Лабораторная работа №7

# Преобразование Hough Line

*Цель*

*В этой главе,*

*Мы поймем концепцию преобразования Хафа.*

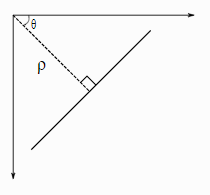
*Мы увидим, как использовать его для обнаружения линий на изображении.*

*Мы увидим следующие функции: cv2.HoughLines(), cv2.HoughLinesP()*

## Преобразование Хафа

Преобразование Хафа-это популярный метод обнаружения любой формы, если вы можете представить эту форму в математической форме. Он может обнаружить форму, даже если она немного нарушена или искажена. Посмотрим, как это работает для линии.

Линия может быть представлена как y = mx+c или в параметрической форме, как \rho = x \cos \theta + y \sin \theta, где \rho-перпендикулярное расстояние от начала координат до линии, а \theta-угол, образованный этой перпендикулярной линией и горизонтальной осью, измеренный против часовой стрелки ( это направление зависит от того, как вы представляете систему координат. Это представление используется в OpenCV). Проверьте изображение ниже:



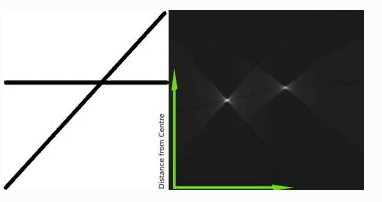
Таким образом, если линия проходит ниже начала координат, то она будет иметь положительный rho и угол меньше 180. Если он идет выше начала координат, то вместо того, чтобы принимать угол больше 180, угол берется меньше 180, а РО принимается отрицательным. Любая вертикальная линия будет иметь 0 градусов, а горизонтальные линии-90 градусов.

Теперь давайте посмотрим, как преобразование Хафа работает для линий. Любая линия может быть представлена в этих двух терминах, (\rho, \theta). Поэтому сначала он создает 2D-массив или аккумулятор (для хранения значений двух параметров), и изначально он устанавливается в 0. Пусть строки обозначают \rho, а столбцы - \theta. Размер массива зависит от необходимой вам точности. Предположим, вы хотите, чтобы точность углов была 1 градус, вам нужно 180 столбцов. Для \rho максимально возможное расстояние-это длина диагонали изображения. Таким образом, принимая точность в один пиксель, количество строк может быть диагональной длиной изображения.

Рассмотрим изображение размером 100х100 с горизонтальной линией посередине. Возьмем первую точку линии. Вы знаете его значения (x,y). Теперь в линейном уравнении поставьте значения \theta = 0,1,2,...., 180 и проверьте \rho,которое вы получите. Для каждой пары (\rho, \theta) вы увеличиваете значение на единицу в нашем аккумуляторе в его соответствующих ячейках (\rho, \theta). Итак, теперь в аккумуляторе ячейка (50,90) = 1 вместе с некоторыми другими ячейками.

Теперь возьмем вторую точку на линии. Сделайте то же самое, что и выше. Увеличьте значения в ячейках, соответствующих (\rho, \theta), которые вы получили. Это время ячейки (50,90) = 2. Что вы на самом деле делать это голосованием (\РО \тета) значений. Вы продолжаете этот процесс для каждой точки на линии. В каждой точке ячейка (50,90) будет увеличена или проголосована, в то время как другие ячейки могут быть или не быть проголосованы. Таким образом, в конце концов, ячейка (50,90) будет иметь максимальное количество голосов. Таким образом, если вы ищете аккумулятор для максимального количества голосов, вы получаете значение (50,90), которое говорит, что на этом изображении есть линия на расстоянии 50 от начала координат и под углом 90 градусов.

Вот как работает преобразование Хафа для линий. Это просто, и, возможно, вы можете реализовать его с помощью Numpy самостоятельно. Ниже приведено изображение, на котором изображен аккумулятор. Яркие пятна в некоторых местах означают, что они являются параметрами возможных линий на изображении.



## Преобразование Хафа в OpenCV

Все, что описано выше, инкапсулировано в функцию OpenCV, **cv2.HoughLines().** Он просто возвращает массив значений (\rho, \theta). \rho измеряется в пикселях, а \theta-в радианах. Первый параметр, входное изображение должно быть двоичным изображением, поэтому примените пороговое значение или используйте осторожное обнаружение ребер, прежде чем найти применение преобразования Хафа. Второй и третий параметры-это точность \rho и \theta соответственно. Четвертый аргумент-это порог, который означает минимальный голос, который он должен получить, чтобы его можно было рассматривать как линию. Помните, что количество голосов зависит от количества точек на линии. Таким образом, он представляет собой минимальную длину линии, которая должна быть обнаружена.

