

Laboratorio 3 - Visión por Computadora

- Paula Barillas - 22764
- Gerardo Pineda - 22880
- Mónica Salvatierra - 22249

Link del repositorio: <https://github.com/paulabaal12/LAB3-VCP>

Task 1

1. Encontrar eigenvalores de ambas matrices

$$M = \begin{bmatrix} 120 & 5 \\ 5 & 115 \end{bmatrix}$$

$$\det(M - \lambda I) = 0$$

$$\begin{bmatrix} 120 & 5 \\ 5 & 115 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$$

$$\begin{bmatrix} 120 - \lambda & 5 \\ 5 & 115 - \lambda \end{bmatrix}$$

$$(120 - \lambda)(115 - \lambda) - 25 \\ 13,800 - 120\lambda - 115\lambda + \lambda^2 - 25$$

Ecuación característica

$$\lambda^2 - 235\lambda + 13,775 = 0$$

Usando la ecuación cuadrática

$$\lambda_{1,2} = \frac{-(-235) \pm \sqrt{(-235)^2 - 4 * 1 * 13,775}}{2(1)}$$

$$\lambda_{1,2} = \frac{235 \pm \sqrt{55,225 - 55,100}}{2}$$

$$\lambda_{1,2} = \frac{235 \pm 5\sqrt{5}}{2}$$

$$\lambda_1 \approx 123.09$$

$$\lambda_2 \approx 111.90$$

$$M' = \begin{bmatrix} 200 & 10 \\ 10 & 1 \end{bmatrix}$$

$$\det(M - \lambda I) = 0$$

$$\begin{bmatrix} 200 & 10 \\ 10 & 1 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$$

$$\begin{bmatrix} 200 - \lambda & 10 \\ 10 & 1 - \lambda \end{bmatrix}$$

$$(200 - \lambda) * (1 - \lambda) - 100$$

$$200 - 200\lambda - \lambda + \lambda^2 - 100$$

Ecuación característica

$$\lambda^2 - 201\lambda + 100 = 0$$

Usando la ecuación cuadrática

$$\lambda_{1,2} = \frac{-(-201) \pm \sqrt{(-201)^2 - 4 * 1 * 100}}{2(1)}$$

$$\lambda_{1,2} = \frac{201 \pm \sqrt{40,401 - 400}}{2}$$

$$\lambda_{1,2} = \frac{201 \pm \sqrt{40,001}}{2}$$

$$\lambda_1 \approx 0.498$$

$$\lambda_2 \approx 200.50$$

2. Obtener respuesta de Harris (asumiendo que k=0.04)

$$R(M) = \det(M) - k(\text{Tr}(M))^2$$

$$R(M) = 13,775 - 0.04(235)^2$$

$$R(M) = 13,775 - 2209$$

$$R(M) = 11,566$$

$$R(M') = \det(M) - k(\text{Tr}(M))^2$$

$$R(M') = 100 - 0.04(201)^2$$

$$R(M') = 100 - 1616.04$$

$$R(M') = -1516.04$$

3. Clasificar que representa cada pixel geométricamente

Para la matriz M, el píxel se clasifica como una **esquina**, ya que ambos eigenvalores son grandes y de magnitud similar, lo que indica variaciones significativas de intensidad en más de una dirección. Este comportamiento se refleja también en la respuesta de Harris, la cual toma un valor alto, consistente con la presencia de cambios fuertes en múltiples direcciones.

Por otro lado, la matriz M' representa un **borde**, dado que sus eigenvalores presentan magnitudes muy distintas. Uno es grande y el otro pequeño. Esto indica un cambio dominante de intensidad en una sola dirección, mientras que la variación en la dirección perpendicular es mínima. En consecuencia, la respuesta de Harris resulta baja y negativa, siendo esta coherente con la geometría de un borde.

Task 2

In [1]:

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

2.1 Carga de imágenes en escala de grises

In [2]:

```
img1_path = 'img/cuaderno2.jpeg'
img2_path = 'img/cuaderno1.jpeg'

# imágenes en escala de grises
img1_gray = cv2.imread(img1_path, cv2.IMREAD_GRAYSCALE)
img2_gray = cv2.imread(img2_path, cv2.IMREAD_GRAYSCALE)

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(img1_gray, cmap='gray')
plt.title('Imagen 1 - Vista frontal')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(img2_gray, cmap='gray')
plt.title('Imagen 2 - Rotada/Escalada')
plt.axis('off')
```

```
plt.tight_layout()
plt.show()
```

Imagen 1 - Vista frontal



Imagen 2 - Rotada/Escalada



2.2 Detección y descripción usando SIFT

```
In [3]: # CSIFT
sift = cv2.SIFT_create()

# se detectan keypoints y descriptores en ambas imágenes
kp1_sift, des1_sift = sift.detectAndCompute(img1_gray, None)
kp2_sift, des2_sift = sift.detectAndCompute(img2_gray, None)

print(f'Número de keypoints SIFT en imagen 1: {len(kp1_sift)}')
print(f'Número de keypoints SIFT en imagen 2: {len(kp2_sift)}')

img1_sift = cv2.drawKeypoints(img1_gray, kp1_sift, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img2_sift = cv2.drawKeypoints(img2_gray, kp2_sift, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(img1_sift, cmap='gray')
plt.title('Keypoints SIFT - Imagen 1')
plt.axis('off')

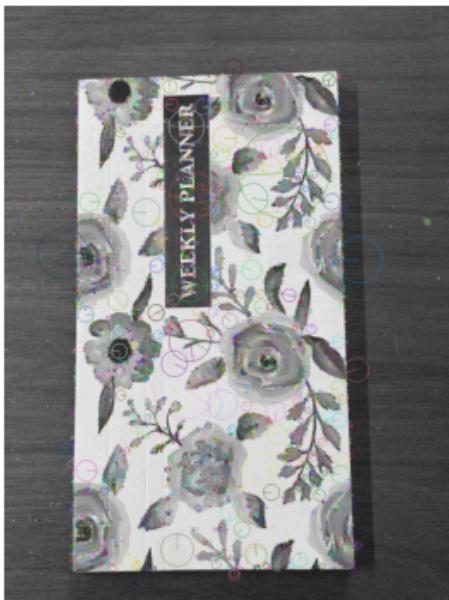
plt.subplot(1, 2, 2)
plt.imshow(img2_sift, cmap='gray')
plt.title('Keypoints SIFT - Imagen 2')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Número de keypoints SIFT en imagen 1: 6019

Número de keypoints SIFT en imagen 2: 4073

Keypoints SIFT - Imagen 1



Keypoints SIFT - Imagen 2



2.3 Detección y descripción usando ORB

In [4]:

```
# ORB
orb = cv2.ORB_create(nfeatures=1000)

# se detectan keypoints y descriptores en ambas imágenes
kp1_orb, des1_orb = orb.detectAndCompute(img1_gray, None)
kp2_orb, des2_orb = orb.detectAndCompute(img2_gray, None)

print(F'Número de keypoints ORB en imagen 1: {len(kp1_orb)}')
print(F'Número de keypoints ORB en imagen 2: {len(kp2_orb)}')

img1_orb = cv2.drawKeypoints(img1_gray, kp1_orb, None, color=(0, 255, 0), flags=0)
img2_orb = cv2.drawKeypoints(img2_gray, kp2_orb, None, color=(0, 255, 0), flags=0)
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(img1_orb, cmap='gray')
plt.title('Keypoints ORB - Imagen 1')
plt.axis('off')

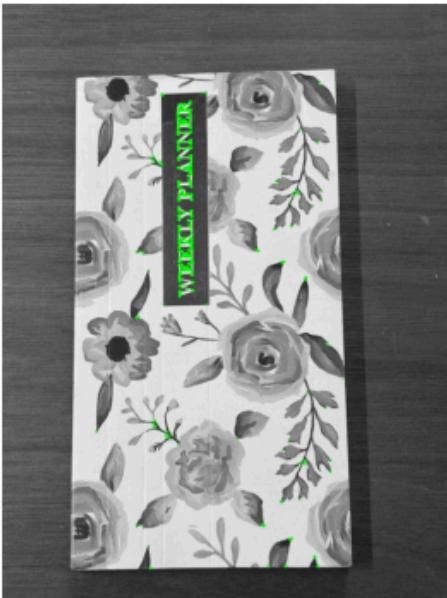
plt.subplot(1, 2, 2)
plt.imshow(img2_orb, cmap='gray')
plt.title('Keypoints ORB - Imagen 2')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Número de keypoints ORB en imagen 1: 1000

Número de keypoints ORB en imagen 2: 1000

Keypoints ORB - Imagen 1



Keypoints ORB - Imagen 2



2.4 Matching con SIFT + Lowe's Ratio Test

```
In [5]: def lowe_ratio_test(knn_matches, ratio=0.75):
    good_matches = []
    for m, n in knn_matches:
        if m.distance < ratio * n.distance:
            good_matches.append(m)
    return good_matches

# SIFT: norma L2 (Eucladiana)
bf_sift = cv2.BFMatcher(cv2.NORM_L2, crossCheck=False)

# KNN matching (k = 2)
matches_sift_knn = bf_sift.knnMatch(des1_sift, des2_sift, k=2)

matches_sift = lowe_ratio_test(matches_sift_knn, ratio=0.75)
print(f'Matches totales SIFT: {len(matches_sift_knn)}')
print(f'Matches SIFT (Lowe 0.75): {len(matches_sift)}')

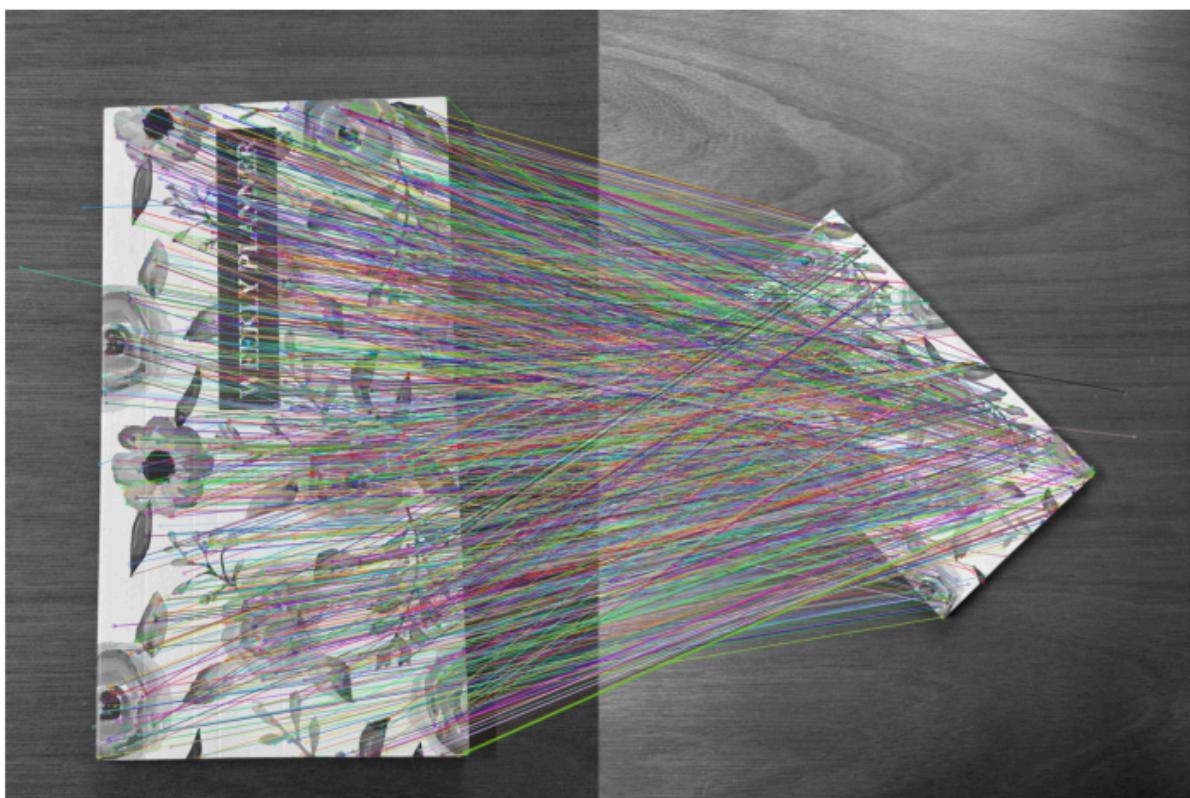
img_matches_sift = cv2.drawMatches(
    img1_gray, kp1_sift,
    img2_gray, kp2_sift,
    matches_sift, None,
    flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
)

plt.figure(figsize=(12, 6))
plt.imshow(img_matches_sift, cmap='gray')
plt.title('Matches SIFT (Lowe 0.75)')
plt.axis('off')
plt.show()
```

Matches totales SIFT: 6019

Matches SIFT (Lowe 0.75): 1360

Matches SIFT (Lowe 0.75)



2.5 Matching con ORB + Lowe's Ratio Test

```
In [6]: # Matcher para ORB: norma de Hamming
bf_orb = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)

# KNN matching (k = 2) sobre los descriptores ORB
matches_orb_knn = bf_orb.knnMatch(des1_orb, des2_orb, k=2)

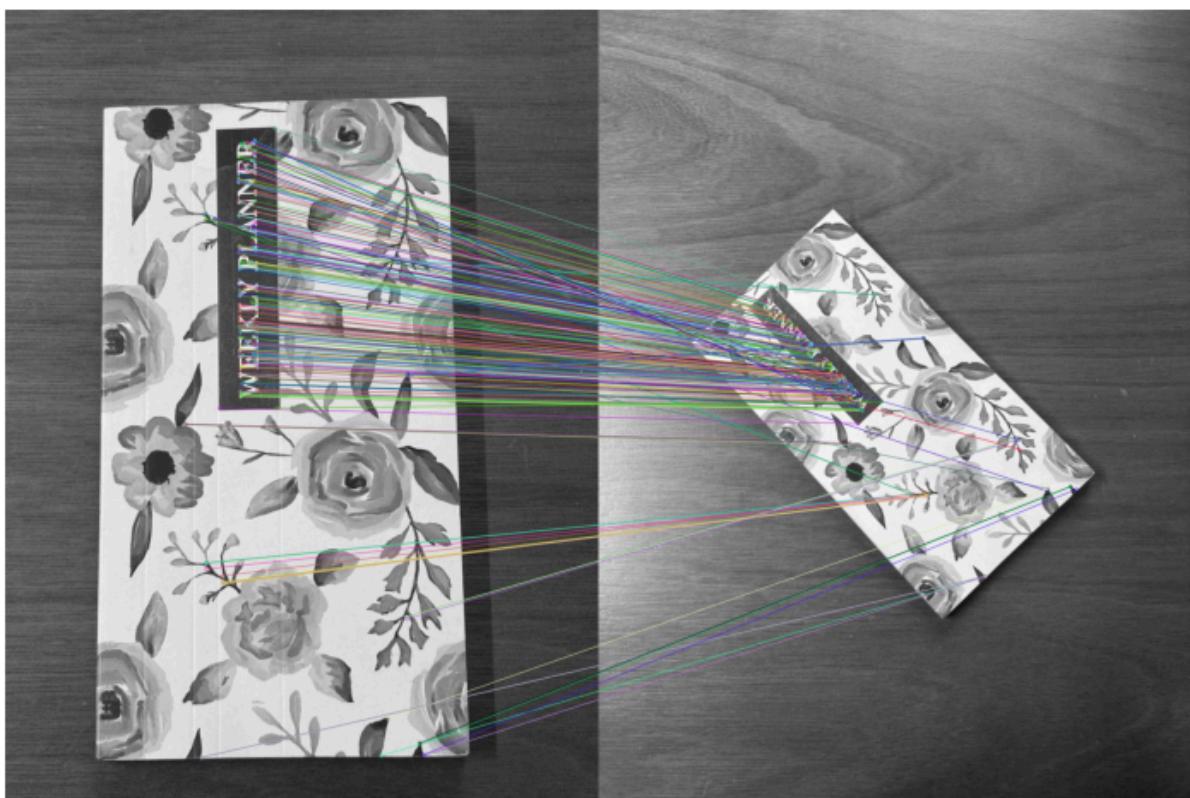
matches_orb = lowe_ratio_test(matches_orb_knn, ratio=0.75)
print(f'Matches totales ORB: {len(matches_orb_knn)}')
print(f'Matches ORB (Lowe 0.75): {len(matches_orb)}')

img_matches_orb = cv2.drawMatches(
    img1_gray, kp1_orb,
    img2_gray, kp2_orb,
    matches_orb, None,
    flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
)

plt.figure(figsize=(12, 6))
plt.imshow(img_matches_orb, cmap='gray')
plt.title('Matches ORB (Lowe 0.75)')
plt.axis('off')
plt.show()
```

Matches totales ORB: 1000
 Matches ORB (Lowe 0.75): 232

Matches ORB (Lowe 0.75)



2.6 SIFT vs ORB

```
In [ ]: # Comparación visual de matches para SIFT y ORB
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.imshow(img_matches_sift, cmap='gray')
plt.title('Matches SIFT')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(img_matches_orb, cmap='gray')
plt.title('Matches ORB')
plt.axis('off')

plt.tight_layout()
plt.show()
```



En la figura se comparan los matches obtenidos con SIFT y con ORB sobre las dos vistas del cuaderno/libreta. Como se puede observar, SIFT suele producir un número mayor de correspondencias consistentes entre ambas imágenes. En donde se ve que hay más líneas que conectan regiones visualmente similares del cuaderno. En donde muchos de los matches de SIFT se mantienen incluso en zonas donde la libreta está rotada y basicamente ligeramente más lejos esto debido a que SIFT está diseñado para ser invariante a escala y rotación. En general consideramos que con SIFT los matches tienden a cubrir mejor toda la superficie de la libreta (la textura que son las flores y el texto de la portada), mientras que ORB a veces concentra las correspondencias en unas pocas áreas y deja otras areas sin cubrir. Ya que en el caso de ORB es más común ver algunas líneas que conectan puntos que claramente no pertenecen a la misma parte del objeto, apesar que se aplicó el Lowe's Ratio Test se siguen apareciendo más outliers que en SIFT.

Task 3

```
In [8]: import cv2
import numpy as np
from matplotlib import pyplot as plt
import time

In [2]: img1_path = 'img/cuaderno2.jpeg'
img2_path = 'img/cuaderno1.jpeg'

# imágenes en escala de grises
img1_gray = cv2.imread(img1_path, cv2.IMREAD_GRAYSCALE)
img2_gray = cv2.imread(img2_path, cv2.IMREAD_GRAYSCALE)

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(img1_gray, cmap='gray')
plt.title('Imagen 1 - Vista frontal')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(img2_gray, cmap='gray')
plt.title('Imagen 2 - Rotada/Escalada')
plt.axis('off')
```

```
plt.tight_layout()
plt.show()
```

Imagen 1 - Vista frontal



Imagen 2 - Rotada/Escalada



In [3]:

```
# CSIFT
sift = cv2.SIFT_create()

# se detectan keypoints y descriptores en ambas imágenes
kp1_sift, des1_sift = sift.detectAndCompute(img1_gray, None)
kp2_sift, des2_sift = sift.detectAndCompute(img2_gray, None)

print(F'Número de keypoints SIFT en imagen 1: {len(kp1_sift)}')
print(F'Número de keypoints SIFT en imagen 2: {len(kp2_sift)}')

img1_sift = cv2.drawKeypoints(img1_gray, kp1_sift, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img2_sift = cv2.drawKeypoints(img2_gray, kp2_sift, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(img1_sift, cmap='gray')
plt.title('Keypoints SIFT - Imagen 1')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(img2_sift, cmap='gray')
plt.title('Keypoints SIFT - Imagen 2')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Número de keypoints SIFT en imagen 1: 6019

Número de keypoints SIFT en imagen 2: 4073

Keypoints SIFT - Imagen 1



Keypoints SIFT - Imagen 2



In [4]:

```
# ORB
orb = cv2.ORB_create(nfeatures=1000)

# se detectan keypoints y descriptores en ambas imágenes
kp1_orb, des1_orb = orb.detectAndCompute(img1_gray, None)
kp2_orb, des2_orb = orb.detectAndCompute(img2_gray, None)

print(f'Número de keypoints ORB en imagen 1: {len(kp1_orb)}')
print(f'Número de keypoints ORB en imagen 2: {len(kp2_orb)}')

img1_orb = cv2.drawKeypoints(img1_gray, kp1_orb, None, color=(0, 255, 0), flags=0)
img2_orb = cv2.drawKeypoints(img2_gray, kp2_orb, None, color=(0, 255, 0), flags=0)
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(img1_orb, cmap='gray')
plt.title('Keypoints ORB - Imagen 1')
plt.axis('off')

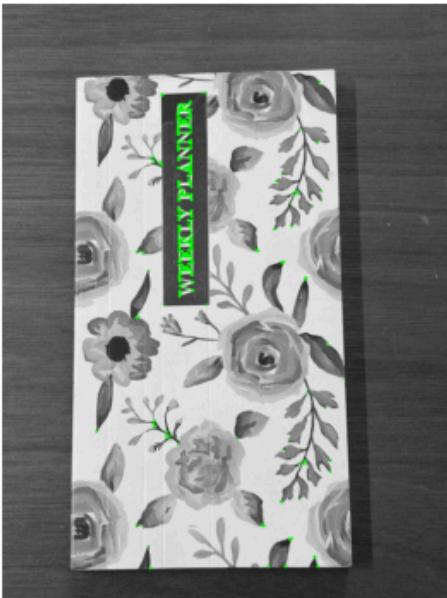
plt.subplot(1, 2, 2)
plt.imshow(img2_orb, cmap='gray')
plt.title('Keypoints ORB - Imagen 2')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Número de keypoints ORB en imagen 1: 1000

