PANORAMA STITCHING

[1] *Wei-Jie Chung* (鍾偉傑)   [1] *Chiou-Shann Fuh* (傅楸善)

[1] Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan.
E-mail: b04902082@ntu.edu.tw   fuh@csie.ntu.edu.tw

ABSTRACT

With the advance of technology, cellphone gradually plays an important role in our lives, especially the convenience and the practicality of the camera. The problem of how to improve the performance of the camera has become an interesting issue. For example, human's full field of view is about $200^{\circ}*135^{\circ}$, while the full field of view of the cellphone is about $50^{\circ}*35^{\circ}$, which would not fulfill the need of user sometimes. Consequently, the idea of panorama stitching shows up, and we aim to implement panorama stitching.

Keywords: *Image stitching.*

1. INTRODUCTION

Image stitching is the process of combining multiple photographic images with overlapping fields of view to produce a segmented panorama image. Nowadays, image stitching is widely used in several commercial applications such as the camera system of iPhone and Samsung Mobile. Consequently, it will be profitable to implement panorama and to improve it.
In the following part, we will present the methods and the expected results of panorama stitching.

1.1. Overview

In the next section, we present the method we use to detect and match the feature points in detail, features with rotation, translation, and illuminance invariant would enable us to match multiple images correctly. In Section 3, we show how to get the transform matrix and process the panorama image to get a good-looking panorama image. In Section 4, we show the final results of our work. Finally, we discuss about the conclusion and our future work.

2. FEATURE DETECTION AND MATCHING

We use the feature detection algorithm Scale-Invariant Feature Transform (SIFT) to accomplish the feature detection. SIFT is a feature detection algorithm used in computer vision to detect and describe the local features in images, which could be used to accomplish object recognition, 3D modeling, and image stitching.

2.1. Feature Detection

To detect features, we will build Gaussian pyramid. Given two constants: set number, which means the number of different $k\sigma$ in each pyramid layer, and layer number of the pyramid. In each layer, we define the initial $k\sigma$ in the first set for the Gaussian Blur $G(x, y, k\sigma)$ at scale $k\sigma$, for the second set we use $k^2*\sigma$, for the third set we use $k^3*\sigma$, and so on. For each layer, we resize the image of the last layer to half, and keep performing Gaussian Blur at different scales.

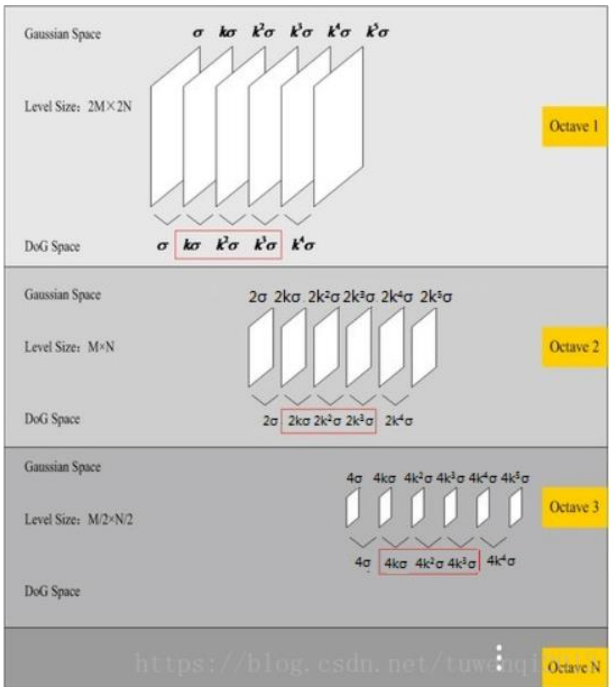


Fig. 1: Gaussian pyramid.

By building the Gaussian Pyramid, we get the Gaussian set of different layers. In each layer, we perform Difference Of Gaussians (DOG) for the targeted set with its previous set and its next set. For every pixel in the target set, we compare it with the pixels around it with the range of 3x3 matrix centered by the index of the pixel. There are 26 pixels for each pixel to compare with in total (8 for the local set, 9 for the last set, and 9 for the next set). After the comparison, if this pixel is the minimum or the maximum of it, then this pixel is the feature point.

