CMPE 362

DIGITAL IMAGE PROCESSING

HOMEWORK 4 REPORT

Instructor: Aslı GENÇTAV
Name - Surname: Neslihan Pelin METİN
Student Number: 71047171244

Introduction

This homework aims to teach us the steps of image stitching using Python libraries and the concepts that we have learned during the classes.

Literature Review

◊ Scale Invariant Feature Transform (SIFT): An algorithm that obtains the initial keypoints of an image and helps to match images if there are similarities.

◊ Affine Geometric Transform: This transformation preserves parallel lines and ratios of distances between points.

Problem Statement & Implementation

In this question, there are five main steps that we must follow.

1. Preparing Data: First, we get the input images from their addresses. Then, we are showing them using "plt.imshow" method.

```
# importing the needed classes to show and process the images
import cv2
from matplotlib import pyplot as plt
import numpy as np

# reading left image
left = cv2.imread('hill_left.JPG')
left_gray = cv2.cvtColor(left, cv2.COLOR_BGR2GRAY)
left = cv2.cvtColor(left, cv2.COLOR_BGR2RGB)               # converting it from BGR to RGB

# reading right image
right = cv2.imread('hill_right.JPG')
right_gray = cv2.cvtColor(right, cv2.COLOR_BGR2GRAY)
right = cv2.cvtColor(right, cv2.COLOR_BGR2RGB)              # converting it from BGR to RGB

plt.imshow(left)
plt.title("Image left")
plt.show()

plt.imshow(right)
plt.title("Image right")
plt.show()
```

2. Detection and Description of Local Features: In the next part, we are computing the keypoints with the help of methods "cv2.SIFT_create" and "sift.detectAndCompute". After that, we are using "cv2.drawKeypoints" method to show keypoints on the image.

```
# Initialize SIFT detector
sift = cv2.SIFT_create()

# Compute keypoints and descriptors for left image
keypoints_left, descriptors_left = sift.detectAndCompute(left_gray, None)

# Compute keypoints and descriptors for right image
keypoints_right, descriptors_right = sift.detectAndCompute(right_gray, None)

# Draw keypoints on the images
left_keypoints = cv2.drawKeypoints(left_gray, keypoints_left, None)
right_keypoints = cv2.drawKeypoints(right_gray, keypoints_right, None)

# Display the images with keypoints
plt.imshow(left_keypoints)
plt.title("Left Image with Keypoints")
plt.show()

plt.imshow(right_keypoints)
plt.title("Right Image with Keypoints")
plt.show()
```

3. Feature Matching: Then, we are initializing “BFMatcher” to be able to use its method that finds the closest keypoints of the images. Afterward, we are finding the good matches with applying ratio test. And we are drawing the good matches between the two images using “drawMatchesKnn” function.

```
# Initialize BFMatcher
bf = cv2.BFMatcher()

# Find matches using knnMatch
matches = bf.knnMatch(descriptors_left, descriptors_right, k=2)

# Apply ratio test and determine good matches
good_matches = []
for m, n in matches:
    if m.distance < 0.5 * n.distance:
        good_matches.append(m)

# Draw good matches
matching_result = cv2.drawMatches(left_gray, keypoints_left, right_gray, keypoints_right, good_matches, None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

# Visualize the good matches
plt.imshow(matching_result)
plt.title("Good Matches")
plt.show()
```

4. Image Registration: Next, we used “estimateAffine2D” and estimated the affine geometric transform to use this transform in aligning the images. Later, we aligned the second image with the first image using “cv2.warpAffine”. Lastly, we have calculated the transformed image’s mask.

```
# Extract matching keypoints coordinates
src_pts = np.float32([keypoints_left[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
dst_pts = np.float32([keypoints_right[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)

# Estimate affine geometric transform
M, _ = cv2.estimateAffine2D(src_pts, dst_pts)

# Transform the second image
transformed_image = cv2.warpAffine(right, M, (left.shape[1] + right.shape[1], max(left.shape[0], right.shape[0])), flags=cv2.WARP_INVERSE_MAP)

# Create a mask image
mask = np.ones_like(right_gray)

# Transform the mask image
transformed_mask = cv2.warpAffine(mask, M, (left.shape[1] + right.shape[1], max(left.shape[0], right.shape[0])), flags=cv2.WARP_INVERSE_MAP)

# Visualize the transformed second image and its mask
print(transformed_image.shape)

plt.imshow(transformed_image)
plt.axis("off")
plt.title("Transformed Image")
plt.show()

plt.imshow(transformed_mask, cmap='gray')
plt.axis("off")
plt.title("Transformed Mask")
plt.show()
```

5. Combining Registered Images: In the last step we have extended the first image to make it as the same size with the transformed image. Also, we have calculated its mask.

```
# Extend the first image to match the dimensions of the transformed second image
extended_first = np.zeros_like(transformed_image)
extended_first[:left.shape[0], :left.shape[1]] = left

# Create a mask for the extended first image
extended_mask3 = np.zeros_like(transformed_image)
extended_mask3[:left.shape[0], :left.shape[1]] = 255

# Visualize the extended first image and its corresponding mask
print(extended_first.shape)

plt.imshow(extended_first)
plt.axis("off")
plt.title("Extended First Image")
plt.show()

plt.imshow(extended_mask3)
plt.axis("off")
plt.title("Extended Mask")
plt.show()
```

After that we have implemented a method that combines left and right images into one. In this method we are creating an image consisting of zero values and in the format of right image. And we are transforming grayscale masks to binary masks. Then, using the 2D array specialty of images, we place the values of the combined image one by one in the nested for loops.

```
def combineLeftRight(LImg, LMask, RImg, RMask):
    newImg = np.zeros_like(RImg)  # Birleştirilmiş görüntüyü depolamak için yeni bir görüntü oluştur

    # Grayscale maskeleri binay maskeye dönüştür
    Lret, Lthresh = cv2.threshold(LMask, 70, 255, cv2.THRESH_BINARY)
    Rret, Rthresh = cv2.threshold(RMask, 70, 255, cv2.THRESH_BINARY)

    for x in range(len(RImg)):
        for y in range(len(RImg[0])):
            if np.any(Lthresh[x, y]) and np.any(Rthresh[x, y]):
                newImg[x, y] = np.mean([LImg[x, y], RImg[x, y]])
            elif np.any(Lthresh[x, y]) and not np.any(Rthresh[x, y]):
                newImg[x, y] = LImg[x, y]
            else:
                newImg[x, y] = RImg[x, y]

    return newImg
```

Finally, we are applying the method “combineLeftRight” and we are showing the image on the screen.

```
# Combine the extended first image and transformed second image
combined_image = combineLeftRight(extended_first, extended_mask3, transformed_image, transformed_mask)

# Visualize the combination result
plt.imshow(combined_image)
plt.axis("off")
plt.title("Combined Image")
plt.show()
```