Laboratorio de OpenCV

Asignatura: Informática Gráfica y Visualización
Datos del alumno: VICTOR PLANAS ORTEGA
Fecha: 12/6/2025

Objetivo:

El ejercicio tiene como finalidad comparar los algoritmos Canny, Sobel y Laplaciano en diferentes tipos de imágenes, específicamente aquellas que presentan características distintivas como ruido, texturas complejas y objetos con contornos bien definidos.

Se ha creado una interfaz basada en trackbars que permite la modificación en tiempo real de parámetros, incluyendo la selección de imagen, tipo de preprocesamiento (blurring), algoritmo de detección, valores de umbralización y un 'switch' para seleccionar si se quiere invertir la imágen binaria con los bordes (blanco sobre negro o negro sobre blanco.

Imports de las librerías necesarias

In [1]:
```
import numpy as np
import cv2
import sys
```

Defición del callback para los trackbars y de variable para controlar si es necesario rehacer cálculos

Esta función se usa en los trackbars. Cuando se modifica un valor de un trackbar la variable needs_update se pone a true. En el bucle principal de la aplicación se mira el valor de esta variable para ver si es necesario o no rehacer los cálculos.

In [2]:
```
# Variable global para controlar cuándo recalcular
needs_update = True

def trackbar_changed(val):
    # Callback que se ejecuta cuando cualquier trackbar cambia
    global needs_update
    needs_update = True
```

Definición de la función setTrackbars

Esta función se llama cuando en la aplicación principal se pulsa la tecla 'o'. Modifica los valores de los trackbars con los parámetros que se le pasan.

In [3]:
```
def setTrackbars(preblurring=0, threshold1=1, threshold2=255, umbral=1, invertir_re
    cv2.setTrackbarPos("preblurring", "Trackbars", preblurring)
    cv2.setTrackbarPos("Filtro", "Trackbars", filtro)
    cv2.setTrackbarPos("threshold1 (Canny)", "Trackbars", threshold1)
    cv2.setTrackbarPos("threshold2 (Canny)", "Trackbars", threshold2)
    cv2.setTrackbarPos("Umbral (Sobel/Laplaciano)", "Trackbars", umbral)
    # Por defecto siempre invertir resultado
    cv2.setTrackbarPos("Invertir resultado", "Trackbars", invertir_resultado)
```

Creación de los trackbars

In [4]:
```
# Crear una ventana con trackbars
cv2.namedWindow("Trackbars")
cv2.setWindowTitle("Trackbars", "Laboratorio CV de Bordes: q: salir, o: optimizar v

cv2.createTrackbar('Imagen', 'Trackbars', 0, 2, trackbar_changed)  # 1:bosqueSencil
cv2.createTrackbar('Filtro', 'Trackbars', 0, 2, trackbar_changed)  # 0:Canny, 1:Sob
cv2.createTrackbar('preblurring', 'Trackbars', 0, 2, trackbar_changed)    # 0: No ,

#Trackbars para los parámetros de Canny y Sobel/Laplaciano
cv2.createTrackbar("threshold1 (Canny)", "Trackbars", 1, 255, trackbar_changed) # U
cv2.setTrackbarMin("threshold1 (Canny)", "Trackbars", 1) # Umbral mínimo 1 para Can
cv2.setTrackbarPos("threshold1 (Canny)", "Trackbars", 1) # Valor inicial para el Um
cv2.createTrackbar("threshold2 (Canny)", "Trackbars", 1, 255, trackbar_changed) # U
cv2.setTrackbarMin("threshold2 (Canny)", "Trackbars", 1) # Umbral mínimo 1 para Can
cv2.setTrackbarPos("threshold2 (Canny)", "Trackbars", 255) # Valor inicial para el
cv2.createTrackbar("Umbral (Sobel/Laplaciano)", "Trackbars", 0, 255, trackbar_chang
cv2.setTrackbarMin("Umbral (Sobel/Laplaciano)", "Trackbars", 1) # Umbral mínimo par
cv2.setTrackbarPos("Umbral (Sobel/Laplaciano)", "Trackbars", 1) # Valor inicial par

# Trackbar para invertir el resultado
cv2.createTrackbar("Invertir resultado", "Trackbars", 0, 1, trackbar_changed)   # 0
```

Lectura de las imágenes

Se leen en escala de grises porque los filtros para detectar bordes funcionan con imágenes en escala de grises
Se comprueba una vez cargadas las imágenes, que se han podido leer todas, si no, sale del programa indicando el error.

