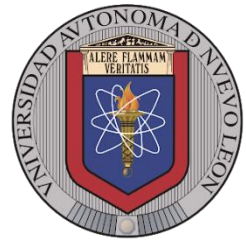



UANL

**UNIVERSIDAD AUTONOMA DE NUEVO LEON**

**FACULTAD DE INGENIERIA MECANICA Y ELECTRICA**


FIME

**Nombre: Roberto Erick Aguilar Morales**

**Matricula: 1871004**

**Carrera: Ingeniero en Tecnologías del Software**

**7.- Círculos**

**Materia: VISION COMPUTACIONAL LABORATORIO**

**Docente: RAYMUNDO SAID ZAMORA PEQUEÑO**

**Hora: N1-N2** **Días: Miércoles**

**Fecha: 24/11/2024**

# Objetivo

Identificar los círculos en una imagen utilizando la Transformada de Hough

# Marco teórico

### Procesamiento de Imágenes

El procesamiento de imágenes es una disciplina que se ocupa de la manipulación y análisis de imágenes digitales a través de computadoras. Se emplea en diversas áreas, desde la medicina y la astronomía, hasta la inteligencia artificial y el control de calidad industrial. Dentro de esta área, el propósito principal es extraer información útil de las imágenes o transformarlas en representaciones más adecuadas para su análisis.

El primer paso en el procesamiento de imágenes digitales suele ser la conversión de la imagen a escala de grises, lo que simplifica el análisis y permite un tratamiento más eficiente de los píxeles. Las imágenes en escala de grises son representaciones bidimensionales donde cada píxel tiene un valor de intensidad que oscila entre 0 (negro) y 255 (blanco).

### Detección de círculos

La detección de círculos en imágenes es una tarea fundamental en el procesamiento de imágenes y visión por computadora. Esta tarea tiene aplicaciones en diversos campos como la robótica, la ingeniería de visión artificial, la medicina, y la inspección industrial, entre otros. Existen diferentes enfoques para la detección de círculos, y uno de los más efectivos y populares es el uso de la transformada de Hough.

### Transformada de Hough

La Transformada de Hough es una técnica matemática utilizada para detectar formas geométricas en una imagen, particularmente líneas rectas y curvas como círculos, elipses, etc. Fue propuesta por Paul Hough en 1962 como un método para la detección de líneas en imágenes, y posteriormente se extendió a la detección de otras figuras geométricas.

En su forma más básica, la transformada de Hough para la detección de líneas convierte una representación espacial de la imagen en un espacio de parámetros, donde cada punto en la imagen se mapea a una curva en el espacio de

parámetros. Esta técnica se puede aplicar también para la detección de círculos, adaptando el concepto a la forma circular.

Detección de Círculos mediante la Transformada de Hough

La ecuación general de un círculo en el espacio cartesiano es:

$(x-a)2+(y-b)2=r2$

donde:

- (a,b)) son las coordenadas del centro del círculo,
- r es el radio del círculo,
- x, y son las coordenadas de cualquier punto sobre la circunferencia del círculo.

Para detectar círculos en una imagen, se utiliza una versión modificada de la transformada de Hough. En lugar de buscar líneas rectas en el espacio de parámetros, se busca un conjunto de tres parámetros: el radio rrr y las coordenadas del centro (a,b) del círculo. El proceso consiste en seguir los siguientes pasos:

1. Preprocesamiento de la Imagen: Primero, la imagen se convierte a escala de grises y se aplica un filtro para reducir el ruido, como el filtro de Canny para detectar bordes. Este paso es esencial, ya que los bordes definen claramente las características de los círculos.
2. Transformada de Hough para Círculos:
    - En lugar de la representación paramétrica de una línea (un par de parámetros), para detectar círculos, se usan tres parámetros: las coordenadas a y b (el centro del círculo) y el radio r.
    - Se calcula un espacio de parámetros 3D, donde cada punto en la imagen que pertenece a un círculo contribuirá a una acumulación en dicho espacio de parámetros. Los puntos que forman un círculo en la imagen coinciden con valores de a, b, y r que acumulan una mayor cantidad de votos en el espacio de parámetros.
3. Búsqueda de picos en el espacio de acumulación: El espacio de parámetros resultante contiene picos en las posiciones correspondientes a los círculos presentes en la imagen. Estos picos se identifican mediante un proceso de umbralización o utilizando un algoritmo de detección de picos, como el algoritmo de detección de máximos locales.
4. Verificación y Extracción de Círculos: Los parámetros (a,b) y r correspondientes a los picos en el espacio de acumulación se mapean nuevamente a la imagen original para marcar los círculos detectados.

## Ventajas de la Transformada de Hough para la Detección de Círculos

La Transformada de Hough ofrece varias ventajas cuando se utiliza para la detección de círculos:

- Robustez ante el ruido: Es capaz de detectar círculos incluso en presencia de ruido o interrupciones parciales en los bordes del círculo.
- Detección de círculos de cualquier tamaño: La técnica puede detectar círculos de diferentes tamaños si se varían adecuadamente los valores de r en el espacio de parámetros.
- No depende de la orientación: A diferencia de otros métodos como el filtro de bordes, la transformada de Hough no depende de la orientación de los círculos en la imagen.

