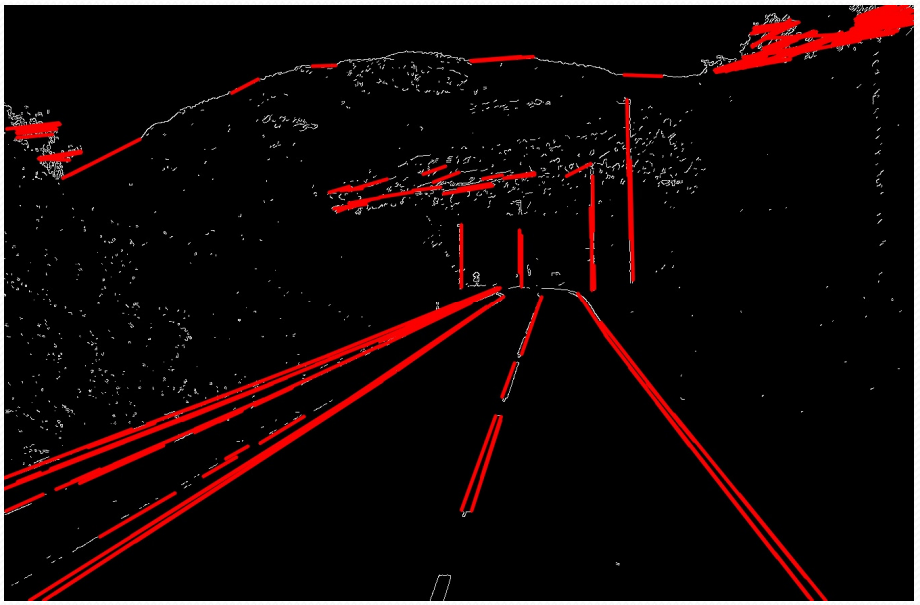
Detección de líneas y círculos

Temas

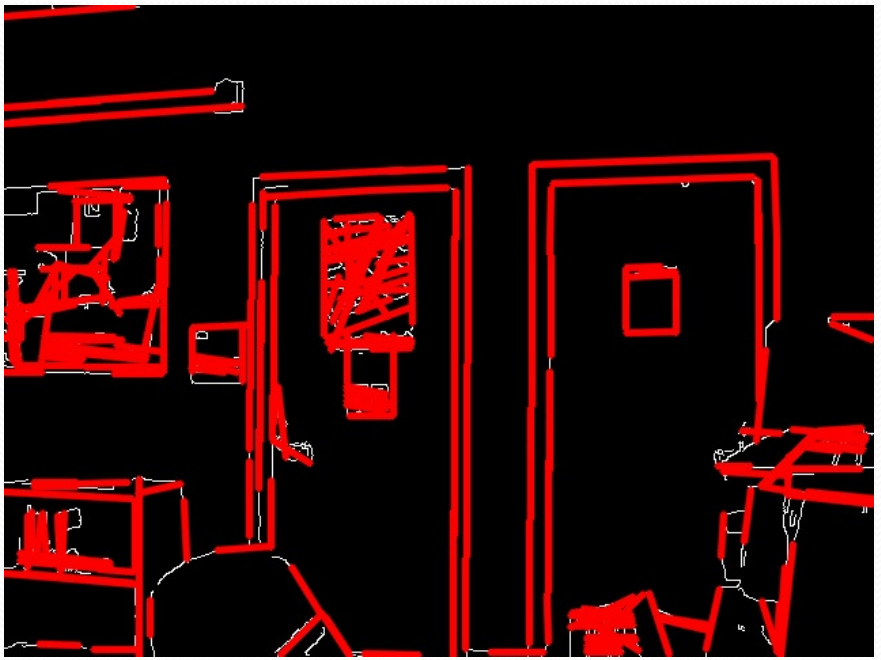
- Detección de líneas rectas.
- Detección de círculos.

Detección de líneas rectas



Fuente: By Piotr Małecki - Friend of mine, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1160928

Detección de líneas rectas



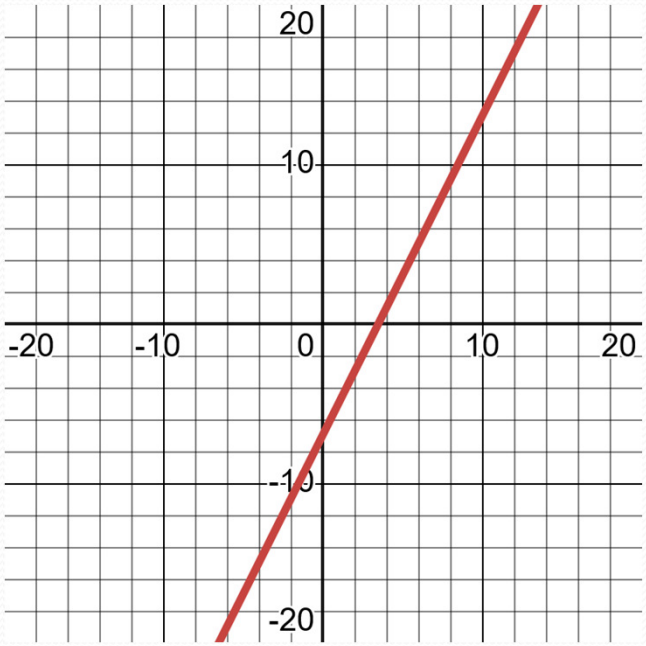
Fuente: https://cs.gmu.edu/~zduric/cs682/Images/

¿Por qué es importante?

- Muchos objetos están delimitados por líneas.
- La detección de líneas permite:
- Detectar carreteras, carriles, texto.
- Rastreo de objetos.
- Reconocimiento de objetos.
- Análisis y entendimiento de la escena.