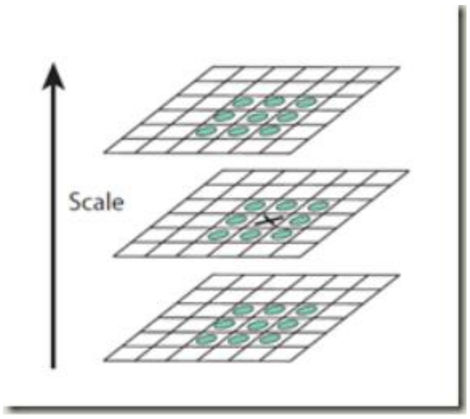


Fig. 2: Difference of Gaussians.

We may find too many feature points in various scale space, so we need to discard some of them by the information of their neighbor pixels and the principal curvatures.

Since the index of extreme point we found in discrete space might not be the same in continuous space, we need to use sub-pixel interpolation to help us locate the index of feature point in the real continuous space from the discrete space. We use the quadratic Taylor Series of DOG with the index of extreme value as the origin to calculate the bias to the index of extreme value in discrete space. If the bias is larger than 0.5, it means that the original index is too far from the true one, so we abandon it and find the others. If the bias is less than 0.5, we will add the bias to get the new index.

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

Fig. 3: Quadratic Taylor Series of DOG.

DOG function has a strong edge response, even if a small amount of noise would affect the choice of feature point along the edge. Consequently, we have to find out the feature points which have high principal curvatures. The principal curvatures across the edge would be much larger than the principal curvatures along it. To figure out the principal curvature, we should solve for the eigenvalues of the second order Hessian matrix.

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Fig. 4: Hessian matrix.

The eigenvalues of Hessian matrix are proportional to the principal curvatures of *D*. It turns out that the ratio of two eigenvalues, say the larger one is α, the smaller one is β. The ratio $\gamma = \alpha/\beta$ is sufficient for the SIFT purposes. As above, the trace of the Hessian matrix *Tr(H)*, which is $D_{xx} + D_{yy}$, equals to $\alpha+\beta$, and the determinant of the Hessian matrix *Det(H)*, which is $D_{xx}*$, $D_{yy} - D_{xy}*D_{xy}$, equals to $\alpha*\beta$. Consequently, we can define a ratio $\Upsilon = Tr(H)^2 / Det(H) = (a/b+1)^2/(a/b)$. We can know that when Υ is larger, there will be a higher absolute difference between α and β. When Υ is larger than the threshold, we can discard that extreme point.

$$\mathrm{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta$$

$$\mathrm{Det}(\mathbf{H}) = D_{xx}D_{yy} - D_{xy}^2 = \alpha\beta$$

$$\mathrm{R} = \mathrm{Tr}(\mathbf{H})^2 / \mathrm{Det}(\mathbf{H}) = (r+1)^2/r$$

Fig. 5: The process of eliminating edge responses.

For each feature point, we calculate the gradient for *x*-axis and *y*-axis, then we use arctangent to get the desired orientation θ. With the orientation on each feature point, we can achieve invariance to rotation as the feature point descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation.

After we find the feature point location at particular scale and assigned orientation, we can achieve invariance to translation, scale, and rotation. Furthermore, we need to build the feature point descriptor to make the feature point also have the invariance to the illuminance and viewpoint. To achieve the invariance to illuminance and viewpoint, we need to formalize each feature point descriptor. First we build a set of orientation histogram on 4*4 pixel neighborhoods with 8 bins, representing 8 directions. For each feature point, we can build 16 sets of orientation histogram with its 16*16 pixel neighborhoods. With the previous way, we can create a 128-dimensional feature point descriptor to represent a feature point.

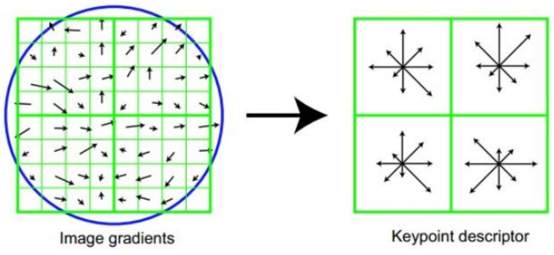


Fig. 6: Gaussians filter on image gradient.

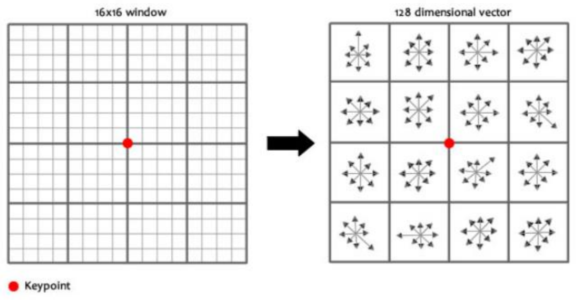


Fig. 7: The creation of feature point descriptor.

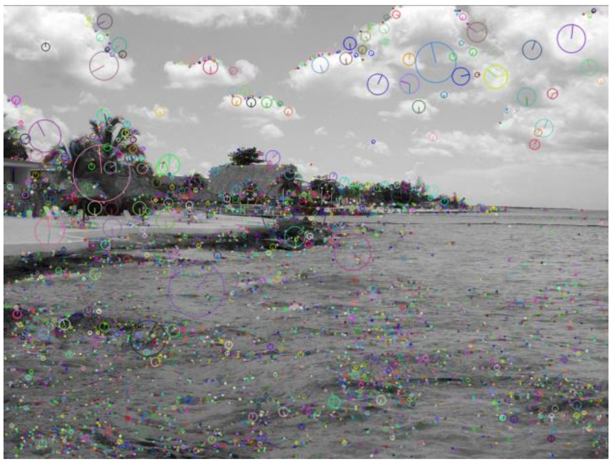
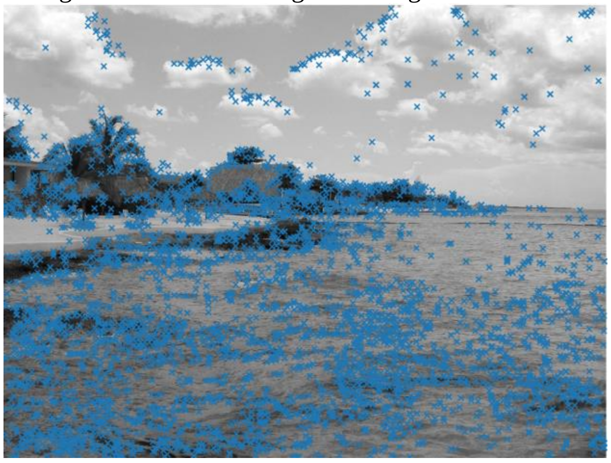


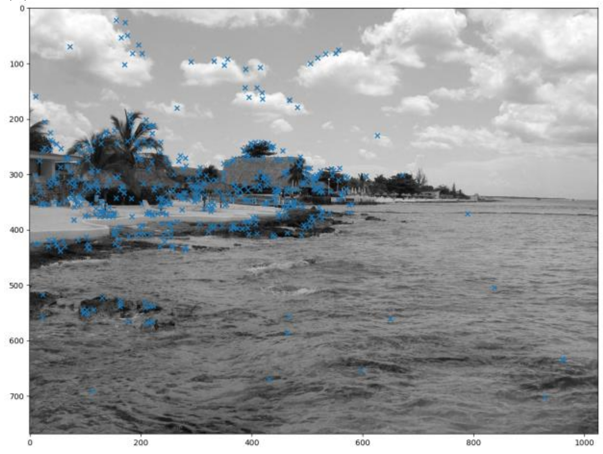
Fig. 8: Result of feature detection.

2.2. Feature Matching

With the feature descriptor, we can easily get the matching of two feature points based on finding the smallest Euclidean distance between feature descriptors on two images. However, according to David G. Lowe [11], we should concern the ratio of the closest and the second closest matching pairs. If the ratio is higher than the threshold, say 0.7 here, then we should discard both of them. Since in the case that the second closest is very near to the closest descriptor, it may happen due to noise or some other reasons. According to David G. Lowe [11], this purpose successfully rejects 90% of wrong matches, while discarding less than 5% of correct matches. Figure 9 (a) is the feature points we detect initially, Figure 9 (b) is the refined feature points after we discard low-quality feature points. We can observe that there's a lot of feature points being rejected, this method ensured the quality of matching pairs while saving a lot of time for image matching afterward.



