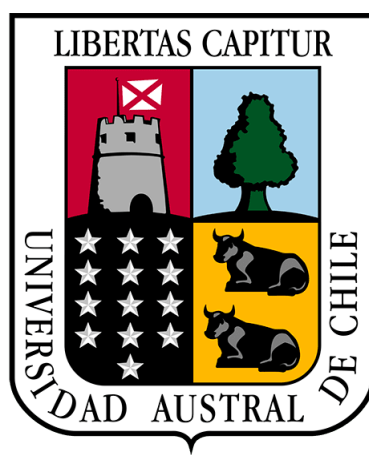

UNIVERSIDAD AUSTRAL DE CHILE

FACULTAD DE CIENCIAS DE LA INGENIERÍA

INGENIERÍA CIVIL ELECTRÓNICA



ELEP 233 - VISIÓN ARTIFICIAL Y REDES NEURONALES

DESARROLLO DE TAREA N°3

PROFESOR:

Gustavo Schleyer.

INTEGRANTES:

Ángel Andrade

Jorge Palavecino.

18 de mayo de 2023

Índice general

Pregunta 1	2
Pregunta 2	5
Pregunta 3	6
Pregunta 4	8
Pregunta 5	10
Pregunta 6	14
Pregunta 7	14

El presente documento desarrolla la tarea 3 del ramo ELEL233. El repositorio en GitHub de la actividad puede encontrarse en <https://github.com/Atrabilis/UACH/tree/main/Trabajos/Vision%20artificial/Tarea%203>.

Las imágenes originales utilizadas para el desarrollo de la tarea fueron las siguientes:

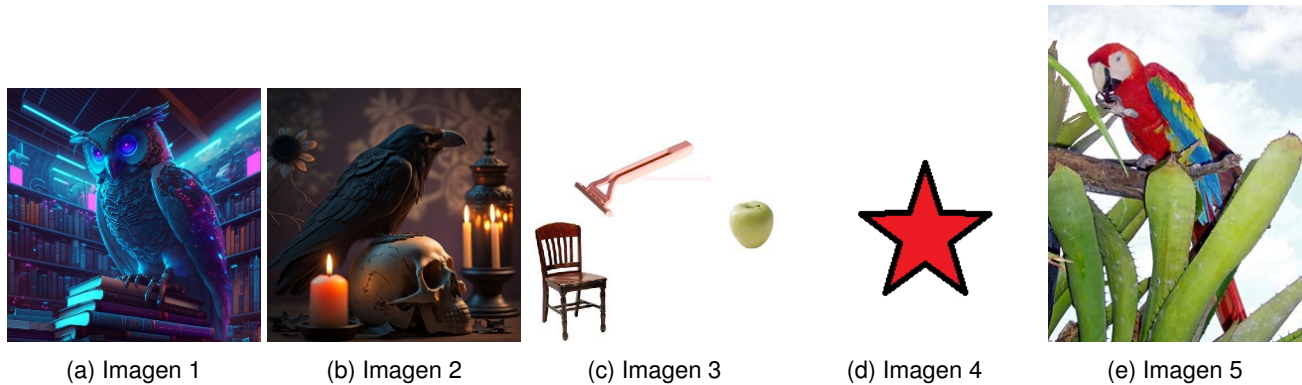


Figura 1: Imágenes de prueba utilizadas en el trabajo.

Pregunta 1

El programa "Tarea 3 P1.py" realiza un análisis de histograma en una imagen y calcula un umbral para generar una imagen binaria. Sin embargo, el cálculo del umbral propuesto en el programa puede no ser óptimo en todos los casos y puede haber una alternativa más precisa y eficiente.

Una alternativa para mejorar el cálculo del umbral es utilizar el método de Otsu (Propuesto en el problema), el cual es una técnica de umbralización automática que busca encontrar el umbral óptimo que minimiza la varianza intraclase de los niveles de gris en la imagen.

Además del método de Otsu, existen otros métodos comunes para calcular automáticamente el umbral en imágenes:

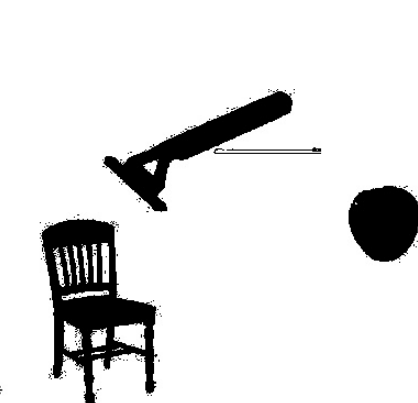
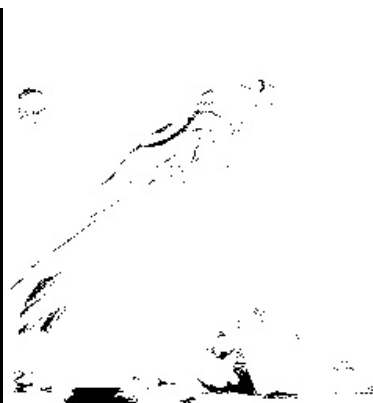
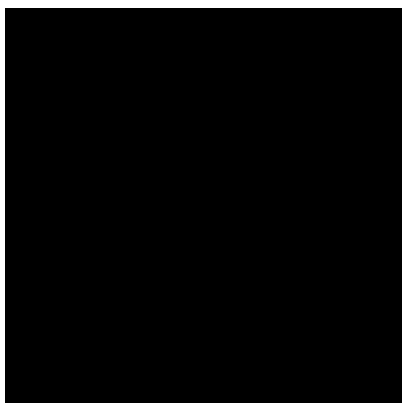
- Método de la entropía: Este método busca encontrar el umbral que maximiza la entropía de la imagen. La entropía mide la incertidumbre o la información promedio en una imagen. Se utiliza para encontrar un umbral que genere una imagen binaria con la máxima información.

- Método de Kapur: También conocido como el método de la entropía máxima, se basa en el concepto de entropía de la información para calcular el umbral óptimo. Busca maximizar la diferencia entre las entropías de las clases de píxeles divididos por el umbral.

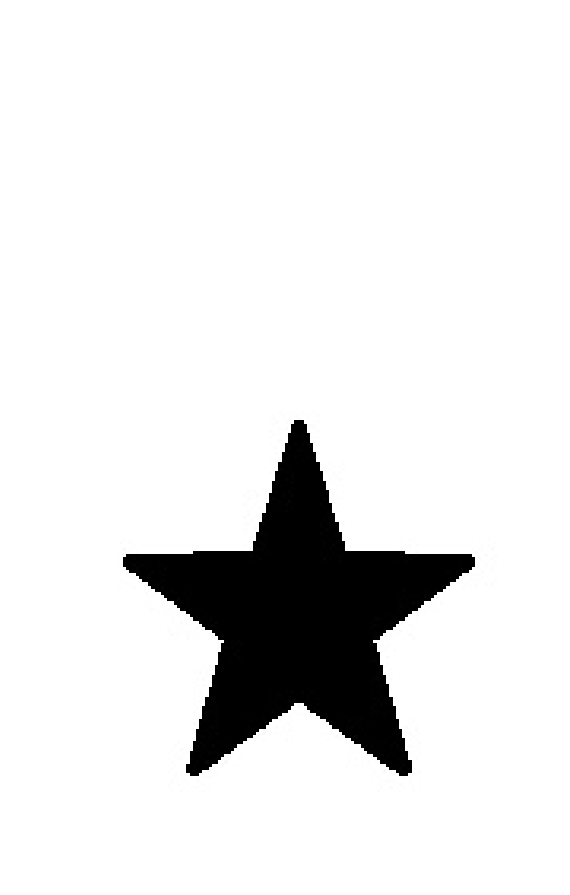
- Método de la varianza mínima de interclase: Este método busca encontrar el umbral que minimiza la varianza entre las clases de píxeles (objeto y fondo) en la imagen. Busca una separación óptima entre los objetos de interés y el fondo.

- Método del percentil: Este método utiliza el análisis de los percentiles de los valores de intensidad de los píxeles. Selecciona el umbral en función de un percentil específico (por ejemplo, el 1 % o el 99 %) de los valores de intensidad.

La prueba del programa con distintas imágenes arroja el siguiente resultado:



(a) Programa aplicado a la imagen 1a (b) Programa aplicado a la imagen 1b (c) Programa aplicado a la imagen 1c



(d) Programa aplicado a la imagen 1c

(e) Programa aplicado a la imagen 1d

Figura 2: Programa aplicado a las imágenes de referencia.