## Limitaciones y Mejoras

Aunque la Transformada de Hough es un enfoque robusto, presenta algunas limitaciones:

- Costo computacional: La implementación de la transformada de Hough puede ser costosa en términos de tiempo de cómputo, especialmente en imágenes grandes o con muchos círculos.
- Espacio de parámetros: La necesidad de explorar un espacio de parámetros tridimensional (para cada combinación de a, b y r) puede ser ineficiente, especialmente cuando hay muchos radios posibles a considerar.

Para mejorar la eficiencia, se pueden aplicar técnicas como la reducción de la resolución del espacio de parámetros o el uso de técnicas de acumulación probabilística.

## Introducción

La detección de círculos en imágenes digitales es un problema crucial en el campo del procesamiento de imágenes y la visión por computadora. Su importancia radica en que los círculos son una de las formas geométricas más comunes en el mundo real, presentes en objetos como ruedas, engranajes, monedas, lentes, entre otros. Identificar y localizar estos círculos de manera precisa puede tener aplicaciones significativas en áreas como la robótica, la inspección industrial, la medicina, y el reconocimiento de objetos.

Uno de los métodos más efectivos para la detección de círculos es la transformada de Hough, una técnica matemática que permite identificar formas geométricas en una imagen a partir de su representación paramétrica. Aunque originalmente se diseñó para detectar líneas rectas, la transformada de Hough se ha adaptado para detectar círculos, utilizando tres parámetros: las coordenadas del centro y el radio del círculo.

El proceso de detección mediante la transformada de Hough implica transformar la imagen original a un espacio de parámetros en el cual se acumulan los puntos que corresponden a círculos. Los picos de acumulación en este espacio indican la presencia de círculos en la imagen, lo que permite su localización y clasificación. Esta técnica es especialmente útil debido a su robustez frente a ruidos y discontinuidades en los bordes de los círculos.

## Desarrollo

Detección de líneas conectadas (detectar_lineas_conectadas)

Esta función busca líneas conectadas en una imagen binaria. Utiliza un algoritmo de búsqueda en profundidad  para explorar y conectar los píxeles que forman una línea. Los pasos son:

- Se recorre cada píxel de la imagen. Si es parte de un borde, se explora su vecindario para formar una línea.
- Las líneas detectadas se almacenan en una lista y se devuelven.

Verificación de centro de un círculo (verificar_centro_candidato)

Esta función verifica si un punto candidato (posible centro de un círculo) está efectivamente cerca de al menos tres líneas de borde. Para ello, extiende líneas desde el punto candidato en ocho direcciones (vertical, horizontal y diagonal) y comprueba si alguna de ellas toca un borde.

Búsqueda de centro de círculo en un rango (encontrar_centro_con_rango_de_busqueda)

Esta función busca el centro de un círculo dentro de un rango de píxeles alrededor de un punto inicial. Para ello:

- Se examinan las líneas conectadas detectadas en la imagen.
- A partir de cada punto en la línea, se calculan las direcciones en las que las líneas pueden extenderse hasta encontrar bordes.
- Se registra el centro de las líneas más largas, y se calcula la posición central de la línea como posible centro del círculo.

Procesamiento de todas las líneas (procesar_todas_las_lineas)

Esta función combina todos los procesos anteriores. Para cada línea detectada:

- Se calcula el centro y el radio de los posibles círculos formados por la línea.
- Se dibujan los círculos encontrados sobre la imagen.
- Los bordes que corresponden a los círculos detectados se eliminan de los bordes restantes, actualizando así la imagen para detectar otros círculos.

Detección de líneas utilizando RANSAC (detectar_varias_lineas_ransac)

RANSAC es un algoritmo robusto para la detección de modelos (como líneas) en presencia de ruido. Este código aplica RANSAC para encontrar varias líneas en un conjunto de puntos (en este caso, bordes de una imagen). Los puntos que se ajustan bien a una línea (inliers) se detectan en cada iteración.

Detección de vecindarios (detectar_vecindarios)

Esta función agrupa los píxeles de la imagen que están cerca unos de otros, basándose en su valor de intensidad. Esto se hace mediante una búsqueda en amplitud (BFS) para agrupar píxeles en vecindarios. Además, los bordes de cada vecindario se identifican.

Procesamiento de la imagen

- Se carga la imagen en color y luego se convierte a escala de grises.
- Se detectan vecindarios y bordes.
- Se aplica la Transformada de Hough para detectar líneas rectas en la imagen binarizada.
- Las líneas rectas detectadas se visualizan en la imagen.
- Luego se procesan las líneas detectadas para encontrar círculos y se dibujan sobre la imagen original.
- Finalmente, las imágenes resultantes (con líneas, círculos, bordes y vecindarios) se visualizan en ventanas y se guardan en archivos de imagen.