**import cv2**

**import numpy as np**

**img = cv2.imread('lines1.jpg')**

**gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)**

**edges = cv2.Canny(gray,50,150,apertureSize = 3)**

**lines = cv2.HoughLines(edges,1,np.pi/180,200)**

**for rho,theta in lines[10]:**

**a = np.cos(theta)**

**b = np.sin(theta)**

**x0 = a\*rho**

**y0 = b\*rho**

**x1 = int(x0 + 1000\*(-b))**

**y1 = int(y0 + 1000\*(a))**

**x2 = int(x0 - 1000\*(-b))**

**y2 = int(y0 - 1000\*(a))**

**cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)**

**#cv2.imwrite('houghlines3.jpg',img)**

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

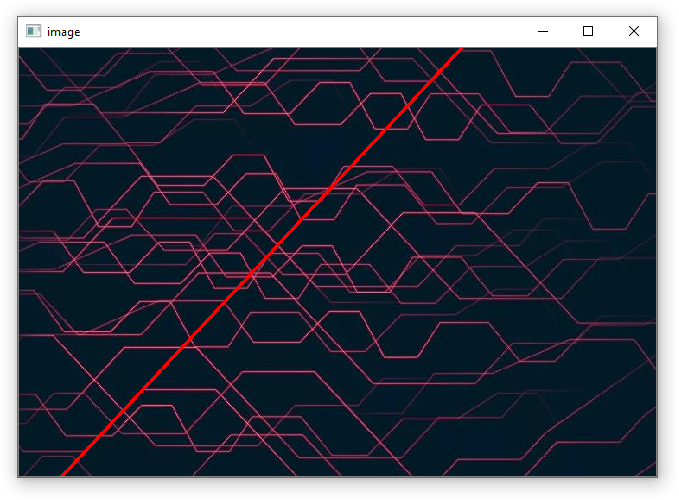
**cv2.imshow('image',img)**

**#cv2.namedWindow("lines", cv2.WINDOW\_NORMAL)**

**#cv2.imshow('lines',lines)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



## Вероятностное Преобразование Хафа

В преобразовании Хафа вы можете видеть, что даже для строки с двумя аргументами требуется много вычислений. Вероятностное преобразование Хафа-это оптимизация преобразования Хафа, которое мы видели. Он не принимает во внимание все точки, А принимает только случайное подмножество точек, и этого достаточно для обнаружения линии. Просто мы должны снизить порог. См. ниже изображение, которое сравнивает преобразование Хафа и вероятностное преобразование Хафа в пространстве Хафа.

Реализация OpenCV основана на надежном обнаружении линий с использованием прогрессивного вероятностного преобразования Хафа Matas, J. and Galambos, C. and Kittler, J. V.. используемая функция-**cv2.HoughLinesP()**. У него есть два новых аргумента.

minLineLength - минимальная длина линии. Отрезки линий короче этого значения отклоняются.

maxLineGap - максимально допустимый зазор между сегментами линии, чтобы рассматривать их как одну линию.

Лучше всего то, что он непосредственно возвращает две конечные точки линий. В предыдущем случае вы получили только параметры линий, и вам нужно было найти все точки. Здесь все прямо и просто.

**import cv2**

**import numpy as np**

**img = cv2.imread('lines1.jpg')**

**gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)**

**edges = cv2.Canny(gray,50,150,apertureSize = 3)**

**minLineLength = 100**

**maxLineGap = 10**

**lines = cv2.HoughLinesP(edges,1,np.pi/180,100,minLineLength,maxLineGap)**

**for line in lines:**

**for x1,y1,x2,y2 in line:**

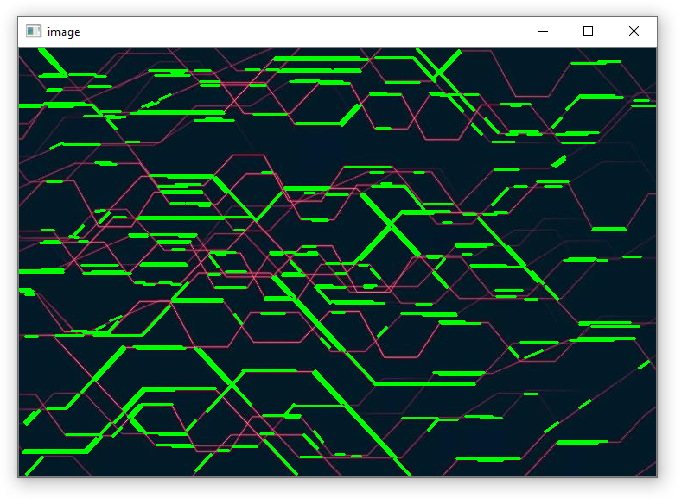
**cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)**

