Topic 6: Geometric Transformation

Geometric transformation using Python is a powerful tool for image processing and computer vision. Some of the key concepts covered in this topic include:

- Affine transformation: a geometric transformation that preserves parallel lines and ratios of distances.
- Non-affine transformation: a geometric transformation that doesn't preserve parallel lines and ratios of distances.
- Homogeneous coordinates: a mathematical concept used to represent points in space, which allows for easy implementation of geometric transformations.
- Transformation matrices: matrices used to apply geometric transformations to an image.
- Translation, rotation, scaling, and shearing: basic geometric transformations that can be applied to an image using transformation matrices.
- OpenCV: a popular Python library for computer vision and image processing that provides a number of functions for geometric transformations and other image processing tasks.
- WarpPerspective and warpAffine functions: OpenCV functions used to apply perspective and affine transformations to an image.
- Edge detection: a process of finding the boundaries of objects in an image.
- Contour detection: a process of finding the contours or outlines of objects in an image.
- Morphological operations: a set of operations that process images based on their shapes.
- Image blending: a process of merging two images into a single image.
- Image thresholding: a process of converting a grayscale image into a binary image.
- Adaptive thresholding: a process of thresholding an image based on local pixel values.
- Color space conversion: a process of converting an image from one color space to another.
- Histogram equalization: a process of enhancing the contrast of an image.
- Gaussian blur: a process of blurring an image using a Gaussian filter.
- Median blur: a process of blurring an image using a median filter.
- Bilateral filtering: a process of smoothing an image while preserving edges.
- Template matching: a process of finding a template image within a larger image.

Detail description of each points are shown below:

1. Affine transformation: Affine transformation is a geometric transformation that preserves parallel lines and ratios of distances. It includes translation, rotation, scaling, and shearing of an image. It is one of the most commonly used geometric transformations in image processing and computer vision.

2. Non-affine transformation: Non-affine transformation is a geometric transformation that doesn't preserve parallel lines and ratios of distances. Some examples of non-affine transformations include perspective transformations, homography transformations, and non-rigid transformations.

3. Homogeneous coordinates: Homogeneous coordinates is a mathematical concept used to represent points in space. It is used to allow easy implementation of geometric transformations, especially affine transformations. Homogeneous coordinates are represented as a three-component vector in which the last component is always 1.

4. Transformation matrices: Transformation matrices are matrices used to apply geometric transformations to an image. They are used to represent different types of transformations such as translation, rotation, scaling, and shearing. The transformation matrix for a particular transformation is computed based on the parameters of the transformation.

5. Translation, rotation, scaling, and shearing: Translation, rotation, scaling, and shearing are basic geometric transformations that can be applied to an image using transformation matrices. Translation involves moving an image in a certain direction, rotation involves rotating an image around a certain point, scaling involves changing the size of an image, and shearing involves slanting an image in a certain direction.

6. OpenCV: OpenCV is a popular Python library for computer vision and image processing. It provides a number of functions for geometric transformations and other image processing tasks.

7. WarpPerspective and warpAffine functions: warpPerspective and warpAffine functions are OpenCV functions used to apply perspective and affine transformations to an image, respectively. These functions take the image and the transformation matrix as input and output the transformed image.

8. Edge detection: Edge detection is a process of finding the boundaries of objects in an image. It is commonly used in computer vision and image processing applications such as object detection, tracking, and recognition.

9. Contour detection: Contour detection is a process of finding the contours or outlines of objects in an image. It is often used in applications such as image segmentation and object recognition.

10. Morphological operations: Morphological operations are a set of operations that process images based on their shapes. They include erosion, dilation, opening, and closing. These operations are commonly used in image processing applications such as noise reduction, object extraction, and shape analysis.