Representación de líneas rectas

- Pendiente – ordenada al origen: $y = mx + b$
- Ejemplo: $y = 2x - 7$



Fuente: https://www.desmos.com/calculator

Problema: líneas verticales

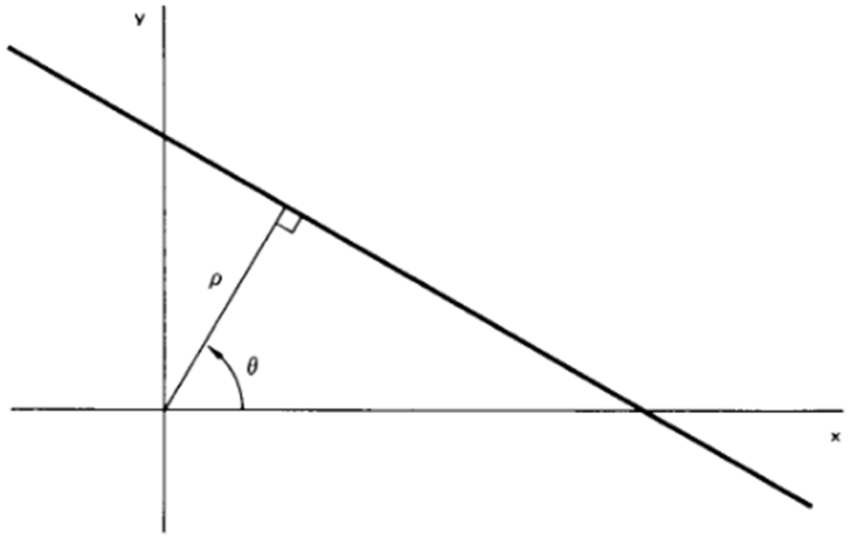
- Dados $p1 = (7, 10)$ y $p2 = (7, 15)$ encontrar la línea recta que los une.
- Se calcula la pendiente:
- $m = \frac{y2-y1}{x2-x1} = \frac{15-10}{7-7} = \frac{5}{0} =$ **no definido**
- Hay que buscar alternativas.

Forma normal de la recta

- También llamada forma normal de Hesse.
- $x \cdot \cos\phi + y \cdot \sin\phi = \rho$
- Donde:
- ρ es la distancia mínima del origen a la recta.
- ϕ es el ángulo que forma el eje x con línea perpendicular entre el origen y la recta.

Forma normal de la recta

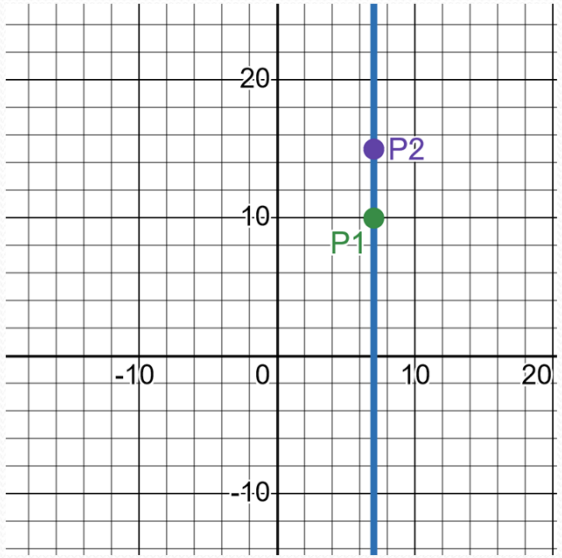
Fig. 1. The normal parameters for a line.



Fuente: https://dl.acm.org/doi/pdf/10.1145/361237.361242

Ejemplo

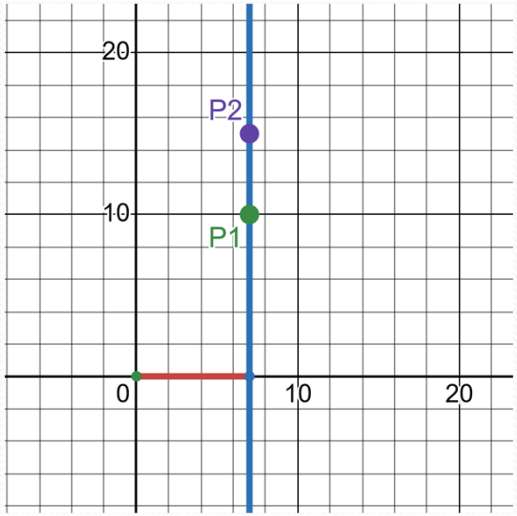
- Para la recta que pasa por $p1 = (7, 10)$ y $p2 = (7, 15)$.



Fuente: https://www.desmos.com/calculator

Ejemplo

- La distancia mínima entre el origen y la recta es $\rho = 7$.
- El ángulo de la perpendicular es $\phi = 0$.



Fuente: https://www.desmos.com/calculator

Ejemplo

- Ecuación de la recta vertical en forma normal:
- $x \cdot \cos\phi + y \cdot \sin\phi = \rho$
- $x \cdot \cos 0 + y \cdot \sin 0 = 7$
- $x = 7$

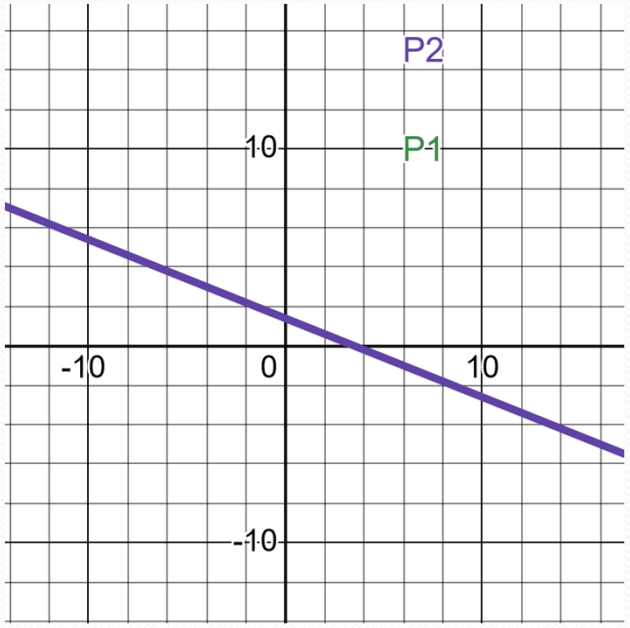
Espacio de Hough

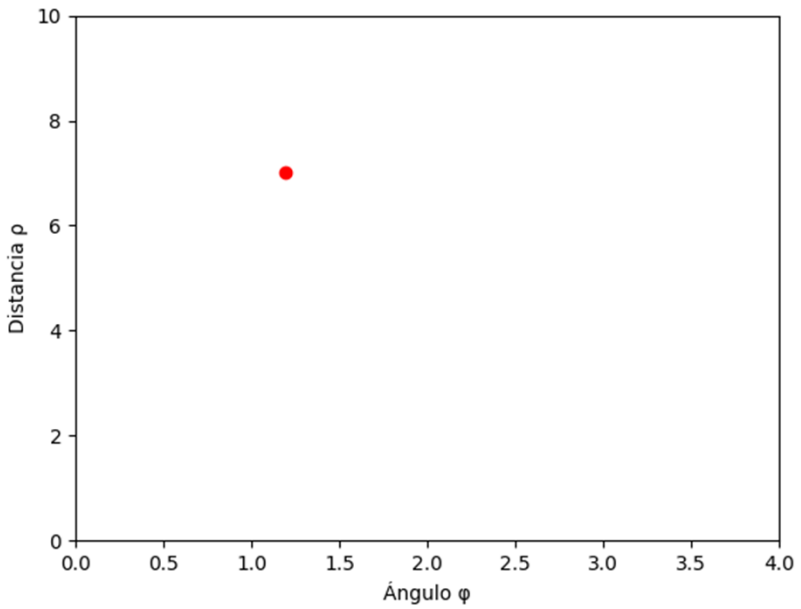
- Permite representar una línea recta como un punto con coordenadas φ, ρ.
- Por ejemplo, la recta $2x + 5y = 7$, se puede escribir en forma normal como $x \cdot \cos(1.19) + y \cdot \sin(1.19) = \frac{7}{\sqrt{29}}$.
- Donde $\varphi = 1.19$ está en radianes y equivale, aproximadamente, a 68.18 grados.

Espacio de Hough

$2x + 5y = 7$

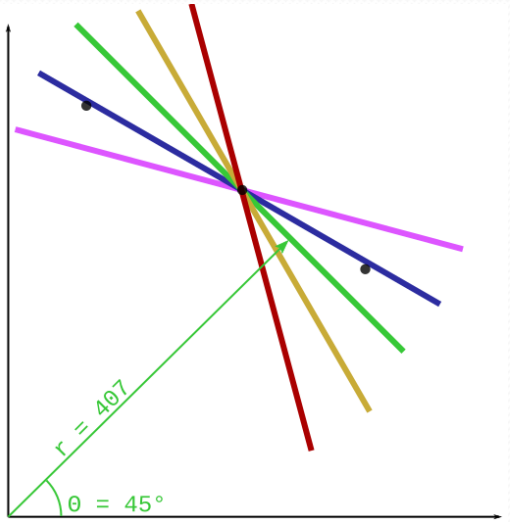
$(1.19, 7)$





Espacio de Hough

- Considerar todas las líneas rectas que pasan por un punto en el espacio cartesiano.



Fuente: https://commons.wikimedia.org/wiki/File:Hough_transform_diagram.svg

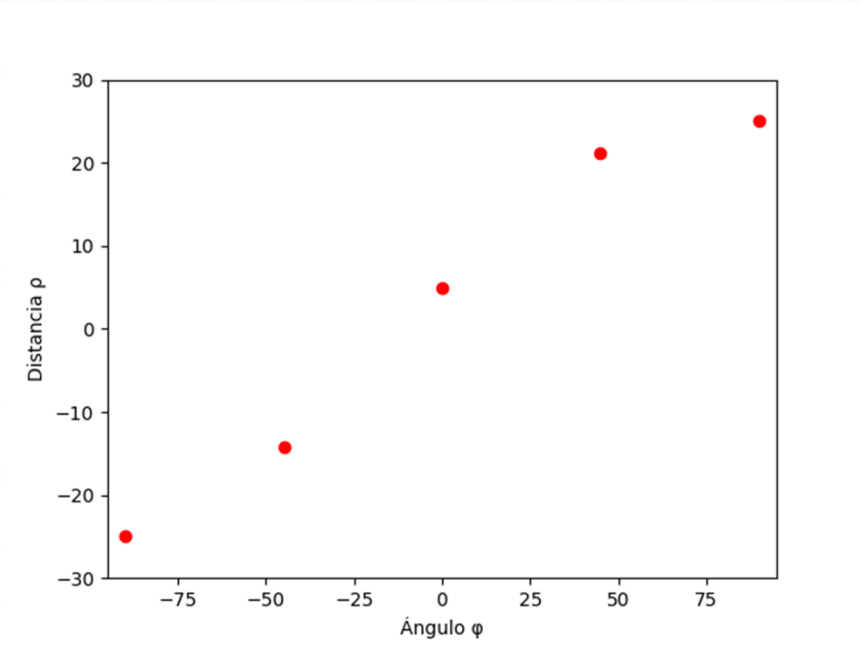
Espacio de Hough

- Ejemplo: dado el punto $(5, 25)$, obtener las rectas con $\phi = -90°, -45°, 0°, 45°, 90°$.

ϕ	ρ
$5 \cdot \cos(-90) + 25 \cdot \sin(-90)$	-25.00
$5 \cdot \cos(-45) + 25 \cdot \sin(-45)$	-14.14
$5 \cdot \cos(0) + 25 \cdot \sin(0)$	5.00
$5 \cdot \cos(45) + 25 \cdot \sin(45)$	21.21
$5 \cdot \cos(90) + 25 \cdot \sin(90)$	25.00

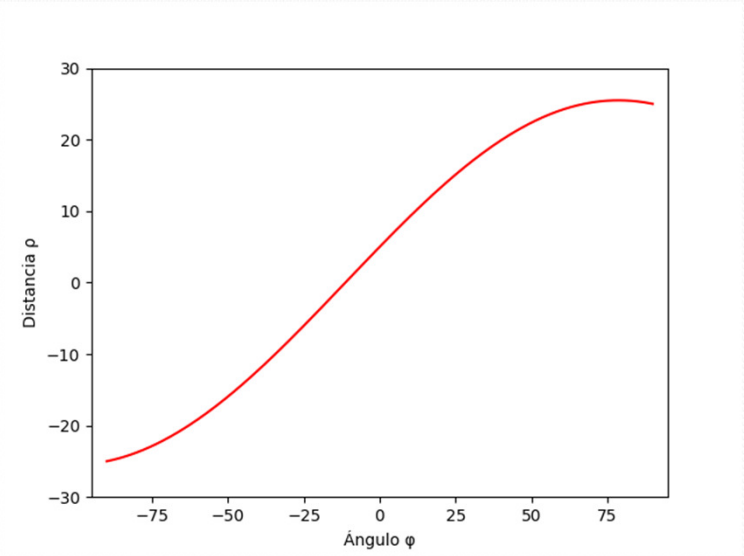
Espacio de Hough

- Al graficar los puntos se nota una curva sinusoidal.



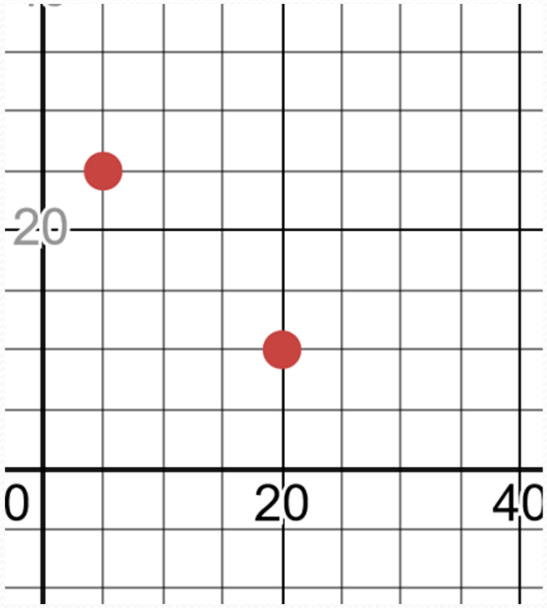
Espacio de Hough

- La curva se nota mejor si se obtienen todas las líneas rectas, que pasan por $(5, 25)$, para $\phi = -90, -89, \ldots, -1, 0, 1, \ldots, 89, 90$ grados.



Espacio de Hough

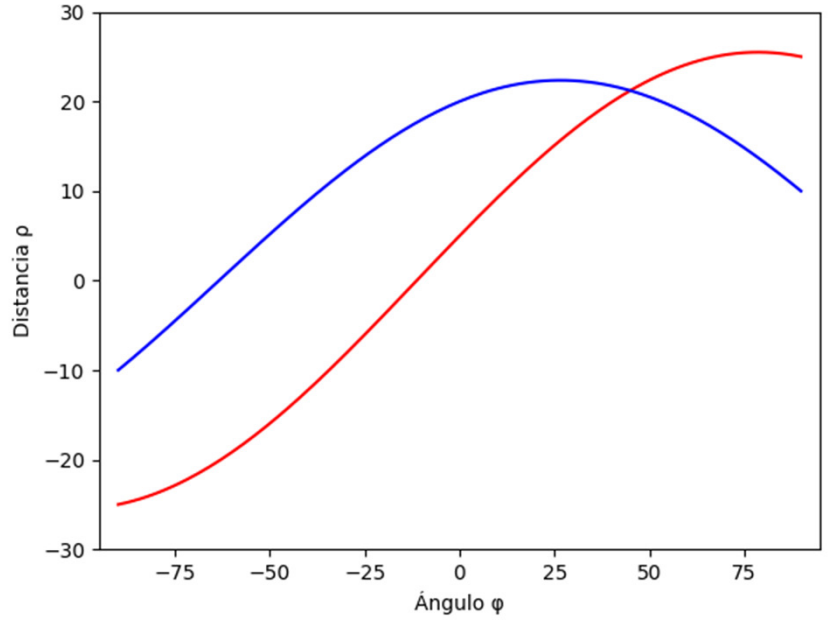
- Considerar dos puntos: $(5, 25)$ y $(20, 10)$.



Fuente: https://www.desmos.com/calculator

Espacio de Hough

- Se repite el procedimiento para los dos puntos.



Se cruzan en

$(\phi = 45, \rho = 21.21)$

Espacio de Hough

- Implicaciones:

a) Una línea entre los dos puntos tiene la ecuación

$$x \cdot \cos(45) + y \cdot \sin(45) = 21.21$$

b) El problema de detectar líneas se convierte en encontrar cruces de curvas.

Algoritmo para detectar líneas

1. Detectar los bordes.
2. Para cada punto en los bordes, encontrar las líneas rectas que pasan por ese punto, generando la curva correspondiente en el espacio de Hough.
3. Contar todas las intersecciones de las distintas curvas.
4. Las intersecciones con un conteo mayor a un umbral arbitrario, se aceptan como línea rectas y pasan al resultado final.

Imagen original

Fuente: https://cs.gmu.edu/~zduric/cs682/Images/

Bordes

Contar cruces

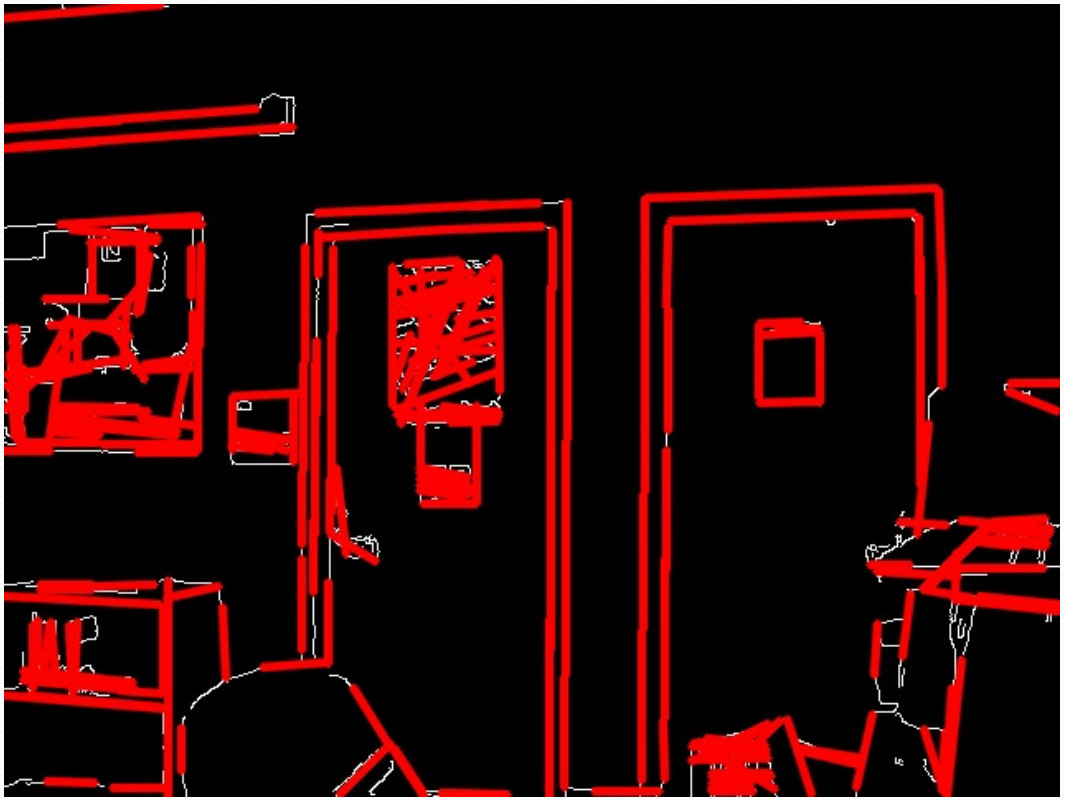
Table I. Accumulator Array for Figure 3(c)

ρ \ θ	0°	20°	40°	60°	80°	100°	120°	140°	160°
85								2	
83							1	2	
81							4	4	
79			2				6	5	
77				2			8	4	2
75							6	6	2
73							4	3	3
71		2		1		1	4	2	3
69			1	4		12	4	3	5
67			3		2	14	2	3	4
65			1			11	1	2	4
63					5	2		2	4
61						1		3	9
59	4	1			11	9	1	8	12
57	4	3		3	10	12	3	10	15
55	9	5		4	5	4	5	11	12
53	6	6		4	10		11	9	14
51	4	9		4	20	2	11	10	8
49	5	6		2	10	3	11	13	8
47	8	4	4	4		2	13	10	10
45	4	7	14	3			11	6	8
43	4	18	21	5		1	12	10	8
41	9	17	21	15		25	18	7	8
39	8	20	21	13		22	11	11	7
37	12	17	22	17		9	10	9	10
35	(38)	14	17	17	38	8	7	9	6
33	37	16	22	21	(42)	10	5	9	9
31	35	11	21	23	23	8	11	9	10
29	13	18	18	23	20	14	13	9	9
27	7	16	12	30	20	20	7	9	6
25	7	18	12	32	19	27	8	7	8
23	8	12	11	20	17	(52)	11	6	7
21	7	17	12	23	8	11	15	11	10
19	9	14	12	16	7	7	14	6	7
17	9	12	12	16	6	9	16	12	7
15	8	13	13	11	7	10	16	14	10
13	10	9	15	11	7	10	16	13	6
11	12	11	13	14	(40)	10	16	13	13
9	10	10	16	14	8	9	14	21	22
7	10	8	22	12	(41)	6	7	12	21
5	11	12	15	11	23	6	11	14	14
3	13	15	15	8	18	7	11	16	15
1	10	14	17	11	7	8	9	10	12