(a)



(b)

Fig. 9: Feature point with and without successful rejection.

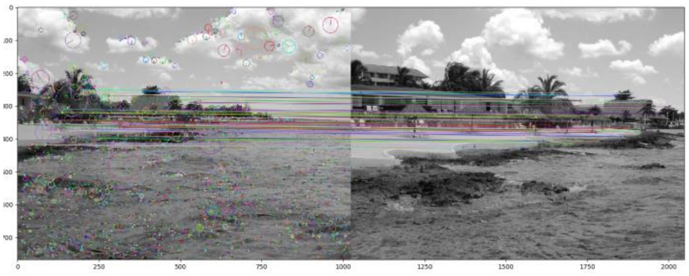


Fig. 10: Feature point matching.

3. IMAGE MATCHING

After we have the desired matching pairs, we use RANSAC, which is an iterative method to estimate the transformation model. After matching up the images, we use pyramid blending to smooth the conspicuous boundaries.

3.1. RANSAC

For image matching, we will use RANdom SAmple Consensus (RANSAC) to implement the image matching. We use some matching pairs to generate a transform matrix. We first express the matching pairs in homogenous coordinate, and then we make the use of numpy.linalg.lstsq to get the least square solution for the linear matrix equation, which is the transform matrix from the matching pairs. After that, we try this transform matrix on every matching pairs to examine the correct rate, we can generate the transform matrix many times and choose the transform matrix which achieves the most pairs of matching with little tolerance as the final transform matrix.

With the final transform matrix, we should first decide the shape of the new panorama image. We have the row and column number of the original image, say they're r and c. We apply the transform matrix on (0,0), (r,0), (0,c), (r,c) to get the transformed corner, and we can find out the new corner with the transformed and original corner. With the new corner, we can get the shape of the new panorama image.

Assume that there are two input images, we choose image in the left side as the base image and put it in the upper left corner. With the desired transform matrix, we can re-project the other images to fit in the base image.

There is a theory about how to determine the number of matching pairs and how many trials we run RANSAC to get the robust answer. Assume we take n pairs and the probability of inliers is p, and the times of trials is k, the probability P we find it success after k times of trial is $P = 1 - (1-p^n)^k$, the following Figure. 11 shows the equation of k times relative to probability.

$$k = \frac{\log(1-P)}{\log(1-p^n)}$$

Fig. 11: Equation for times of trial.

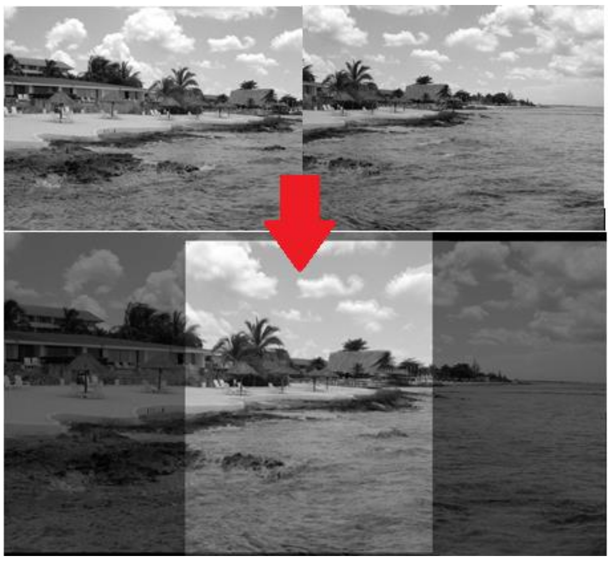


Fig. 12: RANSAC.

With more and more images stitched on the panorama image, we may find the output more and more curved
If we want to create the panorama of 360-degree field of view, we should first warp the input images to the cylinder to get a better performance. As in Figure. 13, the input images are similar to the black edges, but we should consider it as the red circle to look better. Without warping the multiple input images to the cylinder, the expected output would look like a rounded and unnatural image.

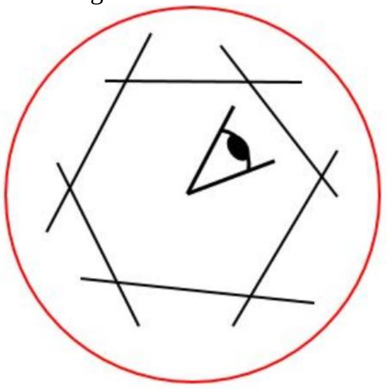
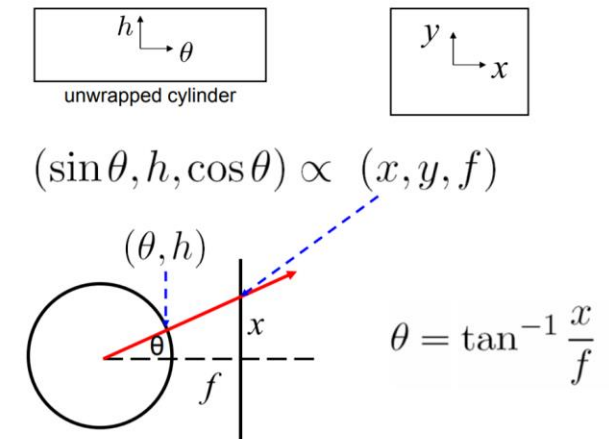
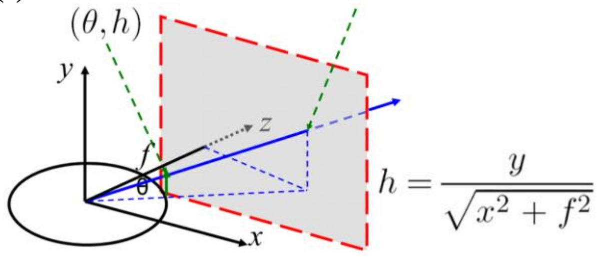


Fig. 13: Warp to cylinder.

We should first project the input images from x-y coordinate to h-theta coordinate, which is the coordinate of the cylinder, with f = the radius of cylinder, theta = $\tan^{-1}(x/f)$, and $h = y/(x^2+f^2)^{0.5}$. With theta and h, we can project the coordinate back to x-y coordinate to get a new index for every pixel.


h
θ
unwrapped cylinder
y
x
$(\sin\theta, h, \cos\theta) \propto (x, y, f)$
(θ, h)
x
θ
f
$\theta = \tan^{-1}\frac{x}{f}$

(a)



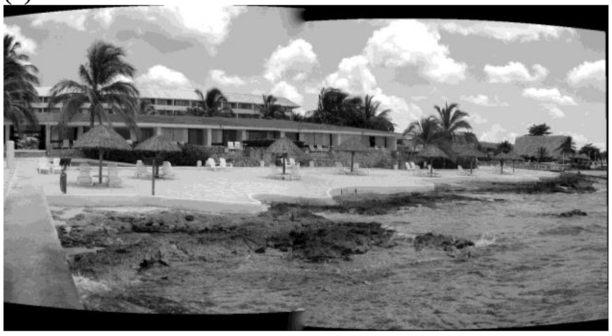
(b)

$$x' = s\theta = s\tan^{-1}\frac{x}{f}$$

$$y' = sh = s\frac{y}{\sqrt{x^2 + f^2}}$$

s=*f* gives less distortion

(c)



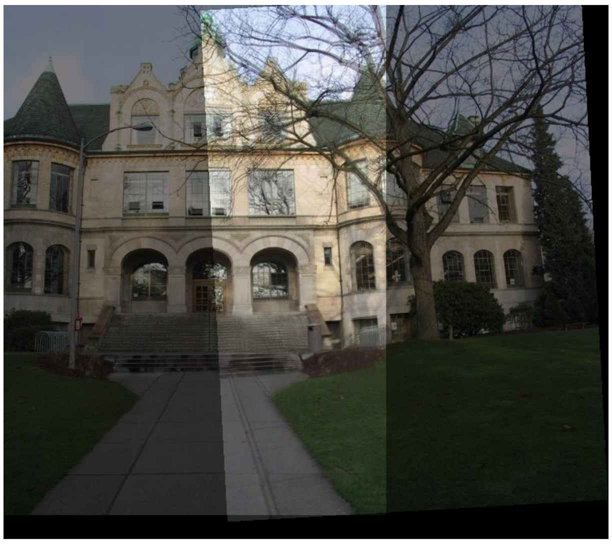
(d)

Fig. 14: Process of warping.

