

Tarea: Detección de Monedas

Serapio Hernández Alexis Arturo
 Facultad de Ingeniería
 UNAM
 Ciudad de México, México
alexis.serapio@ingenieria.unam.mx

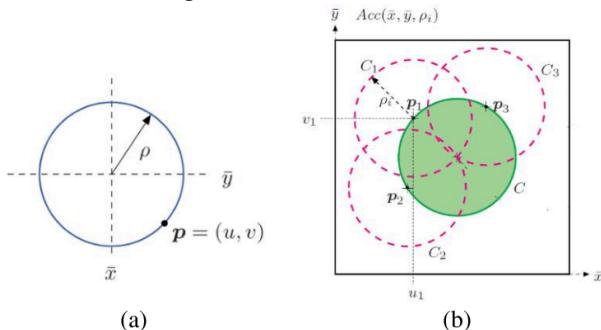
García López Erik
 Facultad de Ingeniería
 UNAM
 Ciudad de México, México
erikpumas999@gmail.com

Reyes Herrera Rogelio
 Facultad de Ingeniería
 UNAM
 Ciudad de México, México
masterfive5of@gmail.com

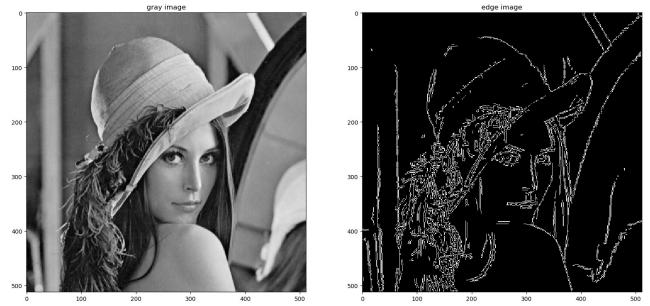
I. INTRODUCCIÓN

En esta tarea abordamos el desarrollo de un sistema automatizado para la detección de monedas en imágenes mediante el uso de la Transformada de Hough y técnicas complementarias de procesamiento de imágenes en Python. Este problema es un excelente ejemplo de cómo los métodos de visión por computadora y procesamiento de imágenes pueden aplicarse en situaciones de la vida real, desde la identificación de formas en imágenes hasta el análisis automatizado en sistemas de vigilancia o control de calidad en la industria.

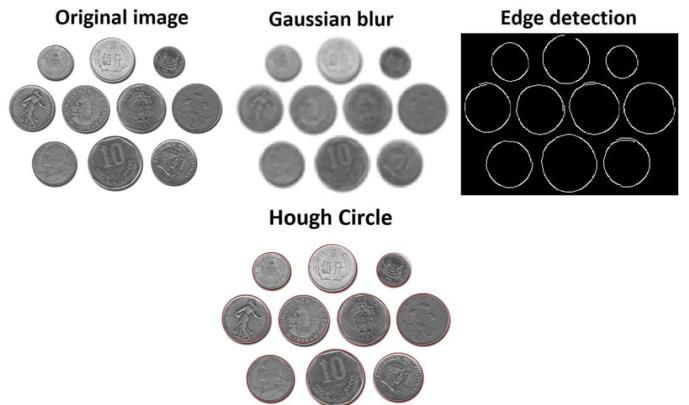
La Transformada de Hough es una técnica poderosa para la detección de formas geométricas en imágenes, como círculos, líneas o elipses. En este caso, su capacidad de localizar círculos se adapta perfectamente a la tarea de identificar monedas en una imagen.



Para lograr esto, es fundamental aplicar ciertas etapas de preprocesamiento, como el suavizado de la imagen y la detección de bordes, para mejorar la precisión de la transformada y reducir el impacto del ruido. En nuestro código, utilizamos la detección de bordes de Canny, que permite resaltar las regiones de interés en la imagen.



Aplicado a monedas, necesitamos un método de esta manera:



Para evaluar el rendimiento del método, aplicaremos métricas de evaluación como precisión, exhaustividad (Recall) y F1 Score, las cuales proporcionan un análisis objetivo de los resultados, ayudándonos a entender tanto las detecciones correctas como los errores del sistema.

