ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

BÁO CÁO
DIGITAL IMAGE PROCESSION AND COMPUTER VISION
PROJECT 4
TRANSFORMATIONS

GVHD: Ths. Võ Thanh Hùng

—o0o—

SVTH: Nguyễn Tiến Phát - 2011797

TP. HỒ CHÍ MINH, 4/2024

Mục lục

Danh sách hình vẽ

Danh sách bảng

Chương 1

Giới thiệu

1.1 Khái niệm về Transformation

Trong lĩnh vực xử lý ảnh và thị giác máy tính, khái niệm "Transformation" đóng một vai trò quan trọng, đặc biệt là trong việc hiểu và áp dụng các phép biến đổi hình học. Hai loại biến đổi hình học phổ biến và cơ bản là affine và projective transformations. Phép biến đổi affine bao gồm các hoạt động như tịnh tiến, xoay, co giãn, và phản xạ, giữ nguyên các đặc tính như đường thẳng, tỉ lệ đoạn thẳng trên cùng một đường, và song song. Điều này làm cho nó trở nên hữu ích trong nhiều ứng dụng như căn chỉnh hình ảnh, biến đổi hình dạng, và nhiều hơn nữa. Mặt khác, phép biến đổi projective mở rộng khái niệm của affine bằng cách thêm khả năng biến đổi góc nhìn, mô phỏng hiệu ứng của việc nhìn một đối tượng từ các vị trí khác nhau, điều này rất quan trọng trong việc tạo ra sự hiểu biết sâu sắc về cấu trúc không gian và quan hệ giữa các đối tượng. Cả hai loại biến đổi này đều có thể được biểu diễn thông qua ma trận và có thể được tích hợp chặt chẽ với các thuật toán xử lý ảnh để cải thiện chất lượng và thông tin chi tiết của hình ảnh, cũng như để hỗ trợ trong việc phát hiện và nhận dạng đối tượng.

Để hiểu rõ hơn về cách thức hoạt động của chúng, cần phải nắm vững các khái niệm toán học cơ bản liên quan đến không gian vector và ma trận, cũng như cách thức chúng tương tác với dữ liệu hình ảnh. Khi đã hiểu được bản chất của các phép biến đổi này, người học có thể áp dụng chúng một cách linh hoạt để giải quyết các vấn đề cụ thể trong lĩnh vực xử lý ảnh và thị giác máy tính, từ đó mở ra những khả năng mới trong việc tạo ra và phân tích hình ảnh.

Chương 2

Hiện thực

2.1 Cơ sở lý thuyết

2.1.1 Affine

Phép biến đổi affine là dạng đơn giản nhất của biến đổi hình ảnh. Chúng là tuyến tính và bao gồm các phép biến đổi như tịnh tiến, co giãn, xoay, cắt và phản xạ. Do có những đặc tính khá giống nhau trong quá trình biến đổi, ta sẽ tổng quát hóa một định nghĩa thống nhất cho các phép biến đổi thuộc nhóm này. Ở đây, chúng ta sẽ hướng về việc sử dụng ma trận làm biến đổi tuyến tính để xác định các biến đổi trong affine. Có nghĩa là phép biến đổi affine ược định nghĩa thông qua việc sử dụng ma trận để biểu diễn các phép biến đổi tuyến tính, sau đó kết hợp với phép tịnh tiến.

Trong không gian 2D, phép biến đổi affine được biểu diễn bằng phương trình:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

Trong đó ma trận $\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ đại diện cho phép biến đổi tuyến tính và vector $\begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$ là phép tịnh tiến. Từ đây, ta có công thức cho các phép biến đổi trong affine.

Scaling matrix:

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

Rotation matrix:

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

Translation matrix:

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a_x \\ 0 & 1 & a_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

2.1.2 Projective

Phép biến đổi projective là một song ánh (one-to-one mapping) biến điểm thành điểm, và bảo toàn sự thẳng hàng. Do sự thẳng hàng được bảo toàn, nên phép biến đổi projective sẽ biến đường thẳng thành đường thẳng, và mặt phẳng thành mặt phẳng. Ngoài ra còn những tính chất khác được bảo toàn là division ratio (với affine transformation) và cross ratio (với projective transformation) nhưng sẽ không đi sâu vào những tính chất này.

Trong không gian hai chiều, một phép biến đổi xạ ảnh có thể được biểu diễn bằng một ma trận (3 x 3) với các phần tử là số thực. Điểm quan trọng là ma trận này phải có định thức khác không để đảm bảo nó khả nghịch, tức là có thể tìm được phép biến đổi ngược lại.