## Cálculos y Resultados

### P07 Circulos.py

```
import cv2
import numpy as np
from sklearn.linear_model import RANSACRegressor

def detectar_lineas_conectadas(bordes_restantes):
    # Crear una copia para ir eliminando líneas una por una
    bordes_temp = np.copy(bordes_restantes)
    lineas_detectadas = []

    # Dimensiones de la imagen
    rows, cols = bordes_temp.shape

    # Función para seguir una línea conectada usando una búsqueda en
profundidad (DFS)
    def seguir_linea(x, y):
        stack = [(x, y)]
        linea_actual = np.zeros_like(bordes_temp)  # Imagen para la línea
actual
        while stack:
            cx, cy = stack.pop()
            if 0 <= cx < cols and 0 <= cy < rows and bordes_temp[cy, cx] ==
255:
                # Marcar el píxel en la imagen de la línea actual y en
bordes_temp
                linea_actual[cy, cx] = 255
                bordes_temp[cy, cx] = 0
                # Agregar píxeles vecinos (4 direcciones) a la pila
                for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
                    nx, ny = cx + dx, cy + dy
                    stack.append((nx, ny))
        return linea_actual

    # Recorrer la imagen para encontrar cada línea
    for y in range(rows):
        for x in range(cols):
            if bordes_temp[y, x] == 255:
                linea_actual = seguir_linea(x, y)
                lineas_detectadas.append(linea_actual)

    return lineas_detectadas

def verificar_centro_candidato(linea_detectada, centro_x, centro_y):
```

```
    # Dimensiones de la imagen
    rows, cols = linea_detectada.shape

    # Definir las 8 direcciones de los píxeles
    direcciones = [
        (-1, 0), (1, 0), (0, -1), (0, 1),
        (-1, -1), (-1, 1), (1, -1), (1, 1)
    ]

    # Contador de líneas que tocan un borde
    contador_lineas_que_toquen_borde = 0

    # Crear una copia de la imagen para dibujar las líneas de prueba
    imagen_lineas = linea_detectada.copy()

    # Verificar cada dirección y dibujar las líneas de prueba
    for dx, dy in direcciones:
        end_x, end_y = centro_x, centro_y

        # Extender la línea en la dirección actual hasta encontrar un borde
o salir de la imagen
        while 0 <= end_x + dx < cols and 0 <= end_y + dy < rows:
            end_x += dx
            end_y += dy
            if linea_detectada[end_y, end_x] == 255:  # Si encontramos un
borde
                contador_lineas_que_toquen_borde += 1
                break

    # Comprobar si al menos 3 líneas tocan un borde
    return contador_lineas_que_toquen_borde >= 3

def encontrar_centro_con_rango_de_busqueda(linea_detectada, intervalo=10,
rango=20):
    acumulador = np.zeros_like(linea_detectada, dtype=np.int32)
    rows, cols = linea_detectada.shape

    # Definir las 8 direcciones de los píxeles
    direcciones = [
        (-1, 0), (1, 0), (0, -1), (0, 1),
        (-1, -1), (-1, 1), (1, -1), (1, 1)
    ]

    contador_pixeles = 0
```

```
    for y in range(rows):
        for x in range(cols):
            if linea_detectada[y, x] == 255:
                contador_pixeles += 1
                if contador_pixeles % intervalo == 0:
                    longitudes_lineas = []
                    puntos_finales = []

                    for dx, dy in direcciones:
                        end_x, end_y = x, y
                        longitud = 0
                        while 0 <= end_x + dx < cols and 0 <= end_y + dy <
rows:
                            end_x += dx
                            end_y += dy
                            longitud += 1
                            if linea_detectada[end_y, end_x] == 255:
                                break

                        longitudes_lineas.append(longitud)
                        puntos_finales.append((end_x, end_y))

                    max_longitud_idx = np.argmax(longitudes_lineas)
                    end_x, end_y = puntos_finales[max_longitud_idx]

                    mid_x = (x + end_x) // 2
                    mid_y = (y + end_y) // 2

                    acumulador[mid_y, mid_x] += 1

