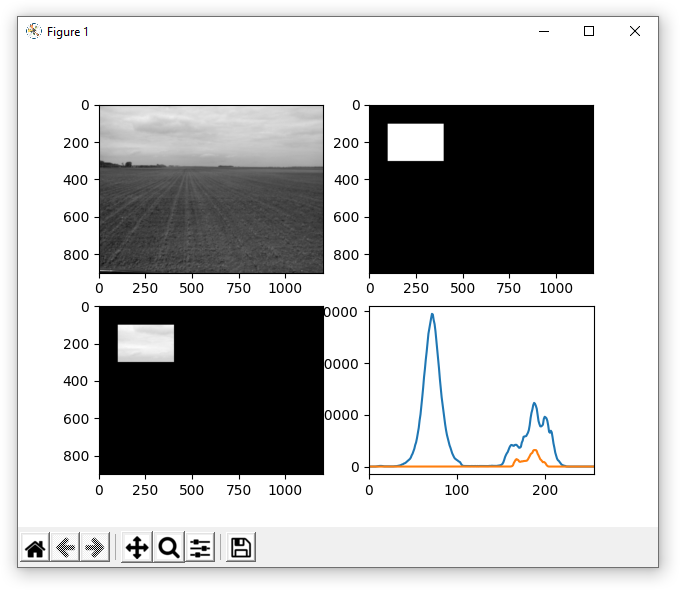
**plt.plot(histr,color = col)**

**plt.xlim([0,256])**

**plt.show()**



Мы использовали **cv2.calcHist ()**, чтобы найти гистограмму полного изображения. Что делать, если вы хотите найти гистограммы некоторых областей изображения? Просто создайте изображение маски с белым цветом на области, которую вы хотите найти гистограмму, и черным в противном случае. Затем передайте это как маску.

**import cv2**

**import numpy as np**

**from matplotlib import pyplot as plt**

**img = cv2.imread('image\_gray.jpg',0)**

**# create a mask**

**mask = np.zeros(img.shape[:2], np.uint8)**

**mask[100:300, 100:400] = 255**

**masked\_img = cv2.bitwise\_and(img,img,mask = mask)**

**# Calculate histogram with mask and without mask**

**# Check third argument for mask**

**hist\_full = cv2.calcHist([img],[0],None,[256],[0,256])**

**hist\_mask = cv2.calcHist([img],[0],mask,[256],[0,256])**

**plt.subplot(221), plt.imshow(img, 'gray')**

**plt.subplot(222), plt.imshow(mask,'gray')**

**plt.subplot(223), plt.imshow(masked\_img, 'gray')**

**plt.subplot(224), plt.plot(hist\_full), plt.plot(hist\_mask)**

**plt.xlim([0,256])**

**plt.show()**

## Гистограммы - 2: Выравнивание Гистограмм

*Цель*

*В этом разделе,*

*Мы изучим концепцию выравнивания гистограмм и используем ее для улучшения контрастности наших изображений.*

Теория

Рассмотрим изображение, значения пикселей которого ограничены только определенным диапазоном значений. Например, более яркое изображение будет иметь все пиксели, ограниченные высокими значениями. Но хорошее изображение будет иметь пиксели из всех областей изображения. Таким образом, вам нужно растянуть эту гистограмму до обоих концов (как показано на рисунке ниже, из Википедии), и именно это делает выравнивание гистограммы (простыми словами). Это обычно улучшает контрастность изображения.

Выравнивание гистограмм в OpenCV

OpenCV имеет функцию для этого, cv2.equalizeHist(). Его вход-это просто изображение в оттенках серого, а выход-это наше уравненное изображение гистограммы.

Ниже приведен простой фрагмент кода, показывающий его использование для того же изображения, которое мы использовали :

**img = cv2.imread('image\_gray.jpg',0)**

**equ = cv2.equalizeHist(img)**

**res = np.hstack((img,equ)) #stacking images side-by-side**

**cv2.imwrite('res.jpg',res)**



Выравнивание гистограммы хорошо, когда гистограмма изображения ограничена определенной областью. Он не будет хорошо работать в тех местах, где есть большие вариации интенсивности, где гистограмма охватывает большую область, то есть присутствуют как яркие, так и темные пиксели. Пожалуйста, проверьте мягкие ссылки в дополнительных ресурсах.

**CLAHE (Contrast Limited Adaptive Histogram Equalization)**

Это правда, что контраст фона улучшился после выравнивания гистограммы. Но сравните лицо статуи на обоих изображениях. Мы потеряли большую часть информации там из-за чрезмерной яркости. Это происходит потому, что его гистограмма не ограничивается определенной областью, как мы видели в предыдущих случаях (попробуйте построить гистограмму входного изображения, вы получите больше интуиции).

Поэтому для решения этой задачи используется адаптивное выравнивание гистограмм. При этом изображение делится на небольшие блоки, называемые “плитками” (размер плитки в OpenCV по умолчанию равен 8x8). Затем каждый из этих блоков гистограммы выравнивается как обычно. Таким образом, в небольшой области гистограмма будет ограничиваться небольшой областью (если только нет шума). Если шум есть, он будет усилен. Чтобы избежать этого, применяется ограничение контрастности. Если какая-либо ячейка гистограммы превышает заданный предел контрастности (по умолчанию 40 в OpenCV), эти пиксели обрезаются и равномерно распределяются по другим ячейкам перед применением выравнивания гистограммы. После выравнивания для удаления артефактов в границах плиток применяется билинейная интерполяция.