Một điểm trên mặt phẳng được biểu diễn bằng tọa độ đồng nhất ((x, y, 1)), và phép biến đổi xạ ảnh được áp dụng lên điểm này thông qua phép nhân ma trận:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax+by+cz \\ dx+ey+fz \\ gx+hy+iz \end{bmatrix}$$

Sau khi thực hiện phép nhân ma trận, ta sẽ thu được một vector mới trong tọa độ đồng nhất. Để chuyển vector này về tọa độ thông thường, ta chia mỗi thành phần của vector kết quả cho thành phần thứ ba (miễn là nó không bằng 0):

$$\frac{[ax+by+c \quad dx+ey+f]}{[gx+hy+i \quad gx+hy+j]}$$

Đây là tọa độ của điểm sau khi đã áp dụng phép biến đổi xạ ảnh.

2.2 Hiện thực với python

Toàn bộ source code có thể tìm thấy tại đây: Google colab

2.2.1 Affine

Cài đặt hàm dùng để transformation

```python
import numpy as np

def transform_image(img, transformation_matrix):
    # Convert image to numpy array
    img = np.array(img)
    # Get image dimensions
    height, width, _ = img.shape

    # Define corner points of the image
    top_left = np.array([[0, 0, 1]], dtype='double').reshape(3, 1)
    top_right = np.array([0, width-1, 1], dtype='double').reshape(3, 1)
    bottom_left = np.array([[height-1, 0, 1]], dtype='double').reshape(3, 1)
    bottom_right = np.array([[height-1, width-1, 1]], dtype='double').reshape(3, 1)

    # Apply transformation to corner points
    new_top_left = transformation_matrix.dot(top_left)
    new_top_left = (new_top_left / new_top_left[2]).astype("int64")
    new_top_right = transformation_matrix.dot(top_right)
    new_top_right = (new_top_right / new_top_right[2]).astype("int64")
    new_bottom_left = transformation_matrix.dot(bottom_left)
    new_bottom_left = (new_bottom_left / new_bottom_left[2]).astype("int64")
    new_bottom_right = transformation_matrix.dot(bottom_right)
    new_bottom_right = (new_bottom_right / new_bottom_right[2]).astype("int64")

    # Calculate dimensions of the transformed image
    new_top = np.max([new_top_left[0], new_top_right[0], new_bottom_left[0], new_bottom_right[0]]) + 1
    new_bottom = np.min([new_top_left[0], new_top_right[0], new_bottom_left[0], new_bottom_right[0]]) - 1
    new_right = np.max([new_top_left[1], new_top_right[1], new_bottom_left[1], new_bottom_right[1]]) + 1
    new_left = np.min([new_top_left[1], new_top_right[1], new_bottom_left[1], new_bottom_right[1]]) - 1

    # Handle cases where new dimensions are negative
    if new_bottom < 0:
        if new_top < 0:
            new_height = -new_bottom + 1
        else:
            new_height = new_top - new_bottom + 1
    else:
        new_height = new_top + 1

    if new_left < 0:
```

```
        if new_right < 0:
            new_width = -new_left + 1
        else:
            new_width = new_right - new_left + 1
    else:
        new_width = new_right + 1

    # Create array for transformed image
    new_img = np.zeros((new_height, new_width, 3), dtype=img.dtype)

    # Get the inverse of the transformation matrix
    transformation_matrix_inverse = np.linalg.inv(transformation_matrix)

    # Iterate through each pixel in the transformed image
    for i in range(new_height):
        for j in range(new_width):
            new_position = np.array([[i, j, 1]]).reshape(3, 1)
            if new_bottom < 0:
                new_position[0, 0] += new_bottom
            if new_left < 0:
                new_position[1, 0] += new_left
            position = transformation_matrix_inverse.dot(new_position).reshape(3)
            if abs(position[2]) > 0.00001:
                position = position / position[2]
            if 0 <= position[0] < height and 0 <= position[1] < width:
                new_img[i, j] = img[int(position[0]), int(position[1])]

    return new_img
```

2.2.1.1 Translation

```
translation_kernel = np.array([[1, 0, 47],
                               [0, 1, 47],
                               [0, 0, 1]])
translated_image = transform_image(img_list[2], translation_kernel)
```

