Ejercicio 1

En este código la principal novedad es que hemos empleado los códigos anteriores para crear funciones, por ejemplo, las sentencias de la función sobel son tal cual las de los ejercicios del tema 7, en este caso del ejemplo 1. En este caso, hemos dejado los valores por defecto ya usados anteriormente. Usamos un low y un high como umbrales en la histéresis, y así evitar la detección de bordes falsos.

La función canny es bastante similar a la que podemos ver en el ejemplo 1 del tema 6, y de igual forma, con la función mostrar, en la que hemos realizado solo unos pequeños ajustes para que al iterar sobre el bucle, si el valor i es par muestre la imagen que le pedimos, y en caso contrario, muestre la matriz correspondiente a esa imagen.

```
def canny(image):
    mapa_bordes = ski.feature.canny(image, sigma=3)
    resultado3 = mapa_bordes

    # Transformada de Hough Probabilística y Progresiva
    segmentos = ski.transform.probabilistic_hough_line(mapa_bordes, threshold=16, line_length=8,
                                                       line_gap=6)  # threshold=10, line_length=5, line_gap=3
```

En esta función, hemos añadido una sigma para controlar la desviación estándar del filtro Gaussiano y, así poder controlar la suavidad de la imagen. Además de esto, también hemos modificado los valores de la transformada Hough para intentar lograr una línea lo más uniforme posible. Threshold determina la sensibilidad del algoritmo para detectar líneas, cuanto más alto sea el valor, mayor precisión. Line length es la longitud mínima para considerar una línea válida. Por último, line gap sirve para agrupar puntos como la misma línea cuando la distancia entre puntos sea menor que la indicia en el parámetro.

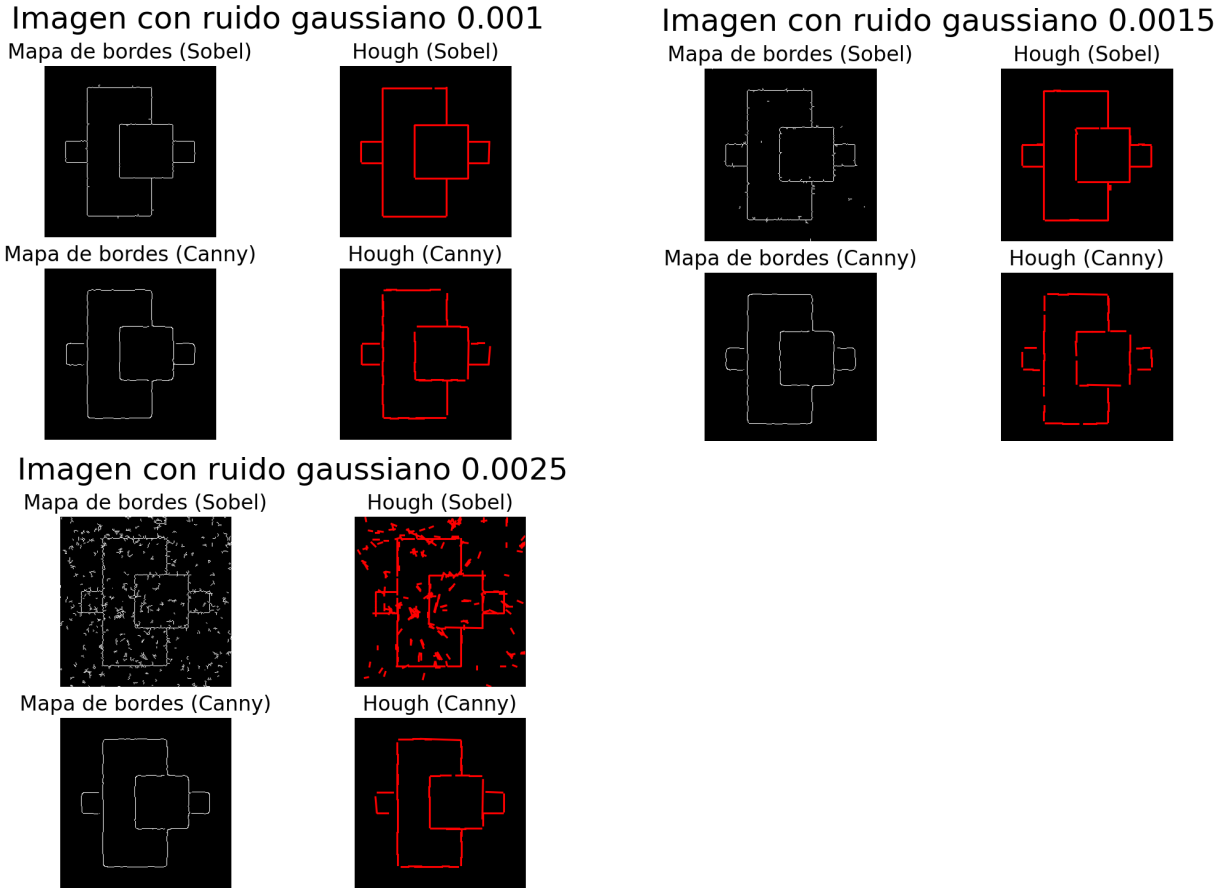
Sin embargo, la principal diferencia se encuentra en el siguiente tramo de código:

```
for idx, img_noise in enumerate(gaussian_noise):
    if idx == 0: continue
    # Resultados para Sobel
    sobel_results = sobel(img_noise)
    sobel_msg = ["Mapa de bordes (Sobel)", "Hough (Sobel)"]

    # Resultados para Canny
    canny_results = canny(img_noise)
    canny_msg = ["Mapa de bordes (Canny)", "Hough (Canny)"]

    tablas = [sobel_results[0], sobel_results[1], canny_results[0], canny_results[1]]
    for x in tablas:
        print(x)

    # Mostrar resultados
    mostrar(sobel_results, canny_results, sobel_msg, canny_msg,
            titulo: f"Imagen con ruido gaussiano {gaussian_values[idx - 1]}")
```

