Introduction to Computer Vision

Final Project Report: Image Stitching

0. Acknowledgement

1. Personal views about computer vision

1.1 Views on CV

Computer vision can be seen as an interdisciplinary field which requires a lot of different background knowledge. Generally speaking, the task of computer vision is to let the computer do something that the human visual system can do. But this is not easy because what the computer can “see” is actually the binary code, instead of the image.

In the past ten years, I think the most important progress on computer vision is the combination between CV and deep learning. The appearance of deep learning can be seen as a revolution of computer vision field. We can see that some classic high level vision problems can be easily solved using deep network. For example, the face related problems, object classification and so on. We may find that some research problems can be transform to a engineering problem which can be solved by training neural networks. However, as the saying goes, every coin has two sides. Although deep learning benefits computer vision problems a lot, some people may think that since we have deep learning methods, should we continue doing research on computer vision methods and algorithms? I think the answer is YES. I don’t believe deep network can fully solve all the CV related problems. It’s because deep learning is focusing on the process of the data, instead of the inner logic of the image and vision.

I think we should not only apply deep learning methods into computer vision problems, but also pay more attention on the CV algorithm itself.

1.2 What I learnt in 332

As an introduction course in computer vision, EECS 332 really helps me build the foundations of CV. From basic image information, image processing to detection, feature extraction, we have learnt the basic low level vision problems in this course. At the same time, we also have some lectures about higher level vision including motion, stereo, tracking and so on, which inspire us for further learning in computer vision. 332 is an interesting and inspiring course because we can not only learn the theory but also implement some demo by coding. The assignments are also a good guide for us to think deeply into what we learnt in the class. It feels good when we turn something we learn into a real demo. In this course, I can also combine what I have learnt in my undergraduate(linear algebra, DSP, coding) and apply them in a new field.

1.3 What I want to learn more

3D related issue (e.g 3D reconstruction) is one of the CV topics that I'm highly interested in. The reason is that it has a various application in real life. For example, in medical fields, 3D reconstruction technique can be used for a 3D model of a patient's organ using CT scanning image, which can be helpful to the diagnosis and treatment process. Besides, 3D reconstruction technique can also be applied to entertainment purposes and make us a better life. We have learnt about the basic ideas about stereo in EECS 332. Although it's not a core topic in 332, I think it's a

challenging and hard problems in computer vision fields. So I hope if I can have further study in computer vision, I can learn more about 3D reconstruction topics.

2. Project Description

2.1 Problem statement

Image stitching is a technique to create a high resolution panorama from a set of images with overlapped fields, which is widely used for producing today's digital maps and satellite photos. Algorithms among aligning images and stitching them into seamless panorama are among the oldest and most widely used in computer vision. Image stitching first originated in photogrammetry fields. In about 1980, manually intensive methods based on surveyed ground control points or manually registeded tie points have long been used to register aerial photos into large scale photo mosaics. One of an important progress in image stitching is the development of bundle adjustment algorithm, which can solve for the locations of all of the camera positions simultaneously. Another recurring problem in stitching is the elimination of visible seams. In 1975, David L.Milgram came up with an idea that allows overlapping images to be combined into a photomosaic in which visual impact of the introduced seam has been minimized. Later, there were many other researches on this topic, including two different basic ideas. One is worked by directly minimizing pixel to pixel dissimilarities and gradient descent methods. Another one is implemented by extract a set of features and match them to each other, which is also a modern way to deal with the problems. Such features based methods have advantage of being more robust against scene movement and also faster. More importantly, it can automatically

discover the overlap relationship among a set of unordered images. The pipeline for today's image stitching are similar: The process of image stitching can be divided into two parts: image registration and image blending. Registration is the process of transforming different image into one coordinate system. Generally, registration consists of the features extraction, features matching and image warping. Then we will smooth the seam through image blending. However, we have different methods and algorithms to implement each step of image stitching.

2.2 Goal

Our goal of this final project is to implement a featured based automatic image stitching demo. When we input two images with overlapped fields, we expect to obtain a wide seamless panorama.

In this final project, we use scale invariant features transform(SIFT) to extract local features of the input images. Then we use K nearest neighbors algorithms to match these features. We will use Random sample consensus(Ransac) to calculate the homograph matrix, which will be used for image warping. Finally we use a weighted matrix as a mask for image blending.

3. Design Description

3.1 Tools

As Python is faster in matrix calculation than Matlab, we use Python 2.7 as our program language in this final project. We also use numpy library for calculation and

OpenCV for image processing.

3.2 Project Structure

We use Python class as our project structure.

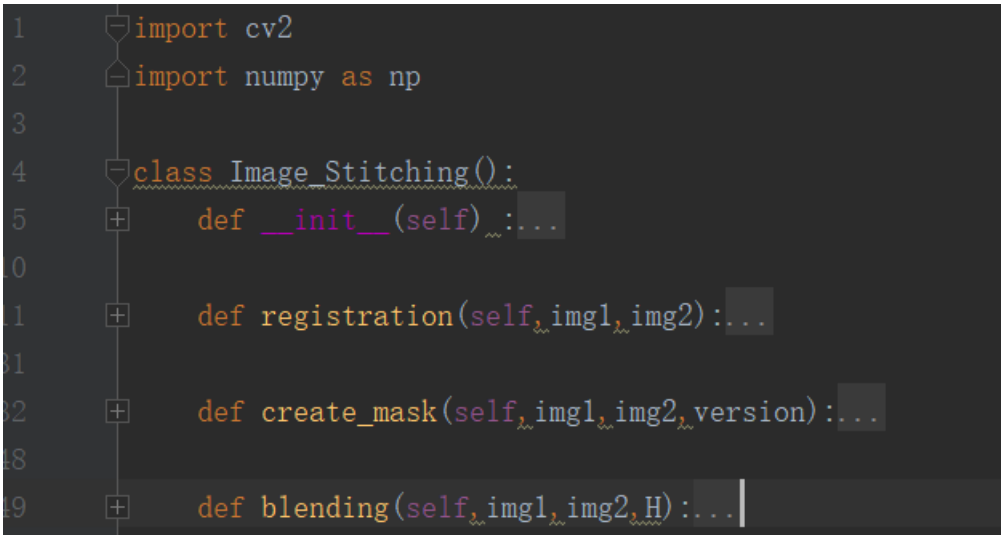

Fig.1. Project Structure

Python class is an object that provide a means of bundling data and function together. By using Python class, we can put all the parameters and functions in one cell. It's also very convenient to call the functions and extend them. In __init__, we initialize some parameters we'll use in the functions. In registration function, we implement feature extraction and feature matching. Then we calculate homography matrix and return it for warp image warping. create_mask function returns a weighted matrix for image blending. In the blending function, we input image 1 and 2 and the homography. Finally it returns the final result image matrix and we can write it as the local file.

4. Algorithms Description

4.1 Image registration

Traditionally, before we can register and align images we must first determine the

appropriate motion model relating pixel coordinates in one image to pixel coordinates in another. A variety of such parametric motion models include translation, affine transform, projective and so on. Then we should find model parameters by Least Square Fitting and use the model for image warping.

In the more recent stitching algorithms, we first extract features and then match them up, often using robust technique like Ransac to compute a good set of inliers. And we finally computer the homography matrix. Here I'll show the steps for featured based image registration. I will use Fig.2 and Fig.3 as our test images.

Fig.2. test image 1

Fig.3. test image 2

1) Features extraction

There are many different methods for features detection: Harris Corner detector, scale invariant features transform(SIFT), speeded up robust features(SURF), Oriented fast and rotated BRIEF(ORB) and so on. SURF is partly inspired by SIFT descriptor. The standard SURF is several times faster than SIFT. ORB is based on the FAST key point detector and the visual descriptor Binary Robust Independent Elementary(BRIEF).

In this project, we'll use the classic SIFT to implement features extraction. There are four core steps for SIFT algorithms:

- **Scale-space extrema detection**

SIFT uses difference of Gaussians(DOG) as an approximation of Laplacian of Gaussian(LOG). DOG is calculated as the difference of Gaussian blurring of an image with two different σ. This process is done for different octaves of the image in Gaussian Pyramid. (Fig.4)

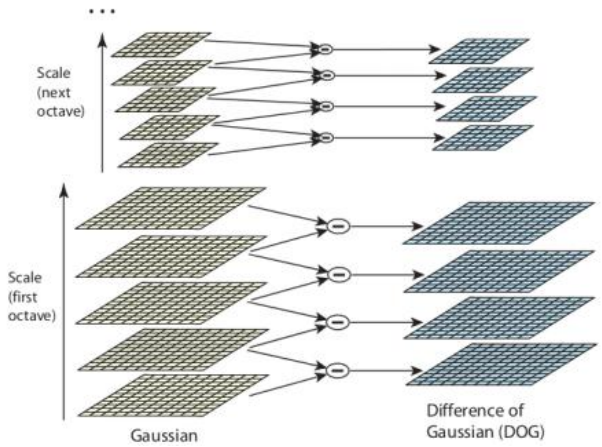

Fig.4. Gaussian Pyramid

This pyramid is so called scale space. Once this DoG is find, images are searched

for local extrema over scale and space. For example, in Fig.5, one pixel in an image is compared with its 8 neighbors as well as 9pixels in next scale and 9pixels in previous scales. If this pixel is a local extrema, it can be regarded as a potential key point, which means that this point is best represented in this scale.

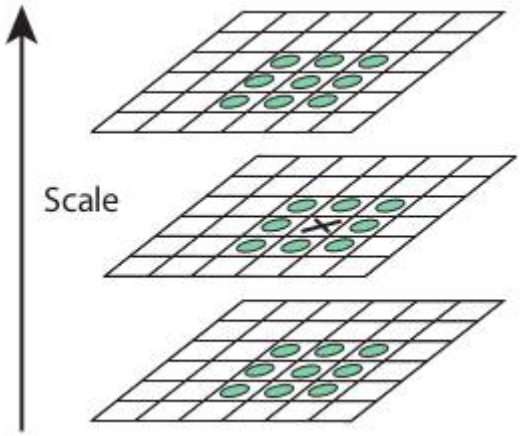

Fig.5. local extrema

- **Keypoint localization**

Scale-space extrema detection produces a set of potential key points. But some of them are unstable. We should apply some constraints to filter some points. First, we'll use Taylor series expansion of acale space to get more accurate location of extrema. If the intensity at this extrema is less than a threshold value, it's rejected. Second, we will also eliminate low-contrast keypoints. At the same time, we will eliminate some points with edge responses using Hessian matrix.

- **Orientation assignment**

In this step, each keypoint will be assigned an orientation based on the local image gradient directions. It's quite similar to what we have done in Canny edge detection. The purpose of this step is to achieve rotation invariance. A neighborhood

is taken around the keypoint, and the gradient magnitude and direction is calculated. An orientation histogram with 36 bins covering 360 degress is created. Among this, the highest peak in the histogram is taken. Any peak above 80% is alo considered to calculate the orientation.

- **Keypoint descriptor**

A 16x16 neighborhood around the keypoint is taken. It's divided into 16 sub-blocks of 4x4 sizes. For each sub-blocks, 8 bin orientation histogram is created. So we have a total 128 bin values. It's represented as a vector to form keypint descriptor, which means that the descriptor for each point is 128 dimensions.

Here is the original images with detected keypoints:

Fig.6. keypoints in image1

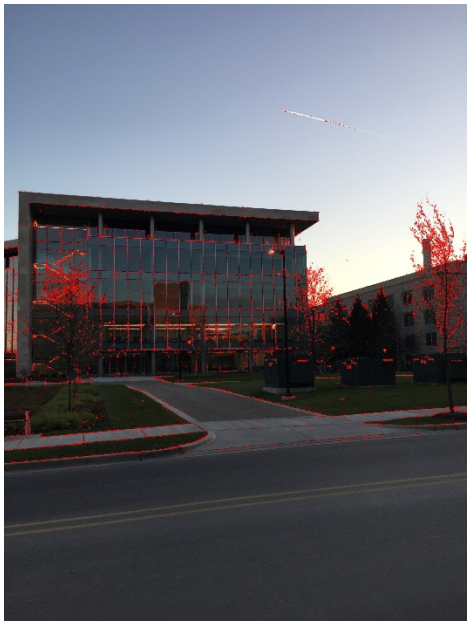

Fig.7. keypoint in image 2

2) Features matching

As we obtain the descriptors for images, we can pair them by some matching methods. Keypoints between two images are matched by identifying their nearest neighbors. In this project, we use K nearest neighbors(KNN) to implement matching. Here we set K=2. By dong so, we'll find two matches in second image for one keypoint in the first image. In some cases, the distance of the second best match may be very close to the distance of the best matches. This may happen due to noise. So we'll apply a ratio test here as Fig.8 shows. The ratio of closest distance to second closest distance is calculated. If the ratio is greater than a threshold ratio we set, this match should be rejected. In this project we set the ratio as 0.85. It means that if it's a good match, the closest match distance should be "smaller enough" than the second closest match distance.

```
for m1, m2 in raw_matches:
    if m1.distance < self.ratio * m2.distance:
        good_points.append((m1.trainIdx, m1.queryIdx))
```

Fig.8. ratio test code

Here is the matching image:

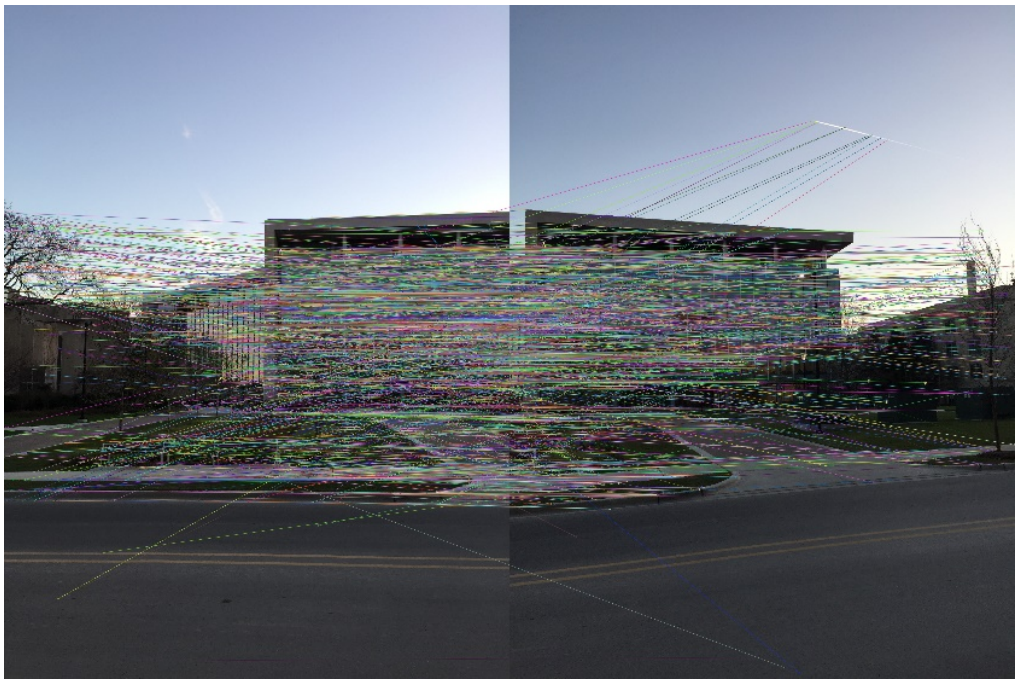

Fig.9. matching image

3) Calculate homograhy using Ransac

As we can in Fig.9, some features pairs are correct, while some other pairs are not the keypoints located in the overlapped region of two images. We shouldn't take the incorrect pairs into homography calculation as the final result will be affected by error. In order to get rid of such incorrect pairs, random sample consensus is applied here. The incorrect matching pairs are called the "outliers" and the correct matching pairs are called "inliers" in Ransac algorithms. Ransac is an iterative method to fit a mathematical model from a set of data that contains outliers. Here we use Ransac to select a subset of inliers matching pairs and discard the outliers. Inliers are pairs that coherent with the homography for overlapping, while outliers don't fit into it. Here we use OpenCV's API cv2.findHomography() to calculate the 3x3 homography matrix.

4.2 Image blending

Once we obtained the homography matrix, it can be applied to the source image to implement warping. One possible way is to simply use cv2.warpPerspective function in OpenCV. But if we use this way, we can't obtain a perfect and seamless panorama image.(as Fig.10 shows)

Fig.10 a rough panorama result

We can see that, in Fig.10, there are obvious distortions in the overlapped fields of the images. The color and light are also uneven. It's because the fact that exposure tiem of the shutters and white balance control of the cameras would change when we capture the same scene by different perspectives. So it's necessary to perform image blending.

As we mentioned before, in the create_mask function, we create a weighted

matrix as a mask to smooth the discontinuities between the two overlapped pictures. Actually, this mask is a matrix with the same size as the output panorama image. It has two versions for this mask. As we want to deal with the seam in the overlapped region, we should keep the region in the left side and the right side of the seam invariant and also modify the overlapped part. So our mask matrix has a so called smoothing window. Meanwhile, there are two versions for this mask as Fig.11 shows.

```
if version== 'left_image':
    mask[:, barrier - offset:barrier + offset] = np.tile(np.linspace(1, 0, 2 * offset).T, (height_panorama, 1))
    mask[:, :barrier - offset] = 1
else:
    mask[:, barrier - offset:barrier + offset] = np.tile(np.linspace(0, 1, 2 * offset).T, (height_panorama, 1))
    mask[:, barrier + offset:] = 1
```

Fig.11. mask creating code

If the version is for the image in the left part of the panorama, the smoothing window is a sub matrix as the Fig.12 shows. In the left part of this mask matrix, there are all ones. In the right part of this mask matrix, there are all zeros.

```
[[ 1.    0.98  0.96 ...,  0.04  0.02  0.  ]
 [ 1.    0.98  0.96 ...,  0.04  0.02  0.  ]
 [ 1.    0.98  0.96 ...,  0.04  0.02  0.  ]
 ...,
 [ 1.    0.98  0.96 ...,  0.04  0.02  0.  ]
 [ 1.    0.98  0.96 ...,  0.04  0.02  0.  ]
 [ 1.    0.98  0.96 ...,  0.04  0.02  0.  ]]
```

Fig.12. smoothing window of left version

If the version is for the image in the right part of the panorama, the smoothing window is a sub matrix as the Fig.13 shows. In the left part of this mask matrix, there are all zeros. In the right part of this mask matrix, there are all ones.

```
[[ 0.    0.02  0.04 ...,  0.96  0.98  1.  ]
 [ 0.    0.02  0.04 ...,  0.96  0.98  1.  ]
 [ 0.    0.02  0.04 ...,  0.96  0.98  1.  ]
 ...,
 [ 0.    0.02  0.04 ...,  0.96  0.98  1.  ]
 [ 0.    0.02  0.04 ...,  0.96  0.98  1.  ]
 [ 0.    0.02  0.04 ...,  0.96  0.98  1.  ]]
```

Fig.13. smoothing window of right version

The final output panorama is formed as the code in Fig.14 shows:

```
panorama1 = np.zeros((height_panorama, width_panorama, 3))
mask1 = self.create_mask(img1,img2,version='left_image')
panorama1[0:img1.shape[0], 0:img1.shape[1], :] = img1
panorama1 *= mask1
mask2 = self.create_mask(img1,img2,version='right_image')
panorama2 = cv2.warpPerspective(img2, H, (width_panorama, height_panorama))*mask2
result=panorama1+panorama2
```

Fig.14. code for creating panorama

First, we multiply image 1 and the left version mask element by element. Then we use homography to warp image 2 and multiply it with right version mask element by element. Then we concatenate them together to obtain the final panorama. Of course we need to crop it to obtain a good frame of the image.

Fig.15. final result

5. Result Analysis

We use three sets of source images to test our stitching demo.

Fig.16. stitching example 1

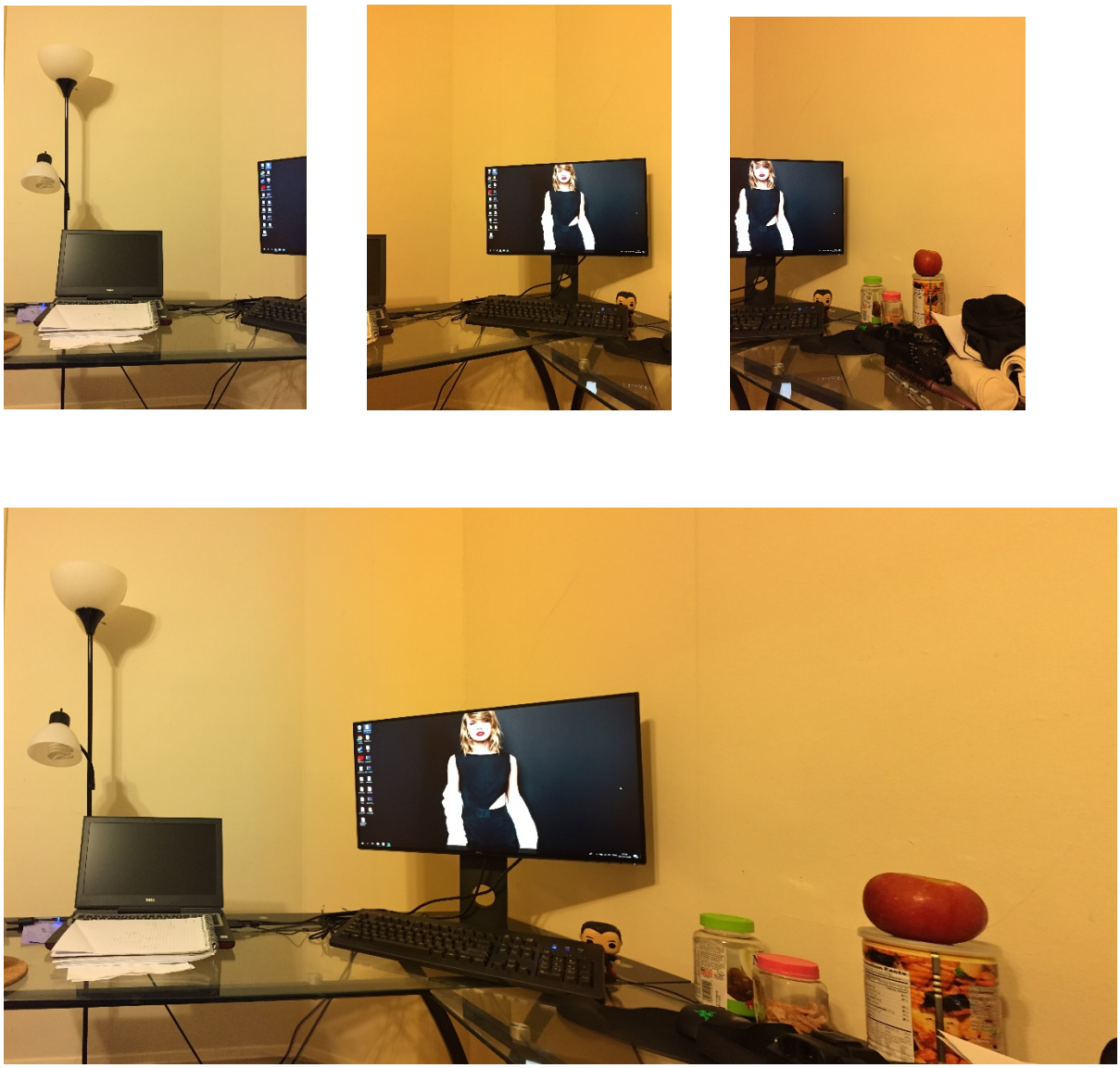

Fig.17. stitching example 2

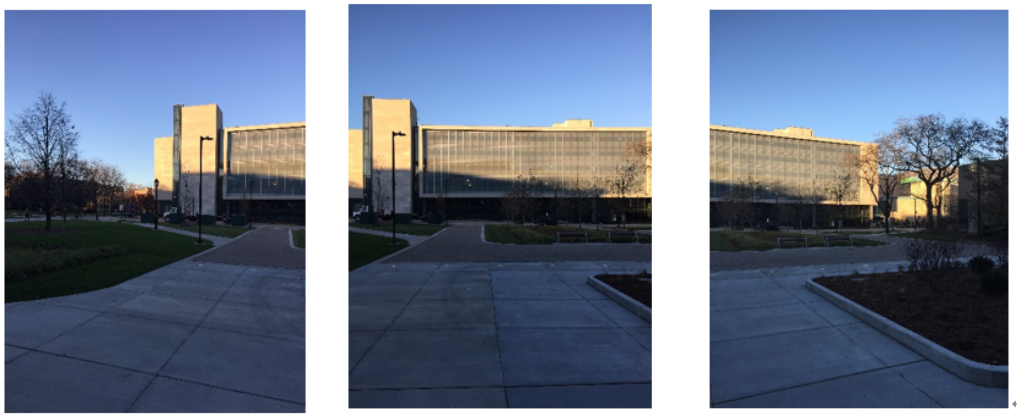

Fig.18. stitching example 3

Analysis: We can see that in the two input situation, the algorithm works well to produce a seamless wide panorama. But when there are three images for stitching, the result is not perfect as there are obvious deformation in the right part of the panorama. I think the problem is due to the fact that the blending algorithm is not robust enough.

6. Remarks and future work

Looking back to this final project, we successfully implement an imperfect demo for image stitching algorithm. We use the modern image stitching pipeline as the process of the whole project and apply some smart methods to implement it. I think in the future there are two things we need to improve:

First, we can try to use less OpenCV API to implement the specific steps of stitching algorithm. It's because if we code in pure python, we can have more controls in every details of the process. By doing so, we can easily tune the code or parameters to obtain a better result.

Second, we should improve the blending algorithm. As we can show in the result analysis part, the algorithm is not robust enough in stitching 3 or more images.