2.2.1.2 Rotation

```
def rotation_matrix(angle):
    cos_theta = np.cos(angle)
    sin_theta = np.sin(angle)
    return np.array([[cos_theta, -sin_theta, 0],
                     [sin_theta, cos_theta, 0],
                     [0, 0, 1]])

angle_rad = np.deg2rad(90)
rotation_kernel = rotation_matrix(angle_rad)
rotation_image = transform_image(img_list[2], rotation_kernel)
```

2.2.1.3 Scaling

```
scale_kernel = np.array([[2, 0, 0],
                         [0, 2, 0],
                         [0, 0, 1]])

scaled_image = transform_image(img_list[2], scale_kernel)
```

2.2.1.4 Shear

```
shear_Matrix = np.array([[1, 0.5, 0],
                        [0.5, 1, 0],
                        [0, 0, 1]])
sheared_image = transform_image(img_list[2], shear_kernel)
```

2.2.2 Projective

Cài đặt một vài hàm dùng để phụ trợ cần thiết

```
def warp_and_combine(poster_img, background_img, anchor_lst):
    # Convert images to numpy arrays
    poster_img = np.array(poster_img)
    background_img = np.array(background_img)

    # Get the shape of the foreground image
    height, width, _ = poster_img.shape

    # Define the coordinates of the corners of the poster image
    origin_lst = np.array([(0, 0), (height-1, 0), (height-1, width-1), (0, width-1)], dtype=np.double)

    # Initialize transformation matrix A and target vector b
    A = np.zeros((8, 8), dtype=np.double)
```

```
    b = np.zeros((8, 1), dtype=np.double)

    for i in range(8):
        id = i // 2
        x, y = origin_lst[id]
        if i % 2 == 0:
            A[i, :3] = [x, y, 1]
            A[i, 6:] = [-x * anchor_lst[id][0], -y * anchor_lst[id][0]]
            b[i, 0] = anchor_lst[id][0]
        else:
            A[i, 3:6] = [x, y, 1]
            A[i, 6:] = [-x * anchor_lst[id][1], -y * anchor_lst[id][1]]
            b[i, 0] = anchor_lst[id][1]

    # Solve for the transformation coefficients
    res = np.linalg.solve(A, b)

    # Create the transformation matrix
    M = np.ones((3, 3), dtype="double")
    for i in range(3):
        for j in range(3):
            if i * 3 + j < 8:
                M[i, j] = res[i * 3 + j, 0]

    # Warp the foreground image
    warped_img = transform_image(poster_img, M)

    # Copy the background image
    return_img = background_img.copy()

    # Combine the warped poster image with the background image
    for i in range(warped_img.shape[0]):
        for j in range(warped_img.shape[1]):
            if np.all(warped_img[i, j] != [0, 0, 0]):  # Check if pixel is not black
                return_img[i, j] = warped_img[i, j]

    return return_img

def plot_image_pointed(img, img_name="", points=[], color="red", size=1):
    # Convert points to the correct format
    converted_points = sorted([(y, x) for x, y in points], key=lambda point: point[0], reverse=True)

    plt.imshow(img)
    plt.title(img_name)

    for point in converted_points:
        x, y = point
        plt.scatter(x, y, c=color, s=size)

    return
```

2.2.3 Image in elevator

```
place_poited_img_tv = [(820, 1210), (2460, 1070), (2530, 1900), (1080, 1940)]

plot_image_pointed(
    img=img_list[1],
    img_name='bk_tv',
    points=place_poited_img_tv,
    size=10
```

```
)

tv_with_poster = warp_and_combine(
    img_list[2],
    img_list[1],
    place_poited_img_tv
)
```