Finalmente, compararemos la denominación de cada una de las monedas

II. DESARROLLO PRÁCTICA 1

3.1.- Utilizando la transformada de Hough, deberá diseñar un método que detecte todas las monedas presentes en todas las imágenes. Podrá auxiliarse de cualquier método de detección de bordes o puntos característicos antes de aplicar la Transformada de Hough. Despliegue sus propias imágenes de acuerdo al material que se le proporcionó.

Para comenzar, tomamos la primera imagen del conjunto proporcionado por la profesora. En este caso tomamos la IMG_B_Foco.jpg.



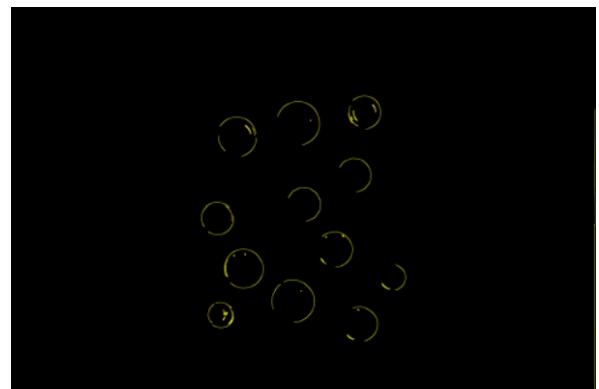
Para la detección de bordes hicimos uso de canny, como vimos, este usa un algoritmo de múltiples etapas para detectar una amplia gama de bordes en imágenes, aunque se puede aplicar a imágenes a color, estas deben convertirse a escala de grises para ser tratadas mediante Canny.

```
imagen = cv2.imread(imagen_path,
cv2.IMREAD_GRAYSCALE)

# Aplicar suavizado para reducir ruido
imagen_suavizada = cv2.GaussianBlur(imagen,
(9, 9), 2)

# Aplicar el detector de bordes de
Canny
bordes = cv2.Canny(imagen_suavizada, 100,
200)
```

Por lo que se le aplica un filtro gaussiano a la imagen y posteriormente se traslada a escala de grises, una vez hecho eso. Obtenemos como resultado lo siguiente:

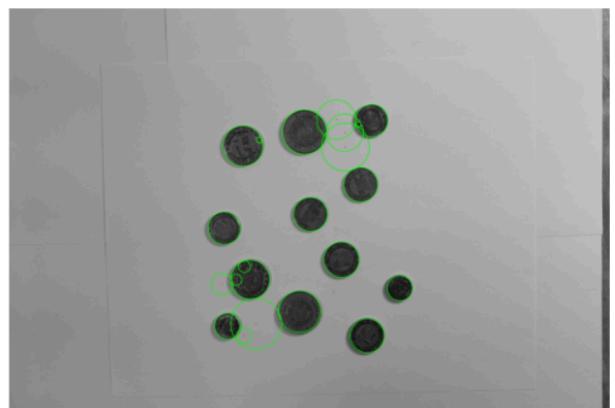


Finalmente aplicamos el algoritmo de Hough, La transformada de Hough se basa en la idea de que cualquier línea puede ser representada por dos parámetros: la distancia desde el origen (ρ) y el ángulo desde el eje x (θ). Para cada píxel de la imagen, puedes calcular los valores de ρ y θ para cada línea posible que pase a través de él.

```
# Aplicar la Transformada de Hough para detectar círculos
monedas = cv2.HoughCircles(
    bordes,
    cv2.HOUGH_GRADIENT,
    dp=2,           # Inversa de la relación de resolución del acumulador de Hough
    minDist=100,    # Distancia mínima entre los centros de los círculos
    param1=75,      # Valor superior para el detector de bordes de Canny
    param2=65,      # Umbral del acumulador para la detección de círculos
    minRadius=0,    # Radio mínimo de las monedas
    maxRadius=200   # Radio máximo de las monedas
```

El primer resultado, nos daba valores muy extraños y “malos” Esto es por que necesitábamos adaptar correctamente los parámetros de la transformada a las curvas de nuestra imagen.

