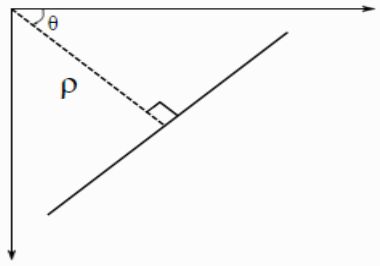
Hough Çizgi Dönüşümü

Teori

Hough Transform, herhangi bir şekli matematiksel formda gösterebiliyorsanız, herhangi bir şekli algılamak için kullanılan popüler bir tekniktir. Biraz kırık veya bozuk olsa bile şekli algılayabilir. Bir hat için nasıl çalıştığını göreceğiz.

Bir çizgi, başlangıçtan çizgiye dikey uzaklık olduğu $y = mx + c$gibi parametrik formda veya bu dikey çizgi ile saat yönünün tersine ölçülen yatay eksenin oluşturduğu açı olarak gösterilebilir (Bu yön, koordinatı nasıl temsil ettiğinize göre değişir. Bu gösterim OpenCV'de kullanılmaktadır). Aşağıdaki resme bakın:$\rho = x \cos\theta + y \sin\theta \rho\theta$

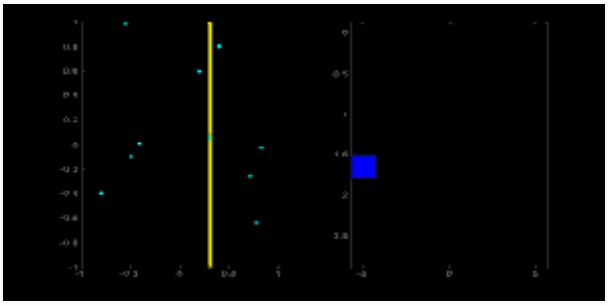


Yani çizgi orijinin altından geçiyorsa, pozitif bir rho ve 180'den küçük bir açıya sahip olacaktır. Orijinin üzerine giderse, 180'den büyük açı almak yerine açı 180'den küçük alınır ve rho negatif alınır. Herhangi bir dikey çizgi 0 dereceye ve yatay çizgiler 90 dereceye sahip olacaktır.

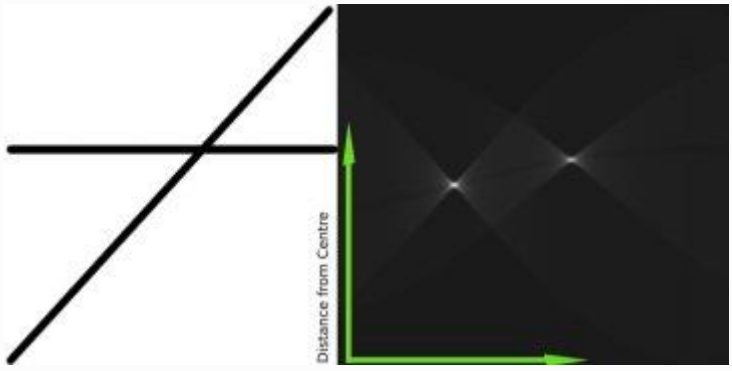
Şimdi Hough Transform'un çizgiler için nasıl çalıştığını görelim. Bu iki terimle herhangi bir satır gösterilebilir (ρ, θ). Bu yüzden önce bir 2D dizi veya akümülatör oluşturur (iki parametrenin değerlerini tutmak için) ve başlangıçta 0'a ayarlanır. Satırların, ρ sütunların θ. Dizinin boyutu, ihtiyacınız olan doğruluğa bağlıdır. Açıların doğruluğunun 1 derece olmasını istediğinizi varsayalım, 180 sütuna ihtiyacınız var. Çünkü ρ, mümkün olan maksimum mesafe, görüntünün çapraz uzunluğudur. Dolayısıyla, bir piksel doğruluğu alındığında, satır sayısı görüntünün çapraz uzunluğu olabilir.

Ortasında yatay bir çizgi bulunan 100x100 boyutunda bir resim düşünün. Çizginin ilk noktasını alın. (X, y) değerlerini biliyorsunuz. Şimdi çizgi denkleminde, değerleri koyun $\theta = 0, 1, 2, ..., 180$ve ρelde ettiğinizi kontrol edin . Her (ρ, θ)çift için, ilgili (ρ, θ) hücrelerdeki akümülatörümüzde değeri bir artırırsınız. Şimdi akümülatörde, hücre (50,90) = 1, diğer bazı hücrelerle birlikte.

