Universidad Nacional de San Antonio Abad del Cusco
Departamento Académico de Ing. Informática
VISION COMPUTACIONAL
Práctica Nº 09

DETECCIÓN DE LINEAS Y CÍRCULOS CON LA TRANSFORMADA DE HOUGH

Iván C. Medrano Valencia

1. OBJETIVO.

- Conocer cómo detectar líneas y círculos con la transformada de Hough en OpenCV

2. CONCEPTOS TEORICOS

2.1. INTRODUCCIÓN.

La transformación Hough es un método de extracción de características para detectar formas simples como círculos, líneas, etc. en una imagen.

Una forma "simple" es aquella que se puede representar con solo unos pocos parámetros. Por ejemplo, una línea se puede representar mediante dos parámetros (pendiente, intersección) y un círculo tiene tres parámetros: las coordenadas del centro y el radio (x, y, r). **Hough transform** hace un excelente trabajo al encontrar tales formas en una imagen.

2.2. LA TRANSFORMADA DE HOUGH

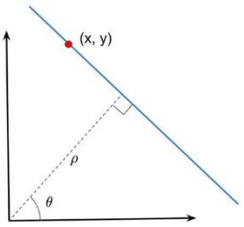
Hough Transform es una técnica para detectar cualquier forma, si se puede representar en forma matemática. Puede detectar la forma incluso si está un poco rota o distorsionada. Veremos cómo funciona para una línea.

Una línea se puede representar como:

$$y = mx + c$$

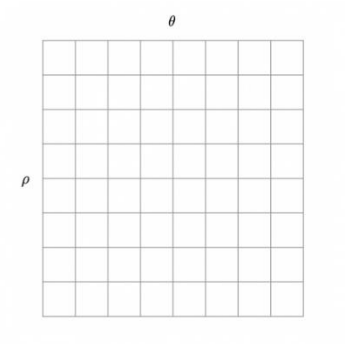
o en forma paramétrica, como:

$\rho = x \cos\theta + y \sin\theta$ donde ρ es la distancia perpendicular desde el origen a la línea, y θ es el ángulo formado por esta línea perpendicular y el eje horizontal medido en sentido antihorario (Esa dirección varía según la forma en que representa el sistema de coordenadas. Esta representación se usa en OpenCV). Como se muestra en la siguiente imagen:



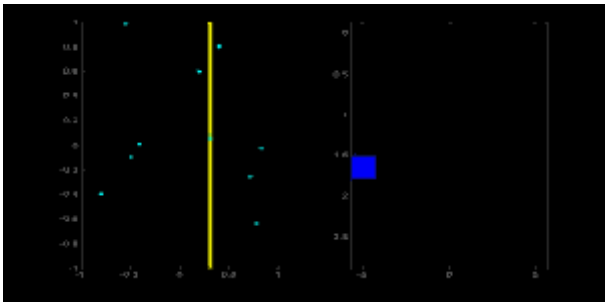
Entonces, si la línea pasa por debajo del origen, tendrá un ρ positivo y un ángulo menor de **180°**. Si va por encima del origen, en lugar de tomar un ángulo mayor que 180°, el ángulo se toma menos de 180° y el ρ es negativo. Cualquier línea vertical tendrá 0° grados y las líneas horizontales 90° grados.

Ahora veamos cómo funciona Hough Transform para líneas. Cualquier línea se puede representar en estos dos términos, (ρ, θ). Entonces, primero crea una matriz o acumulador 2D (para mantener los valores de dos parámetros) y se establece en 0 inicialmente. Las filas denotan ρ y las columnas denotan θ. El tamaño de la matriz depende de la precisión que necesite. Suponga que desea que la precisión de los ángulos sea de 1 grado, necesita 180 columnas. Para ρ, la distancia máxima posible es la longitud diagonal de la imagen. Entonces, tomando una precisión de un píxel, el número de filas puede ser la longitud diagonal de la imagen.

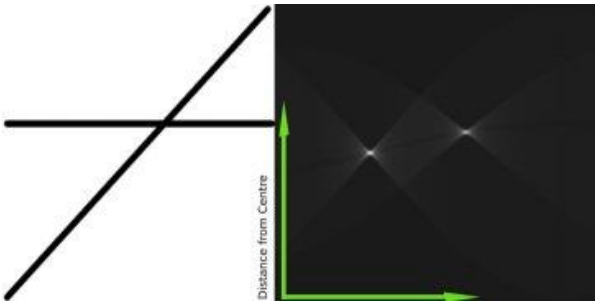


Considere una imagen de 100x100 con una línea horizontal en el medio. Toma el primer punto de la línea. Conocemos sus valores (x, y). Ahora, en la ecuación lineal, coloque los valores θ = 0, 1, 2, ... ,180 y verifique el ρ que obtiene. Para cada par (ρ, θ), incrementas el valor en uno en nuestro acumulador en sus celdas correspondientes (ρ, θ). Entonces ahora en el acumulador, la celda (50,90) = 1 junto con algunas otras celdas.

Ahora toma el segundo punto de la línea. Haz lo mismo que arriba. Incrementa los valores en las celdas correspondientes a (ρ, θ)que obtuviste. Esta vez, la celda (50,90) = 2. Lo que realmente hacemos es votar los valores (ρ, θ). Continúa este proceso para cada punto de la línea. En cada punto, la celda (50,90) se incrementará o votará a favor, mientras que otras celdas pueden o no votar a favor. De esta forma, al final, la celda (50,90) tendrá el máximo de votos. Entonces, si busca en el acumulador el máximo de votos, obtiene el valor (50,90) que dice, hay una línea en esta imagen a una distancia de 50 del origen y en un ángulo de 90 grados. Se muestra bien en la siguiente animación

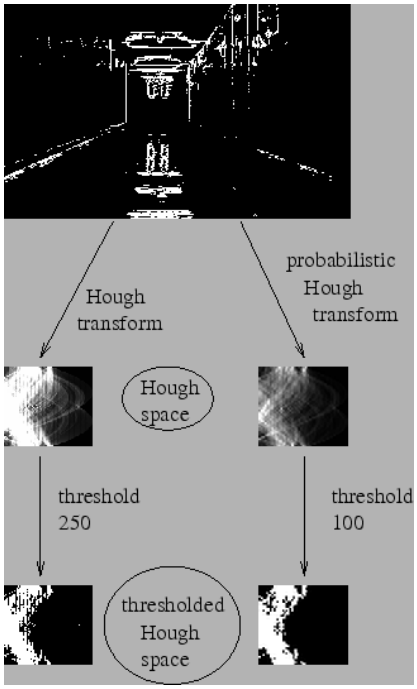


Así es como funciona la transformación de hough para líneas. A continuación, se muestra una imagen que muestra el acumulador. Los puntos brillantes en algunas ubicaciones denotan que son los parámetros de posibles líneas en la imagen.



2.3. TRANSFORMADA PROBABILÍSTICA DE HOUGH

En la transformación hough, puede ver que incluso para una línea con dos argumentos, se necesita mucho cálculo. La transformada probabilística de Hough es una optimización de la transformada de Hough que vimos. No toma en consideración todos los puntos, sino que solo toma un subconjunto aleatorio de puntos y eso es suficiente para la detección de líneas. Solo tenemos que disminuir el umbral. Vea la imagen a continuación que compara la Transformada de Hough y la Transformada de Hough probabilística en un espacio reducido.



2. ACTIVIDAD I: DETECTAR LINEAS EN UNA IMAGEN

en OpenCV, la detección de líneas mediante Hough Transform se implementa en la función **HoughLines** y **HoughLinesP** (Probabilistic Hough Transform). Esta función toma los siguientes argumentos:

- edges: Salida del detector de bordes.
- lines: Un vector para almacenar las coordenadas del inicio y el final de la línea.
- rho: el parámetro de resolución ρ en píxeles.
- theta: La resolución del parámetro θ en radianes.
- threshold: el número mínimo de puntos de intersección para detectar una línea.