Este fragmento de código itera sobre una lista de imágenes con ruido gaussiano. Para cada imagen, calcula los resultados de los operadores de detección de bordes Sobel, su

transformada de Hough, Canny y su correspondiente transformada de Hough. Posteriormente, definimos lo que queremos mostrar por pantalla encima de cada una de esas imágenes, y se agrupan los resultados en una lista de tablas. Luego, imprime estas tablas. Finalmente, muestra los resultados utilizando una función llamada mostrar, pasando los resultados de Sobel y Canny, los mensajes correspondientes, y un título que indica el ruido gaussiano presente en cada imagen.

Los resultados obtenidos son las tablas y las siguientes imágenes:



Viendo estos resultados, podemos afirmar que el más afectado por el ruido será el mapa de bordes Sobel. Y aunque los bordes de Canny sean más redondeados, soportan mejor el ruido.

Ejercicio 2

Este código es prácticamente el mismo que el que tenemos en los ejemplos, tan solo hemos añadido las siguientes cosas.

```
image1 = ski.util.random_noise(image1, mode="gaussian", var=0.001)

images = []
angles = [0, 22.5, 45, 67.5, 90]
for angle in angles:
    images.append(ski.transform.rotate(image1, angle=angle))
```

Primero, hemos añadido ruido gaussiano a la imagen y también, al tener que rotar las imágenes, hemos creado una lista con los ángulos a los que pretendemos girar la imagen y luego lo hemos aplicado.

```
filtros = ["kitchen_rosenfeld", "foerstner", "moravec", "harris", "fast"]
nombres = ["0°", "22.5°", "45°", "67.5°", "90°"]

umbral = [0.85, 0.05, 0.07, 0.005, 0.045]
for id, image in enumerate(images):
    images1 = filtrar(image, filtros)
    picos1 = detectar_picos(images1, umbral)
    mostrar(titulo: f"Imagen ruidosa rotada {nombres[id]} (umbrales usados = {umbral})", picos1, [nombres[id]] + filtros)
```

En esta parte, hemos definido una lista de umbrales asociados a cada filtro. Luego, se itera sobre una lista de imágenes y para cada imagen se aplican los filtros de detección de bordes. Se detectan los picos de los bordes en las imágenes filtradas y se muestran visualmente los resultados de la detección de bordes para cada imagen, con la rotación y umbrales concretos.

En cuanto a los resultados, cabe decir que no todos los umbrales funcionan igual en cada filtro, por lo que tras muchos intentos, hemos llegado a una serie de umbrales que van bastante bien en todas las rotaciones:

```
kitchen_rosenfeld = ninguno termina de ir bien, foerstner = 0.05,
moravec = 0.07, harris = 0.005, fast = 0.05
```

El filtro kitchen no es especialmente bueno, pues aun habiendo probado multitud de umbrales o detecta prácticamente todos los segmentos, no solo los puntos o detecta puntos muy escasos, hemos escogido en este caso un umbral de 0.85 aunque faltarían varios bordes.

En el ángulo de 45º todos los filtros funcionan perfectamente, sin embargo, en el caso de moravec depende el ángulo detecta mejor o peor. No obstante, los filtros restantes, foerstner, fast y harris funcionan realmente bien en todos los ángulos, aunque los umbrales de los dos primeros son los mismos, en el caso de harris es mucho menor.

Ejercicio 3

Este código tampoco ha sido muy diferente al de los ejercicios anteriores o incluso de los ejemplos de clase. El principal reto ha sido el determinar los umbrales de los radios, así como calcular los radios, y una vez obtenido esto era muy similar a lo que ya teníamos.

```python
def detector_contornos(image):
    mapa_bordes = ski.feature.canny(image, sigma=3)

    # Transformada de Hough para círculos
    radios_posibles = np.arange(16, 25, 1)  # Buscará círculos con radios entre 8 y 14 de 1 en 1
    hough_res = ski.transform.hough_circle(mapa_bordes, radios_posibles)
    accums, cx, cy, radii = ski.transform.hough_circle_peaks(hough_res, radios_posibles, min_xdistance=10,
                                                             min_ydistance=10,
                                                             threshold=hough_res.max() / 2)  # El threshold

    # Dibujar los círculos en la imagen resultado
    resultado = np.zeros(image.shape + (3,))
    for fila, col, radio in zip(cy, cx, radii):
        if radio > 21:
            circy, circx = ski.draw.circle_perimeter(fila, col, radio, shape=image.shape)  # Dibuja un círc
            resultado[circy, circx] = [1, 0, 0]  # Moneda de 1 euro en rojo
        else:
            circy, circx = ski.draw.circle_perimeter(fila, col, radio, shape=image.shape)  # Dibuja un círc
            resultado[circy, circx] = [0, 1, 0]  # Moneda de 10 céntimos en verde

    return resultado
```

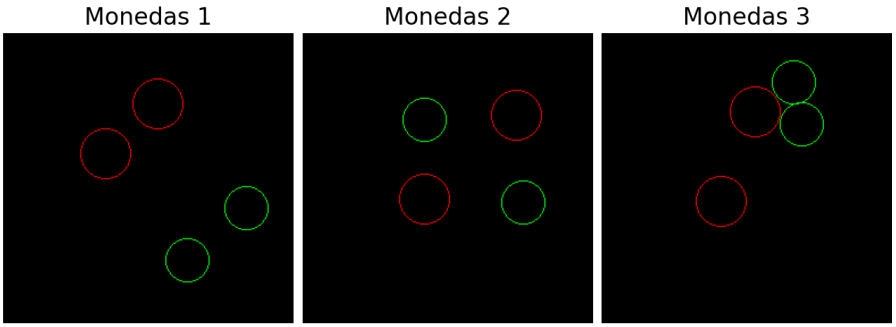
En este caso he realizado el preprocesamiento de la imagen con el detector de bordes Canny, pues como he mencionado anteriormente, la obtención del mapa de bordes con este método me ha resultado la mejor ante ruidos externos y cómo generaba bordes circulares y hablábamos de monedas, lo he visto más apropiado. Como parámetro tenía la imagen y sigma para controlar la suavidad de la imagen.

Luego hemos usado el código proporcionado en el ejemplo 2 del tema 7 para calcular la transformada de Hough para círculos, habiendo calculado el valor de cada radio. Esto se hacía con una simple regla de 3, si 50 puntos por pulgada equivale a 25,4 mm de la imagen, si queremos obtener el radio de la moneda de euro, haremos la operación de la imagen, de igual forma con el otro radio. Una vez obtenido este valor, lo dividimos entre 2 para obtener el radio y no el diámetro.

$$x = \frac{50 \cdot 23}{25,4}$$

Por último, hemos calculado los círculos en la imagen resultante, inicializando primero una matriz de ceros pero con tres canales de color. Luego al realizar el ***if*** hemos dado un valor entre medias de los 2 radios de modo que las monedas de 1 euro se van rojas y las de 10 cts verdes.

Contornos de las monedas detectadas



Estos son los resultados obtenidos. Como podemos ver se ha logrado el objetivo del ejercicio.

Estudiante: Irene Mejía Hurtado
Profesor: Pedro García Sevilla