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

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



## Алгоритм Хафа для детектирования окружностей

*Цель*

*В этой главе,*

*Мы научимся использовать преобразование Хафа для нахождения кругов на изображении.*

*Мы увидим эти функции: cv2.HoughCircles()*

Теория

Окружность представляется математически как (x-x\_{center})^2 + (y - y\_{center})^2 = r^2, где (x\_{center},y\_{center}) - центр окружности, а r - радиус окружности. Из уравнения мы видим, что у нас есть 3 параметра, поэтому нам нужен 3D-аккумулятор для преобразования Хафа, что было бы крайне неэффективно. Таким образом, OpenCV использует более сложный метод, метод градиента Хафа, который использует информацию о градиенте ребер.

Функция, которую мы используем здесь, - это **cv2.HoughCircles()**.

**import cv2**

**import numpy as np**

**img = cv2.imread('mons2.png',0)**

**#img = cv2.medianBlur(img,5)**

**ret,img = cv2.threshold(img,240,255,cv2.cv2.THRESH\_TRUNC)**

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

**cv2.waitKey(0)**

**cimg = cv2.cvtColor(img,cv2.COLOR\_GRAY2BGR)**

**circles = cv2.HoughCircles(img,cv2.HOUGH\_GRADIENT,1,20,**

**param1=180,param2=90,minRadius=40,maxRadius=250)**

**circles = np.uint16(np.around(circles))**

**for i in circles[0,:]:**

**# draw the outer circle**

**cv2.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)**

**# draw the center of the circle**

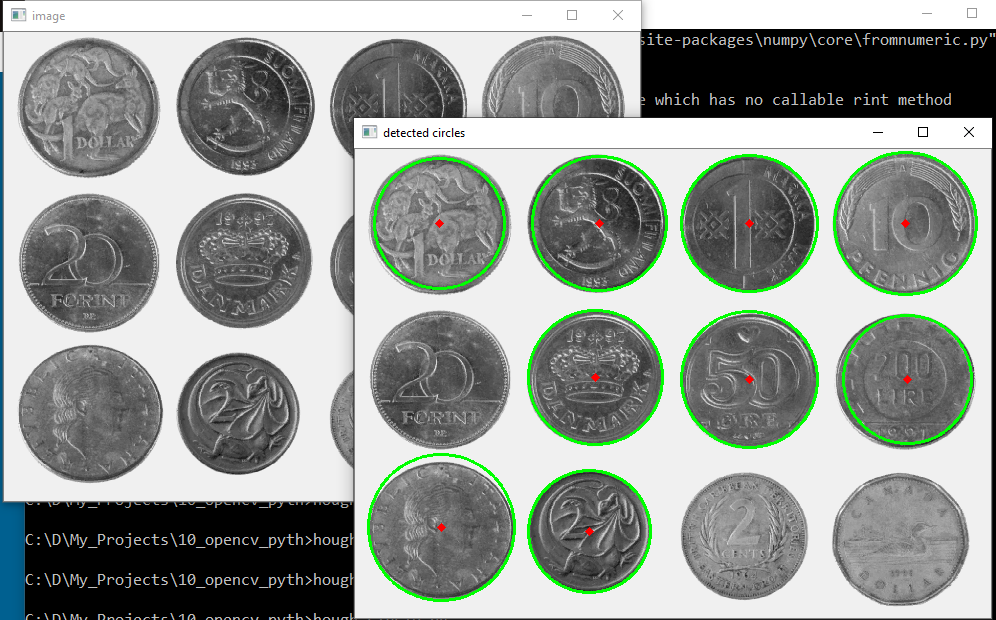
**cv2.circle(cimg,(i[0],i[1]),2,(0,0,255),3)**

**cv2.imshow('detected circles',cimg)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**





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)