2.2.4 Image on BK H6

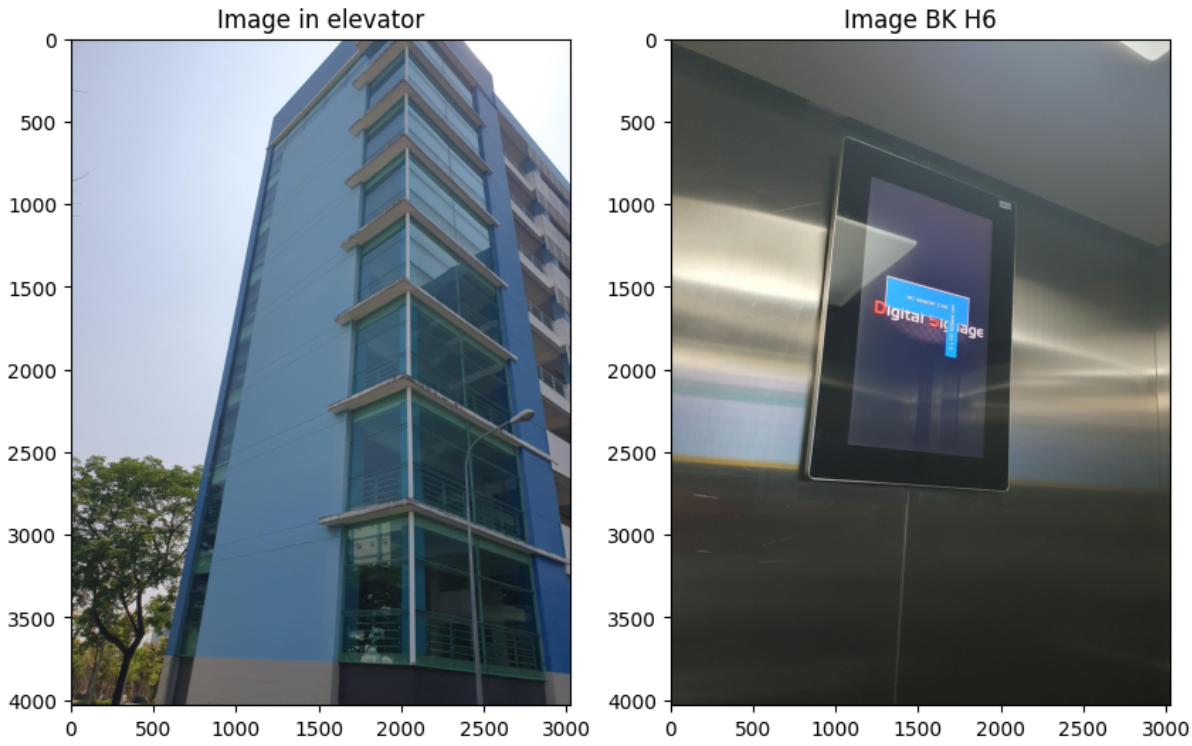
```
place_poited_img_h6 = [(2140, 950), (3120, 750), (2950, 1500), (1800, 1550)]
plot_image_pointed(
    img=img_list[0],
    img_name='bk_h6',
    points=place_poited_img_h6,
    size=10
)

transform_img = warp_and_combine(
    img_list[2],
    img_list[0],
    place_poited_img_h6
)
```