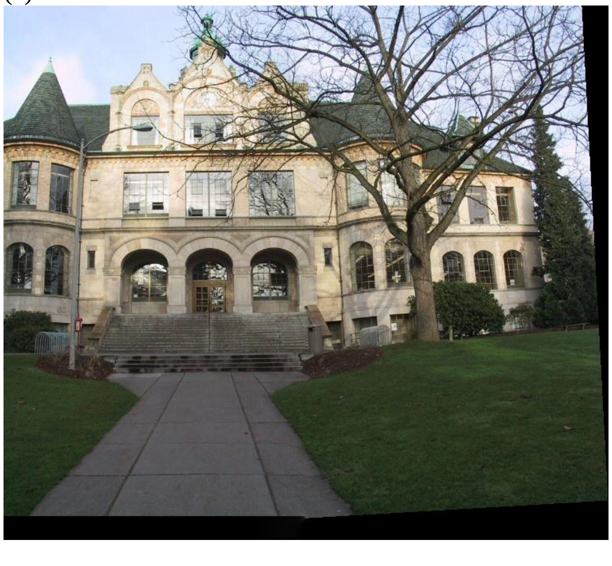
3.2. Pyramid Blending

Due to parallax, distortion, exposure difference, and so on, we may see conspicuous boundaries at the overlapping area of two images, just as Fig. 15 (a). To fix this problem, we will apply pyramid blending on these images, which can blend the images while keeping the significant features in the mean time.

First we take out the overlapping part from two images, say they are subA and subB, and we apply Gaussian blur on them to get the blurred images, say they are GA and GB. Second, we downsize GA and GB into multiple sizes with Gaussian to build up the Gaussian pyramid (we use cv2.pyrDown() here). Third, we expand the pyramid and let original layers subtract the expanded layers to get the Laplacian images in each layer, the Laplacian pyramid is built up here. After generating Laplacian pyramid for overlap images subA and subB, we combine two images in different Laplacian levels by combining partial images from each of them, say the result is LS. Finally, we expand the LS from the top level to the next level and add it to the original Laplacian image in the corresponding layer to generate the latest image. We repeat this step until reaching the ground level and the result will be the successful blending image. With input image in Figure 15 (a), Figure 15 (b) is the output with successful blending.



(a)

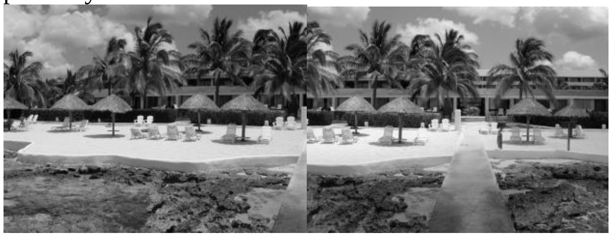


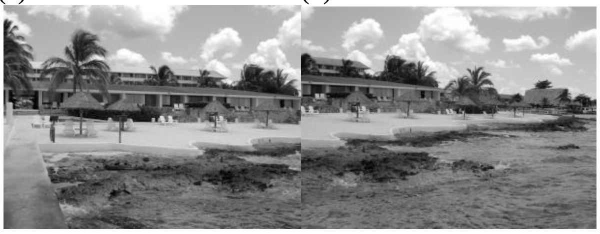
(b)

Fig. 15: Pyramid blending.

4. RESULT

With feature detection, feature matching, and image matching, we can now successfully generate a good-looking panorama with a set of images. With input images in Figure 16 (a), (b), (c), and (d), we expect to gain the output of Figure 16 (e) which is stitched perfectly.


