Universidad del Valle de Guatemala

Facultad de Ingeniería

Laboratorio 1

CC3182 – Visión por Computadora

Profesor:
Luis Alberto Suriano Saravia

Integrantes:

Diederis Solis — 22952
Andy Fuentes — 22944
Christian Echeverria — 221441

Fecha: 27/01/2026

Repositorio

El código fuente del proyecto se encuentra disponible en el siguiente enlace: https://github.com/Andyfer004/Lab1_VC

1. Task 1: Análisis Teórico y Analítico

Usted ha sido contratado como Ingeniero de Visión Computacional Junior en AutonoVision, una startup que desarrolla sistemas de navegación para robots de almacén. El equipo de hardware ha instalado nuevas cámaras, pero las imágenes llegan con mucho ruido térmico debido a las condiciones de luz del almacén. El módulo actual de detección de obstáculos está fallando: detecta la textura del suelo de concreto como si fueran "bordes"de obstáculos, frenando el robot innecesariamente. Su Director de Proyecto le ha asignado la tarea de construir un pipeline de pre-procesamiento robusto desde cero para entender el problema a nivel matemático y ajustar los parámetros óptimos para el despliegue.

Considerando el escenario previamente planteado, conteste:

1.1. Pregunta 1: Box Filter vs Filtro Gaussiano

Pregunta:
Su jefe sugiere usar un filtro de media (Box Filter) de 7x7 para eliminar el ruido rápido. Usted cree que es una mala idea. Explique matemáticamente y con un diagrama visual (dibujado) por qué un Box Filter de ese tamaño es perjudicial para la detección precisa de la posición de un obstáculo comparado con un filtro Gaussiano del mismo tamaño.

Respuesta:
Consideramos que un **Box Filter** de 7×7 es perjudicial porque promedia **con pesos iguales** toda la ventana, lo cual desplaza/ensancha transiciones y hace que la localización de bordes sea menos precisa (y en este caso, confunde textura del concreto con bordes reales).

Matemáticamente, el Box Filter 7×7 se define como:

$$h_{\text{box}}(i,j) = \frac{1}{49}, \qquad i,j \in \{-3,-2,-1,0,1,2,3\}.$$

Mientras que un Gaussiano del mismo tamaño se define por:

$$h_{\text{gauss}}(i,j) = \frac{1}{C}\exp\left(-\frac{i^2+j^2}{2\sigma^2}\right), \quad C = \sum_{i=-3}^{3}\sum_{j=-3}^{3}\exp\left(-\frac{i^2+j^2}{2\sigma^2}\right),$$

y cumple $\sum h_{\text{gauss}}(i,j) = 1$.

(i) Precisión en la posición del obstáculo (localización de borde). En un pipeline típico (suavizado + gradiente), lo que importa es **dónde** aparece el máximo del gradiente, porque eso me da una estimación de la posición del borde del obstáculo. Si modelo un borde ideal 1D como un escalón $u(x)$ y aplico un suavizado $h(x)$, el resultado es:

$$(u * h)(x).$$

El gradiente que se calcula después es proporcional a la derivada:

$$\frac{d}{dx}(u * h) = u * h'.$$

- Con Box, $h_{\text{box}}(x)$ es una ventana rectangular; su derivada $h'_{\text{box}}(x)$ tiene cambios bruscos en los extremos. Esto produce un perfil del borde **más extendido** y menos definido, por lo que la posición del borde queda menos precisa (se çorre.º se ensancha).
- Con Gauss, $h_{\text{gauss}}(x)$ es suave y su derivada también es suave; el máximo del gradiente se mantiene **mejor localizado** y más estable, lo cual es clave para estimar bien dónde está el obstáculo.

(ii) Textura del suelo y artefactos (dominio frecuencia). El Box Filter tiene respuesta en frecuencia tipo **sinc** con **lóbulos secundarios**:

$$H_{\text{box}}(\omega) \propto \text{sinc}\left(\frac{7\omega}{2}\right),$$

lo que puede introducir oscilaciones (*ringing*) y dejar pasar componentes no deseadas asociadas a texturas repetitivas (como el concreto). Eso aumenta los **falsos positivos** de bordes. En cambio, el Gaussiano en frecuencia también es Gaussiano:

$$H_{\text{gauss}}(\omega) = \exp\left(-\frac{\sigma^2\omega^2}{2}\right),$$

sin lóbulos secundarios, por lo que atenúa altas frecuencias de forma más "limpiaz reduce la probabilidad de convertir textura en "bordes".

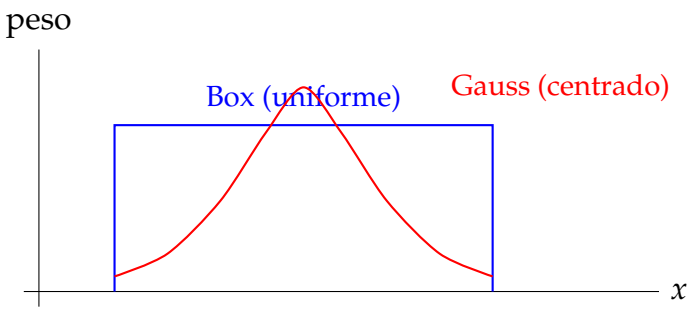


Figura 1: Comparación conceptual de pesos: el Box reparte peso uniforme; el Gaussiano concentra peso en el centro, lo cual mejora estabilidad y localización de bordes reales.

Diagrama visual (dibujado).

1.2. Pregunta 2: Estrategias de Padding

Pregunta:
Al realizar la convolución en los bordes de la imagen (por ejemplo, en el píxel 0,0), el kernel "se sale"de la imagen.

a) Si el robot navega por pasillos oscuros con luces brillantes al final, ¿por qué el Zero-Padding podría generar falsos positivos de bordes en la periferia de la imagen?

b) ¿Qué estrategia de padding (Reflect, Replicate, Wrap) recomendaría para evitar esto y por qué?

Respuesta:

a) Con **Zero-Padding**, todo lo que queda fuera de la imagen se asume como intensidad 0:

$$I_{\text{pad}}(x,y) = \begin{cases} I(x,y), & \text{dentro de la imagen} \\ 0, & \text{fuera} \end{cases}$$

Entonces, cuando el kernel cae parcialmente fuera en la periferia, se combinan valores reales con ceros artificiales y se crea una **discontinuidad** cerca del borde (ïmagen real"vs çero"). Operadores que miden cambios (Sobel/Laplaciano) interpretan esa discontinuidad como un cambio fuerte, generando **bordes falsos** pegados al marco. En pasillos oscuros con luces brillantes (alto contraste), este efecto se vuelve más evidente y el robot puede frenar por detecciones falsas.

b) Yo recomendaría **Reflect padding**, porque refleja el contenido en el borde y mantiene mejor la **continuidad** de la señal (evita saltos bruscos), reduciendo artefactos en gradientes.

- **Reflect**: transición más natural, menos bordes artificiales.
- **Replicate**: mejora vs ceros, pero puede crear zonas planas extendidas que alteran derivadas cerca del borde.
- **Wrap**: suele ser poco realista (conecta con el lado opuesto) y puede introducir bordes falsos por mezclar contenido no relacionado.

1.3. Pregunta 3: Convolución con Kernel Laplaciano

Pregunta:
Dada la siguiente sub-imagen I de 3×3 y el kernel K:

$$I = \begin{bmatrix} 10 & 10 & 10 \\ 10 & 0 & 10 \\ 10 & 10 & 10 \end{bmatrix}, \qquad K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

a) Calcule el valor del píxel central resultante de la convolución

b) ¿Qué tipo de estructura detecta este filtro K (conocido como Laplaciano)?