Chương 3

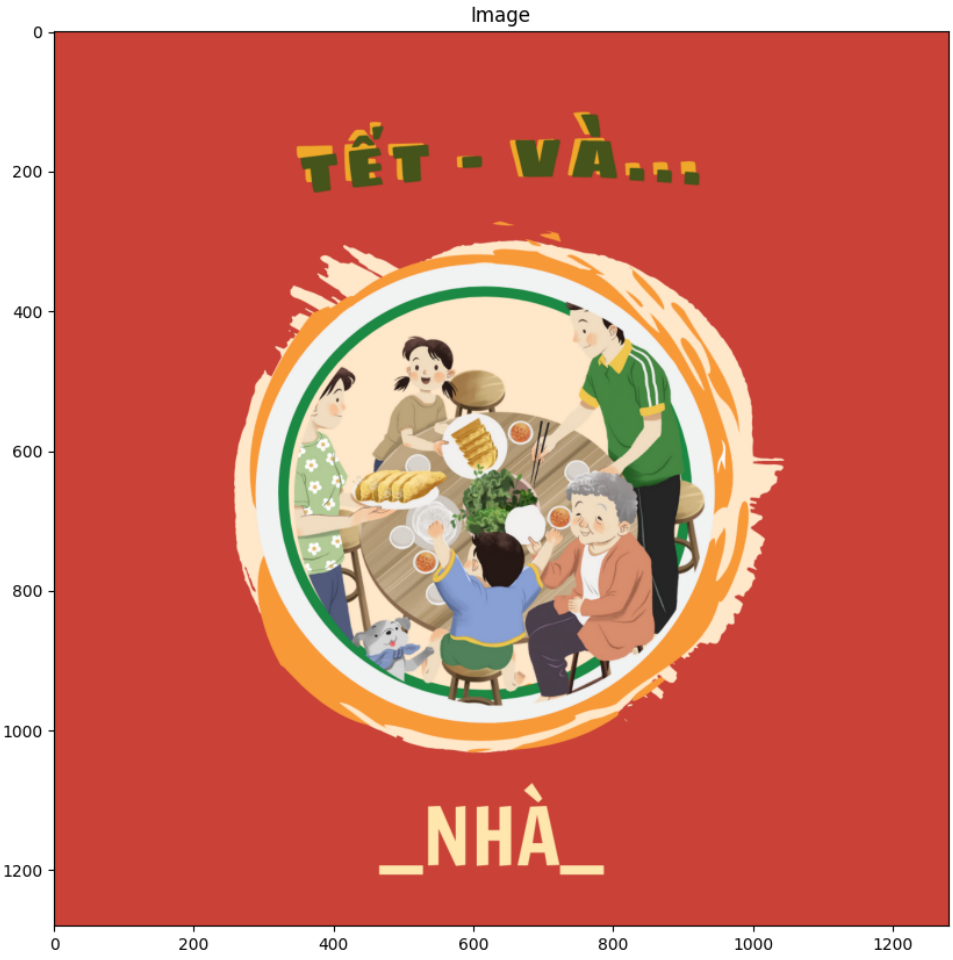
Kết quả đạt được và tổng kết

3.1 Kết quả

3.1.1 Bộ ảnh sử dụng



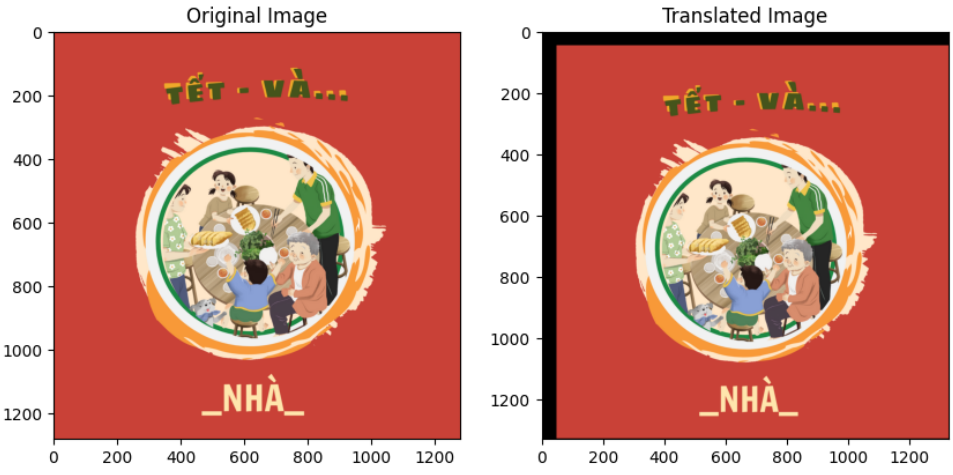
Hình 3.1: *Background*



Hình 3.2: *Poster*

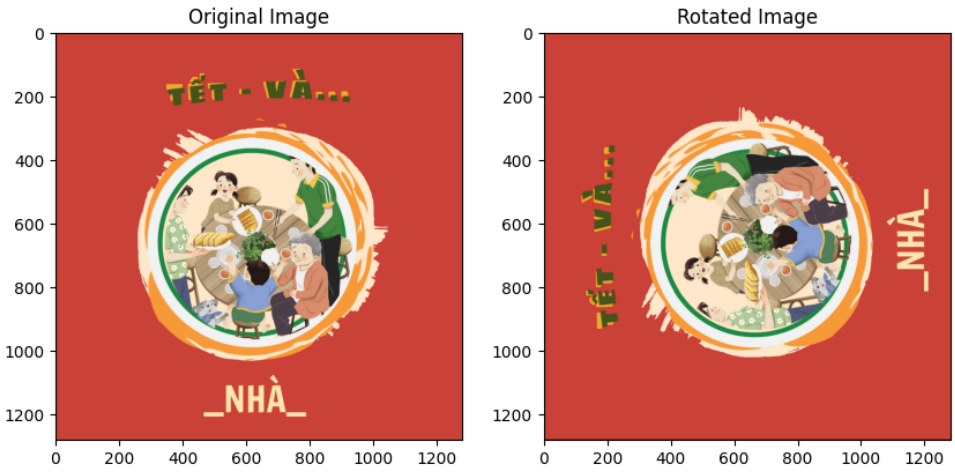