Şimdi çizgideki ikinci noktayı alın. Yukarıdakinin aynısını yapın. Sahip olduğunuza karşılık gelen hücrelerdeki değerleri artırın (ρ, θ). Bu sefer, hücre (50,90) = 2. Gerçekte yaptığınız şey (ρ, θ)değerleri oylamaktır . Çizgideki her nokta için bu işleme devam ediyorsunuz. Her noktada, hücre (50,90) artırılır veya oylanırken, diğer hücreler oylanabilir veya verilmeyebilir. Bu şekilde, sonunda, hücre (50,90) maksimum oy hakkına sahip olacaktır. Dolayısıyla, maksimum oy için toplayıcıyı ararsanız, bu görüntüde orijinden 50 uzaklıkta ve 90 derece açıda bir çizgi olduğunu söyleyen (50,90) değerini elde edersiniz. Aşağıdaki animasyonda iyi gösterilmiştir.



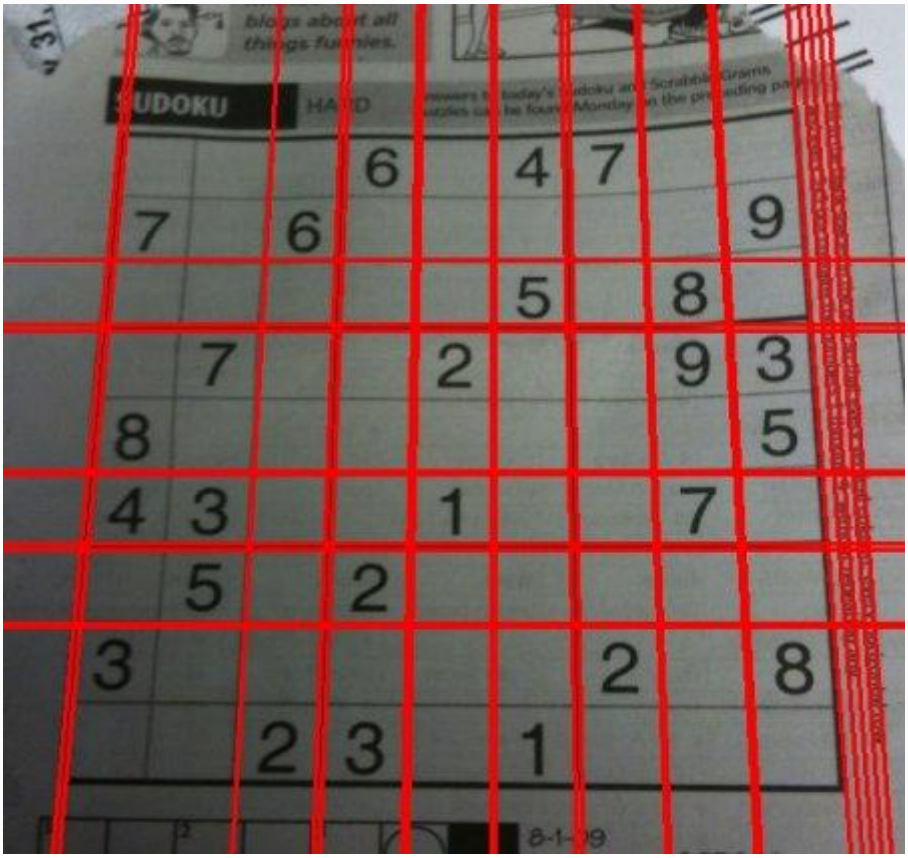
Hatlar için dönüşüm bu şekilde çalışır. Basittir ve Numpy kullanarak kendi başınıza uygulayabilirsiniz. Aşağıda akümülatörü gösteren bir resim bulunmaktadır. Bazı konumlardaki parlak noktalar, bunların görüntüdeki olası çizgilerin parametreleri olduğunu gösterir.