ρ \ θ	0°	20°	40°	60°	80°	100°	120°	140°	160°
−85									
−83									
−81									
−79				3	1				
−77				1	3				
−75									
−73									
−71		2							
−69		2							
−67									
−65			1	2					
−63		1			2				
−61	5								
−59	7		2	1		1			
−57	6					2	1		2
−55	0	10	6			1	1		4
−53	16	13	12		18	4	1		6
−51	(40)	15	11		15	16			6
−49	32	18	11	(57)	15	23	1	1	5
−47	10	16	11	22	14	16	21	9	5
−45	7	17	11	11	16	18	(41)	21	6
−43	8	12	14	10	13	17	12	17	6
−41	6	7	14	11	14	14	7	19	12
−39	7	10	9	8	12	8	11	20	23
−37	7	7	14	8	17	9	12	18	24
−35	8	9	17	8	10	7	10	23	23
−33	6	12	15	8	12	9	11	22	26
−31	5	9	19	9	8	11	16	18	15
−29	9	10	12	9	8	9	18	18	15
−27	7	12	10	8	6	9	18	19	19
−25	5	10	8	8	7	7	22	9	14
−23	6	11	9	9	6	11	19	12	9
−21	7	15	9	7	10	10	16	10	11
−19	6	13	8	16	9	11	17	9	10
−17	7	17	9	15	7	11	16	14	13
−15	6	15	10	17	8	13	10	14	9
−13	10	15	9	15	9	17	11	13	12
−11	10	13	10	7	8	17	9	11	15
−9	7	14	8	7	8	23	8	12	15
−7	9	15	12	7	8	21	7	13	12
−5	(77)	13	15	9	7	14	10	12	15
−3	26	14	14	6	8	12	9	11	18
−1	10	13	18	9	8	8	11	12	15

Fuente: https://dl.acm.org/doi/pdf/10.1145/361237.361242

Resultado



- Umbral = 30.

Implementación

- Python:
- https://github.com/alyssaq/hough_transform
- https://github.com/AlbertoFormaggio1/Hough-Transform-From-Scratch

- OpenCV:
- cv2.HoughLines(image, rho, theta, threshold) -> lines
- cv2.HoughLinesP(image, rho, theta, threshold[, minLineLength[, maxLineGap]]) -> lines

Diferencias

- HoughLines devuelve una representación de cada línea como un punto y un ángulo, sin información sobre los extremos.
- HoughLinesP
  - Devuelve los dos extremos de cada línea detectada.
  - Permite rechazar líneas menores a cierto tamaño.
  - Permite controlar la separación entre líneas.
  - Es más rápido.
- Conclusión: hay pocas razones para preferir HoughLines sobre HoughLinesP.

Función HoughLines

cv.HoughLines(image, rho, theta, threshold) -> lines

- image – imagen binaria de entrada; la imagen puede ser modificada
- lines – lista de líneas detectadas; cada línea se representa por un vector de elementos (ρ, θ), donde ρ es la distancia al origen (esquina superior izquierda) y θ es el ángulo de rotación en radianes (0 – línea vertical; $\pi/2$ – línea horizontal)
- rho – resolución (paso) de distancia en pixeles
- theta – resolución (paso) del ángulo en radianes
- threshold – umbral del acumulador; solo se devuelven aquellas líneas que obtienen suficientes votos (> threshold)

Función HoughLinesP

cv.HoughLinesP(image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]]) -> lines

- image – imagen binaria de entrada; la imagen puede ser modificada
- lines – vector de salida de líneas; cada línea se representa por un vector de elementos (x_1, y_1, x_2, y_2), donde (x_1, y_1) y (x_2, y_2) son los extremos de cada segmento de línea detectado
- rho – resolución (paso) de distancia en pixeles
- theta – resolución (paso) del ángulo en radianes
- threshold – umbral del acumulador; solo se devuelven aquellas líneas que obtienen suficientes votos (> threshold)

Función HoughLinesP

- minLineLength – longitud de línea mínima; los segmentos de línea más cortos que este valor son rechazados
- maxLineGap – espacio máximo permitido entre puntos en la misma línea para unirlos

Ejemplo

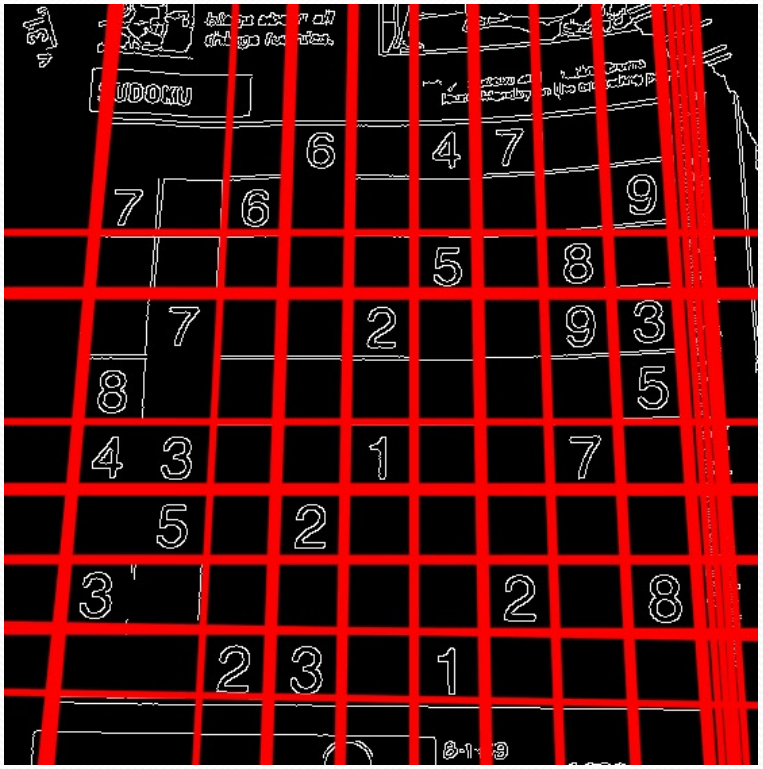
```
# File: hough_lines.py

# standard Hough
# searches for lines that are separated by as little as 1 pixel and 1 degree
# if a candidate line has at least the threshold number of votes, it is retained;
# otherwise, it is discarded.
lines = cv.HoughLines(image_edges, rho=1, theta=np.pi / 180, threshold=180)
```

Ejemplo

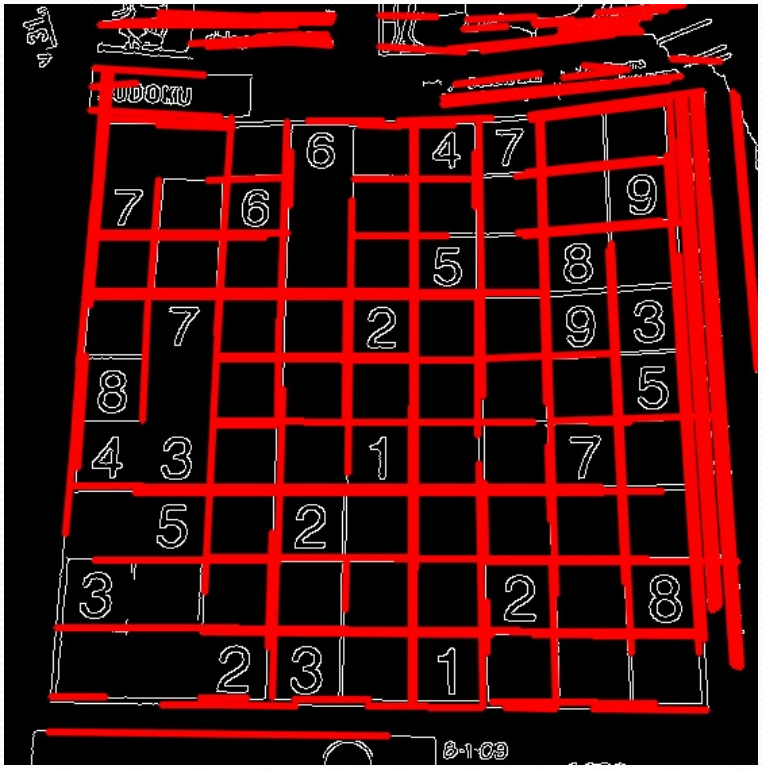
```
# File: hough_lines.py

# rho, theta, and threshold as in HoughLines
# shorter lines than minLineLength are discarded
# maxLineGap is the maximum size of a gap in a line before the two
# segments start being considered as separate lines
linesP = cv.HoughLinesP(image_edges, rho=1, theta=np.pi / 180, threshold=80,
minLineLength=30, maxLineGap=10)
```

Imagen original

Resultado HoughLines



Resultado HoughLinesP



Detección de círculos

- La ecuación del círculo en 2D es:
- $(x-a)^2+(y-b)^2=r^2$
- Donde:
- (x,y) es un punto cualquiera en el círculo.
- (a,b) es el centro de círculo.
- r es el radio del círculo.
- En el espacio de Hough, un círculo se representa como un punto en 3D con coordenadas (a,b,r).

Algoritmo para detectar círculos

- Detectar bordes, por ejemplo mediante el algoritmo de Canny.
- Para cada punto (x, y) en el borde, usar la ecuación del círculo para calcular todos los valores posibles de (a, b) para un radio dado r.
- Incrementar las entradas correspondientes en el acumulador para cada posible centro del círculo r.
- Si el radio no es fijo, considerar múltiples radios, lo que hace que el acumulador sea tridimensional.

Algoritmo para detectar círculos

- Después de la votación, se buscan máximos locales en el acumulador que correspondan a los centros y radios de los círculos más probables.
- Las coordenadas (a, b, r) de estos máximos proporcionan los parámetros de los círculos detectados.

Ventajas y desventajas

- Ventajas:
- Puede detectar círculos de distintos tamaños.
- Es resistente al ruido, especialmente cuando se utiliza con una detección de bordes adecuada.
- Desventajas:
- Es costoso desde el punto de vista computacional, especialmente con imágenes grandes y radios múltiples.
- Los picos en el acumulador pueden ser difíciles de diferenciar si los círculos están cerca uno del otro o son ruidosos.

Función HoughCircles

cv2.HoughCircles(image, method, dp, minDist[, circles[, param1[, param2[, minRadius[, maxRadius]]]]]) -> circles

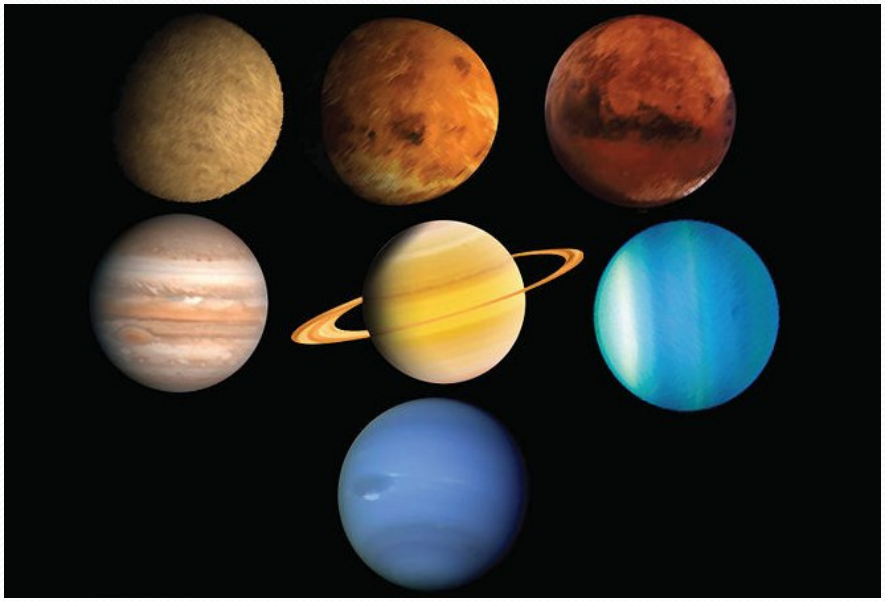
- image – imagen de entrada en escala de grises, de un solo canal y de 8 bits
- circles – vector de salida de los círculos encontrados; cada vector se codifica como un vector de elementos (x, y, radius)
- method – método de detección: HOUGH_GRADIENT o HOUGH_GRADIENT_ALT
- dp – relación inversa de la resolución del acumulador a la resolución de la imagen

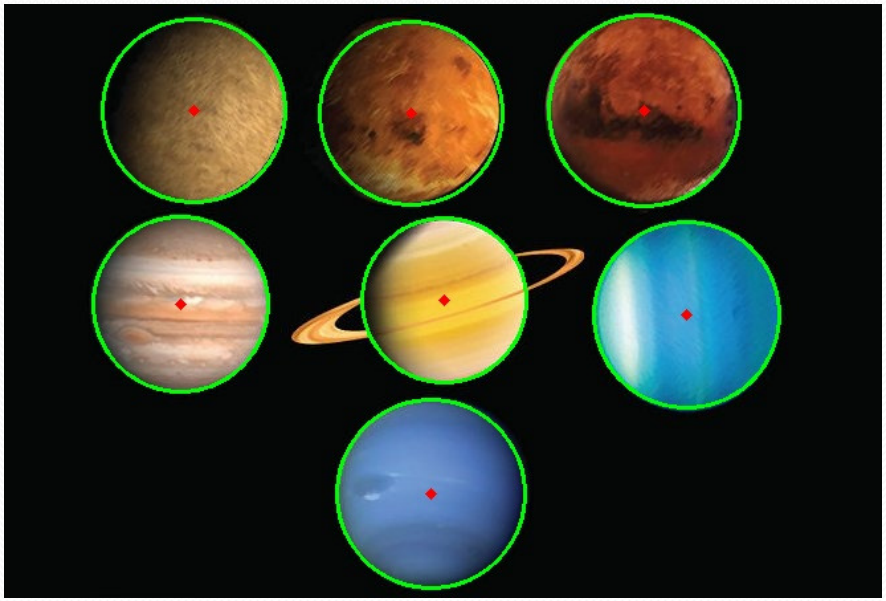
Función HoughCircles

- minDist – distancia mínima entre los centros de los círculos detectados
- param1 – primer parámetro específico del método
- param2 – segundo parámetro específico del método
- minRadius – radio mínimo
- maxRadius – radio máximo

Ejemplo

- Disponible en object_detection/hough_circles.py.





Detección de elipses

- La ecuación de una elipse es
- $\frac{(x-a)^2}{r_x^2} + \frac{(y-b)^2}{r_y^2} = 1$
- Donde:
- (a, b) es el centro de la elipse.
- r_x es el semieje de abscisas.
- r_y es el semieje de ordenadas.
- θ es la rotación de la elipse con respecto al eje x.

Espacio de Hough

- El espacio de Hough y el acumulador tienen 5 dimensiones: a, b, r_x, r_y, θ.
- Los requerimientos de memoria y de poder de cómputo son altos.
- El proceso es sensible al ruido si hay bordes que no pertenecen a elipses.
- Conclusión: para detectar elipses es preferible usar métodos como direct least-squares fitting o machine learning.

Detección de rectángulos

- Un rectángulo se puede definir por cuatro líneas rectas que forman cuatro ángulos rectos (90 grados).
- En términos de parámetros, un rectángulo en el espacio 2D se puede caracterizar por sus cuatro vértices $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$.
- Se puede usar el algoritmo para detectar rectas.
- El siguiente paso es encontrar parejas de rectas paralelas.
- Para un rectángulo:
  - Los lados opuestos deben ser paralelos.
  - Los lados adyacentes deben ser ortogonales.

Detección de rectángulos

- Para las parejas de líneas paralelas y ortogonales, calcular sus puntos de intersección.
- Estas intersecciones dan los posibles vértices del rectángulo.
- Con los cuatro vértices, verificar que formen un rectángulo:
  - Los cuatro lados deben tener dos longitudes distintas.
  - Los lados opuestos deben tener la misma longitud.
  - Las diagonales deben interceptarse en su punto medio y deben tener longitudes iguales.
  - Los ángulos entre lados adyacentes deben ser cercanos a 90°.

Detección de rectángulos

- Se utilizar un acumulador para votar por conjuntos de cuatro líneas (o vértices) que cumplan con los criterios para formar un rectángulo.
- Una vez que se completa el proceso de votación, los rectángulos con mayor número de votos corresponden a los candidatos más probables.

Conclusión

- Hay métodos más prácticos para detectar rectángulos:
- Encontrar los contornos y revisar si forman un rectángulo.
- RANSAC (Random Sample Consensus). Se puede utilizar para detectar rectángulos ajustando segmentos de líneas a los puntos de los bordes y verificando la estructura rectangular.
- Machine learning.