11. Image blending: Image blending is a process of merging two images into a single image. It is often used in applications such as image compositing and panorama stitching.
12. Image thresholding: Image thresholding is a process of converting a grayscale image into a binary image. It involves setting all pixel values above a certain threshold to white and all pixel values below the threshold to black.
13. Adaptive thresholding: Adaptive thresholding is a process of thresholding an image based on local pixel values. It is often used in applications where the lighting conditions vary across the image.
14. Color space conversion: Color space conversion is a process of converting an image from one color space to another. It is often used in applications such as image processing, computer vision, and digital photography.
15. Histogram equalization: Histogram equalization is a process of enhancing the contrast of an image. It involves adjusting the pixel values in an image to spread out the intensity levels.
16. Gaussian blur: Gaussian blur is a process of blurring an image using a Gaussian filter. It is often used in applications such as noise reduction and edge detection.
17. Perspective transformations: Perspective transformations are a type of non-affine transformation that are used to correct the perspective distortion in images. OpenCV provides the getPerspectiveTransform() function to compute the transformation matrix for a perspective transformation.
18. Homography transformations: Homography transformations are a type of non-affine transformation that are used to map one image onto another image. They are often used in applications such as image stitching, panorama creation, and augmented reality. OpenCV provides the findHomography() function to compute the transformation matrix for a homography transformation.
19. Non-rigid transformations: Non-rigid transformations are a type of non-affine transformation that are used to deform an image in a non-linear way. They are often used in applications such as facial expression recognition and image morphing.
20. Interpolation methods: Interpolation methods are used to estimate the pixel values of a transformed image based on the pixel values of the original image. OpenCV provides several interpolation methods such as nearest neighbor, bilinear, bicubic, and Lanczos.
21. Image warping: Image warping is a process of transforming an image into a new coordinate system. It involves computing the transformation matrix for a given transformation and applying it to the image using an interpolation method.
22. Image registration: Image registration is a process of aligning two or more images of the same scene. It involves finding the transformation that maps one image onto the other image. This is often used in applications such as medical imaging and remote sensing.

23. Image segmentation: Image segmentation is a process of dividing an image into multiple regions or segments. It is often used in applications such as object detection, tracking, and recognition.

24. Image feature extraction: Image feature extraction is a process of extracting meaningful features from an image. This can be used for applications such as object detection, recognition, and tracking. OpenCV provides several feature extraction algorithms such as SIFT, SURF, and ORB.

25. Deep learning-based approaches: Deep learning-based approaches are becoming increasingly popular for image processing and computer vision tasks, including geometric transformations. Popular deep learning frameworks for Python include TensorFlow and PyTorch.

26. Real-time image processing: Real-time image processing is the process of processing images in real-time, typically at video frame rates. This is often used in applications such as robotics, autonomous vehicles, and video surveillance.

27. GPU acceleration: GPU acceleration can be used to speed up the processing of geometric transformations in Python. Popular libraries for GPU acceleration include TensorFlow-GPU and PyTorch.

28. Image data augmentation: Image data augmentation is a technique used to increase the size of a training dataset by generating new images from the existing images. This can be done by applying various geometric transformations such as rotation, scaling, and flipping.

29. Optimization techniques: Optimization techniques can be used to improve the performance of geometric transformations in Python. This can include optimizing the code for better performance, using parallel processing, and optimizing the parameters of the transformation for better results.

Examples:

1. Translation transformation example:

```
import cv2


image = cv2.imread('image.jpg')

rows, cols = image.shape[:2]


# Define the translation matrix

translation_matrix = np.float32([[1, 0, 100], [0, 1, 50]])


# Apply the translation transformation

translated_image = cv2.warpAffine(image, translation_matrix, (cols, rows))
```

```
# Display the original and translated image
cv2.imshow('Original Image', image)
cv2.imshow('Translated Image', translated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2. Rotation transformation example:

```
import cv2

image = cv2.imread('image.jpg')
rows, cols = image.shape[:2]

# Define the rotation angle and scale
angle = 45
scale = 1

# Define the rotation matrix
rotation_matrix = cv2.getRotationMatrix2D((cols/2, rows/2), angle, scale)

# Apply the rotation transformation
rotated_image = cv2.warpAffine(image, rotation_matrix, (cols, rows))

# Display the original and rotated image
cv2.imshow('Original Image', image)
cv2.imshow('Rotated Image', rotated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. **Scaling transformation example**:

```
import cv2
```

```
image = cv2.imread('image.jpg')


# Define the scaling factor
scaling_factor = 0.5


# Apply the scaling transformation
scaled_image = cv2.resize(image, None, fx=scaling_factor, fy=scaling_factor,
interpolation=cv2.INTER_LINEAR)


# Display the original and scaled image
cv2.imshow('Original Image', image)
cv2.imshow('Scaled Image', scaled_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

4. ***Shearing transformation example:***

import cv2

```
import numpy as np


image = cv2.imread('image.jpg')
rows, cols = image.shape[:2]


# Define the shearing matrix
shearing_matrix = np.float32([[1, 0.5, 0], [0, 1, 0]])


# Apply the shearing transformation
sheared_image = cv2.warpAffine(image, shearing_matrix, (cols, rows))


# Display the original and sheared image
cv2.imshow('Original Image', image)
cv2.imshow('Sheared Image', sheared_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

5. ***Affine transformation example:***

```
import cv2
import numpy as np

image = cv2.imread('image.jpg')
rows, cols = image.shape[:2]

# Define the affine transformation matrix
affine_matrix = np.float32([[1, 0.5, 100], [0.2, 1, 50]])

# Apply the affine transformation
affine_image = cv2.warpAffine(image, affine_matrix, (cols, rows))

# Display the original and transformed image
cv2.imshow('Original Image', image)
cv2.imshow('Affine Image', affine_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

6. ***Flip transformation example:***

```
import cv2

image = cv2.imread('image.jpg')

# Apply the horizontal flip transformation
flipped_image = cv2.flip(image, 1)

# Display the original and flipped image
cv2.imshow('Original Image', image)
cv2.imshow('Flipped Image', flipped_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

7. ***Perspective transformation example:***

```
import cv2
import numpy as np


image = cv2.imread('image.jpg')

rows, cols = image.shape[:2]


# Define the perspective transformation points

source_points = np.float32([[0, 0], [cols-1, 0], [cols-1, rows-1], [0, rows-1]])

destination_points = np.float32([[0, 0], [cols-1, 0], [int(0.6*cols), rows-1], [int(0.4*cols), rows-1]])


# Define the perspective transformation matrix

perspective_matrix = cv2.getPerspectiveTransform(source_points, destination_points)


# Apply the perspective transformation

perspective_image = cv2.warpPerspective(image, perspective_matrix, (cols, rows))


# Display the original and perspective transformed image

cv2.imshow('Original Image', image)

cv2.imshow('Perspective Transformed Image', perspective_image)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

8. **Homography transformation example:**

```
import cv2
import numpy as np

image = cv2.imread('image.jpg')
rows, cols = image.shape[:2]

# Define the homography transformation points
source_points = np.float32([[0, 0], [cols-1, 0], [cols-1, rows-1], [0, rows-1]])
destination_points = np.float32([[0, 0], [cols-1, 0], [int(0.7*cols), rows-1], [int(0.3*cols), rows-1]])

# Define the homography transformation matrix
```

```
homography_matrix, _ = cv2.findHomography(source_points, destination_points)

# Apply the homography transformation
homography_image = cv2.warpPerspective(image, homography_matrix, (cols, rows))

# Display the original and homography transformed image
cv2.imshow('Original Image', image)
cv2.imshow('Homography Transformed Image', homography_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

9. ***Image registration example:***

```
import cv2

import numpy as np


image1 = cv2.imread('image1.jpg')

image2 = cv2.imread('image2.jpg')


# Convert the images to grayscale

gray_image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)

gray_image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)


# Define the feature detector

sift = cv2.xfeatures2d.SIFT_create()


# Detect the keypoints and compute the descriptors for both images

keypoints1, descriptors1 = sift.detectAndCompute(gray_image1, None)

keypoints2, descriptors2 = sift.detectAndCompute(gray_image2, None)


# Define the matcher

matcher = cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_FLANNBASED)


# Match the descriptors of the two images

matches = matcher.match(descriptors1, descriptors2)
```

```
# Sort the matches by distance
matches = sorted(matches, key=lambda x: x.distance)

# Define the points of correspondence for image registration
source_points = np.float32([keypoints1[m.queryIdx].pt for m in matches])
destination_points = np.float32([keypoints2[m.trainIdx].pt for m in matches])

# Compute the homography transformation matrix
homography_matrix, _ = cv2.findHomography(source_points, destination_points)

# Apply the homography transformation to image1
registered_image = cv2.warpPerspective(image1, homography_matrix, (image2.shape[
```

Conclusion:

In conclusion, geometric transformations are powerful techniques that can be used to manipulate images in various ways. Using Python and OpenCV, we can easily perform a variety of geometric transformations such as translation, rotation, scaling, shearing, affine transformation, perspective transformation, homography transformation, and image registration. Each transformation has its own advantages and use cases, and by understanding them, we can create visually stunning images and achieve our desired results.

In this presentation, we have covered the basic theory behind geometric transformations, their applications, and examples of how to perform these transformations using Python and OpenCV. With these tools and techniques at our disposal, we can continue to explore and push the boundaries of what is possible in image manipulation and computer vision.