3.1.2 Affine

3.1.2.1 Translation



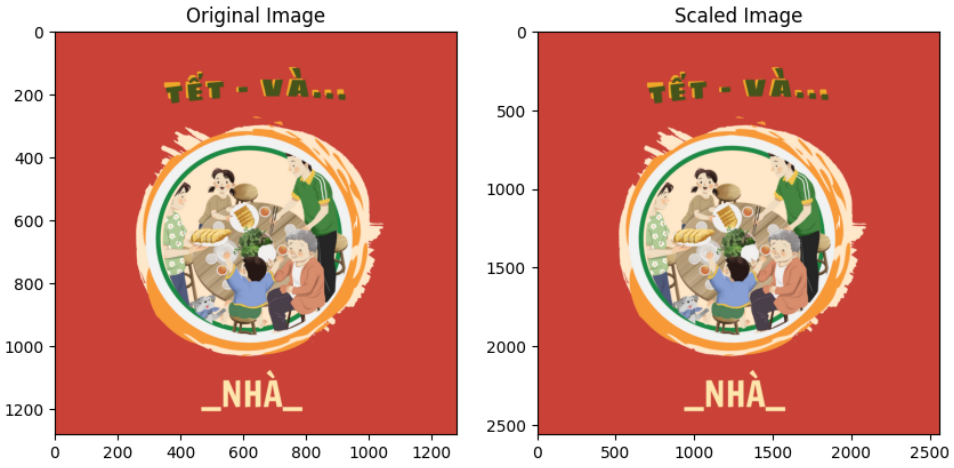
Hình 3.3: *Image before and after using Translation*

3.1.2.2 Rotation



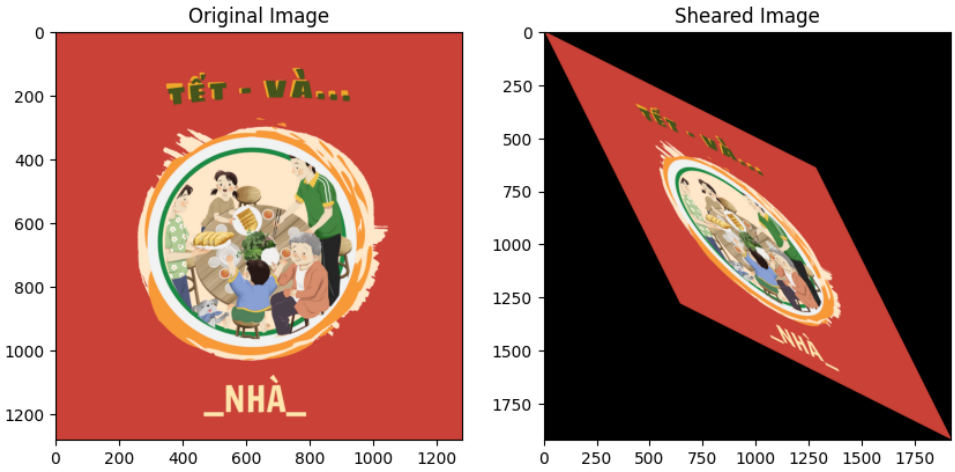
Hình 3.4: *Image before and after using Rotation*