Se puede apreciar que en las imágenes con poco contraste entre fondo y objeto el algoritmo tiende a fallar. Utilizando el método de Otsu propuesto:



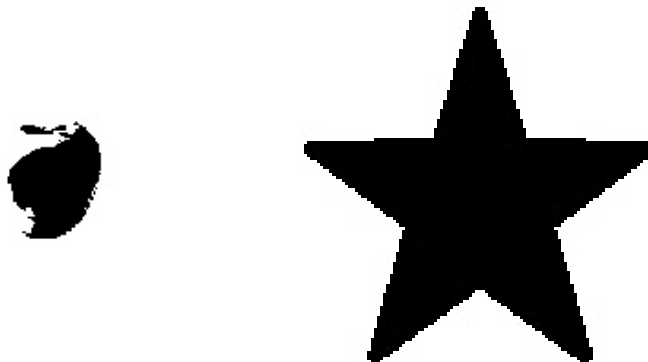
(a) Método de Otsu aplicado a la imagen 1a

(b) Método de Otsu aplicado a la imagen 1b

(c) Método de Otsu aplicado a la imagen 1e



(d) Método de Otsu aplicado a la imagen 1c



(e) Método de Otsu aplicado a la imagen 1d

Figura 3: Método de Otsu aplicado a las imágenes de referencia.

Este umbral es mucho mas sensible y menos radical a la hora de binarizar, como puede observarse cuando se aplica a 1a y 1b.

Pregunta 2

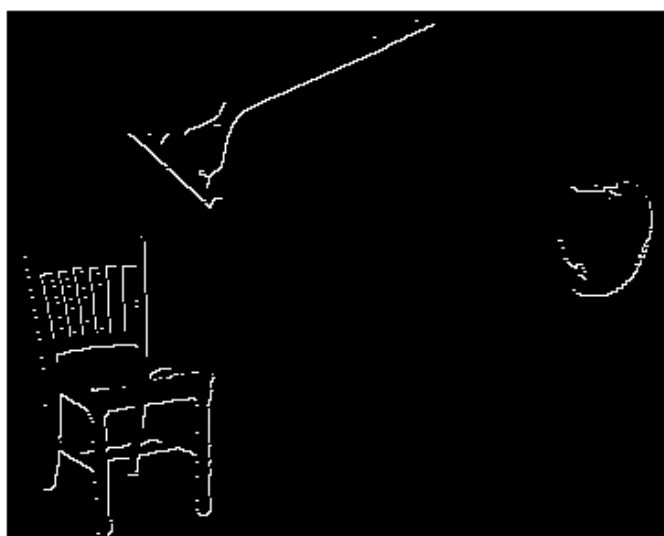
Este apartado pide implementar el operador Roberts en Python. Los resultados son los siguientes:



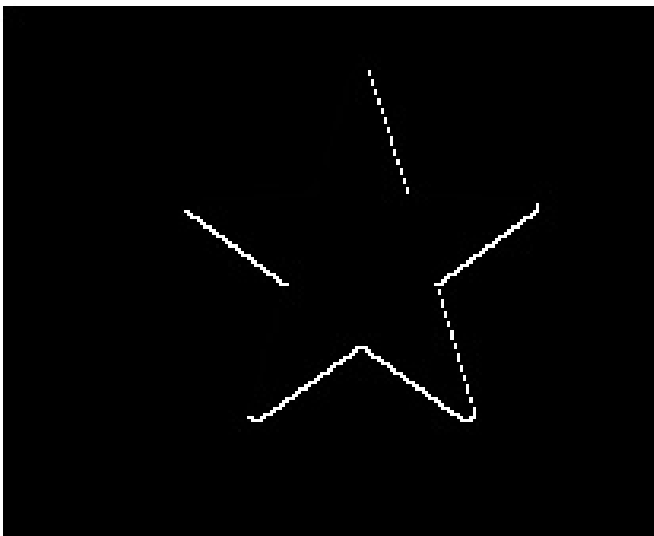
(a) Operador Roberts aplicado a la imagen 1a

(b) Operador Roberts aplicado a la imagen 1b

(c) Operador Roberts aplicado a la imagen 1e



(d) Operador Roberts aplicado a la imagen 1c



(e) Operador Roberts aplicado a la imagen 1d

Figura 4: Operador Roberts aplicado a las imágenes de referencia.

Pregunta 3

Este apartado pide implementar el operador Frei-Chen y Prewitt en Python. Los resultados son los siguientes:



(a) Operador Frei-Chen aplicado a la imagen 1a

(b) Operador Frei-Chen aplicado a la imagen 1b

(c) Operador Frei-Chen aplicado a la imagen 1e



(d) Operador Frei-Chen aplicado a la imagen 1c

(e) Operador Frei-Chen aplicado a la imagen 1d

Figura 5: Operador Frei-Chen aplicado a las imágenes de referencia.



(a) Operador Prewitt aplicado a la imagen 1a

(b) Operador Prewitt aplicado a la imagen 1b

(c) Operador Prewitt aplicado a la imagen 1e



(d) Operador Prewitt aplicado a la imagen 1c

(e) Operador Prewitt aplicado a la imagen 1d

Figura 6: Operador Prewitt aplicado a las imágenes de referencia.

Pregunta 4

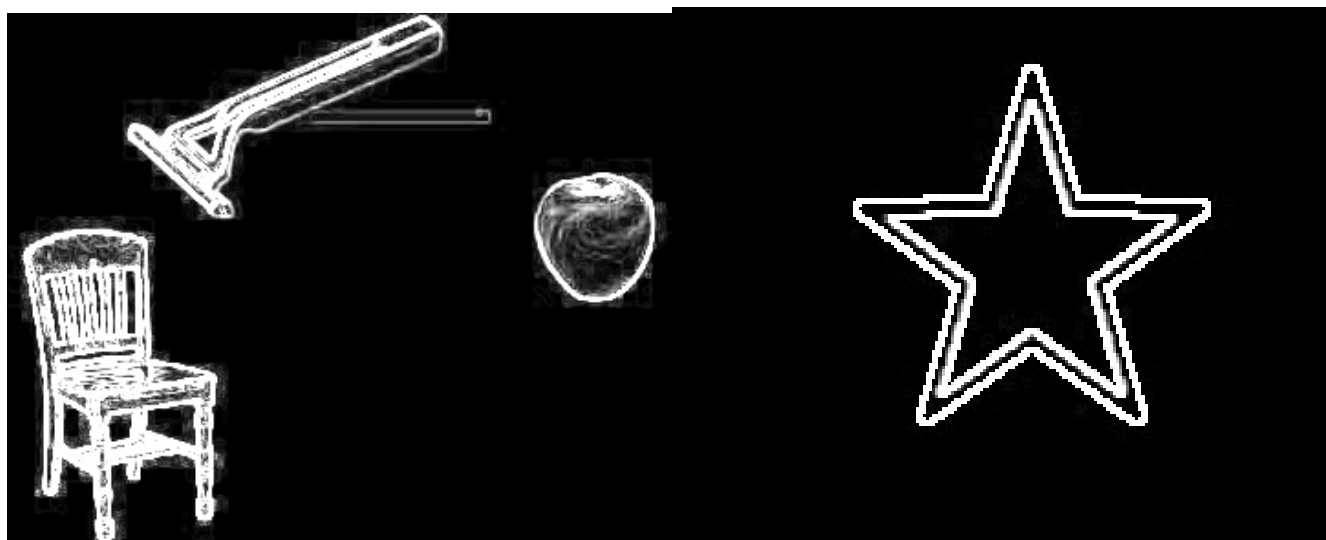
Este apartado pide implementar el operador Kirsch y Robinson en Python. Los resultados son los siguientes:



(a) Operador Kirsch aplicado a la imagen 1a

(b) Operador Kirsch aplicado a la imagen 1b

(c) Operador Kirsch aplicado a la imagen 1e



(d) Operador Kirsch aplicado a la imagen 1c

(e) Operador Kirsch aplicado a la imagen 1d

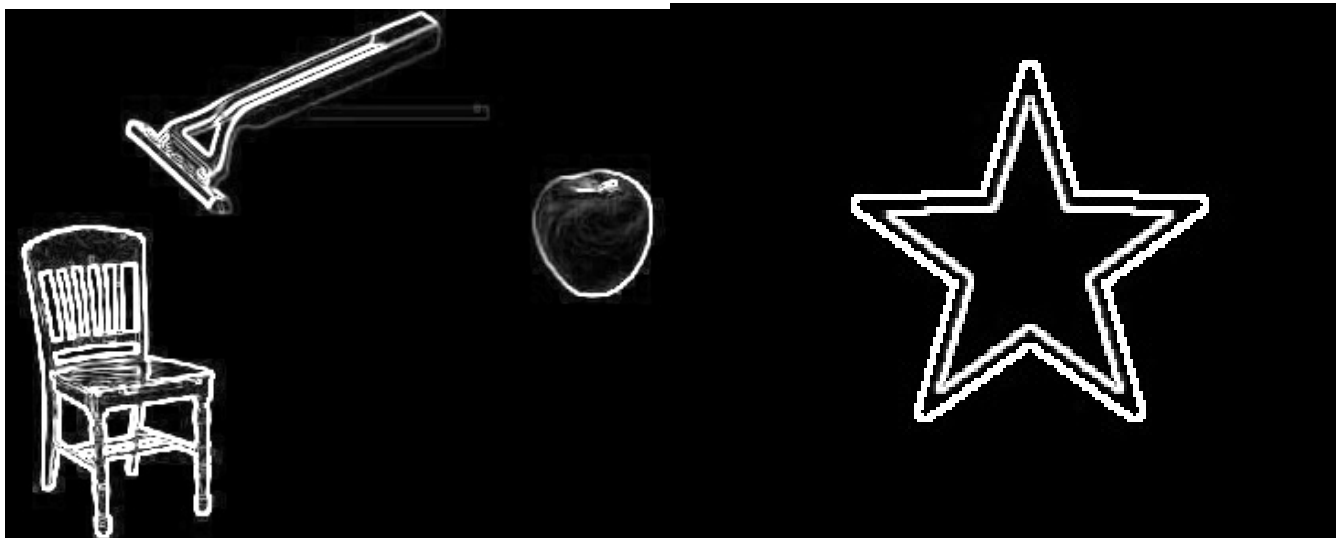
Figura 7: Operador Kirsch aplicado a las imágenes de referencia.



(a) Operador Robinson aplicado a la imagen 1a

(b) Operador Robinson aplicado a la imagen 1b

(c) Operador Robinson aplicado a la imagen 1e



(d) Operador Robinson aplicado a la imagen 1c

(e) Operador Robinson aplicado a la imagen 1d

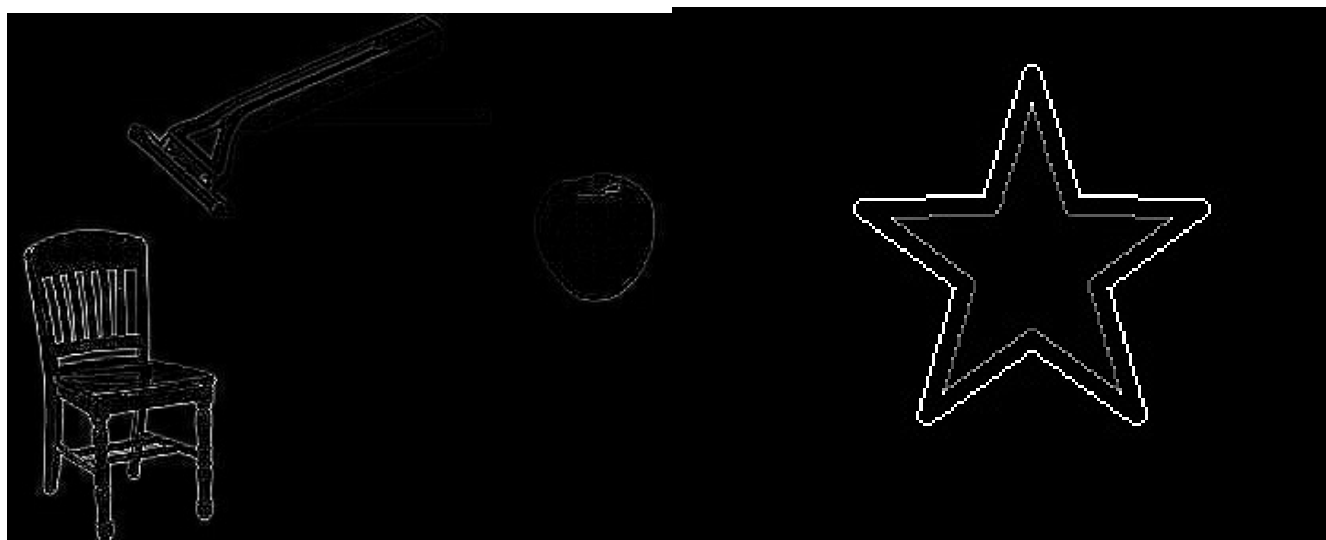
Figura 8: Operador Robinson aplicado a las imágenes de referencia.

Pregunta 5

El objetivo de este apartado es implementar los operadores con 3 máscaras Laplacianas. Los resultados son los siguientes:

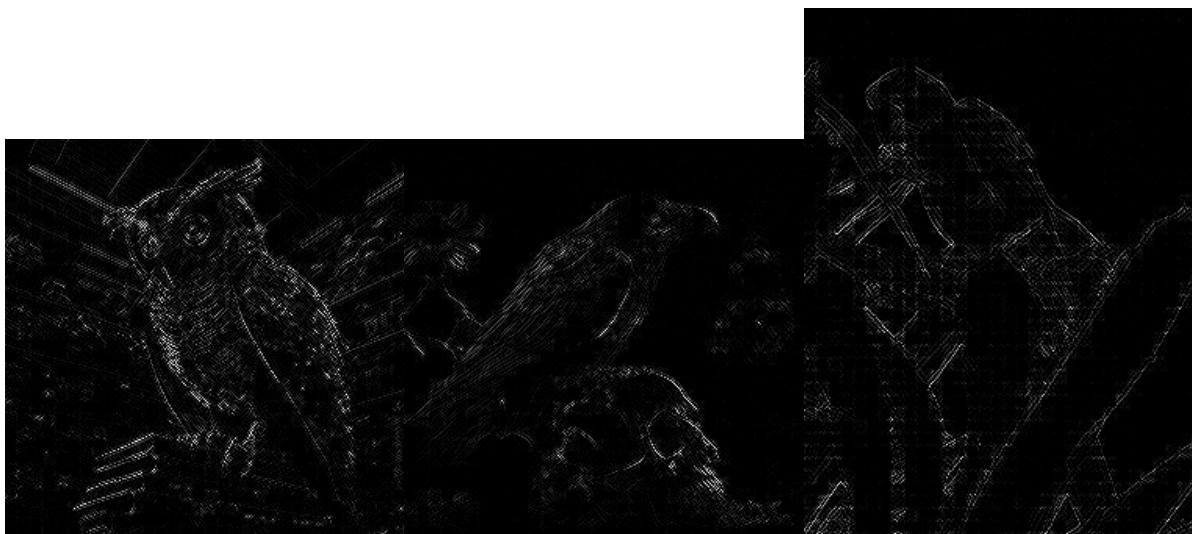


(a) Operador Laplaciano con la máscara 1 aplicado a la imagen 1a (b) Operador Laplaciano con la máscara 1 aplicado a la imagen 1b (c) Operador Laplaciano con la máscara 1 aplicado a la imagen 1c

