AI6121: IMAGE STITCHING ASSIGNMENT

UWINEZA Joseph : G2303477F
Program: MSAI
Nanyang Technological University - Singapore
Academic year: 2023-2024

October 2023

1 Abstract

Image stitching is a technique in computer vision that combines multiple overlapping images to produce a single, seamless panoramic image. This process is commonly employed in various applications, from creating panoramic photos on smartphones to constructing high-resolution satellite imagery maps [1].

It's primary goal is to identify common features in overlapping images and use this information to align and merge the images seamlessly.

In this assignment we are going to experiment how pair images can be stitched together to form panoramic image. we are going to implement image stitching just for single pair images then after we will apply the same algorithm to the rest of pair images.

For doing report i divide it into three main parts:

Introduction which will discuss the main steps adopted,

Image stitching development where we will show the lines of codes and out put for making image stitching, and *Discussion* where we will implement the algorithm to the rest of the pair images.

2 Introduction

There are 5 major processes that are required while making image stitching[2]:

Image acquisition: This involves in capturing image and align it wisely in the way the features will be best matched.

Feature Detection: This identifies distinct features or keypoints in each image, such as corners or distinct patterns. Various algorithms, like SIFT (Scale-Invariant Feature Transform) and SURF (Speeded Up Robust Features), have been developed for this purpose. In our assignment we are going to use SIFT

Feature Matching: After detecting features, the next step is to match these features across different images. This provides an understanding of how two images relate spatially.

The matched features help in identifying the transformation required to align the images.

Homography Estimation: It is a $3X3$ matrix defines how one image can be warped to align with another. Algorithms like RANSAC (RANdom SAmple Consensus) are used to estimate a robust homography by minimizing the effect of outlier matches. In assignment we are going to use RANSAC.

Image Warping: Once the homography is calculated, the images can be warped or transformed so that they align correctly. This involves adjusting the perspective of one image so that it matches with the adjacent image.

3 Image Stitching Implementation

3.1 Image Acquisition

Importing the important libraries to be used.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import imageio
cv2.ocl.setUseOpenCL(False)
import warnings
warnings.filterwarnings('ignore')
```

Source Code 1: Libraries to be used

Reading a pair images to be used. we have query and train images, where the train image is the one to be transformed so as to match or combine it with query image.

```
#for query imagae
query_image = cv2.imread('./img_01_01.jpg')
query_image = cv2.cvtColor(query_image, cv2.
    COLOR_BGR2RGB)
query_image_gray = cv2.cvtColor(query_image,
     cv2.COLOR_RGB2GRAY)

#train image, image to be transformed
train_photo = cv2.imread('./img_01_02.jpg')
train_photo = cv2.cvtColor(train_photo, cv2.
    COLOR_BGR2RGB)
```

```
train_photo_gray = cv2.cvtColor(train_photo,
     cv2.COLOR_RGB2GRAY)

#view image
fig,(ax1,ax2) = plt.subplots(nrows=1, ncols
    =2, constrained_layout=False, figsize
    =[16,9])
ax1.imshow(query_image, cmap='gray')
ax1.set_xlabel('query photo', fontsize=14)

ax2.imshow(train_photo, cmap='gray')
ax2.set_xlabel('Train photo', fontsize=14)
```

Source Code 2: Image acquisition

(a) [Query image

(b) Train image

Figure 1: Graph (a) and (b) show a pair of both query and train (image that is going to be transformed) images

3.2 Feature Detection

Firstly we are going to standardize each of the feature vectors by subtract the mean and divide by the standard deviation as shown in the following code snippets.

```
# Step 1: Define the standardization
    function.
def standardize_features(features):
    """Standardize the given feature vectors
    ."""
    # Calculate mean and standard deviation
    mean = np.mean(features, axis=1,
    keepdims=True)
    std_dev = np.std(features, axis=1,
    keepdims=True) + 1e-8  # Adding a small
    value to avoid division by zero

    # Standardize features
    standardized_features = (features - mean
    ) / std_dev
    return standardized_features
```

Source Code 3: Libraries to be used

From many algorithm available as said above in introduction, we are going to use SIFT to detect the features of the images.

```
#choose descriptor
def select_descriptor_method(image, method=
    None):
    assert method is not None, "Please
    define a descriptor. 'sift','surf', 'orb
    ','brisk'"
    if method == 'sift':
        descriptor = cv2.SIFT_create()
    if method == 'surf':
        descriptor = cv2.SURF_create()

    if method == 'brisk':
        descriptor = cv2.BRISK_create()

    if method == 'orb':
        descriptor = cv2.ORB_create()

    (keypoints, features) = descriptor.
    detectAndCompute(image, None)
    return (keypoints, features)
feature_extraction_algo = 'sift'
feature_to_match = 'bf'
```

Source Code 4: Choosing Descriptor

The following code snippets is for extracting and standarize the features for both Train and Query images.

```
# Step 2: After extracting features,
    standardize them.

# Feature extraction and standardization for
     train image
keypoints_train_img, feature_train_img =
    select_descriptor_method(
    train_photo_gray, method=
    feature_extraction_algo)
feature_train_img = standardize_features(
    feature_train_img)

# Feature extraction and standardization for
     query image
keypoints_query_img, feature_query_img =
    select_descriptor_method(
    query_image_gray, method=
    feature_extraction_algo)
feature_query_img = standardize_features(
    feature_query_img)

print("keypoints_query_image:",
    keypoints_query_img)
```

Source Code 5: Key points

The figure below shows the extracting features.

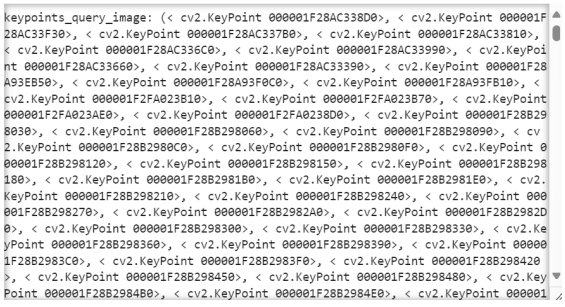

Figure 2: Keypoints for image pair 1

```
print("feature_query_img", feature_query_img
    )
```

Source Code 6: Key features

The figure below shows the extracting keypoints. The following is the code snippets for visualizing the keypoints

```
feature_query_img [[ 0.25219133 -0.45504946 -0.42979085 ... -0.55608386 -0.55608386
  -0.55608386]
 [-0.41779575 -0.5196583  -0.5196583  ... -0.57058966 -0.57058966
  -0.57058966]
 [-0.5553853  -0.5553853  -0.52986866 ... -0.5809019  -0.5809019
  -0.5809019 ]
 ...
 [-0.5944795  -0.5944795  -0.25159773 ... -0.64723057 -0.64723057
  -0.62085503]
 [-0.5715565  -0.5715565  -0.49511006 ... -0.5205922  -0.41866365
  -0.5715565 ]
 [ 1.5765176  -0.74520725 -0.7172347  ...  1.6324629   2.2198873
   0.34572372]]
```

Figure 3: Keypoints for image pair 1

