Assignment-1

Image Stitching

K S Ashin Shanly
20250019
M. Tech CSE

Procedure:

In simple terms, image stitching for a group of images outputs a composite image which is a culmination of all the individual scenes, in such a way that the logical flow between the input images is preserved.

An **assumption** is made that the given images are in the left to right order of orientation. Also here, the best 4 images from each of the image folder in the dataset are chosen for stitching.

An abstract procedure for the task is:

1. Load input images to list_of_ images[] array.
2. Assuming, that the center image is no_of_images / 2
   let centerIndex = length(images)/2
3. for each image at positions 0 to centerIndex :
   Do leftward stitching
4. for each image at positions centerIndex to last image in the array:
   Do rightward stitching

To perform both the leftward or rightward stitching between any 2 images, we need to calculate the **homography matrix** first. Note that here dst_image(destination) is kept as it is, and the src_image(source image) is the one that is chosen for warping.

To calculate the points needed for calculating the homography, we first convert both the src_image and dst_image into greyscale and compute the key points and features of both the images using **ORB**. Once we have the descriptors and key points of both images, we match the correspondences between them using BFMatcher and 2-Nearest Neighbor(here). Any four random correspondences are passed into the function that calculates homography. The RANSAC function will calculate a homography by choosing four random correspondences and keep the homography if it is better than any homography yet found. There is also a threshold parameter to the function that sets a minimum percentage of image points that the current best homography estimation must account for before RANSAC can stop.
Once we have estimated a homography, we **warp** the src_image according to it. After obtaining a warped image, **blending** is done to remove any differences in the intensities or exposure of both the images and thereby obtain a smooth transient when stitching. Next, we add the warped source image along with the destination image and **crop** out the black regions from the composite image. Repeat this over through leftward stitching and rightward stitching, which will eventually yield to us the panorama image.

Visual Results

Folder-1

- Using homography **implementation**:

- Using **inbuilt** homography estimation:

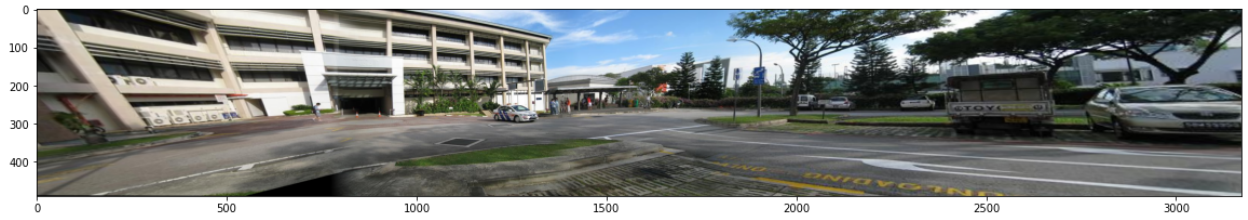
Folder-2

- Using homography **implementation**:

```
images = ['2_2.JPG','2_3.JPG','2_4.JPG','2_5.JPG']
list_of_images = []

for i in images:
  list_of_images.append(cv2.imread(i))

panorama=stitchMultiple(list_of_images, opt='my')
plt.figure(figsize=(20,20))
panorama = np.array(panorama,dtype=float)/float(255)
panorama = panorama[:,:,::-1]
plt.imshow(panorama)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers
<matplotlib.image.AxesImage at 0x7f9f25045290>
```

Image:



- Using **inbuilt** homography estimation:

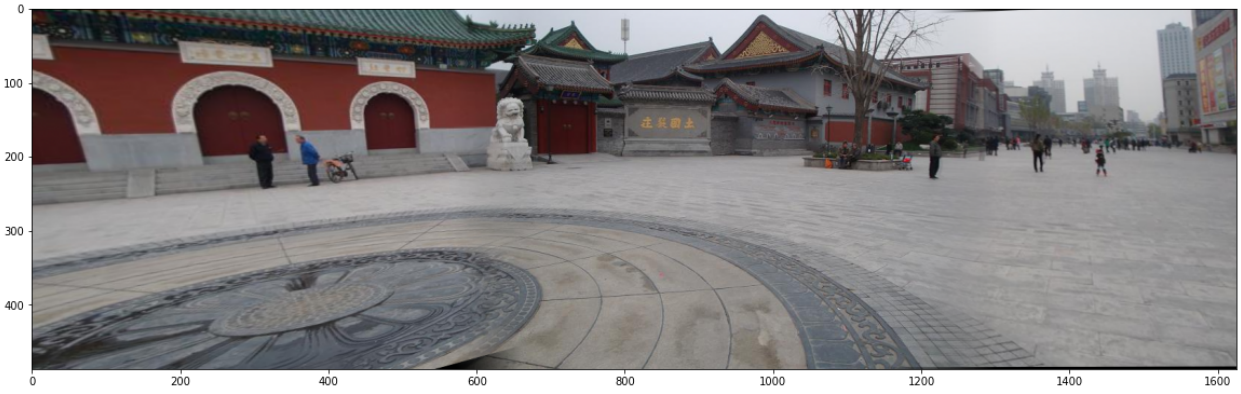
```
images = ['2_2.JPG','2_3.JPG','2_4.JPG','2_5.JPG']
list_of_images = []

for i in images:
  list_of_images.append(cv2.imread(i))

panorama=stitchMultiple(list_of_images, opt='inbuilt')
plt.figure(figsize=(20,20))
panorama = np.array(panorama,dtype=float)/float(255)
panorama = panorama[:,:,::-1]
plt.imshow(panorama)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers
<matplotlib.image.AxesImage at 0x7fdf9f51c110>
```

