画像と空間の幾何学変換

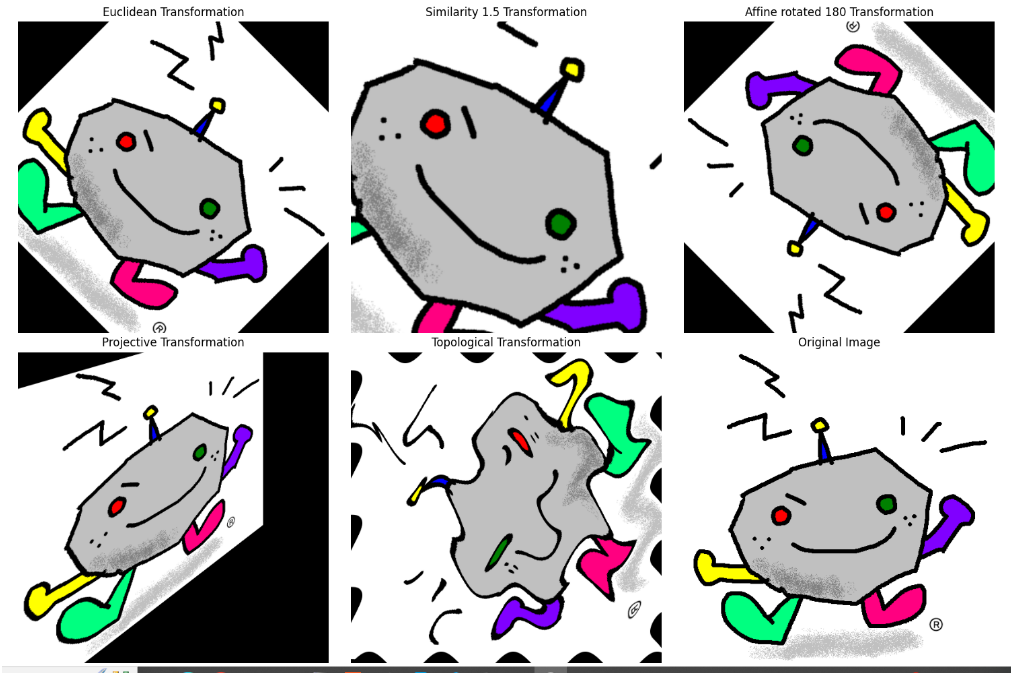
This time learned how to rotate, zoom, compress, invert images... And so on many operations, while using cv2 and the use of matrix calculation to realize.
Learned in the picture binarization, a variety of methods to find the bipolar value, I think the OTSU method is the best!

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

# Load an image
image = cv2.imread('images/kan.png')
# Get the image size
height, width = image.shape[:2]
# Define the transformation matrix for rotation and scaling
M = cv2.getRotationMatrix2D((width//2, height//2), -45, 1)  # Rotate by 45 degrees
euclidean_image  = cv2.warpAffine(image, M, (width, height))
# Define the similarity transformation matrix (scaling, rotation, translation)
M_similarity = cv2.getRotationMatrix2D((width // 2, height // 2), 0, 1.5)
similarity_image = cv2.warpAffine(euclidean_image, M_similarity, (width, height))
rotated_image = cv2.rotate(euclidean_image, cv2.ROTATE_180)
# Define the source and destination points for projective transformation
pts1 = np.float32([[50, 50], [200, 50], [50, 200], [200, 200]])
pts2 = np.float32([[50, 100], [200, 50], [50, 250], [200, 180]])
# Compute the homography matrix
M = cv2.getPerspectiveTransform(pts1, pts2)
projective_image = cv2.warpPerspective(image, M, (width, height))
# Example of a simple topological transformation: image warping
map_x, map_y = np.indices((height, width), dtype=np.float32)
map_x = map_x + 20 * np.sin(map_y / 20)  # Apply a sine distortion in the x-direction
map_y = map_y + 20 * np.cos(map_x / 20)  # Apply a cosine distortion in the y-direction
topological_image = cv2.remap(image, map_x, map_y, cv2.INTER_LINEAR)

fig, axs = plt.subplots(2, 3, figsize=(15, 10))
axs[0, 0].imshow(cv2.cvtColor(euclidean_image, cv2.COLOR_BGR2RGB))
axs[0, 0].set_title('Euclidean Transformation')
axs[0, 1].imshow(cv2.cvtColor(similarity_image, cv2.COLOR_BGR2RGB))
axs[0, 1].set_title('Similarity 1.5 Transformation')
axs[0, 2].imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB))
axs[0, 2].set_title('Affine rotated 180 Transformation')
axs[1, 0].imshow(cv2.cvtColor(projective_image, cv2.COLOR_BGR2RGB))
```