```
#draw keypoints of two images

fig,(ax1, ax2) = plt.subplots(nrows=1, ncols
    =2, figsize = [20,8], constrained_layout
    =False)
ax1.imshow(cv2.drawKeypoints(
    train_photo_gray,keypoints_train_img,
    None,color=(0,255,0)))
ax1.set_xlabel('(a)', fontsize=14)
ax2.imshow(cv2.drawKeypoints(
    query_image_gray,keypoints_query_img,
    None,color=(0,255,0)))
ax2.set_xlabel('(b)', fontsize=14)
#plt.savefig('./Output/' +
    feature_extraction_algo +"feature_img
    "+'.png',bbox_inches='tight', dpi=300,
    format='png')
```

Source Code 7: Drawing key points

The following pair images shows the keypoints found for each to be matched later on.

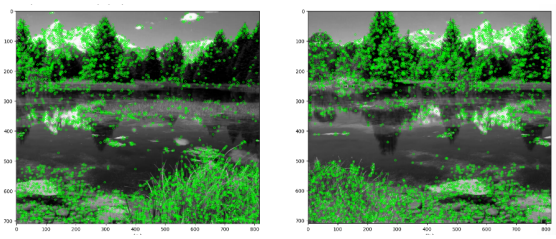

Figure 4: Draw of keypoints

3.3 Feature Matching

After detecting features in both query and train images, the next step is to find matches between these features across different images.

This gives us an understanding of how two images relate spatially. From many method available as said above in introduction we are going to use Brute-Force matcher

Below is the code snippets show how we have used Brute-Force matcher to match the keypoints.

```
def create_matching_object(method,
    crossCheck):
    if method == 'sift' or method == 'surf':
        bf = cv2.BFMatcher(cv2.NORM_L2,
    crossCheck=crossCheck)

    if method == 'orb' or method == 'brisk':
        bf = cv2.BFMatcher(cv2.NORM_HAMMING,
     crossCheck=crossCheck)
    return bf

def key_points_matching(feature_tain_img,
    feature_query_img, method):
    bf = create_matching_object(method,
    crossCheck=True)
    best_matches = bf.match(
    feature_train_img, feature_query_img)
    raw_matches =sorted(best_matches, key=
    lambda x: x.distance)
    print("Raw matches with Brute Force",
    len(raw_matches))

    return raw_matches

#drawing the matched features
feature_to_match = 'bf'
print('Drawing matched features for',
    feature_to_match)
fig = plt.figure(figsize=[20,8])

if feature_to_match == 'bf':
    matches = key_points_matching(
    feature_train_img, feature_query_img,
    method=feature_extraction_algo)
    mapped_feature_image = cv2.drawMatches(
    train_photo, keypoints_train_img,
    query_image, keypoints_query_img,
    matches[:100], None, flags=cv2.
    DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
    plt.imshow(mapped_feature_image)
```

Source Code 8: codes for drawing features that matched

The figure below shows clearly how the image keypoints have been matched.

Figure 5: Matching pair images features

3.4 Homography Estimation

Once feature matches are found, a transformation matrix (called the Homography) that maps points in one image to another can be calculated. This is usually done using methods like RANSAC (RANdom SAmple Consensus) to ensure a robust estimation by ignoring outlier matches.

Below is the code snippets that shows how we adopted the Homography matrix.

```
def homography_stitching(keypoints_train_img
    , keypoints_query_img, matches,
    reprojThresh):
    #convert to numpy array
```

```
    keypoints_train_img = np.float32([
    keypoint.pt for keypoint in
    keypoints_train_img])
    keypoints_query_img = np.float32([
    keypoint.pt for keypoint in
    keypoints_query_img])

    if len(matches) > 4:
        points_train = np.float32([
    keypoints_train_img[m.queryIdx] for m in
     matches])
        points_query = np.float32([
    keypoints_query_img[m.trainIdx] for m in
     matches])

        (H, status) = cv2.findHomography(
    points_train, points_query, cv2.RANSAC,
    reprojThresh)
        return (matches, H, status)
    else:
        return None

M = homography_stitching(keypoints_train_img
    , keypoints_query_img, matches,
    reprojThresh=4)

if M is not None:
    (matches, Homography_Matrix, status) = M
    print(Homography_Matrix)
else:
    print('Error in calculating Homography
    Matrix')
```

Source Code 9: Homography matrix

3.5 Image Warping

Once the homography is calculated, the images can be warped or transformed so that they align correctly.

Image warping, also known as geometric distortion, is a transformation that maps every point in one image to a point in a destination image. In simpler terms, it's the process of changing the geometry of an image to transform its perspective or shape

Below is the code snippets that shows how to warps image train image to query image.