    contador_vecindario = np.zeros_like(acumulador)
    for y in range(rango, rows - rango):
        for x in range(rango, cols - rango):
            ventana = acumulador[y - rango:y + rango + 1, x - rango:x +
rango + 1]
            contador_vecindario[y, x] = np.sum(ventana)

    centro_y, centro_x = np.unravel_index(np.argmax(contador_vecindario),
contador_vecindario.shape)
    max_votos = contador_vecindario[centro_y, centro_x]

    # Crear una copia para la imagen de prueba de verificación
    if verificar_centro_candidato(linea_detectada, centro_x, centro_y):
        max_distancia = 0
        for dx, dy in direcciones:
```

```
            end_x, end_y = centro_x, centro_y
            while 0 <= end_x + dx < cols and 0 <= end_y + dy < rows:
                end_x += dx
                end_y += dy
                if linea_detectada[end_y, end_x] == 255:
                    distancia = np.sqrt((end_x - centro_x) ** 2 + (end_y -
centro_y) ** 2)
                    if distancia > max_distancia:
                        max_distancia = distancia
                    break

        if max_distancia > 0:
            return (centro_x, centro_y), max_distancia
    return None, None

# Procesar todas las líneas en bordes_restantes y dibujar los círculos en la
imagen final
def procesar_todas_las_lineas(bordes_restantes, imagen_original):
    lineas_detectadas = detectar_lineas_conectadas(bordes_restantes)
    imagen_resultado = imagen_original.copy()  # Imagen final para todos los
círculos
    imagen_solo_circulos = np.zeros_like(bordes_restantes)  # Imagen final
para solo los círculos
    bordes_restantes_2 = bordes_restantes.copy()  # Bordes restantes despues
de los circulos

    for idx, linea in enumerate(lineas_detectadas):
        print(f"Procesando línea {idx + 1}")
        centro, radio = encontrar_centro_con_rango_de_busqueda(linea)
        if centro:
            print(f"Centro del círculo en línea {idx + 1}: {centro}")
            print(f"Diámetro del círculo en línea {idx + 1}: {2 * radio}")

            # Dibujar el círculo detectado en la imagen de resultado
            cv2.circle(imagen_resultado, centro, int(radio), (255, 0, 0), 2)
            cv2.circle(imagen_resultado, centro, 2, (0, 0, 255), 3)

            # Dibujar solo los círculos detectados en la imagen nueva
            cv2.circle(imagen_solo_circulos, centro, int(radio), (255), 2)

            # Borrar circulos de los bordes restantes
            cv2.circle(bordes_restantes_2, centro, int(radio), (0), 2)
        else:
            print(f"No se encontró un círculo válido en línea {idx + 1}.")
```

```
    return imagen_resultado, imagen_solo_circulos, bordes_restantes_2

# Función para aplicar RANSAC en múltiples iteraciones
def detectar_varias_lineas_ransac(puntos_X, puntos_Y, min_puntos=5,
max_iteraciones=50):
    lineas_ransac = []
    iteracion = 0

    while len(puntos_X) > min_puntos and iteracion < max_iteraciones:
        # Aplicar RANSAC para detectar una línea
        ransac = RANSACRegressor()
        ransac.fit(puntos_X, puntos_Y)

        # Obtener los inliers, es decir, los puntos que se ajustan bien a la
línea
        inlier_mask = ransac.inlier_mask_

        # Extraer los puntos que corresponden a la línea detectada
        puntos_inliers_X = puntos_X[inlier_mask]
        puntos_inliers_Y = puntos_Y[inlier_mask]

        # Guardar los puntos de la línea detectada
        lineas_ransac.append((puntos_inliers_X, puntos_inliers_Y))

        # Eliminar los inliers de la lista de puntos (quitar la línea ya
detectada)
        puntos_X = puntos_X[~inlier_mask]
        puntos_Y = puntos_Y[~inlier_mask]

        iteracion += 1

    return lineas_ransac

# Función para detectar vecindarios (ya existente)
def detectar_vecindarios(image, rango=50):
    rows, cols = image.shape
    visitado = np.zeros((rows, cols), dtype=bool)
    vecindarios = []
    bordes_vecindarios = []

    def obtener_vecindario(r, c, pivote_valor, rango):
        vecindario = [(r, c)]
        borde = set()
        cola = [(r, c)]
        visitado[r, c] = True
```