3.1.2.3 Scale



Hình 3.5: *Image before and after using Scale*

3.1.2.4 Shear

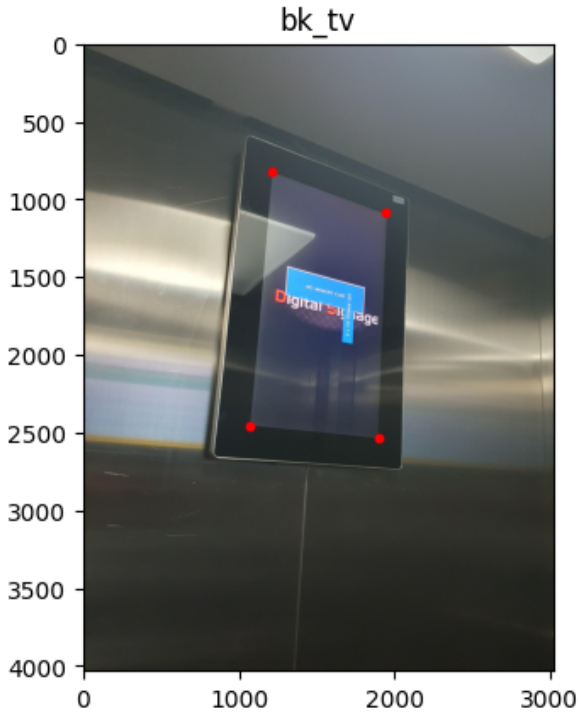


Hình 3.6: *Image before and after using Shear*

3.1.3 Projective

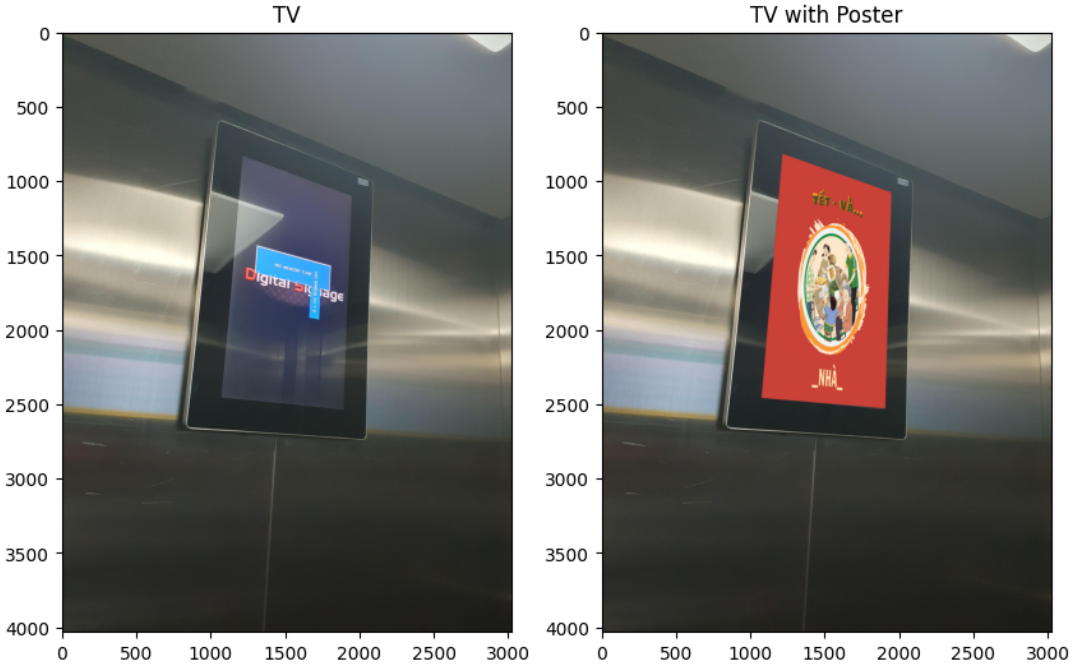
3.1.3.1 Ảnh 1

Xác định khu vực thêm ảnh:



Hình 3.7: *Background with 4 pointed corner*

Kết quả:



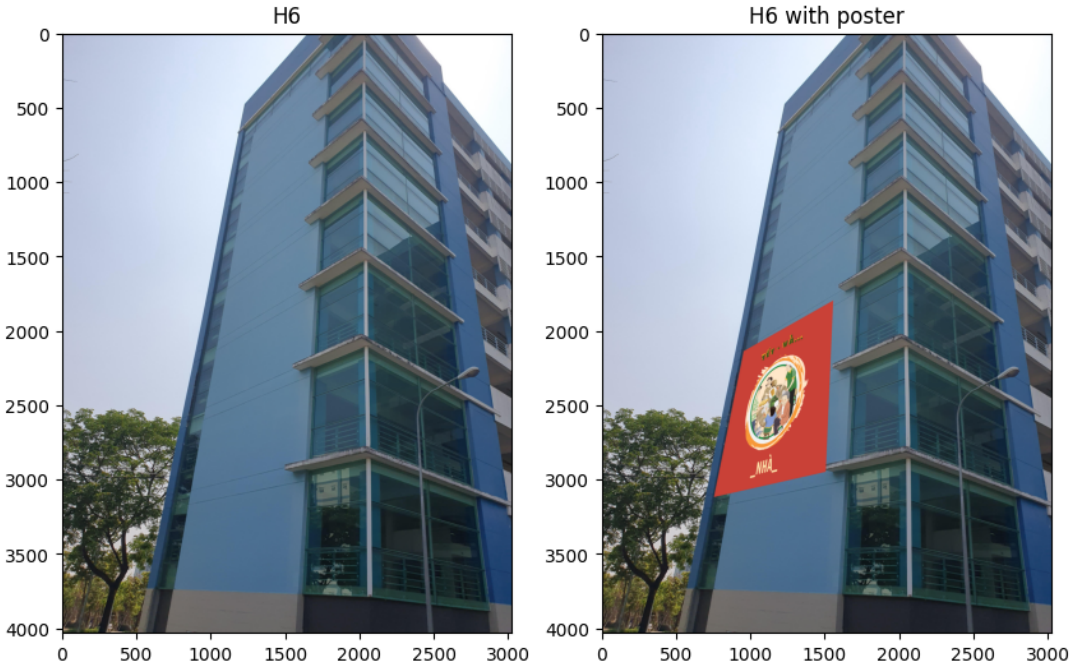
Hình 3.8: *Background before and after add Poster*