In [5]:
```
# Lectura de la imagenes en escala de grises
bosqueSencillo = cv2.imread('bosqueSencillo.jpg', cv2.IMREAD_GRAYSCALE)
bricks = cv2.imread('bricks.jpg', cv2.IMREAD_GRAYSCALE)
carRuido = cv2.imread('carRuido.jpg', cv2.IMREAD_GRAYSCALE)

# Comprobar la existencia del fichero
if bosqueSencillo is None or carRuido is None or bricks is None :
    sys.exit("Alguna de las imágenes no se han podido cargar.")
```

Resize de las imágenes

No seria necesario si las imágenes fueran más pequeñas en origen, però sirve para ver como funciona el redimensionamiento y hace que se ajusten mejor a pantallas pequeñas

In [6]:
```
# resize de las imágenes
factor_escala_x = 0.7
factor_escala_y = 0.7
interpolation_method = cv2.INTER_CUBIC

#El segundo parámetro se ignora si se seleccionar factores de escala.
resized_carRuido = cv2.resize(carRuido, (0,0), fx=factor_escala_x, fy=factor_escala
resized_bricks = cv2.resize(bricks, (0,0), fx=factor_escala_x, fy=factor_escala_y,
resized_bosqueSencillo = cv2.resize(bosqueSencillo, (0,0), fx=factor_escala_x, fy=f
```

Bucle principal del programa

Mientras no se pulse la tecla 'q' del teclado el programa estará en ejecución detectando los movimientos de los trackbars para realizar las funciones deseadas

- El trackbar 'Imagen' permite seleccionar 3 imágenes distintas
- El trackbar 'Filtro' permite elegir entre los filtros canny, sobel y laplaciano
- El trackbar 'preblurring' permite elegir entre sin preblurring, con preblurring gaussiano (3x3) o mediana (3x3)
- Los 'threshold 1 i 2' son los umbrales de la función canny
- 'Umbral' sirve tanto para Sobel como para Laplaciano
- Invertir resultado permite conumtar entre blanco sobre negro y negro sobre blanco

q: sale de programa o: pone los valores de los trackbars a una posición donde el creador de esta aplicación se detectan mejor los bordes para cáda imágen/filtro. Por tanto define el tipo de preblurring, los umbrales y siempre muestra la imágen invertida.