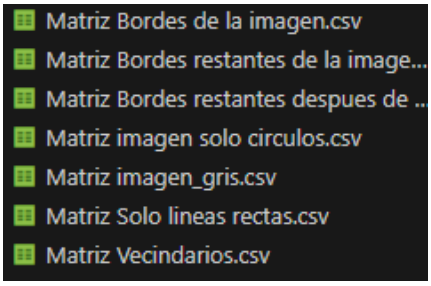
```
        pivote_valor = int(pivote_valor)

        while cola:
            x, y = cola.pop(0)
            for dx in [-1, 0, 1]:
                for dy in [-1, 0, 1]:
                    nx, ny = x + dx, y + dy
                    if 0 <= nx < rows and 0 <= ny < cols and not
visitado[nx, ny]:
                        if pivote_valor - rango <= int(image[nx, ny]) <=
pivote_valor + rango:
                            visitado[nx, ny] = True
                            vecindario.append((nx, ny))
                            cola.append((nx, ny))
                        else:
                            borde.add((nx, ny))
        return vecindario, borde

    for r in range(rows):
        for c in range(cols):
            if not visitado[r, c]:
                pivote_valor = image[r, c]
                vecindario, borde = obtener_vecindario(r, c, pivote_valor,
rango)
                vecindarios.append(vecindario)
                bordes_vecindarios.append(borde)

    return vecindarios, bordes_vecindarios

# Cargar imagen en escala de grises
imagen_a_color = cv2.imread('circulos.png')
if imagen_a_color is None:
    print("Error: No se pudo cargar la imagen.")
else:
    imagen = cv2.cvtColor(imagen_a_color, cv2.COLOR_BGR2GRAY)

    # Detectar vecindarios y bordes
    vecindarios, bordes_vecindarios = detectar_vecindarios(imagen)

    # Mostrar cuántos vecindarios se encontraron
    print(f"Número de vecindarios encontrados: {len(vecindarios)}")

    # Para visualizar los vecindarios en la imagen (opcional)
    output_image = np.zeros_like(imagen)
    for i, vecindario in enumerate(vecindarios):
```

```
        color_value = (255 - (i * 25)) % 256  # Asegurar que los valores
estén entre 0 y 255
        for (r, c) in vecindario:
            output_image[r, c] = color_value  # Colorear cada vecindario de
manera diferente
    # Crear imagen binaria para los bordes
    bordes_imagen = np.zeros_like(imagen)

    # Marcar los píxeles de borde en la imagen binaria
    for borde in bordes_vecindarios:
        for (r, c) in borde:
            bordes_imagen[r, c] = 255  # Borde en blanco

    # Aplicar la Transformada de Hough directamente sobre la imagen
binarizada de bordes
    # threshold: Decide que el minimo de votos que debe tener para
detectarse como linea recta
    # minLineLength: La minima cantidad de pixeles para poder considerarse
linea recta
    # masLineGap: maxima separacion permitida entre dos puntos de una linea
    lineas = cv2.HoughLinesP(bordes_imagen, rho=1, theta=np.pi / 180,
threshold=75, minLineLength=3, maxLineGap=1)

    solo_lineas = np.zeros_like(bordes_imagen)
    bordes_restantes = np.copy(bordes_imagen)

    if lineas is not None:
        for linea in lineas:
            for x1, y1, x2, y2 in linea:
                cv2.line(imagen_a_color, (x1, y1), (x2, y2), (0, 255, 0),
2)  # Dibuja cada línea con color verde
                cv2.line(solo_lineas, (x1, y1), (x2, y2), (255), 2)  #
Dibuja cada línea con color verde
                cv2.line(bordes_restantes, (x1, y1), (x2, y2), (0), 2) #
Quita las lineas rectas

    imagen_circulos, imagen_solo_circulos, bordes_restantes_2 =
procesar_todas_las_lineas(bordes_imagen, imagen_a_color)

    # Mostrar las imagenes procesadas
    cv2.imshow('Vecindarios', output_image)
    cv2.imshow("Bordes de la imagen", bordes_imagen)
    cv2.imshow('Lineas rectas sobrepuestas', imagen_a_color)
    cv2.imshow('Solo lineas rectas', solo_lineas)
    cv2.imshow("Bordes restantes de la imagen", bordes_restantes)
```

```
    cv2.imshow("Circulos detectados", imagen_circulos)
    cv2.imshow("Solo circulos", imagen_solo_circulos)
    cv2.imshow("Bordes restnantes despues de circulos", bordes_restantes_2)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Guardar las imagenes procesadas
    cv2.imwrite('Imagen Vecindarios.png', output_image)
    cv2.imwrite("Imagen Bordes de la imagen.png", bordes_imagen)
    cv2.imwrite('Imagen Lineas rectas sobrepuestas.png', imagen_a_color)
    cv2.imwrite('Imagen Solo lineas rectas.png', solo_lineas)
    cv2.imwrite("Imagen Bordes restantes de la imagen.png",
bordes_restantes)
    cv2.imwrite("Imagen Circulos detectados.png", imagen_circulos)
    cv2.imwrite("Imagen Solo circulos.png", imagen_solo_circulos)
    cv2.imwrite("Imagen Bordes restantes despues de circulos.png",
bordes_restantes_2)

    # Guardar la matriz de la imagen en escala de grises en un archivo CSV
    np.savetxt('Matriz imagen_gris.csv', imagen, delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'Matriz imagen_gris.csv'.")

    # Guardar la matriz de la imagen vecindarios en un archivo CSV
    np.savetxt('Matriz Vecindarios.csv', output_image, delimiter=',',
fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'Matriz Vecindarios.csv'.")

    # Guardar la matriz de la imagen Bordes de la imagen en un archivo CSV
    np.savetxt('Matriz Bordes de la imagen.csv', bordes_imagen,
delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'Matriz Bordes de la imagen.csv'.")

    # Guardar la matriz de la imagen Solo lineas rectas en un archivo CSV
    np.savetxt('Matriz Solo lineas rectas.csv', solo_lineas, delimiter=',',
fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'Matriz Solo lineas rectas.csv'.")

    # Guardar la matriz de la imagen Bordes restantes de la imagen en un
archivo CSV
    np.savetxt('Matriz Bordes restantes de la imagen.csv', bordes_restantes,
delimiter=',', fmt='%d')
```

```
    print("La matriz de la imagen en escala de grises se ha guardado en
'Matriz Bordes restantes de la imagen.csv'.")

    # Guardar la matriz de la imagen Bordes restantes de la imagen en un
archivo CSV
    np.savetxt('Matriz imagen solo circulos.csv', imagen_solo_circulos,
delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'Matriz Bordes restantes de la imagen.csv'.")

    # Guardar la matriz de la imagen Bordes restantes de la imagen en un
archivo CSV
    np.savetxt('Matriz Bordes restantes despues de circulos.csv',
bordes_restantes_2, delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'Matriz Bordes restantes despues de circulos.csv'.")
```