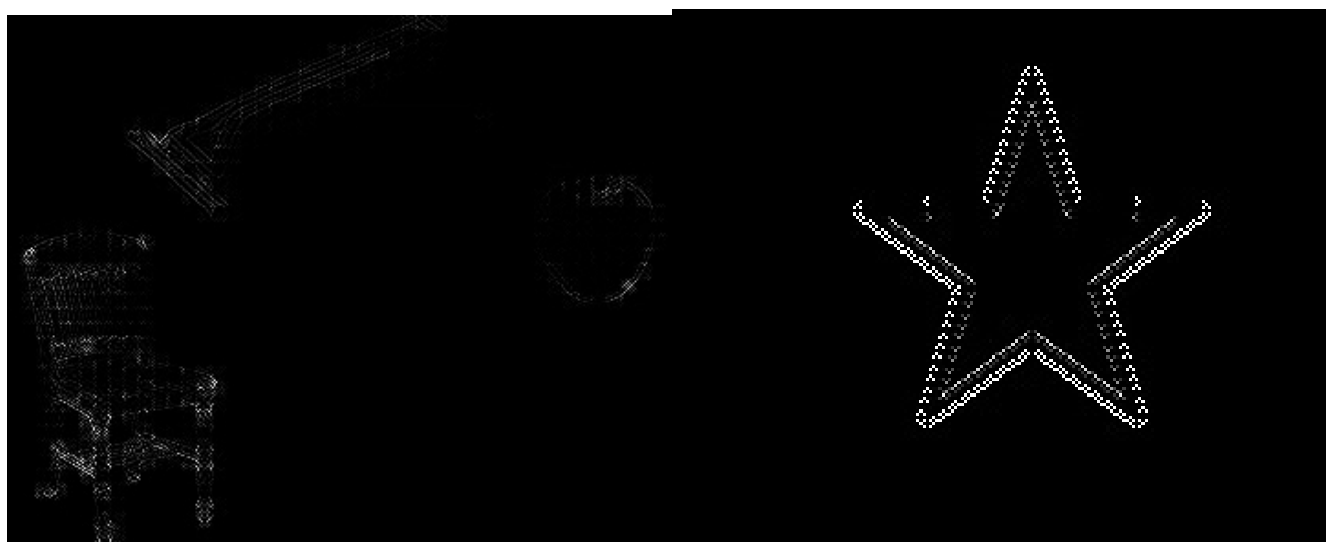


(d) Operador Laplaciano con la máscara 1 aplicado a la imagen 1c (e) Operador Laplaciano con la máscara 1 aplicado a la imagen 1d

Figura 9: Operador Laplaciano con la máscara 1 aplicado a las imágenes de referencia.



(a) Operador Laplaciano con la máscara 2 aplicado a la imagen 1a (b) Operador Laplaciano con la máscara 2 aplicado a la imagen 1b (c) Operador Laplaciano con la máscara 2 aplicado a la imagen 1e

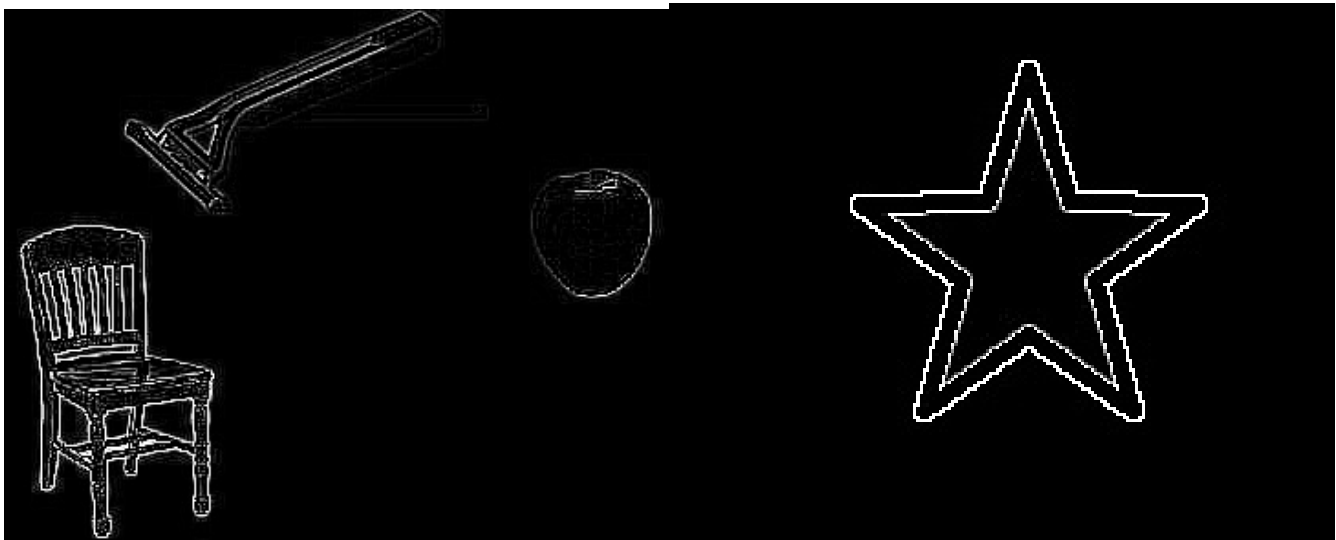


(d) Operador Laplaciano con la máscara 2 aplicado a la imagen 1c (e) Operador Laplaciano con la máscara 2 aplicado a la imagen 1d

Figura 10: Operador Laplaciano con la máscara 2 aplicado a las imágenes de referencia.



(a) Operador Laplaciano con la máscara 3 aplicado a la imagen 1a (b) Operador Laplaciano con la máscara 3 aplicado a la imagen 1b (c) Operador Laplaciano con la máscara 3 aplicado a la imagen 1c



(d) Operador Laplaciano con la máscara 3 aplicado a la imagen 1d (e) Operador Laplaciano con la máscara 3 aplicado a la imagen 1e

Figura 11: Operador Laplaciano con la máscara 3 aplicado a las imágenes de referencia.

Las máscaras son las siguientes:

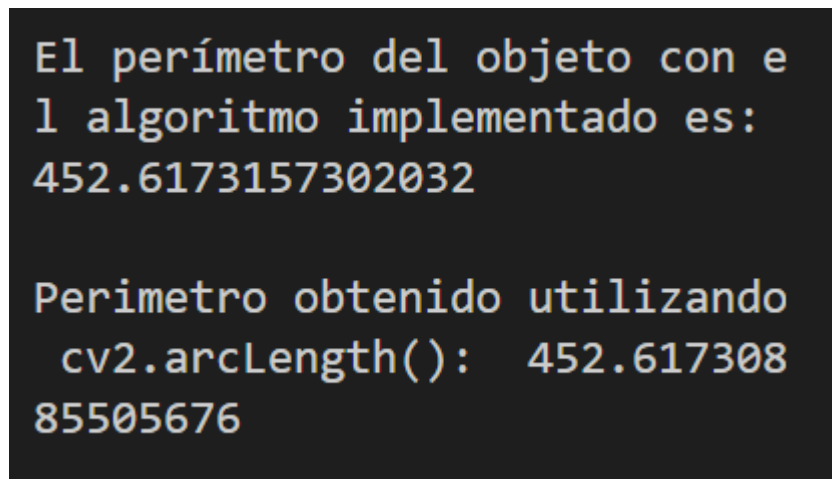
$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad (1)$$

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad (2)$$

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \quad (3)$$

Pregunta 6

Este apartado requiere calcular el perímetro de un objeto en una imagen. La imagen de referencia utilizada para este apartado fue la imagen de la figura 1d. Los resultados son los siguientes:



```
El perímetro del objeto con el
algoritmo implementado es:
452.6173157302032

Perímetro obtenido utilizando
cv2.arcLength(): 452.617308
85505676
```

Figura 12: Comparación entre el algoritmo implementado y el de openCV

La diferencia mínima puede deberse a :

1. Método de aproximación de la curva: El método de cálculo de perímetro utilizando la función `cv2.arcLength()` utiliza un enfoque de aproximación de la curva mediante técnicas de polígonos o curvas suavizadas. Esto implica que la función puede realizar ciertas suposiciones o aproximaciones para obtener un perímetro más preciso.
2. Precisión de los cálculos internos: La función `cv2.arcLength()` utiliza algoritmos y cálculos internos optimizados en OpenCV. Estos cálculos internos pueden incluir manejo de puntos flotantes de alta precisión, corrección de errores de redondeo y métodos de optimización numérica.

Pregunta 7

Este apartado requiere calcular el centro de masa de un objeto en una imagen. La imagen de referencia utilizada para este apartado fue la imagen de la figura 1d. Los resultados son los siguientes:

```
Centro de masa obtenido media  
nte moments de OpenCV: (104,  
84)  
Centro de masa calculado manu  
almente: (104, 84)
```

Figura 13: Comparación entre el algoritmo implementado y el de openCV

A pesar de no observarse diferencias entre un calculo y otro, existen ciertas diferencias técnicas entre una implementación y otra:

- Método cv2.moments de OpenCV:

1. Utiliza los momentos de la imagen para calcular el centro de masa del objeto.
2. La función cv2.moments de OpenCV calcula automáticamente los momentos necesarios.
3. Proporciona resultados precisos y considera el área del objeto (momento m00) para calcular las coordenadas del centro de masa.
4. Este método es más robusto y adecuado para casos donde el objeto puede tener formas irregulares o contornos complejos.

- Cálculo manual del centro de masa:

1. En este enfoque, se calcula el centro de masa manualmente utilizando el promedio de las coordenadas x y y de los puntos del contorno del objeto.
2. Se asume que los puntos del contorno están almacenados en una matriz bidimensional.
3. Este método es más simple pero puede ser menos preciso que el método de cv2.moments ya que no tiene en cuenta el área del objeto.
4. No considera otros momentos de la imagen, como momentos centrados o momentos de orden superior, que pueden proporcionar más información sobre la forma del objeto.