```
result = cv2.warpPerspective(train_photo,
    Homography_Matrix,(width, height))
#print(result)
result[0:query_image.shape[0], 0:query_image
    .shape[1]] = query_image

plt.figure(figsize=(20,10))
plt.axis('off')
plt.imshow(result)
```

Figure below shows the resulted warped image that can be considered as panoramic image.

4 DISCUSSION

After working through on how we can stitch two image together by using just one pair images, Now we are going to discuss on all of rest image pair given following the workflow to implement our stitched images.

Figure 6: Stitched images

4.1 Image acquisition

In image acquisition, it require intuitive ways to load the image in logical and efficient way in sense of maximizing the matches of detected features in shorter distance as much as possible.

Below are the Image Acquisition for pair 2, pair 3 and pair4.

(a) Query image

(b) Train image

Figure 7: Graph (a), and (b) show both query image and train image (image to be transformed) for PAIR 2

(a) Query image

(b) Train image

Figure 8: Graph (a), and (b) show both query image and train image (image to be transformed) for PAIR 3

4.2 Feature detection

As shown below, we are going to show the features that have been detected to each pair images, the green dots shows the features.

(a) Query image (b) Train image

Figure 9: Graph (a), and (b) show both query image and train image (image to be transformed) for PAIR 3. Specifically to this image we have rotated 90° degree clockwise the left image and 90° degree anticlockwise the right image in sense of getting the better results.

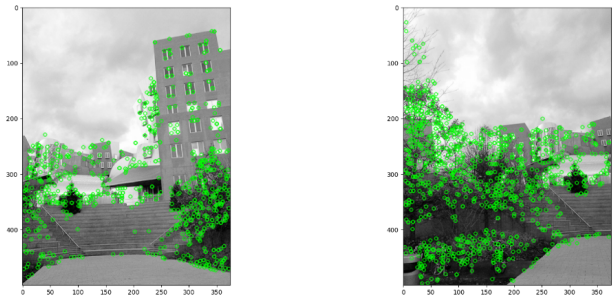

Figure 10: keypoints detected for PAIR 2: it is obvious that the green small circle indicated the important features present in the images.

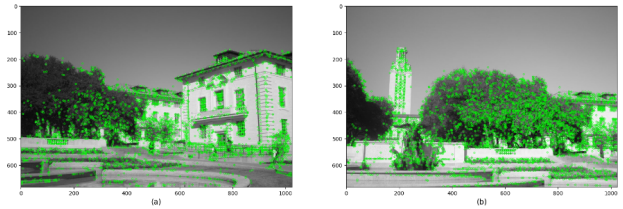

Figure 11: keypoints detected for PAIR 3: This figures shows the important features that have been identified that will be matched for making panoramic images

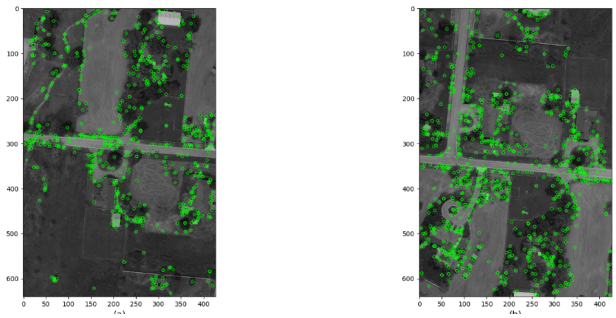

Figure 12: keypoints detected for PAIR 4: the figure shows the features detected after rotating the image in wise way to maximize the values of best features match

As an overview of features detection, it is initial step that identifies distinct features or keypoints in each image.

These features, such as corners or distinct patterns, are points that the algorithm can reliably recognize in different images.

The figures 10, 11,12 shows the detected features.