## Matriz de imágenes



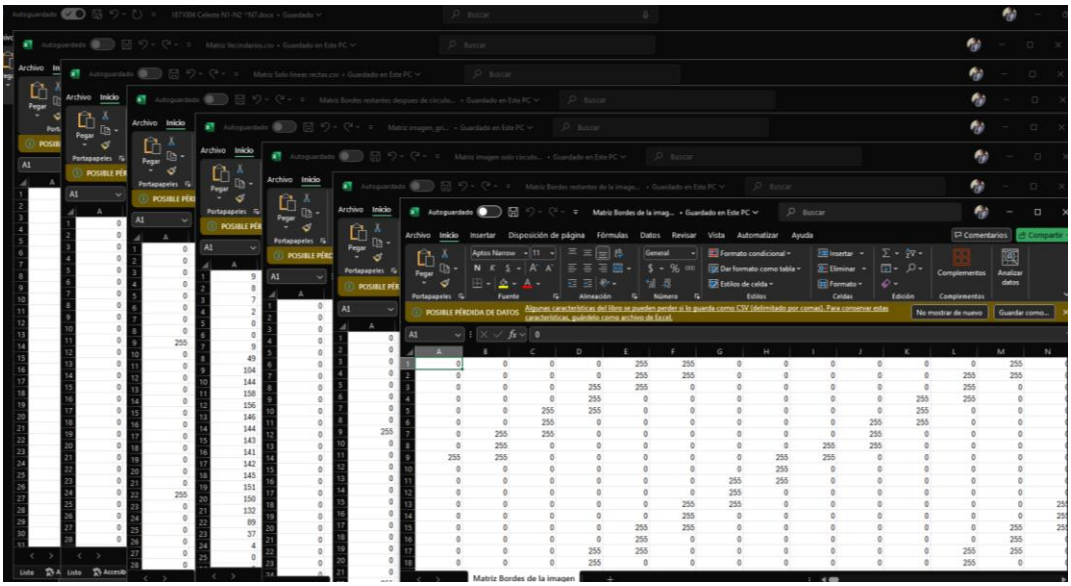



**Imagen círculos.jpg**

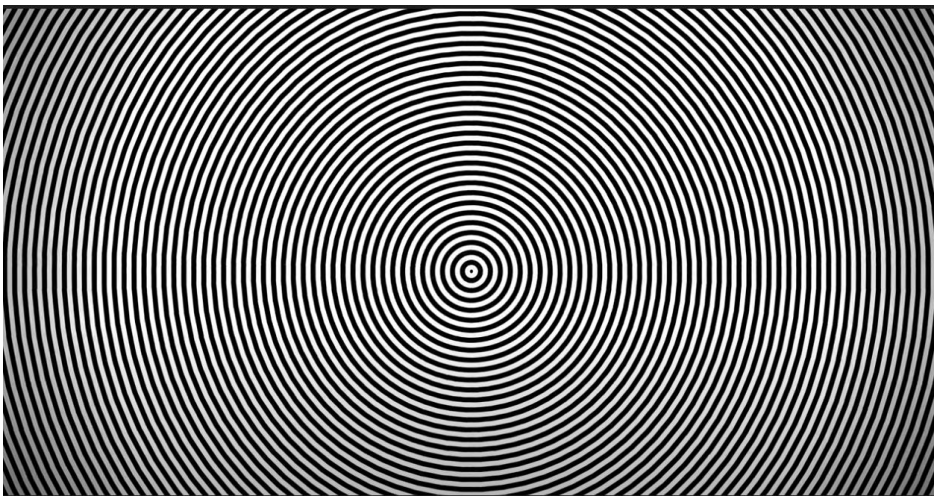


**Imagen Vecindarios.png**

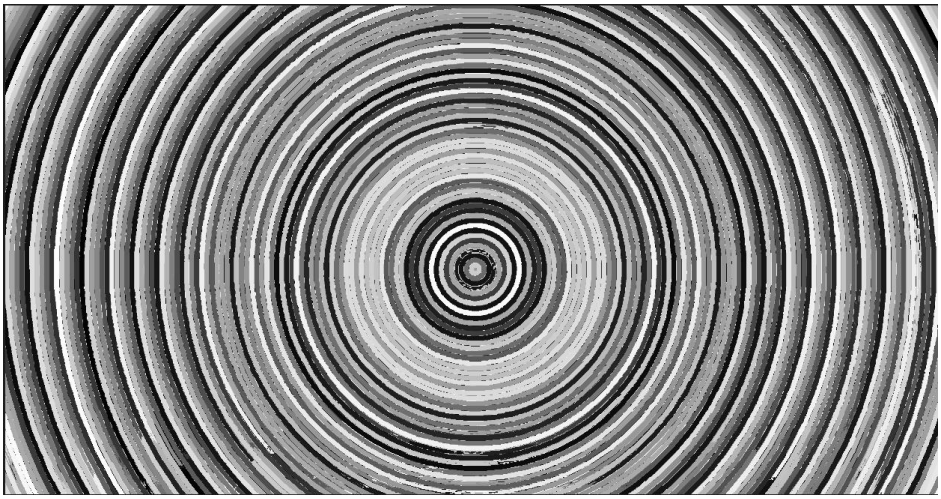


**Imagen Bordes de la imagen.png**

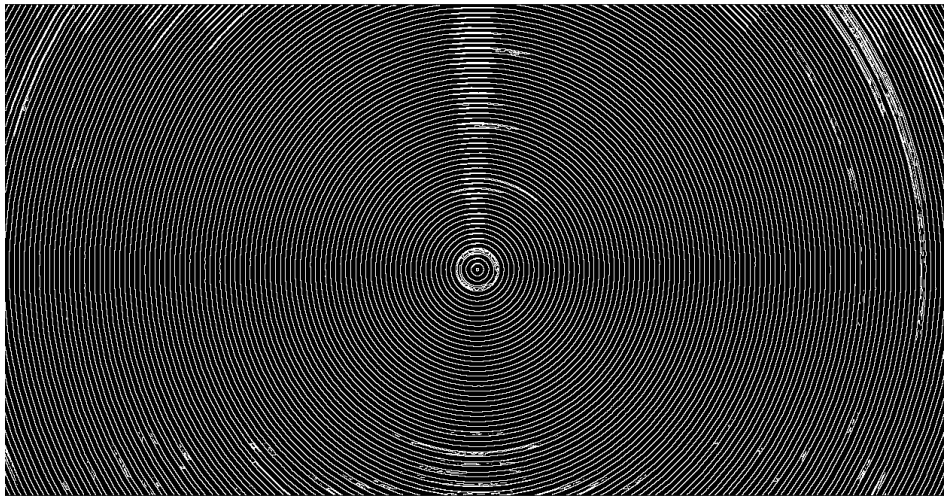


**Imagen Líneas rectas sobrepuestas.png**

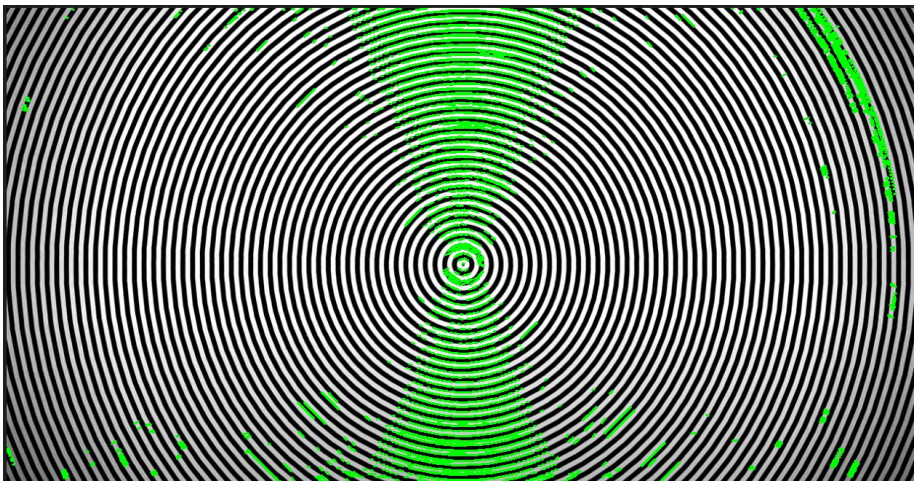


**Imagen Solo líneas rectas.png**

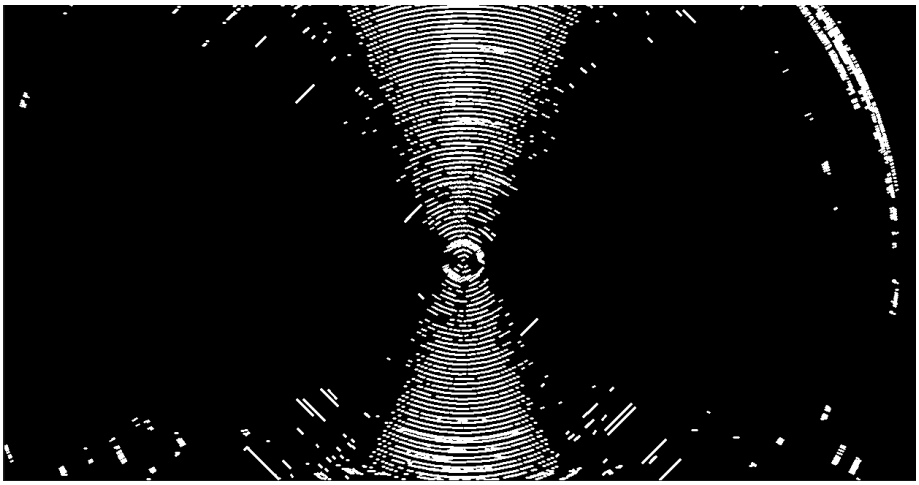


**Imagen Bordes restantes de la imagen.png**

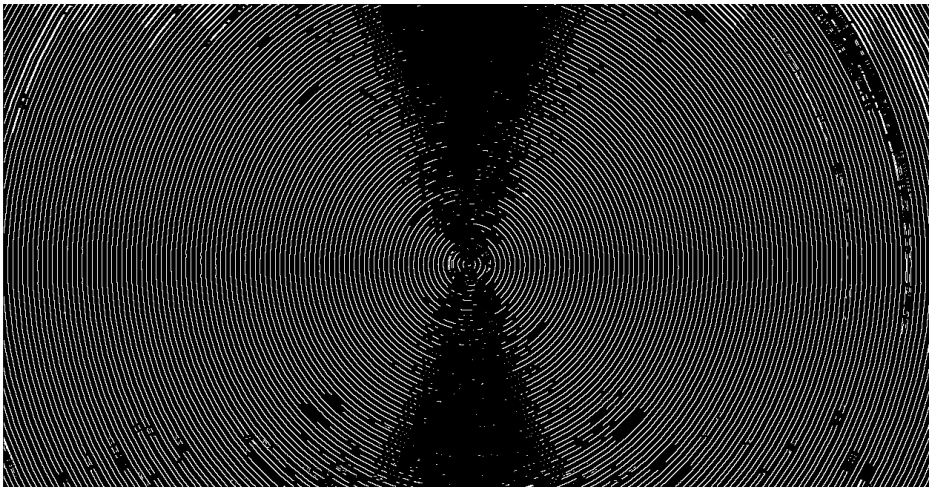


**Imagen Círculos detectados.png**

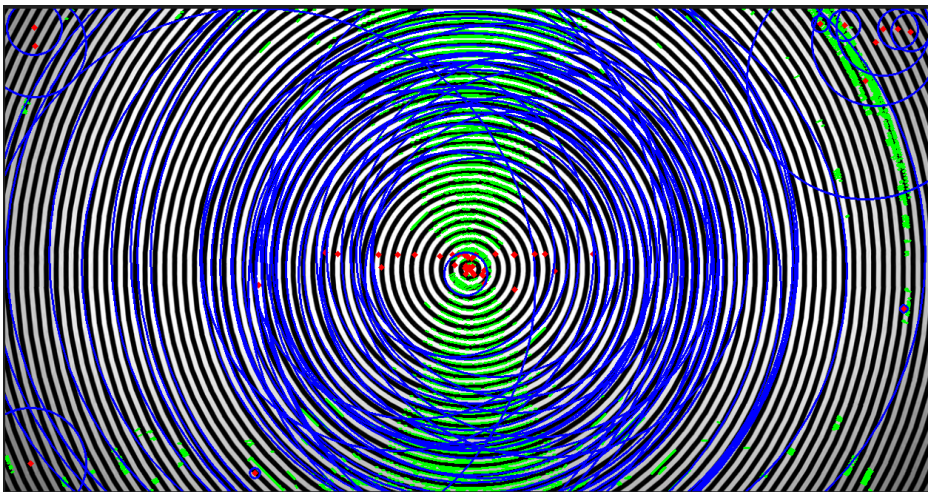


**Imagen Solo circulos.png**

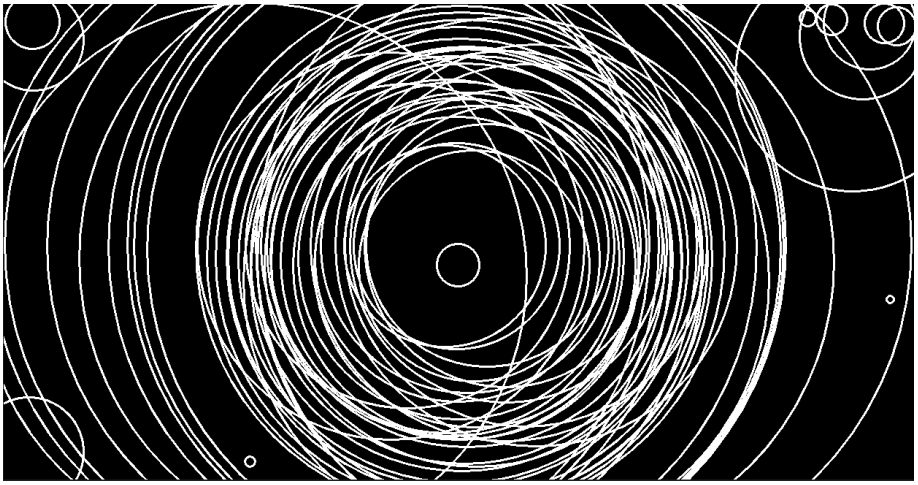
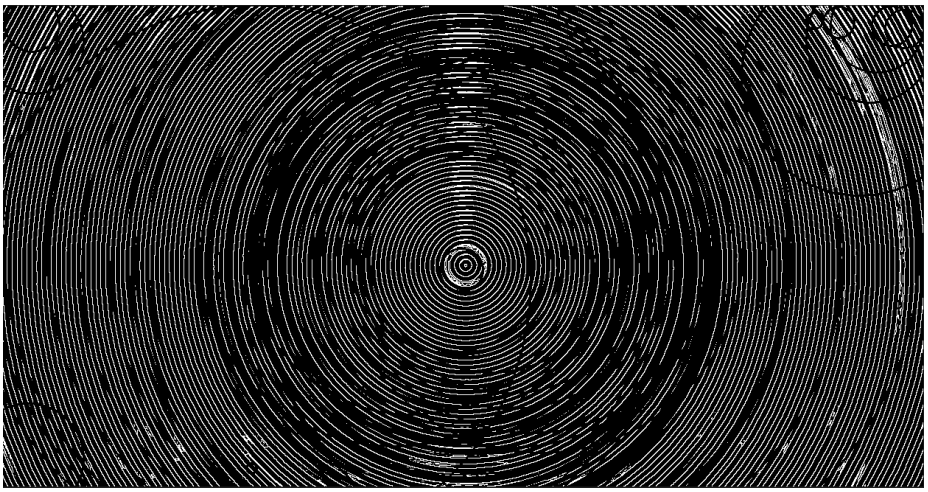