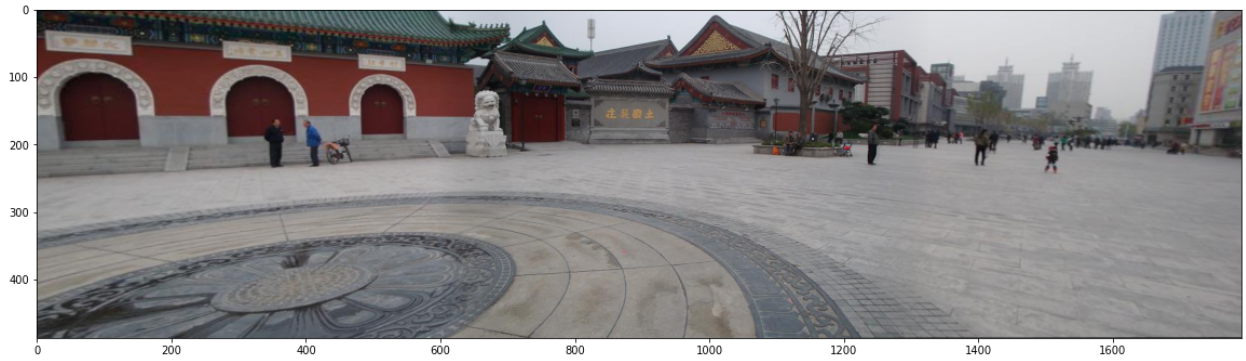
Image:

Folder-3

- Using homography **implementation**:

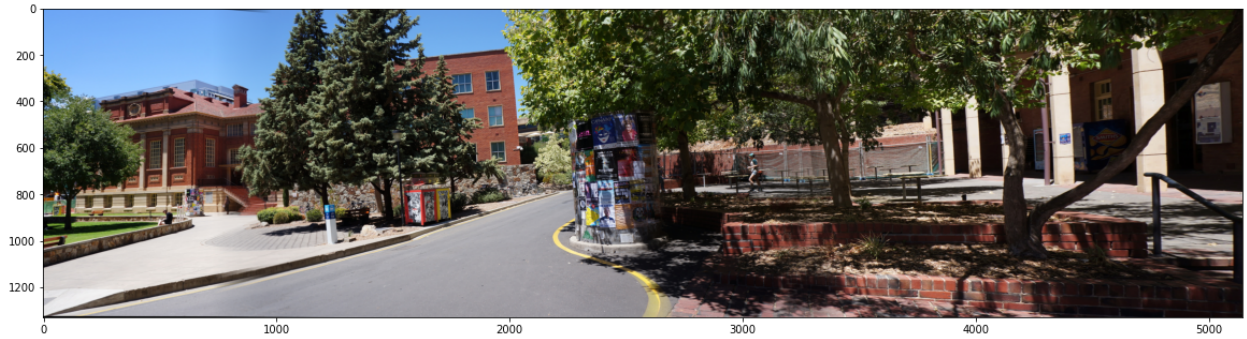


- Using **inbuilt** homography estimation:



Folder-4

- Using homography **implementation**:



- Using **inbuilt** homography estimation:

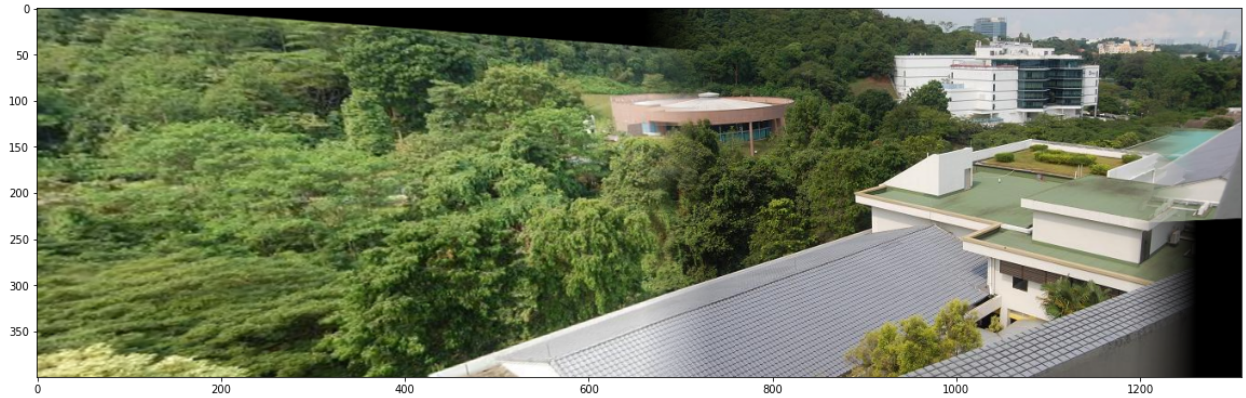
Folder-5

- Using homography **implementation**:

- Using **inbuilt** homography estimation:

Folder-6

- Using homography **implementation**:



- Using **inbuilt** homography estimation: