Universidad Autónoma de San Luis Potosí

Facultad de Ingeniería

Área Mecánica y Eléctrica

Tratamiento de imágenes

Hora de la Materia:

18:00 – 19:00, 20:00

Nombre del profesor:

Jully Kado Mercado Elías

Proyecto #3

Nombre del alumno:

Martínez Carrillo Carlos Ignacio

315671

Índice

Introducción

Mediante el presente proyecto, se realiza un programa en el cual, de una imagen se hacen recortes de regiones como si fuera un rompecabezas. Se deben aplicar métodos para localizar las piezas dentro de la imagen recortada y acomodarlas.

Desarrollo

Explicación del código

Para el programa en cuestión, primero se deben cargar las librerías a utilizar, siendo estas “cv2”, “matplotlib.pyplot” y “numpy”.

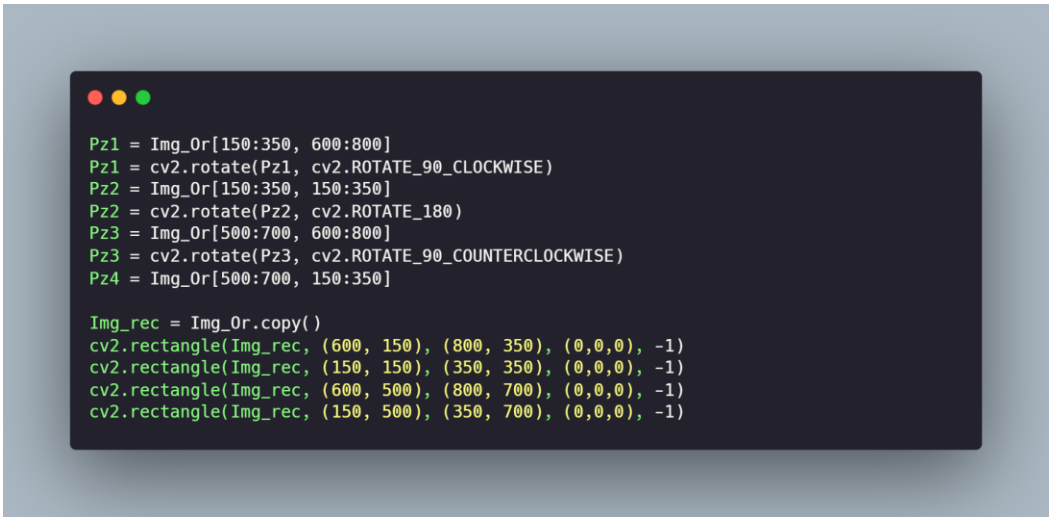
Seguido de esto se debe hacer la lectura de la imagen mediante su ruta.

```
import numpy as np
import cv2

Img_Or = cv2.imread("Img_Original.jpg", cv2.IMREAD_ANYCOLOR)
```

Ilustración 1: carga de librerías e imágenes.

De esta imagen original, se obtienen las “piezas” haciendo recortes. Se crea un espacio en negro en la imagen original donde irán las piezas.



```
Pz1 = Img_Or[150:350, 600:800]
Pz1 = cv2.rotate(Pz1, cv2.ROTATE_90_CLOCKWISE)
Pz2 = Img_Or[150:350, 150:350]
Pz2 = cv2.rotate(Pz2, cv2.ROTATE_180)
Pz3 = Img_Or[500:700, 600:800]
Pz3 = cv2.rotate(Pz3, cv2.ROTATE_90_COUNTERCLOCKWISE)
Pz4 = Img_Or[500:700, 150:350]

Img_rec = Img_Or.copy()
cv2.rectangle(Img_rec, (600, 150), (800, 350), (0,0,0), -1)
cv2.rectangle(Img_rec, (150, 150), (350, 350), (0,0,0), -1)
cv2.rectangle(Img_rec, (600, 500), (800, 700), (0,0,0), -1)
cv2.rectangle(Img_rec, (150, 500), (350, 700), (0,0,0), -1)
```

Ilustración 2: recorte de piezas..