```
axs[1, 0].set_title('Projective Transformation')
axs[1, 1].imshow(cv2.cvtColor(topological_image, cv2.COLOR_BGR2RGB))
axs[1, 1].set_title('Topological Transformation')
axs[1, 2].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
axs[1, 2].set_title('Original Image')
for ax in axs.flat:
    ax.axis('off')
plt.tight_layout()
plt.show()
```

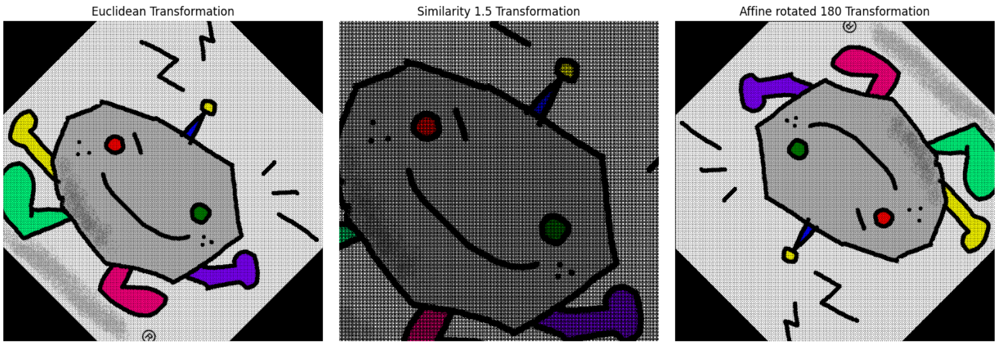



```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load an image
image = cv2.imread('images/kan.png')
image = cv2.imread('image.png')
height, width = image.shape[:2]
theta = -45
theta_rad = np.deg2rad(theta)
cos_theta = np.cos(theta_rad)
sin_theta = np.sin(theta_rad)
rotation_matrix = np.array([[cos_theta, -sin_theta], [sin_theta, cos_thet
a]])
center = np.array([width // 2, height // 2])
euclidean_image = np.zeros_like(image)
for y in range(height):
    for x in range(width):
```

```
        offset = np.array([x, y]) - center
        new_offset = np.dot(rotation_matrix, offset)
        new_x, new_y = new_offset + center
        new_x, new_y = int(round(new_x)), int(round(new_y))
        if 0 <= new_x < width and 0 <= new_y < height:
            euclidean_image[new_y, new_x] = image[y, x]
scale_x = 1.5
scale_y = 1.5
scaling_matrix = np.array([[scale_x, 0], [0, scale_y]])
similarity_image = np.zeros_like(euclidean_image)
for y in range(height):
    for x in range(width):
        offset = np.array([x, y]) - center
        new_offset = np.dot(scaling_matrix, offset)
        new_x, new_y = new_offset + cente
        new_x, new_y = int(round(new_x)), int(round(new_y))
        if 0 <= new_x < width and 0 <= new_y < height:
            similarity_image[new_y, new_x] = euclidean_image[y, x]
rotation_matrix = np.array([[-1, 0], [0, -1]])
center = np.array([width // 2, height // 2])
rotated_image = np.zeros_like(euclidean_image)
for y in range(height):
    for x in range(width):
        offset = np.array([x, y]) - center
        new_offset = np.dot(rotation_matrix, offset)
        new_x, new_y = new_offset + center
        new_x, new_y = int(round(new_x)), int(round(new_y))
        if 0 <= new_x < width and 0 <= new_y < height:
            rotated_image[new_y, new_x] = euclidean_image[y, x]

fig, axs = plt.subplots(2, 3, figsize=(15, 10))
axs[0, 0].imshow(cv2.cvtColor(euclidean_image, cv2.COLOR_BGR2RGB))
axs[0, 0].set_title('Euclidean Transformation')
axs[0, 1].imshow(cv2.cvtColor(similarity_image, cv2.COLOR_BGR2RGB))
axs[0, 1].set_title('Similarity 1.5 Transformation')
axs[0, 2].imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB))
axs[0, 2].set_title('Affine rotated 180 Transformation')
for ax in axs.flat:
    ax.axis('off')
plt.tight_layout()
plt.show()
```

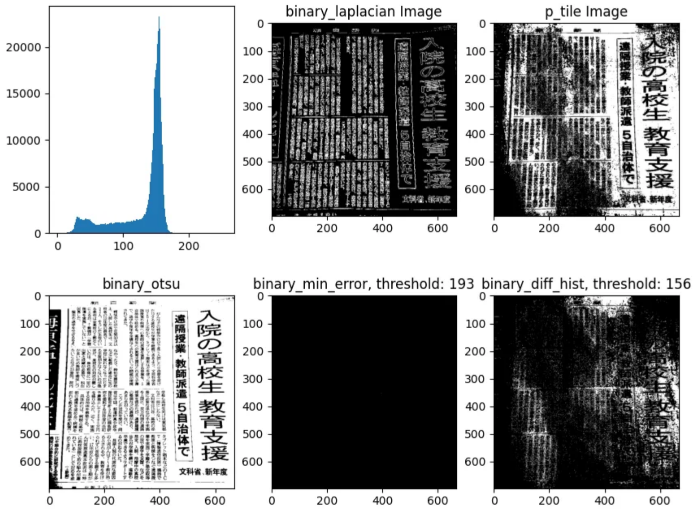


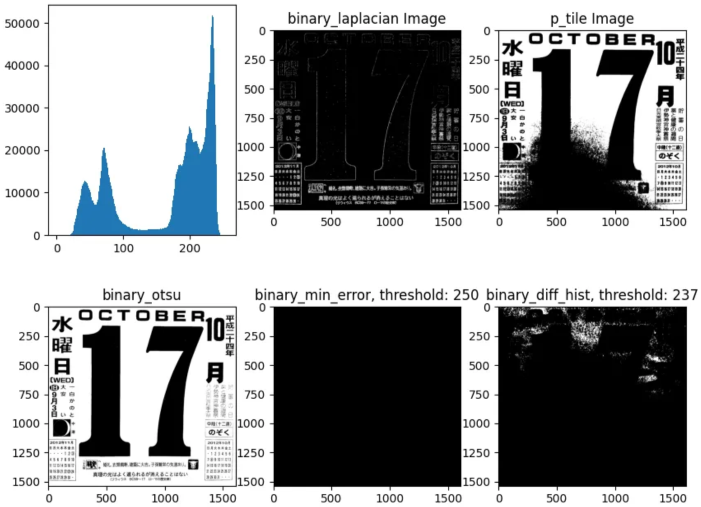

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
image_path = 'images/news.png'
image = cv2.imread(image_path, 1)
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
def p_tile_method(image, p=50):
    flattened = image.flatten()
    threshold = np.percentile(flattened, p)
    _, binary_image = cv2.threshold(image, threshold, 255, cv2.THRESH_BINAR
Y)
    return binary_image
_, binary_otsu = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY + cv2.
THRESH_OTSU)

def minimum_error_threshold(image):
    hist = cv2.calcHist([image], [0], None, [256], [0, 256]).flatten()
    hist = hist / hist.sum()
    bins = np.arange(256)
    total_mean = (bins * hist).sum()
    min_error = float('inf')
    best_threshold = 0
    for t in bins:
        w0 = hist[:t].sum()
        w1 = hist[t:].sum()
        if w0 == 0 or w1 == 0:
            continue
        mean0 = (bins[:t] * hist[:t]).sum() / w0
        mean1 = (bins[t:] * hist[t:]).sum() / w1
        error = w0 * (mean0 - total_mean)**2 + w1 * (mean1 - total_mean)**2
        if error < min_error:
            min_error = error
            best_threshold = t
    print(f"Best threshold: {best_threshold}")
    _, binary_image = cv2.threshold(image, best_threshold, 255, cv2.THRESH_
BINARY)
```

```
    return binary_image, best_threshold
binary_min_error, binary_min_error_threshold = minimum_error_threshold(gray
_image)

def differential_histogram_method(image):
    # Calculate histogram
    hist = cv2.calcHist([image], [0], None, [256], [0, 256]).flatten()
    diff = np.diff(hist)
    # Find minimum index in the derivative (local minimum)
    threshold = np.argmin(diff)
    _, binary_image = cv2.threshold(image, threshold, 255, cv2.THRESH_BINAR
Y)
    return binary_image, threshold
binary_diff_hist, binary_diff_hist_threshold = differential_histogram_metho
d(gray_image)
def laplacian_histogram_method(image):
    # Apply Laplacian operator
    laplacian = cv2.Laplacian(image, cv2.CV_64F)
    laplacian = cv2.convertScaleAbs(laplacian)
    # Threshold using Otsu's method
    _, binary_image = cv2.threshold(laplacian, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
    return binary_image
binary_laplacian = laplacian_histogram_method(gray_image)
plt.figure(figsize=(10, 8))
plt.subplot(2, 3, 1)
plt.hist(gray_image.ravel(), 256, [0, 256])
plt.subplot(2, 3, 2)
plt.imshow(binary_laplacian, cmap='gray')
plt.title('binary_laplacian Image')
plt.subplot(2, 3, 3)
plt.imshow(p_tile_method(gray_image, 50), cmap='gray')
plt.title('p_tile Image')
plt.subplot(2, 3, 4)
plt.imshow(binary_otsu, cmap='gray')
plt.title('binary_otsu')
plt.subplot(2, 3, 5)
plt.imshow(binary_min_error, cmap='gray')
plt.title(f'binary_min_error, threshold: {binary_min_error_threshold}')
plt.subplot(2, 3, 6)
plt.imshow(binary_diff_hist, cmap='gray')
plt.title(f'binary_diff_hist, threshold: {binary_diff_hist_threshold}')
plt.show()
cv2.waitKey(0)
```


binary_laplacian Image
p_tile Image
binary_otsu
binary_min_error, threshold: 193
binary_diff_hist, threshold: 156


binary_laplacian Image
p_tile Image
binary_otsu
binary_min_error, threshold: 250
binary_diff_hist, threshold: 237