Se detectaron 22 monedas



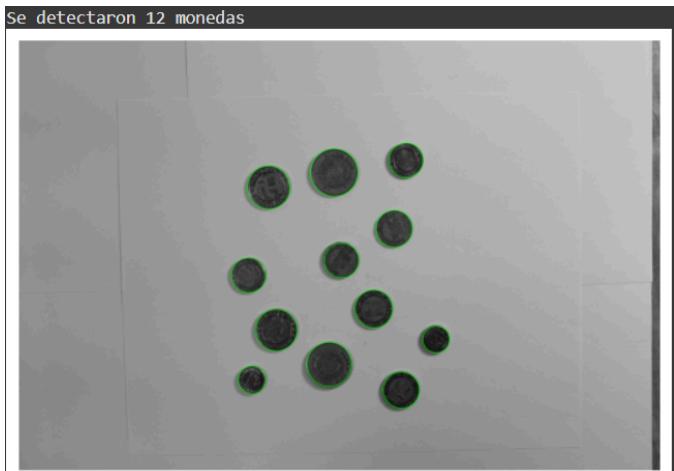
Modificamos todos los atributos de los parámetros.

Un valor alto de param1 hace que el detector de bordes sea más estricto, capturando solo los bordes más fuertes. Un valor bajo hace que se detecten más bordes, lo cual puede incluir ruido.

Un valor bajo de param2 permite detectar más círculos, incluidos algunos falsos positivos, mientras que un valor alto

detecta sólo los círculos que tienen un alto grado de coincidencia en el espacio de acumulación.

Finalmente, llegamos a los valores de 75 para param1 y 65 para param2, dándonos como resultado lo siguiente:



3.2.- Deberá presentar una evaluación objetiva de su método, que permita tener cifras de desempeño de su método. ¿Cómo se le ocurre que lo va a evaluar? Puede utilizar el paso 3 para esto.

Nosotros vamos a utilizar 3 parámetros, **Precisión**, **Exhaustividad (Recall)** y **F1 Score**.

- **Precisión:** Mide la proporción de detecciones correctas entre todas las detecciones realizadas. Nos dice cuántas de las monedas detectadas son realmente monedas.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Exhaustividad (Recall):** Mide la proporción de monedas verdaderas detectadas con respecto al total de monedas en la imagen. Nos indica qué tan completo es el método en la detección de monedas.

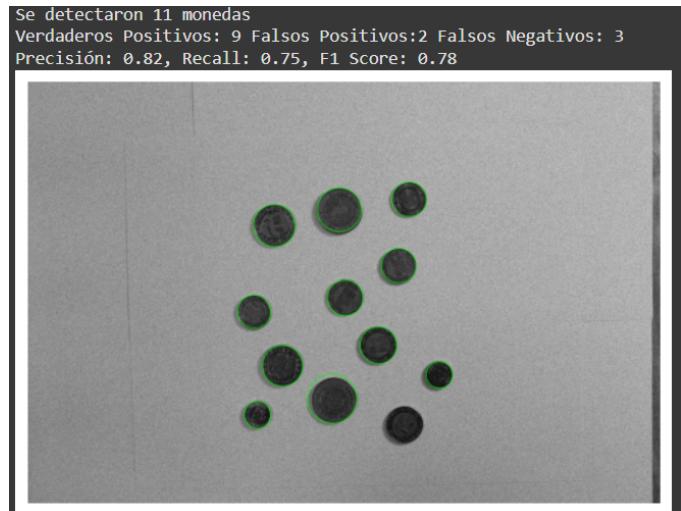
$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **F1 Score:** Es la media de precisión y exhaustividad. Es útil si buscamos un balance entre las dos métricas.

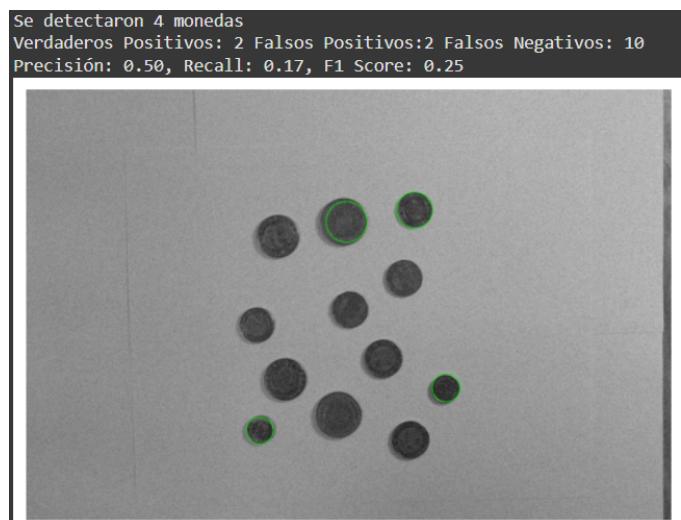
$$F1 = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Ahora pasando al código modificado de esta práctica, le daremos las imágenes con los filtros/enfocado y desenfocado y compararemos estos valores para observar el desempeño.

Foco Speckle:



Foco SaltPepper:

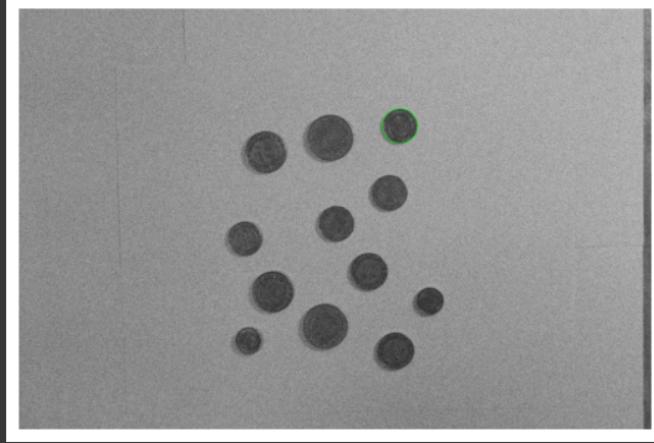


Foco Gaussian:

Se detectaron 1 monedas

Verdaderos Positivos: 1 Falsos Positivos:0 Falsos Negativos: 11

Precisión: 1.00, Recall: 0.08, F1 Score: 0.15



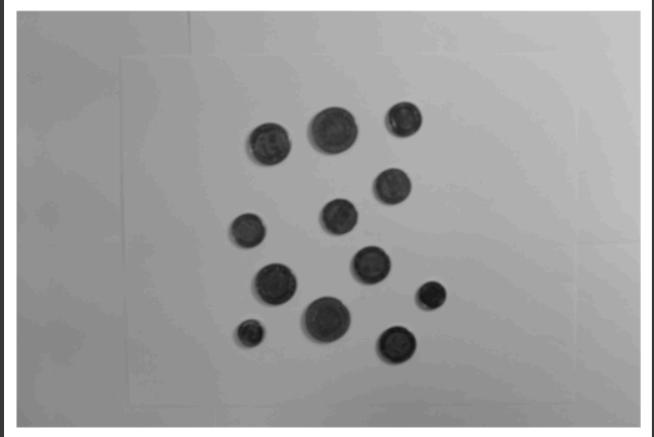
Ahora utilizaremos el segundo conjunto de imágenes:

No Foco:

Se detectaron 0 monedas

Verdaderos Positivos: 0 Falsos Positivos:0 Falsos Negativos: 12

Precisión: 0.00, Recall: 0.00, F1 Score: 0.00

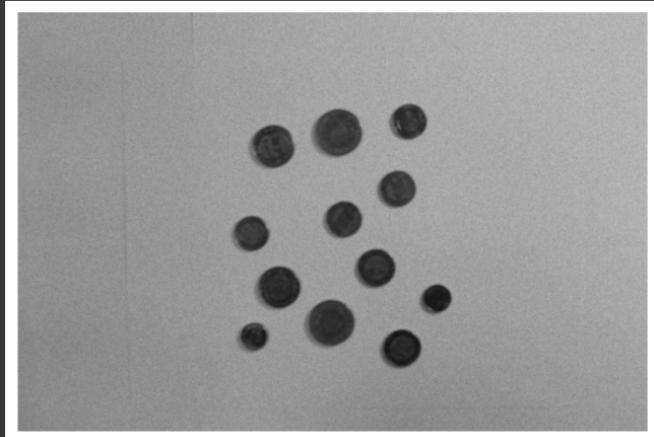


No Foco speckle:

Se detectaron 0 monedas

Verdaderos Positivos: 0 Falsos Positivos:0 Falsos Negativos: 12

Precisión: 0.00, Recall: 0.00, F1 Score: 0.00

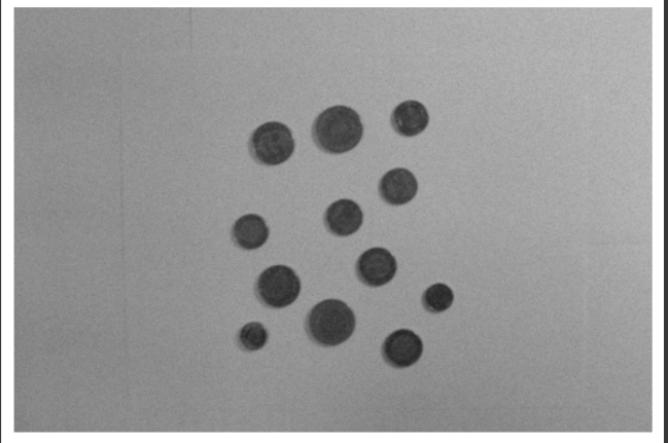


No Foco saltpepper:

Se detectaron 0 monedas

Verdaderos Positivos: 0 Falsos Positivos:0 Falsos Negativos: 12

Precisión: 0.00, Recall: 0.00, F1 Score: 0.00

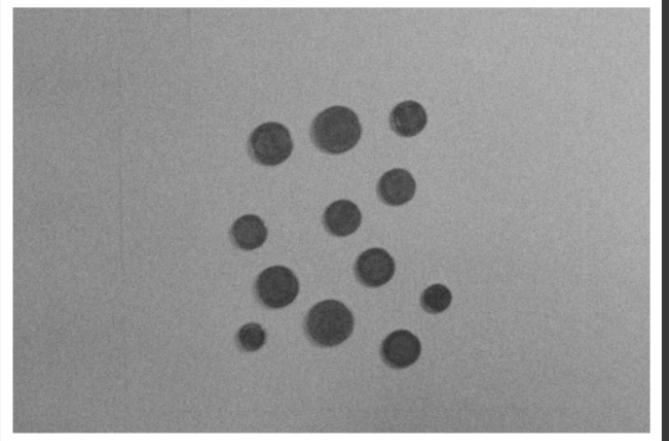


No Foco gaussian:

Se detectaron 0 monedas

Verdaderos Positivos: 0 Falsos Positivos:0 Falsos Negativos: 12

Precisión: 0.00, Recall: 0.00, F1 Score: 0.00



III. CÓDIGOS

En este apéndice, se anexará el contenido del código para los ejercicios desarrollados en la presente práctica.

Código:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def detectar_monedas(imagen_path,
                     ground_truth, tolerancia=10):
    # Cargar la imagen en escala de grises
```

```

        imagen = cv2.imread(imagen_path,
cv2.IMREAD_GRAYSCALE)

        # Aplicar suavizado para reducir
ruido

        imagen_suavizada = cv2.GaussianBlur(imagen, (9, 9), 2)

        # Aplicar el detector de bordes de
Canny

        bordes = cv2.Canny(imagen_suavizada, 100, 200)

        # Aplicar la Transformada de Hough
para detectar círculos

        monedas = cv2.HoughCircles(
            bordes,
            cv2.HOUGH_GRADIENT,
            dp=2,
            minDist=100,
            param1=75,
            param2=65,
            minRadius=0,
            maxRadius=200
        )