Respuesta:

a) Para la convolución, matemáticamente el kernel se invierte (flip). En este caso, como el kernel Laplaciano mostrado es simétrico, el *flip* no cambia la matriz. Entonces, el valor del píxel central se obtiene multiplicando y sumando los vecinos (arriba, abajo, izquierda, derecha) y el centro:

$$(I * K)_{\text{centro}} = (10)(1) + (10)(1) + (10)(1) + (10)(1) + (0)(-4) + (\text{esquinas}) \cdot 0$$

$$(I * K)_{\text{centro}} = 10 + 10 + 10 + 10 + 0 = 40$$

Por lo tanto, **el valor del píxel central resultante es 40**.

b) Como grupo, concluimos que el Laplaciano detecta **cambios bruscos de intensidad** porque es un operador de **segunda derivada**:

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}.$$

En práctica, resalta **bordes finos** y también **puntos/anomalías** donde un píxel es muy diferente a sus vecinos.

En esta sub-imagen, el centro tiene intensidad 0 y sus vecinos directos tienen 10, entonces el filtro produce una respuesta positiva alta (40), lo que indica que hay una **discontinuidad fuerte** en el centro: es decir, el Laplaciano está detectando un **"punto.º cambio abrupto** (una especie de hueco o detalle muy marcado) respecto al entorno.

2. Task 2: Práctica - Implementación de Algoritmos

2.1. Ejercicio 1: Convolución 2D Genérica

2.1.1. Descripción del Problema

Se requirió implementar una función:
`mi_convolucion(imagen, kernel, padding_type='reflect')` que realice la operación de convolución 2D, cumpliendo con las siguientes restricciones:

- **Restricción 1:** Manejar únicamente imágenes en escala de grises.
- **Restricción 2:** Implementar el padding manualmente antes de operar.
- **Reto de optimización:** Reducir los bucles anidados de 4 a 2 usando slicing de NumPy.
- **Nota matemática:** Implementar el flip del kernel (convolución matemática real).

2.1.2. Solución Implementada

La función implementada sigue el siguiente algoritmo:

1. **Validación:** Se verifica que la imagen sea 2D (escala de grises).
2. **Cálculo de padding:** Se determina el padding necesario como $\lfloor \text{kernel\_size}/2 \rfloor$.
3. **Aplicación de padding:** Se utiliza `np.pad` con modo reflejo por defecto.
4. **Flip del kernel:** Se invierte el kernel en ambos ejes para realizar convolución matemática (no correlación).
5. **Convolución optimizada:** Se utilizan solo 2 bucles sobre las dimensiones del kernel.

2.1.3. Análisis de la Optimización

La estrategia de optimización reduce la complejidad computacional:

Método	Bucles	Complejidad
Ingenuo (4 bucles)	$H \times W \times K_h \times K_w$	$O(n^2 \cdot k^2)$
Optimizado (2 bucles)	$K_h \times K_w$ iteraciones vectorizadas	$O(k^2)$ operaciones vectoriales

Tabla 1: Comparación de complejidad entre métodos de convolución

La fórmula matemática de la convolución implementada es:

$$I_{out}[x, y] = \sum_{i=0}^{K_h-1} \sum_{j=0}^{K_w-1} K_{flip}[i, j] \cdot I_{pad}[x + i, y + j] \tag{1}$$

donde K_{flip} es el kernel invertido y I_{pad} es la imagen con padding aplicado.

2.2. Ejercicio 2: Generador de Gaussianos

2.2.1. Descripción del Problema

Implementar una función `generar_gaussiano(tamano, sigma)` que genere un kernel Gaussiano 2D normalizado.

2.2.2. Fundamento Teórico

La distribución Gaussiana 2D centrada está definida por:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{2}$$

donde σ es la desviación estándar que controla el "ancho" de la campana.

2.2.3. Solución Implementada

La implementación:

1. Valida que el tamaño sea impar (para tener un centro definido).
2. Crea una malla de coordenadas centrada usando `np.meshgrid`.
3. Aplica la fórmula Gaussiana 2D.
4. Normaliza el kernel para que $\sum G[i, j] = 1{,}0$ (preservación de energía).

2.2.4. Importancia de la Normalización

La normalización garantiza que:

- La intensidad promedio de la imagen se preserve.
- No haya saturación ni oscurecimiento artificial.
- El filtro actúe como un promedio ponderado puro.

2.3. Ejercicio 3: Pipeline de Detección de Bordes (Sobel)

2.3.1. Descripción del Problema

Crear una función `detectar_bordes_sobel(imagen)` que aplique los kernels de Sobel y calcule la magnitud y dirección del gradiente.

2.3.2. Fundamento Teórico

Los operadores de Sobel aproximan las derivadas parciales de la imagen:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I \tag{3}$$

La magnitud del gradiente se calcula como:

$$G = \sqrt{G_x^2 + G_y^2} \tag{4}$$

Y la dirección del gradiente:

$$\theta = \arctan 2(G_y, G_x) \tag{5}$$

2.3.3. Solución Implementada

La función:

1. Define los kernels de Sobel G_x y G_y.
2. Aplica `mi_convolucion` con cada kernel.
3. Calcula la magnitud usando la fórmula euclidiana.
4. Normaliza la magnitud al rango [0, 255] para visualización.
5. Calcula la dirección usando `arctan2`.

3. Task 3: Evaluación de Ingeniería y Criterio

3.1. Experimento A: El Efecto de Sigma (σ)

3.1.1. Metodología

Se generaron tres versiones de detección de bordes variando el pre-procesamiento Gaussiano:

1. **Sin suavizado:** Detección Sobel directa sobre la imagen original.
2. **Gaussiano $\sigma = 1$:** Kernel 5×5, suavizado moderado.
3. **Gaussiano $\sigma = 5$:** Kernel 31×31, suavizado intenso.

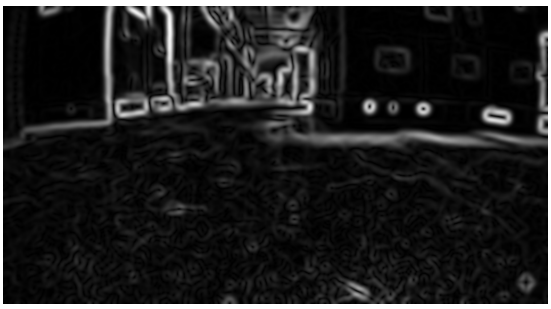
3.1.2. Resultados Visuales

(a) Imagen Original

(b) Sobel sin Filtro Gaussiano

Figura 2: Comparación: Original vs Sin Suavizado

(a) Gaussiano $\sigma = 1$ + Sobel



(b) Gaussiano $\sigma = 5$ + Sobel

Figura 3: Efecto del valor de Sigma en la detección de bordes

3.1.3. Análisis de Resultados

¿Qué pasa con los bordes finos cuando σ es muy alto? Cuando σ aumenta significativamente (como $\sigma = 5$):

- Los **bordes finos desaparecen** porque el suavizado excesivo "difumina" las transiciones abruptas de intensidad.
- La imagen pierde **detalles de alta frecuencia** como texturas, grietas pequeñas y bordes delgados.
- Solo se preservan los **bordes dominantes** y estructuras de gran escala.
- Los bordes restantes aparecen más **suaves y gruesos**, perdiendo precisión en su localización.

¿Qué pasa con la textura del suelo cuando no hay suavizado? Sin aplicar filtro Gaussiano:

- La **textura del suelo se detecta como bordes**, creando una imagen muy ruidosa.
- Cada pequeña variación de intensidad (granos, polvo, imperfecciones) genera respuesta en el detector.
- Es **imposible distinguir** entre bordes reales (objetos) y ruido de textura.
- La imagen resultante tiene **demasiada información** que dificulta el análisis.

Criterio de Ingeniería: ¿Cuál elegir para detectar pallets ignorando grietas? Recomendación: Gaussiano con σ entre 3 y 5.

Justificación:

- Un σ moderado-alto **elimina las grietas pequeñas** del suelo de almacén.
- **Preserva los bordes de los pallets** que son estructuras de mayor escala.
- Reduce el ruido de la textura del concreto sin perder la geometría de los objetos de interés.
- Para una aplicación robótica, esto permite una **segmentación más limpia** de obstáculos.

El trade-off es: **a mayor σ, mejor supresión de ruido pero menor precisión en la localización exacta del borde**.

3.2. Experimento B: Umbral Simple vs. Histéresis (Canny)

3.2.1. Metodología

Se implementó una función de umbralización simple y se comparó con el detector Canny de OpenCV:

```
def umbral_simple(magnitud, T):
    """Umbralizaci n binaria simple."""
    binaria = np.zeros_like(magnitud)
    binaria[magnitud >= T] = 255
    return binaria
```

Listing 1: Implementación de Umbral Simple

Se utilizó $T = 80$ como umbral y se comparó contra `cv2.Canny` con umbrales $(50, 150)$.

3.2.2. Resultados Visuales

(a) Magnitud del Gradiente (b) Umbral Simple $T = 80$ (c) Canny (Referencia)

Figura 4: Comparación entre Umbral Simple y Detector Canny

3.2.3. Análisis: ¿Se rompen las líneas de los bordes?

Observación crítica: Con el umbral simple, **sí se rompen las líneas de los bordes**.

- Los bordes aparecen **fragmentados y discontinuos**.
- Donde la magnitud del gradiente fluctúa cerca del umbral T, aparecen "huecos".

- Los bordes de objetos no forman contornos cerrados, dificultando su uso para segmentación.

En contraste, Canny produce:

- Bordes **delgados y bien conectados**.
- Contornos más **continuos y definidos**.
- Mejor supresión de respuestas múltiples.

3.2.4. Pregunta Crítica: ¿Por qué el umbral simple nunca será tan efectivo como la histéresis?

Problema fundamental del umbral simple: Un umbral único T crea una decisión binaria rígida:

$$\text{Borde}(x,y) = \begin{cases} 255 & \text{si } G(x,y) \geq T \\ 0 & \text{si } G(x,y) < T \end{cases} \tag{6}$$

Este enfoque tiene limitaciones intrínsecas:

- **Si T es bajo:** Se detecta mucho ruido como bordes falsos.
- **Si T es alto:** Se pierden bordes débiles pero válidos, causando discontinuidades.
- **No existe un T óptimo** que simultáneamente elimine ruido y preserve todos los bordes reales.

Ventaja de la Histéresis (Canny): La histéresis utiliza **dos umbrales** T_{low} y T_{high}:

1. Píxeles con $G \geq T_{high}$ son **bordes fuertes** (definitivamente bordes).
2. Píxeles con $T_{low} \leq G < T_{high}$ son **bordes débiles** (candidatos).
3. Un borde débil se **acepta solo si está conectado** a un borde fuerte.

Problema específico en robótica (vibraciones e iluminación): En el contexto de un robot moviéndose y vibrando en un almacén:

- Las **vibraciones mecánicas** causan pequeños desplazamientos en la imagen capturada.
- Los **cambios de iluminación** (sombras, reflejos) modifican localmente la intensidad.
- Ambos factores hacen que la magnitud del gradiente **fluctúe alrededor del umbral**.

Con umbral simple:

- Un borde que en un frame tiene $G = 81$ (detectado) puede tener $G = 79$ en el siguiente (perdido).
- Esto causa **parpadeo temporal** (flickering) y **fragmentación espacial**.
- Los contornos de objetos aparecen y desaparecen erráticamente.

Con histéresis:

- La **conectividad** garantiza que si parte del borde es fuerte, los segmentos débiles conectados se preservan.

- Esto proporciona **estabilidad temporal** y **continuidad espacial**.
- Los contornos permanecen coherentes aunque haya fluctuaciones locales de intensidad.

4. Referencias

- Canny, J. (1986). "A Computational Approach to Edge Detection". IEEE Transactions on Pattern Analysis and Machine Intelligence.
- Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th ed.). Pearson.
- Sobel, I., & Feldman, G. (1968). "A 3×3 Isotropic Gradient Operator for Image Processing".