OpenCV'de Hough Tranform

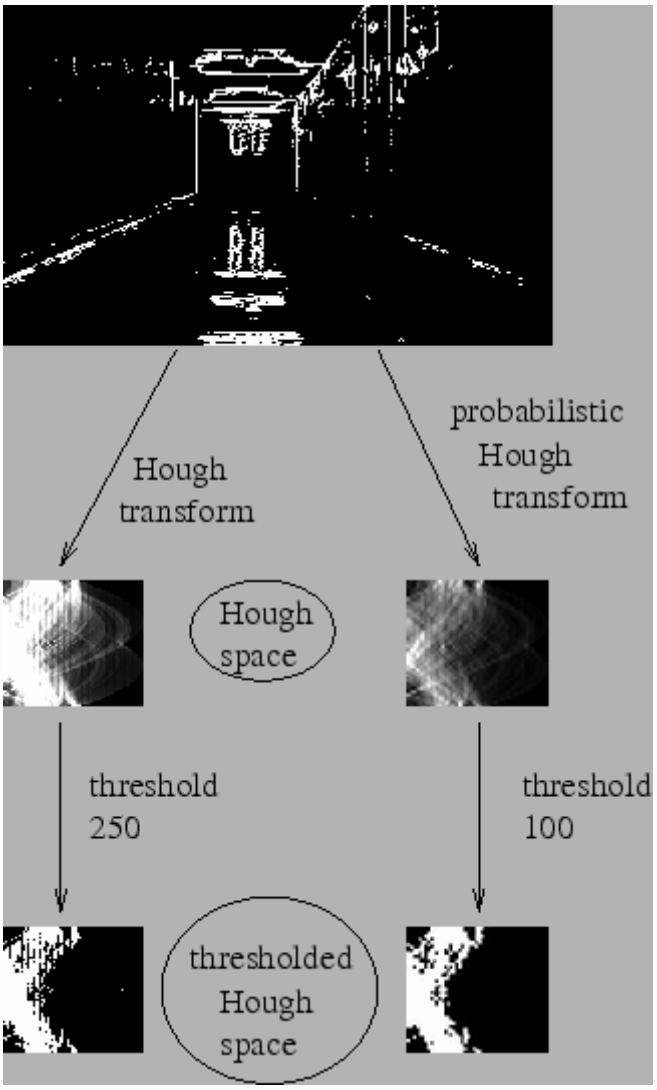
Yukarıda açıklanan her şey OpenCV işlevi, **cv2.HoughLines () içinde özetlenmiştir** . Basitçe bir dizi (ρ, θ)değer döndürür . ρpiksel θcinsinden ölçülür ve radyan cinsinden ölçülür. İlk parametre olan Girdi görüntüsü bir ikili görüntü olmalıdır, bu nedenle eşik uygulayın veya hough dönüşümü uygulamayı bulmadan önce canny edge algılamayı kullanın. İkinci ve üçüncü parametreler sırasıyla ρve θ doğruluklardır. Dördüncü argüman *eşiktir* , bu da bir çizgi olarak kabul edilmesi için alması gereken asgari oy anlamına gelir. Unutmayın, oy sayısı hattaki puan sayısına bağlıdır. Bu nedenle, tespit edilmesi gereken minimum çizgi uzunluğunu temsil eder.

```
import cv2
import numpy as np

img = cv2.imread('dave.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[0]:
    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)
```



Olasılıksal Hough Dönüşümü

Hough dönüşümünde, iki bağımsız değişkenli bir satır için bile çok fazla hesaplama gerektirdiğini görebilirsiniz. Olasılıksal Hough Dönüşümü, gördüğümüz Hough Dönüşümünün bir optimizasyonudur. Tüm noktaları hesaba katmaz, bunun yerine yalnızca rastgele bir nokta alt kümesini alır ve bu, hat tespiti için yeterlidir. Sadece eşiği düşürmeliyiz. Hough uzayında Hough Dönüşümü ve Olasılıksal Hough Dönüşümü karşılaştıran aşağıdaki resme bakın.