3.1.3.2 Ảnh 2

Xác định khu vực thêm ảnh:

Hình 3.9: *Background with 4 pointed corner*

Kết quả:



Hình 3.10: *Background before and after add Poster*

3.2 Tổng kết

Sau khi thực hiện nghiên cứu và thực hiện các phép biến đổi affine và projective, chúng ta có thể rút ra một số điểm so sánh và đánh giá giữa hai loại biến đổi này

Bảng 3.1: *So sánh Affine và Projective Transform*

	Affine	**Projective**
Tính chất tuyến tính	Có	Không
Bảo toàn đường thẳng	Có	Có
Ứng dụng	Xử lý ảnh, biến đổi hình dạng	Hình chiếu, biến đổi phức tạp
Homography	Không	Có
Độ khó	Dễ	Phức tạp

Trong dự án nghiên cứu về biến đổi affine và projective, chúng ta đã khám phá những khái niệm toán học sâu sắc và ứng dụng của chúng trong nhiều lĩnh vực khác nhau, từ xử lý ảnh đến thị giác máy tính. Biến đổi affine, với khả năng bảo toàn các điểm, đường thẳng, và tỷ lệ, đã cho thấy sự linh hoạt trong việc thay đổi kích thước, xoay, và dịch chuyển các đối tượng. Mặt khác, biến đổi projective mở rộng khả năng này bằng cách thêm vào khả năng biến đổi góc nhìn, tạo ra hiệu ứng góc nhìn thực tế hơn trong các ứng dụng 3D.

Bài học lớn nhất từ dự án này là sự quan trọng của việc hiểu rõ cơ sở toán học đằng sau các công nghệ không quá phức tạp này. Sự hiểu biết này không chỉ giúp chúng ta sử dụng công nghệ một cách hiệu quả hơn mà còn mở ra khả năng sáng tạo và đổi mới trong cách chúng ta áp dụng các công cụ này để giải quyết vấn đề. Ngoài ra, dự án cũng nhấn mạnh tầm quan trọng của việc đọc và nghiên cứu tài liệu liên qua nhiều đến toán.

Kết luận, dự án transformation đã cung cấp một cái nhìn toàn diện về biến đổi affine và projective, từ lý thuyết đến thực hành, và từ đó mở rộng hiểu biết của chúng ta trong lĩnh vực xử lí ảnh nói riêng và ngành khoa học máy tính nói chung. Điều này không chỉ làm sâu sắc thêm kiến thức của chúng ta mà còn khuyến khích chúng ta tiếp tục tìm tòi và học hỏi, không ngừng nâng cao kỹ năng của mình để đối mặt với những thách thức mới trong tương lai.

3.3 Tài liệu tham khảo

1. Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision* (2nd Edition). Cambridge University Press, 2004.
2. Szeliski, Richard. *Computer Vision: Algorithms and Applications*. Springer, 2010.
3. Ma, Yi, Stefano Soatto, Jana Kosecka, and S. Shankar Sastry. *An Invitation to 3D Vision: From Images to Geometric Models*. Springer, 2004.
4. Gonzalez, Rafael C., and Richard E. Woods. *Digital Image Processing* (4th Edition). Pearson, 2022.
5. Sonka, Milan, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision* (4th Edition). Cengage Learning, 2019.
6. Fattal, Raanan; Lischinski, Dani; Werman, Michael (2002). "Gradient domain high dynamic range compression" (PDF). Proceedings of the 29th annual conference on Computer graphics and interactive techniques - SIGGRAPH '02. p. 249. doi:10.1145/566570.566573. ISBN 1581135211. S2CID 1650337.

3.4 Phụ lục