**Imagen Bordes restantes después de circulos.png**



## Conclusiones

Este código representa una implementación sólida y versátil de procesamiento de imágenes, que combina múltiples técnicas avanzadas para la detección y análisis de estructuras geométricas. Integra métodos como la identificación de líneas conectadas, la detección de círculos y el uso de RANSAC para localizar líneas rectas de forma precisa, incluso en escenarios con datos complejos o ruidosos. Además, la combinación de técnicas de agrupamiento y la Transformada de Hough contribuye a mejorar la precisión en la identificación de características geométricas, proporcionando una visualización clara y detallada de las estructuras presentes en la imagen.

El enfoque también incorpora la acumulación y el análisis de vecindarios, lo que facilita la detección de patrones espaciales y bordes, permitiendo una representación procesada más comprensible. Estas capacidades no solo generan imágenes procesadas más informativas, sino que también producen matrices exportables, ideales para un análisis exhaustivo y para aplicaciones que requieren procesamiento posterior.

## Bibliografía


Gonzalez, R. C., & Woods, R. E. (2008). *Digital image processing* (3rd ed.). Pearson.

Szeliski, R. (2010). *Computer vision: Algorithms and applications*. Springer.

Pratt, W. K. (2007). *Digital image processing: PIKS scientific inside* (4th ed.). Wiley-Interscience.