In [7]:
```
while True:
     # Solo procesar si hay cambios
    if needs_update:
        # Se obtienen los valores de los trackbars
        # Se obtiene el código de la imagen a procesar
        # Según el código de la imagen se carga la imagen a procesar
        codigo_imagen = cv2.getTrackbarPos("Imagen", "Trackbars")
        invertir_resultado = cv2.getTrackbarPos("Invertir resultado", "Trackbars")
        if codigo_imagen == 0:
            imagen_a_procesar = resized_bosqueSencillo
            nombre_imagen = "bosqueSencillo"
        elif codigo_imagen == 1:
            imagen_a_procesar = resized_bricks
            nombre_imagen = "bricks"
        else:
            imagen_a_procesar = resized_carRuido
            nombre_imagen = "carRuido"
```

```
        # Se obtienen los valores de los trackbars para preblurring, filtro
        preblurring = cv2.getTrackbarPos("preblurring", "Trackbars")
        filtro =  cv2.getTrackbarPos("Filtro", "Trackbars")

        # Según el preblurring seleccionado se aplica el filtro correspondiente (o
        if preblurring == 1:  # Aplicar preblurring Gaussiano
            image = cv2.GaussianBlur(imagen_a_procesar, (3,3), 0)
            preblurringStyle = "Gaussiano"
        elif preblurring == 2:  # Aplicar preblurring mediana
            image = cv2.medianBlur(imagen_a_procesar, 3)  # Aplicar un filtro de me
            preblurringStyle = "Mediana"
        else:
            image = imagen_a_procesar
            preblurringStyle = "Sin preblurring"
        # Segñun el filtro seleccionado se aplica el filtro correspondiente     con
        if filtro == 0:  # Canny
            # Obtener los valores de los trackbars para los umbrales de Canny
            threshold1 = cv2.getTrackbarPos("threshold1 (Canny)", "Trackbars")
            threshold2 = cv2.getTrackbarPos("threshold2 (Canny)", "Trackbars")
            imgCanny = cv2.Canny(image, threshold1, threshold2)
            # Invertir el resultado si se ha seleccionado
            if invertir_resultado == 1:
                #No se puede invertir con imgCanny = -1.0 * imgCanny + 1.0 porque n
                #No se  normalizado a [0:1], se ha dejado con [0:255] porque los tr
                imgCanny =  cv2.bitwise_not(imgCanny)
            # Definir el nombre del filtro para mostrarlo en la imagen
            nombre_filtro = "Canny"
            # Concatenar la imagen original y la imagen con los filtros aplicados p
            img_combined = cv2.hconcat([image, imgCanny])
        elif filtro == 1: # Sobel
            # Obtener los valores del trackbar para el umbral de Sobel
            umbral = cv2.getTrackbarPos("Umbral (Sobel/Laplaciano)", "Trackbars")
            # Aplicar el filtro Sobel en x (bordes verticales) e y (bordes horizont
            # Usar cv2.CV_64F para evitar cortar valores negativos    q
            sobelx = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)
            sobely = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)
            # Calcular el modulo del gradiente
            edges = np.sqrt(sobelx**2 + sobely**2)  # Mantiene float64
            # Normalizar a [0, 255] y convertir a uint8
            edges = cv2.normalize(edges, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.C
            # Aplicar el umbral lo que esta por encima del umbral se pone a 255 y l
            _, imgSobel = cv2.threshold(edges, umbral, 255, cv2.THRESH_BINARY)
            # Invertir el resultado si se ha seleccionado
            if invertir_resultado == 1:
                # No se puede invertir con imgSobel = -1.0 * imgSobel + 1.0 porque
                imgSobel =  cv2.bitwise_not(imgSobel)
            # Definir el nombre del filtro para mostrarlo en la imagen
            nombre_filtro = "Sobel"
            # Concatenar la imagen original y la imagen con los filtros aplicados p
            img_combined = cv2.hconcat([image, imgSobel])
        else: # Laplaciano
            # Obtener el valor del trackbar para el umbral de Laplaciano
            umbral = cv2.getTrackbarPos("Umbral (Sobel/Laplaciano)", "Trackbars")
            #Funcion laplaciana con cv2.CV_64F para evitar cortar valores negativos
            laplacian = cv2.Laplacian(image, cv2.CV_64F)
```

```
            #Pasar a valor absoluto
            laplacian_abs = np.absolute(laplacian)  # Mantiene float64
            # Normalizar a [0, 255] y convertir a uint8
            edges = cv2.normalize(laplacian_abs, None, 0, 255, cv2.NORM_MINMAX, dty
            # Aplicar el umbral lo que esta por encima del umbral se pone a 255 y l
            _, imgLaplacian = cv2.threshold(edges, umbral, 255, cv2.THRESH_BINARY)
            # Invertir el resultado si se ha seleccionado
            if invertir_resultado == 1:
            # No se puede invertir con imgLaplacian = -1.0 * imgLaplacian + 1.0 por
                imgLaplacian =   cv2.bitwise_not(imgLaplacian)
            # Definir el nombre del filtro para mostrarlo en la imagen
            nombre_filtro = "Laplaciano"
            # Concatenar la imagen original y la imagen con los filtros aplicados p
            img_combined = cv2.hconcat([image, imgLaplacian])

        # Mostrar la imagen combinada con el nombre de la imagen y el filtro aplica
        cv2.putText(img_combined, f"Imagen: {nombre_imagen} | Filtro: {nombre_filtr
        # Mostrar la ventana con los trackbars y la imagen combinada
        cv2.imshow('Trackbars', img_combined)

        # Marcar que ya se actualizó
        needs_update = False

    # Esperar a que se pulse una tecla
    key = cv2.waitKey(1) & 0xFF
    # Si se pulsa 'q' se sale del bucle
    if key == ord('q'):
        break
    # Si se pulsa 'o' se optimizan los valores de los trackbars para la imagen y fi
    elif key == ord('o'):

        #Miramos que imagen estamos usando
        if codigo_imagen == 0:
            #Miramos que filtro estamos usando
            if filtro == 0: #Canny
                # Imagen bosqueSencillo con Canny
                # No es necesario preblurring
                setTrackbars(preblurring=0, threshold1=31, threshold2=43)
            elif filtro == 1:
                 # Imagen bosqueSencillo con Sobel
                # No es necesario preblurring
                setTrackbars(preblurring=0, umbral=8)
            else:
                # Imagen bosqueSencillo con Laplaciano
                # Le va mejor el preblurring Gaussiano
                setTrackbars(preblurring=1, umbral=6)
        elif codigo_imagen == 1:
            # Miramos que filtro estamos usando
            if filtro == 0: #Canny
                # La imagen bricks con Canny
                # Mejor preblurring Gaussiano para difuminar texturas
                setTrackbars(preblurring=1, threshold1=160, threshold2=218)
            elif filtro == 1:
                #La imagen bricks con Sobel
                # Mejor preblurring Gaussiano para difuminar texturas
                setTrackbars(preblurring=1, umbral=51)
```

```
            else:
                #La imagen bricks con laplaciano
                # Mejor preblurring Gaussiano para difuminar texturas
                # No funciona muy bien con laplaciano... con un fitro de blurring g
                setTrackbars(preblurring=1, umbral=45)
        else:
            # Miramos que filtro estamos usando
            if filtro == 0: #Canny
                # La imagen carRuido con Canny
                # Mejor preblurring Mediana para difuminar ruido
                setTrackbars(preblurring=2, threshold1=34, threshold2=86)
            elif filtro == 1:
                #La imagen carRuido con Sobel
                # Mejor preblurring Mediana para difuminar ruido
                setTrackbars(preblurring=2, umbral=12)
            else:
                #La imagen carRuido con laplaciano
                # Mejor preblurring Mediana para difuminar ruido
                # Parece que laplacian es muy sensible al ruido, por lo que no se v
                # de 5 el laplaciano se ve un poco mejor, pero no lo he puesto porq
                setTrackbars(preblurring=2, umbral=7)

        needs_update = True  # Forzar actualización después de optimizar

cv2.destroyAllWindows()
```