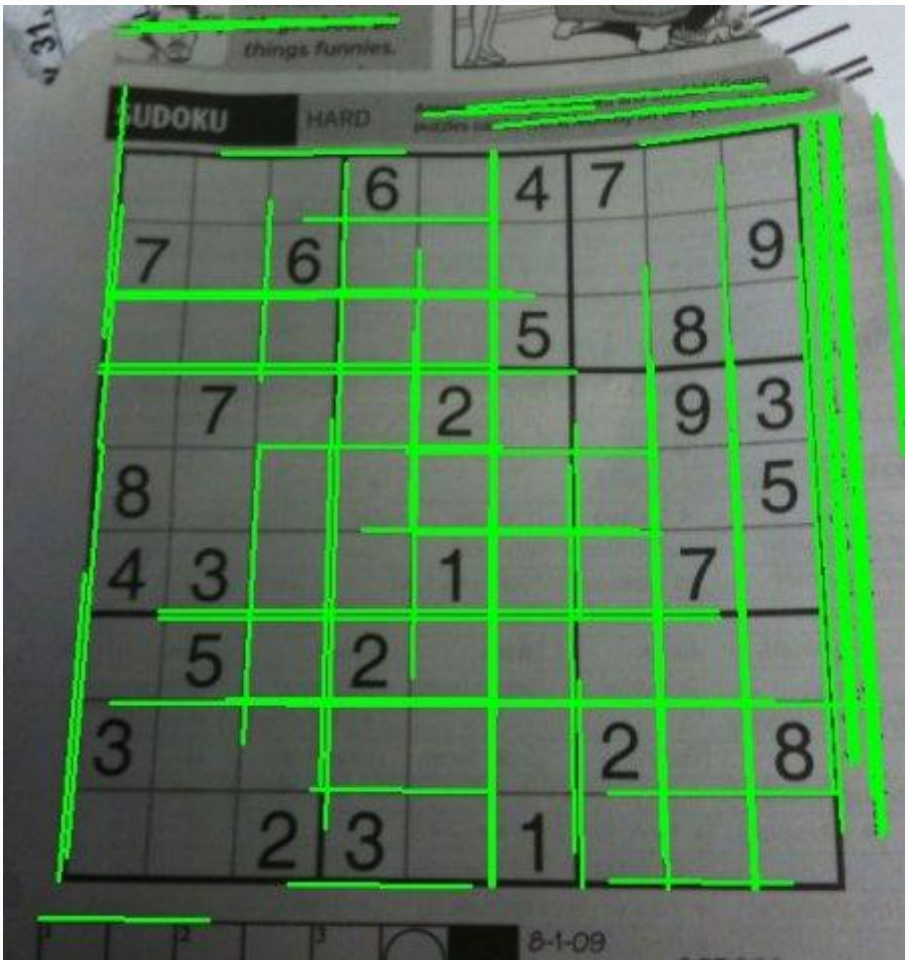


OpenCV uygulaması, Matas, J. ve Galambos, C. ve Kittler, JV tarafından Aşamalı Olasılıklı Hough Dönüşümü Kullanılarak Hatların Sağlam Algılamasına dayanmaktadır. Kullanılan işlev **cv2.HoughLinesP ()** 'dir . İki yeni argümanı var.

- **minLineLength** - Minimum satır uzunluğu. Bundan daha kısa çizgi segmentleri reddedilir.
- **maxLineGap** - Tek satır olarak ele almak için çizgi segmentleri arasında izin verilen maksimum boşluk.

En iyisi, çizgilerin iki uç noktasını doğrudan döndürmesidir. Önceki durumda, yalnızca çizgilerin parametrelerini aldınız ve tüm noktaları bulmanız gerekiyordu. Burada her şey doğrudan ve basittir.

```
import cv2
import numpy as np

img = cv2.imread('dave.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 x1,y1,x2,y2 in lines[0]:
    cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)

cv2.imwrite('houghlines5.jpg',img)
```



Hough Circle Dönüşümü

Bu bölümde,

- Bir görüntüdeki çemberleri bulmak için Hough Transform'u kullanmayı öğreneceğiz.
- Şu işlevleri göreceğiz: **cv2.HoughCircles ()**

Teori

Bir daire, matematiksel olarak , dairenin merkezinin $(x - x_{center})^2 + (y - y_{center})^2 = r^2$nerede (x_{center}, y_{center})olduğu ve dairenin $r$yarıçapı olarak temsil edilir . Denklemden, 3 parametremiz olduğunu görebiliriz, bu yüzden hough dönüşümü için oldukça etkisiz olan bir 3B akümülatöre ihtiyacımız var. Bu nedenle OpenCV , kenarların gradyan bilgisini kullanan **Hough Gradient Metodu** olan daha karmaşık bir metot kullanır.

Burada kullandığımız işlev **cv2.HoughCircles ()** . Dokümantasyonda iyi açıklanan birçok argümana sahiptir. Yani doğrudan koda gidiyoruz.

```
import cv2
import numpy as np

img = cv2.imread('opencv_logo.png',0)
img = cv2.medianBlur(img,5)
cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)

circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,20,
                           param1=50,param2=30,minRadius=0,maxRadius=0)

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()
```