4.3 Feature matching

After detecting features, the next step is to match these features across different images. This provides an understanding of how two images relate spatially. The matched features help in identifying the transformation required to align the images.

Below are the images of the matched features for each pair images.

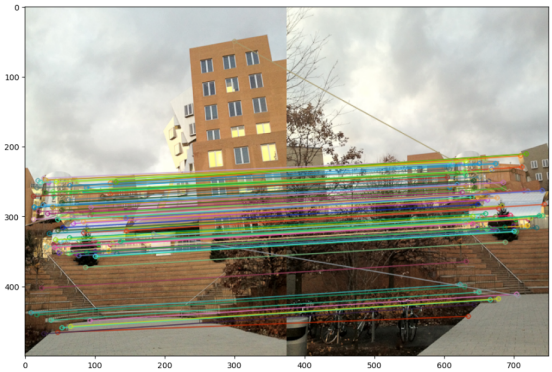

Figure 13: Matched features for PAIR 2: This figure clearly shows how the features have been matched accordingly with the standardized vector.

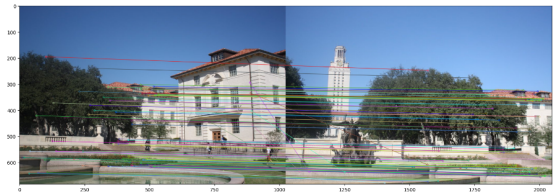

Figure 14: Matched features for PAIR 3: it shows the feature matched across two images. Most of the time the clear image with distinct features shows, the parallel straight line of matching the features

The figures 13, 14,15 shows the matched features across the images pair.

4.4 Warping Images

The following are results of the matched features and warped to form one panoramic image.

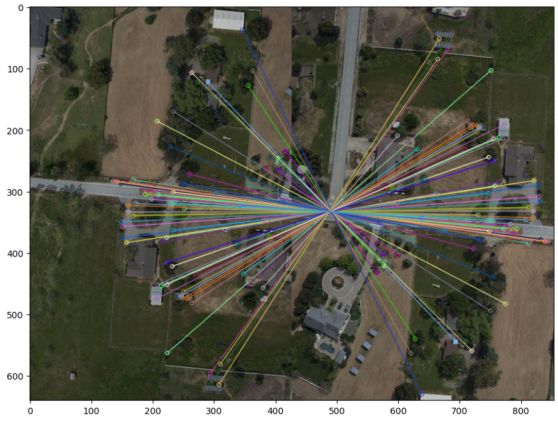

Figure 15: Matched features for PAIR 4: This unusual appearance of features match which is caused by natural of images where they have been taken by sliding camera position in the shortest distance.

Figure 16: Stitched Images for PAIR 2: this is the resulted panoramic image after applying homograph estimation to the pair image.

Figure 17: Stitched Images for PAIR 3: As you see the image view has been extended after matching the two image pair.

5 Future Work

1. In this work we have used only two images, may be the stitching together three image, or several image might be useful.

2. In the stitched image it appear that the images are combined together, they might be a way to minimize the different luminosity in the image that it may seem to be one image.

3. during image acquisition, we make sure the image is properly arranged, in the future it will be more efficient if this tasks is done by the computer by arranging different image after detecting its features and estimate its best matches.

Figure 18: Stitched Images for PAIR 4

6 Tools

To accomplish this task we have used the following tools:

1. We use this format `https://www.overleaf.com/project` to structure and present our work.
2. We used `https://chat.openai.com/` to structure and understand codes
3. We used `https://quillbot.com/` to check any grammatical error

References

[1] OpenAI. Chatgpt-4, 2022. Accessed: 2023-October-21.

[2] Adrian Rosebrock. Opencv panorama stitching, 2016. Accessed: 2023-10-24.

-end-