Número de keypoints ORB en imagen 2: 1000

Keypoints ORB - Imagen 1



Keypoints ORB - Imagen 2



```
In [5]: def lowe_ratio_test(knn_matches, ratio=0.75):
    good_matches = []
    for m, n in knn_matches:
        if m.distance < ratio * n.distance:
            good_matches.append(m)
    return good_matches

# SIFT: norma L2 (Euclidiana)
bf_sift = cv2.BFMatcher(cv2.NORM_L2, crossCheck=False)

# KNN matching (k = 2)
matches_sift_knn = bf_sift.knnMatch(des1_sift, des2_sift, k=2)

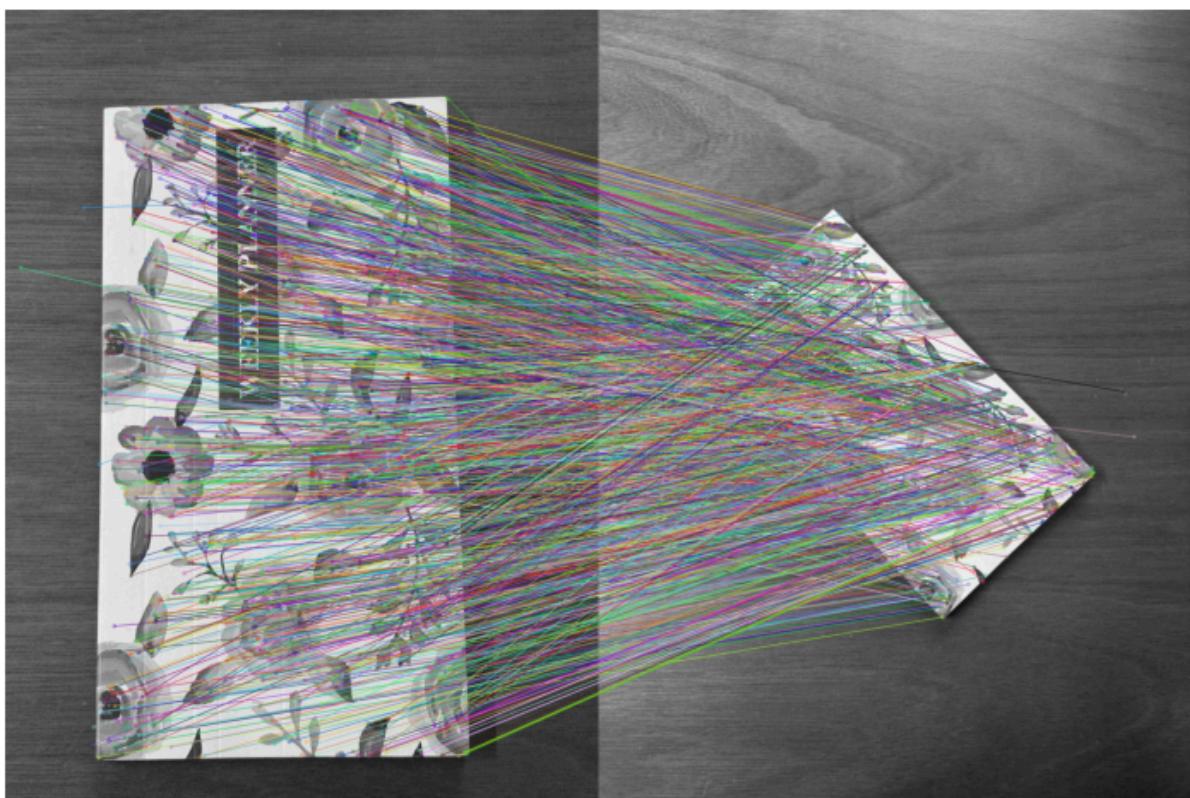
matches_sift = lowe_ratio_test(matches_sift_knn, ratio=0.75)
print(f'Matches totales SIFT: {len(matches_sift_knn)}')
print(f'Matches SIFT (Lowe 0.75): {len(matches_sift)}')

img_matches_sift = cv2.drawMatches(
    img1_gray, kp1_sift,
    img2_gray, kp2_sift,
    matches_sift, None,
    flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
)

plt.figure(figsize=(12, 6))
plt.imshow(img_matches_sift, cmap='gray')
plt.title('Matches SIFT (Lowe 0.75)')
plt.axis('off')
plt.show()
```

Matches totales SIFT: 6019
 Matches SIFT (Lowe 0.75): 1360

Matches SIFT (Lowe 0.75)



```
In [6]: # Matcher para ORB: norma de Hamming
bf_orb = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)

# KNN matching (k = 2) sobre los descriptores ORB
matches_orb_knn = bf_orb.knnMatch(des1_orb, des2_orb, k=2)

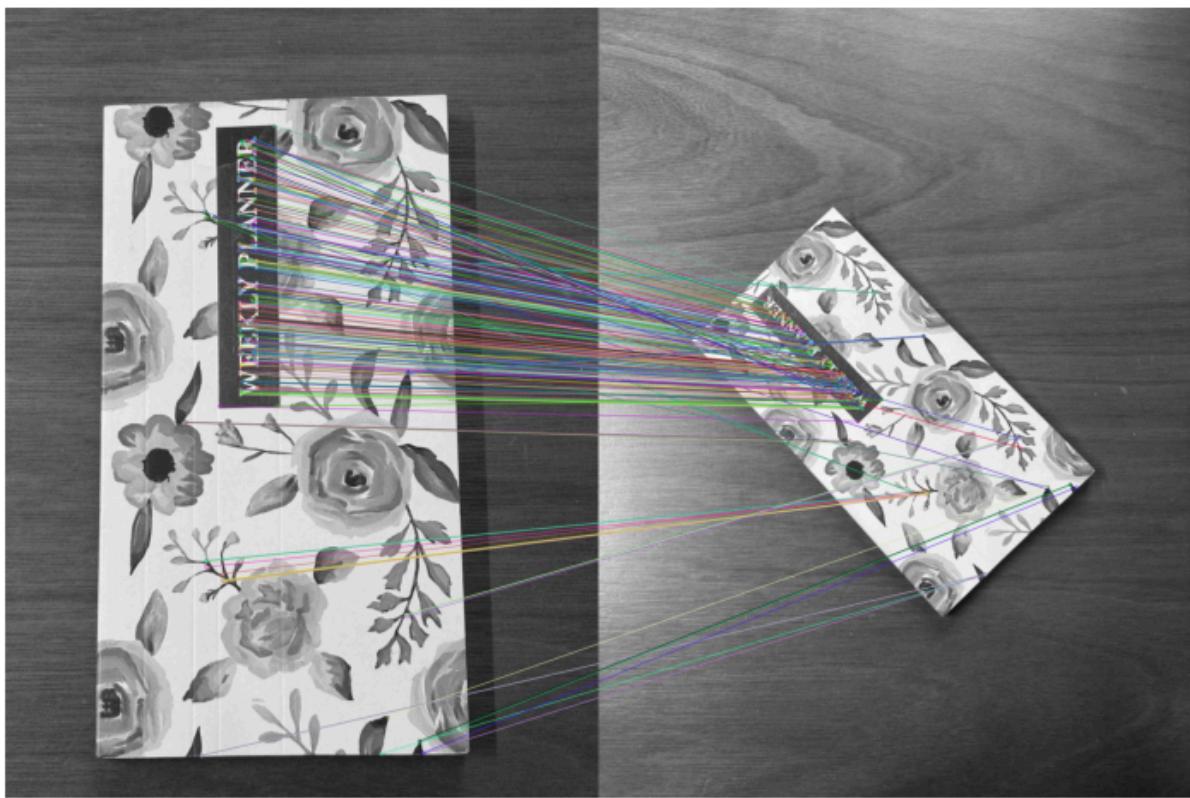
matches_orb = lowe_ratio_test(matches_orb_knn, ratio=0.75)
print(f'Matches totales ORB: {len(matches_orb_knn)}')
print(f'Matches ORB (Lowe 0.75): {len(matches_orb)}')

img_matches_orb = cv2.drawMatches(
    img1_gray, kp1_orb,
    img2_gray, kp2_orb,
    matches_orb, None,
    flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
)

plt.figure(figsize=(12, 6))
plt.imshow(img_matches_orb, cmap='gray')
plt.title('Matches ORB (Lowe 0.75)')
plt.axis('off')
plt.show()
```

Matches totales ORB: 1000
 Matches ORB (Lowe 0.75): 232

Matches ORB (Lowe 0.75)



```
In [7]: # Comparación visual de matches para SIFT y ORB
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.imshow(img_matches_sift, cmap='gray')
plt.title('Matches SIFT')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(img_matches_orb, cmap='gray')
plt.title('Matches ORB')
plt.axis('off')

plt.tight_layout()
plt.show()
```



```
In [ ]: def measurements(detector, matcher, name, repeats=20):
    for _ in range(3):
        detector.detectAndCompute(img1_gray, None)
        detector.detectAndCompute(img2_gray, None)

    t_detect = []
    t_match = []
    kp_counts = []
    match_counts = []

    for _ in range(repeats):
        t0 = time.perf_counter()
        kp1, des1 = detector.detectAndCompute(img1_gray, None)
        kp2, des2 = detector.detectAndCompute(img2_gray, None)
        t1 = time.perf_counter()

        good = []
        if des1 is not None and des2 is not None:
            knn = matcher.knnMatch(des1, des2, k=2)
            good = Lowe_ratio_test(knn)

        t2 = time.perf_counter()

        t_detect.append((t1 - t0) * 1000)
        t_match.append((t2 - t1) * 1000)
        kp_counts.append((len(kp1) + len(kp2)) / 2)
        match_counts.append(len(good))

    avg_detect = np.mean(t_detect)
    avg_match = np.mean(t_match)
    total = avg_detect + avg_match
    fps = 1000 / total

    print(f'\n{name}')
    print(f'Detect + Compute: {avg_detect:.2f} ms')
    print(f'Matching: {avg_match:.2f} ms')
    print(f'Total: {total:.2f} ms')
    print(f'FPS approximado: {fps:.2f}')
    print(f'Keypoints promedio: {np.mean(kp_counts):.1f}')
    print(f'Matches promedio: {np.mean(match_counts):.1f}'
```

```
In [10]: sift = cv2.SIFT_create()
bf_sift = cv2.BFMatcher(cv2.NORM_L2)

orb = cv2.ORB_create(nfeatures=1000)
bf_orb = cv2.BFMatcher(cv2.NORM_HAMMING)

benchmark(sift, bf_sift, "SIFT")
benchmark(orb, bf_orb, "ORB")
```

SIFT

Detect + Compute: 669.28 ms
 Matching: 147.82 ms
 Total: 817.09 ms
 FPS aproximado: 1.22
 Keypoints promedio: 5046.0
 Matches promedio: 1360.0

ORB

Detect + Compute: 69.77 ms
 Matching: 7.39 ms
 Total: 77.16 ms
 FPS aproximado: 12.96
 Keypoints promedio: 1000.0
 Matches promedio: 232.0

1. Ninguno de los dos algoritmos sería útil para los dos casos por los pocos fps. Pero, el que más fps puede alcanzar es ORB. Entonces para el producto A se usa. Se podrían aplicar a distintas modificaciones de la imagen para que los fps suban. Se recomienda ORB.
2. Para el Producto B, SIFT produjo más detección de keypoints de 5046 y match. Al no ser importante, el tiempo de procesamiento y la precisión es importante. Para este producto se recomienda SIFT

Prompt Utilizado

Uso de ChatGPT

Task 2

En la segunda tarea usamos ChatGPT como guía para diseñar el pipeline. Le preguntamos qué norma de distancia correspondía en el `BFMatcher` (L2 para SIFT y Hamming para ORB) y cómo implementar correctamente el Lowe's Ratio Test con un umbral de 0.75 usando `knnMatch`. Con esa guía, construimos la lógica para filtrar los matches y luego armamos la visualización de los "buenos matches" para comparar el desempeño entre detectores/descriptores.

Task 3

Se uso ChatGPT como guía para diseñar un metodo que simulara un entorno real en movimiento para saber cuantos fps por algoritmo se tienen. La idea fue definir una forma consistente de repetir el experimento (mismo tamaño de imagen, misma cantidad de frames y mismas condiciones) para poder comparar el rendimiento de manera justa.

Por qué funcionó este prompt

Ambos prompts funcionaron, dado que se definió con claridad lo que se buscaba lograr y las herramientas a utilizar, de manera que la respuesta fuese accionable y pudiese ser totalmente modificable por nosotros. Esto con el propósito de poder realizar los ajustes necesarios en cuestiones de lógica, performance y resultados.