Se crean los objetos SIFT y BF, esto con el fin de obtener características y hacer el matching más adelante. Se define una función para hacer estos matches con las características y con eso crear la homografía. Una vez hecha la homografía para cada imagen, se utiliza la función “warpPerspective()” para acomodar las piezas dentro de la imagen recortada donde van las piezas. Seguido de esto, se utiliza la función “cv2.add()” para añadir la imagen de recortes con las piezas y sobreponerlas. Se muestra la imagen con cada pieza puesta.

```
sift = cv2.SIFT_create()
kp1, desc1 = sift.detectAndCompute(Img_Or, None)
bf = cv2.BFMatcher()

def imprimirMatches(img_pieza, nombre):

    kp2, desc2 = sift.detectAndCompute(img_pieza, None)
    matches = bf.knnMatch(desc1, desc2, k=2)

    good_matches = []
    for m, n in matches:
        if m.distance < 0.40 * n.distance:
            good_matches.append(m)

    src_pts = np.float32([kp1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1,
2)
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in good_matches]).reshape(-1, 1,
2)

    H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 3.0)

    h1, w1 = Img_Or.shape[:2]
    h2, w2 = img_pieza.shape[:2]
    result_width = max(w1, w2)
    result_height = max(h1, h2)
    img_pieza_warped = cv2.warpPerspective(img_pieza, H, (result_width,
result_height))
    Img_res = cv2.add(img_pieza_warped, Img_rec)
    cv2.imshow(nombre, Img_res)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Ilustración 3: Función principal.

Después de esto, se llama a mandar la función principal para cada pieza y visualizarla.

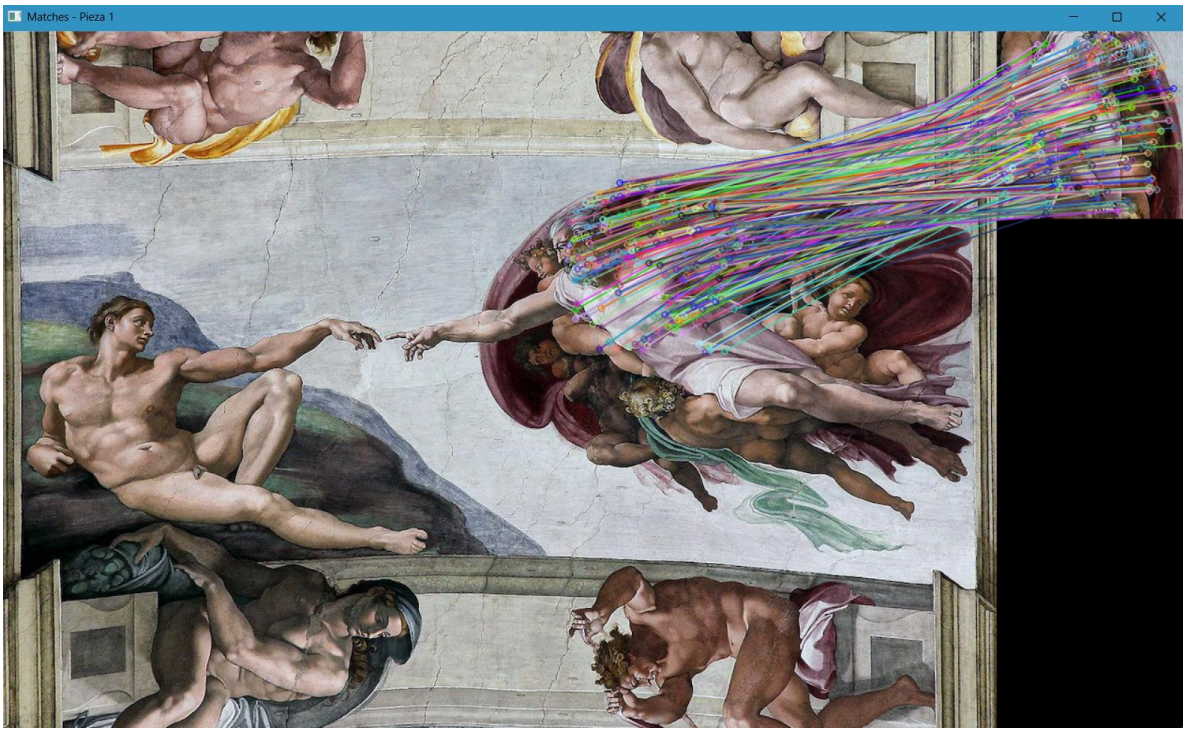
```
imprimirMatches(Pz1, "Pieza 1")
imprimirMatches(Pz2, "Pieza 2")
imprimirMatches(Pz3, "Pieza 3")
imprimirMatches(Pz4, "Pieza 4")
```

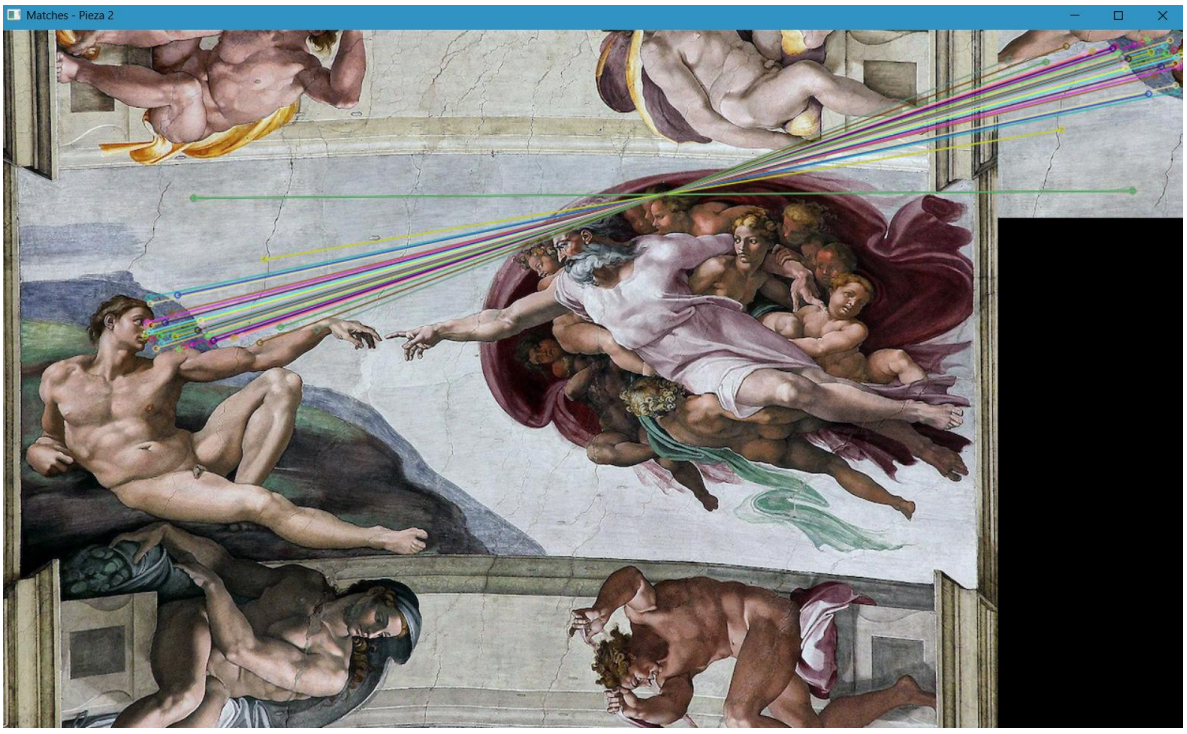
Ilustración 4: Llamadas a función.

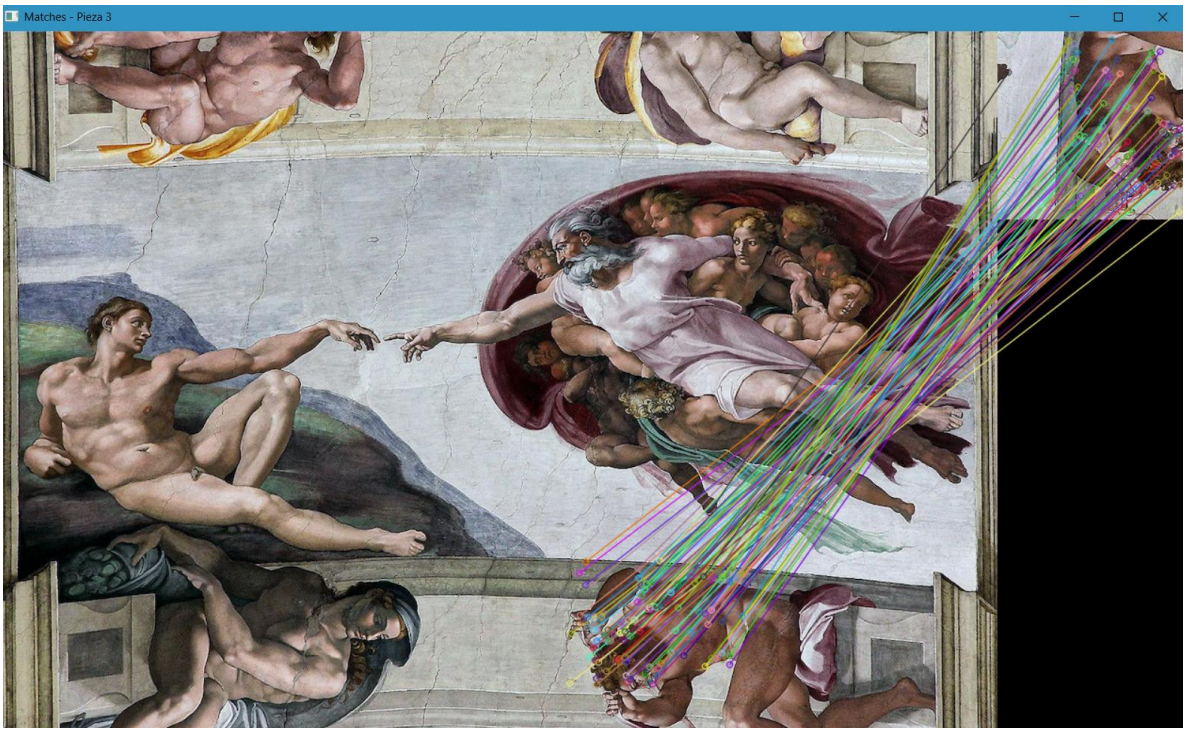
Posibles mejoras

Mediante el código, solo se logró acomodar una pieza, aunque todas tuvieron un buen matching. La mejora del código sería averiguar la razón de esto.

Resultados




Matches - Pieza 2


Matches - Pieza 3

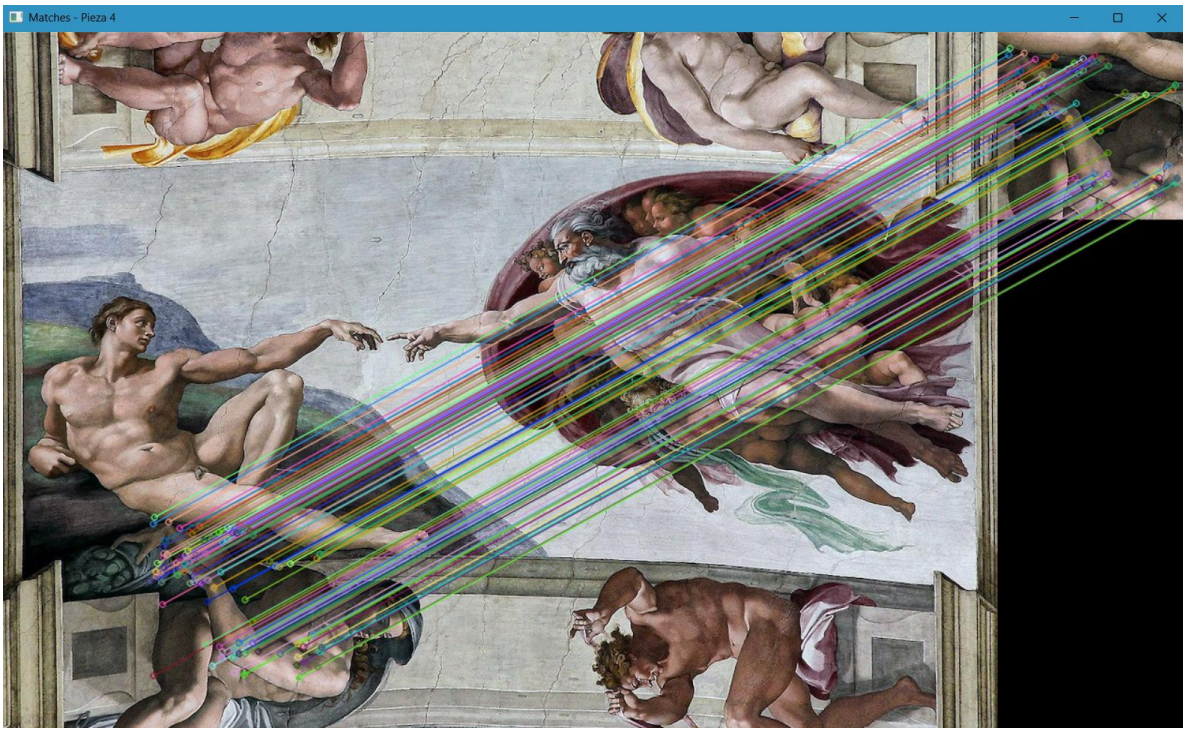


Ilustración 5: Matches con cada pieza en imagen original.

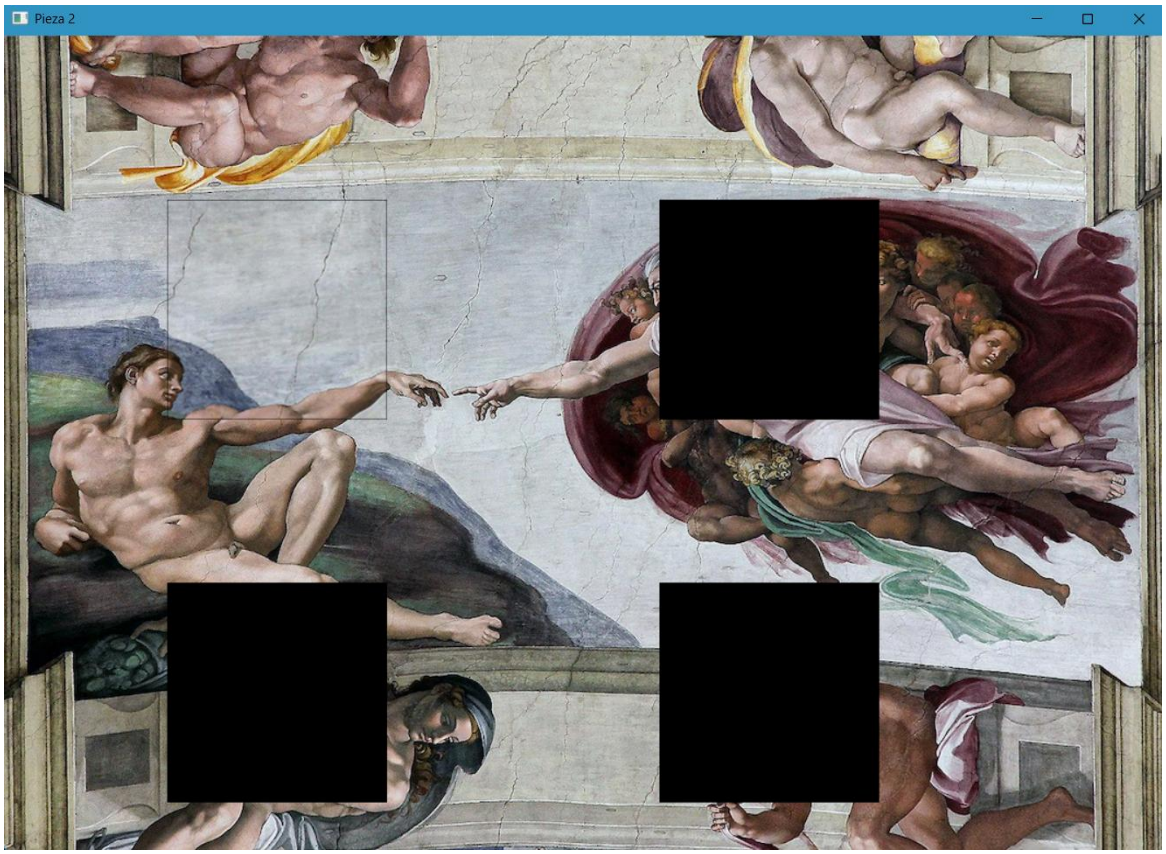


Ilustración 6: Superposición de pieza en imagen recortada.

Conclusiones

En este proyecto intentamos reconstruir una imagen completa usando piezas recortadas y rotadas. Utilizamos SIFT para detectar características, BFMatcher para emparejar puntos clave y homografía para alinear las piezas en su posición original.

Aunque logramos acomodar bien una de las piezas, el resto no quedó en su lugar como esperábamos. El problema estuvo en que la homografía no siempre fue precisa, lo que hizo que algunas piezas no encajaran bien. Esto nos deja claro que el método que usamos necesita algunos ajustes o quizás probar técnicas más robustas para mejorar el resultado.

A pesar de no lograr el ensamblaje completo, el proyecto nos permitió experimentar con procesamiento de imágenes y reflexionar sobre cómo abordar mejor este tipo de problemas en el futuro.

Referencias y bibliografía

- *OpenCV: Geometric Image Transformations. (s. f.). https://docs.opencv.org/4.x/da/d54/group__imgproc__transform.html*
- OpenAI. (2025). ChatGPT (marzo 14, versión GPT-4). https://chat.openai.com/