        # Dibujar los círculos detectados y
contarlos

        salida = cv2.cvtColor(imagen,
cv2.COLOR_GRAY2BGR)
        detecciones = []
        if monedas is not None:
            monedas = np.round(monedas[0,
:]).astype("int")
            for (x, y, r) in monedas:
                detecciones.append((x, y,
r))
        # Dibujar el círculo
        cv2.circle(salida, (x, y),
r, (0, 255, 0), 4)

```

```

# Dibujar el centro del
círculo
cv2.circle(salida, (x, y),
2, (0, 0, 255), 3)
print(f"Se detectaron
{len(detecciones)} monedas")

# Evaluar la detección de monedas
precision, recall, f1 =
evaluar_deteccion(detecciones,
ground_truth, tolerancia)
print(f"Precisión: {precision:.2f},
Recall: {recall:.2f}, F1 Score:
{f1:.2f}")

# Mostrar la imagen con los
círculos detectados
plt.imshow(cv2.cvtColor(salida,
cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.show()

def evaluar_deteccion(detecciones,
ground_truth, tolerancia=10):
    """
    Evalúa la detección de monedas.

    Parámetros:
        - detecciones: Lista de círculos
detectados [(x, y, radio)].
        - ground_truth: Lista de círculos
verdaderos [(x, y, radio)].
        - tolerancia: Tolerancia en píxeles
para considerar una detección como
correcta.

    Retorna:
        - Precisión, Exhau...
    """
    TP, FP, FN = 0, 0, 0

```

```

        # Para cada círculo verdadero,
verificamos si fue detectado
    for gt in ground_truth:
        encontrado = False
        for det in detecciones:
            distancia = np.sqrt((gt[0]
- det[0]) * 2 + (gt[1] - det[1]) * 2)
            diferencia_radio =
abs(gt[2] - det[2])

                # Si la distancia entre
centros y el radio son menores que la
tolerancia, cuenta como TP
            if distancia <= tolerancia
and diferencia_radio <= tolerancia:
                TP += 1
                encontrado = True
                break
            if not encontrado:
                FN += 1
        print(f"Verdaderos Positivos: {TP}
Falsos Positivos:{FP} Falsos Negativos:
{FN}" )

            # Cada detección que no se
encuentra en el ground truth es un FP
    FP = len(detecciones) - TP

    # Calcular métricas
    precision = TP / (TP + FP) if (TP +
FP) > 0 else 0
    recall = TP / (TP + FN) if (TP +
FN) > 0 else 0
    f1 = 2 * precision * recall /
(precision + recall) if (precision +
recall) > 0 else 0

    return precision, recall, f1

# Ejemplo de uso:
```

```

# Supongamos que tenemos una lista de
ground truth para la imagen
ground_truth = [(1715, 1919, 138),
(2099, 875, 158), (2363, 1779, 126),
(1663, 973, 143), (2579, 793, 114),
(1531, 1555, 115), (2067, 2147, 148),
(1555, 2249, 92), (2499, 1247, 122),
(2547, 2317, 121), (2779, 1981, 89),
(2149, 1457, 115)] # Coordenadas y
radios de las monedas reales
detectar_monedas("IMG_B_Foco.JPG",
ground_truth)
```

REFERENCES

- [1] colaboradores de Wikipedia. (2024, 22 junio). Transformada de Hough. Wikipedia, la Encyclopædia Libre. https://es.wikipedia.org/wiki/Transformada_de_Hough
- [2] Datahacker.Rs. (2019, 26 octubre). OpenCV #010 Circle Detection using Hough Transform. Master Data Science. <https://datahacker.rs/opencv-circle-detection-hough-transform/>
- [3] Del Valle Hernández, L. (2021, 1 diciembre). Detector de bordes Canny, cómo contar objetos con OpenCV y Python. Programafacil Arduino y Home Assistant. <https://programafacil.com/blog/vision-artificial/detector-de-bordes-canny-opencv/>