Ниже фрагмент кода показывает, как применить CLAHE в OpenCV:



**import numpy as np**

**import cv2**

**img = cv2.imread('image\_gray.jpg',0)**

**# create a CLAHE object (Arguments are optional).**

**clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))**

**cl1 = clahe.apply(img)**

**res = np.hstack((img,cl1)) #stacking images side-by-side**

**cv2.imwrite('clahe\_2.jpg',res)**

## Гистограммы - 3 : 2D гистограммы

*Цель*

*В этой главе мы научимся находить и строить 2D гистограммы. Это будет полезно в следующих главах.*

*Вступление*

*В первой статье мы рассчитали и построили одномерную гистограмму. Он называется одномерным, потому что мы учитываем только одну особенность, то есть значение интенсивности оттенков серого пикселя. Но в двумерных гистограммах вы рассматриваете две особенности. Обычно он используется для поиска цветовых гистограмм, где два объекта являются значениями оттенка и насыщенности каждого пикселя.*

**2D гистограмма в OpenCV**

Он довольно прост и вычисляется с помощью той же функции cv2.calcHist(). Для цветных гистограмм нам нужно преобразовать изображение из BGR в HSV. (Помните, что для 1D гистограммы мы преобразовали BGR в оттенки серого). Для 2D гистограмм его параметры будут изменены следующим образом:

channels = [0,1] because we need to process both H and S plane.

bins = [180,256] 180 for H plane and 256 for S plane.

range = [0,180,0,256] Hue value lies between 0 and 180 & Saturation lies between 0 and 256.

**import cv2**

**import numpy as np**

**img = cv2.imread('home.jpg')**

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

**hist = cv2.calcHist([hsv], [0, 1], None, [180, 256], [0, 180, 0, 256])**

## Гистограмма - 4 : Обратная Проекция Гистограммы

*Цель*

*В этой главе мы познакомимся с обратной проекцией гистограммы.*

Теория

Данный метод был предложен Майклом Дж. Суэйном , Даной Х. Баллард в их работе индексирование с помощью цветных гистограмм.

Что же это такое на самом деле простыми словами? Он используется для сегментации изображений или поиска объектов, представляющих интерес на изображении. Проще говоря, он создает изображение того же размера (но одноканальное), что и наше входное изображение, где каждый пиксель соответствует вероятности принадлежности этого пикселя нашему объекту. В более простых мирах выходное изображение будет иметь наш объект интереса в более белом цвете по сравнению с остальной частью. Ну, это интуитивное объяснение. (Я не могу сделать это проще). Обратное проецирование гистограммы используется с алгоритмом переключения передач и т. д.

Как нам это сделать ? Мы создаем гистограмму изображения, содержащего наш объект интереса (в нашем случае землю, уходящего игрока и другие вещи). Объект должен заполнить изображение настолько, насколько это возможно для лучшего результата. И цветовая гистограмма предпочтительнее гистограммы оттенков серого, потому что цвет объекта-это более лучший способ определить объект, чем его интенсивность в оттенках серого. Затем мы “проецируем” эту гистограмму на наше тестовое изображение, где нам нужно найти объект, то есть другими словами, мы вычисляем вероятность того, что каждый пиксель принадлежит земле, и показываем его. Результирующий выход при правильном пороговом удержании дает нам только почву.

### Алгоритм в Numpy

1. Сначала нам нужно вычислить цветовую гистограмму как объекта, который нам нужно найти (пусть это будет "M"), так и изображения, которое мы собираемся искать (пусть это будет "I").

**import cv2**

**import numpy as np**

**from matplotlib import pyplot as plt**

**#roi is the object or region of object we need to find**

**roi = cv2.imread('rose\_red.png')**

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

**#target is the image we search in**

**target = cv2.imread('rose.png')**

**hsvt = cv2.cvtColor(target,cv2.COLOR\_BGR2HSV)**

**# Find the histograms using calcHist. Can be done with np.histogram2d also**

**M = cv2.calcHist([hsv],[0, 1], None, [180, 256], [0, 180, 0, 256] )**

**I = cv2.calcHist([hsvt],[0, 1], None, [180, 256], [0, 180, 0, 256] )**

2. Найдите соотношение R = \frac{M}{I}. Затем backproject R, то есть использовать R в качестве палитры и создать новое изображение с каждым пикселем в качестве соответствующей вероятности быть целью. то есть B(x,y) = R[h(x,y),s(x,y)], где h-оттенок, а s - насыщенность пикселя в точке (x,y). После этого примените условие

B(x,y) = min[B(x,y), 1].

**B = R[h.ravel(),s.ravel()]**

**B = np.minimum(B,1)**

**B = B.reshape(hsvt.shape[:2])**

3. теперь примените сверток с круговым диском,

B = D \ast B

где D - ядро диска.

**disc = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(5,5))**

**cv2.filter2D(B,-1,disc,B)**

**B = np.uint8(B)**

**cv2.normalize(B,B,0,255,cv2.NORM\_MINMAX)**

4. Теперь местоположение максимальной интенсивности дает нам местоположение объекта. Если мы ожидаем область на изображении, то пороговое значение для подходящего значения дает хороший результат.

**ret,thresh = cv2.threshold(B,50,255,0)**

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)