Ejemplo 9.1 Detectar líneas


```
import cv2
import numpy as np
img = cv2.imread('images/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[5]:
    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.imshow('houghlines’,img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Ejemplo 9.1. Detectar las líneas en la siguiente imagen:

```
import cv2
import numpy as np
import sys

def onTrackbarChange(max_slider):
    global img
    global dst
    global gray
```

```
    dst = np.copy(img)

    th1 = max_slider
    th2 = th1 * 0.4
    edges = cv2.Canny(img, th1, th2)

    #Aplicar la transformada probabilística de hough para detectar líneas
    lines = cv2.HoughLinesP(edges, 2, np.pi/180.0, 50, minLineLength=10,
maxLineGap=100)

    #--dibujar líneas sobre los puntos detectados
    for line in lines:
        x1, y1, x2, y2 = line[0]
        cv2.line(dst, (x1, y1), (x2, y2), (0,0,255), 1)

    cv2.imshow("Imagen Resultante", dst)
    cv2.imshow("Edges",edges)

if __name__ == "__main__":

    #--Leer la imagen
    img = cv2.imread('images/lanes.jpg')

    #--Crear una copia para su posterior uso
    dst = np.copy(img)

    #--Convertir a gris
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Crear las ventanas
    cv2.namedWindow("Edges")
    cv2.namedWindow("Imagen Resultante")


    # Inicializar el valor del umbral
    initThresh = 500

    # Maximo valor del umbral
    maxThresh = 1000
```

```
    cv2.createTrackbar("threshold", "Imagen Resultante", initThresh, maxT
hresh, onTrackbarChange)
    onTrackbarChange(initThresh)

    while True:
        key = cv2.waitKey(1)
        if key == 27:
            break

    cv2.destroyAllWindows()
```

ACTIVIDAD II. DETECTAR CÍRCULOS EN UNA IMAGEN

En OpenCV la función HoughCircles se utiliza para detectar los círculos en una imagen.

Toma los siguientes parámetros:

Image : la imagen de entrada.

Method : método de detección.

Dp : la relación inversa entre la resolución del acumulador y la resolución de la imagen.

Mindst : distancia mínima entre centros de círculos detectados.

param_1 y

param_2 : estos son parámetros específicos del método.

min_Radius: radio mínimo del círculo a detectar.

max_Radius: radio máximo a detectar.

Ejemplo 2. Detectar círculos en la siguiente imagen

```
import cv2
import numpy as np
```

```
import sys

def onTrackbarChange(max_slider):
    cimg = np.copy(img)

    p1 = max_slider
    p2 = max_slider * 0.4

    #--
Detectar circulos usando la transformada de Hough para detectar círculos
    circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, cimg.shape[0]
/64, param1=p1, param2=p2, minRadius=25, maxRadius=50)

    #--Si por lo menos se ha detectado un círculo
    if circles is not None:
        cir_len = circles.shape[1] # almacenar la longitud de los círculo
s encontrados
        circles = np.uint16(np.around(circles))
        for i in circles[0, :]:
            # Dibuja el círculo exterior
            cv2.circle(cimg, (i[0], i[1]), i[2], (0, 255, 0), 2)
            # Dibuja el centro del círculo
            cv2.circle(cimg, (i[0], i[1]), 2, (0, 0, 255), 3)
    else:
        cir_len = 0 # no se detectaron círculos

    # Mostrar imagen de salida
    cv2.imshow('Image', cimg)

    # Imagen de borde para depuración
    edges = cv2.Canny(gray, p1, p2)
    cv2.imshow('Edges', edges)

if __name__ == "__main__":
    # Leer imagen
    img = cv2.imread("images/brown-eyes.jpg", 1)

    # Convertir a escala de grises
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Crear ventanas de visualización
    cv2.namedWindow("Edges")
```

```
    cv2.namedWindow("Image")
        # La barra de seguimiento se utilizará para cambiar el umbral del
 borde
    initThresh = 105
    maxThresh = 200

    # Crear barra de seguimiento
    cv2.createTrackbar("Threshold", "Image", initThresh, maxThresh, onTra
ckbarChange)
    onTrackbarChange(initThresh)

    while True:
        key = cv2.waitKey(1)
        if key == 27:
            break
    cv2.destroyAllWindows()
```

3. REFERENCIAS BIBLIOGRÁFICAS

- Dawson-Howe, K. (2014). A Practical Introduction to Computer Vision With OpenCV, Wiley & Sons Ltd.
- Hafsa Asad, W. R., Nikhil Singh (2020). The Computer Vision Workshop. Birmingham U.K., Pack Publishing.
- Szeliski, R. (2011). Computer Vision Algorithms and Applications. London, Springer.