(a) (b)


(c) (d)

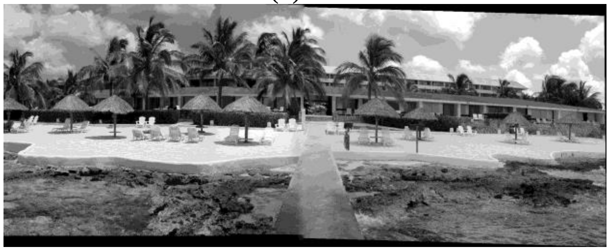
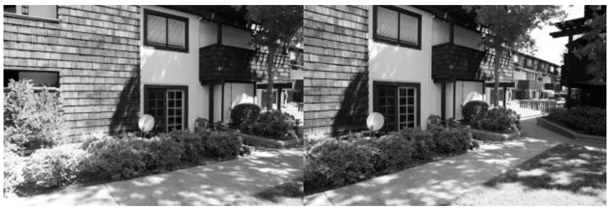

(e)

Fig. 16: Result.


(a) (b)

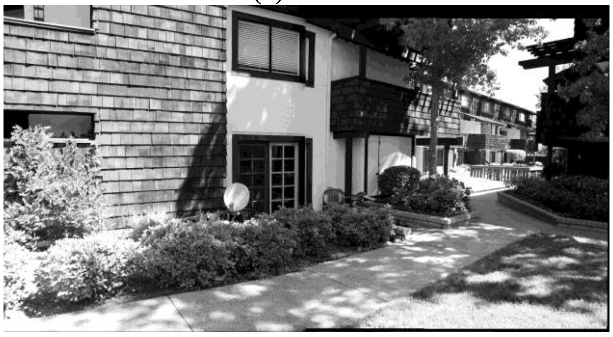
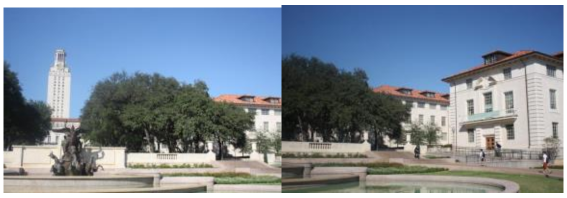

(c)

Fig. 17: Result.


(a) (b)

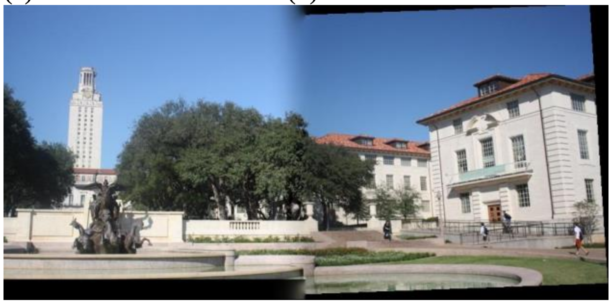
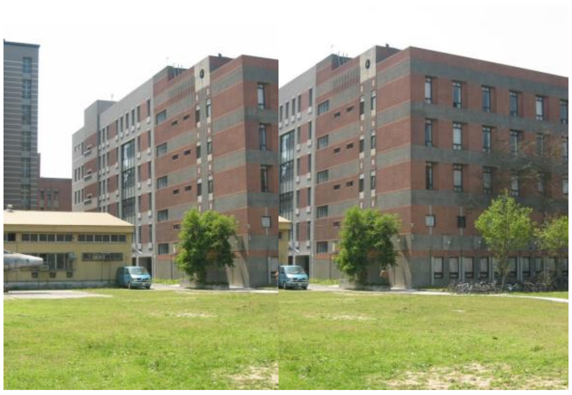
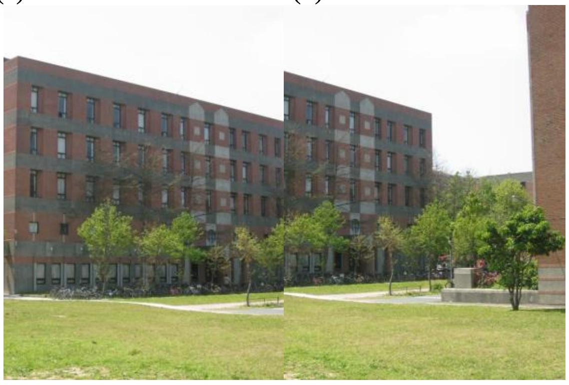

(c)

Fig. 18: Result.


(a) (b)


(c) (d)

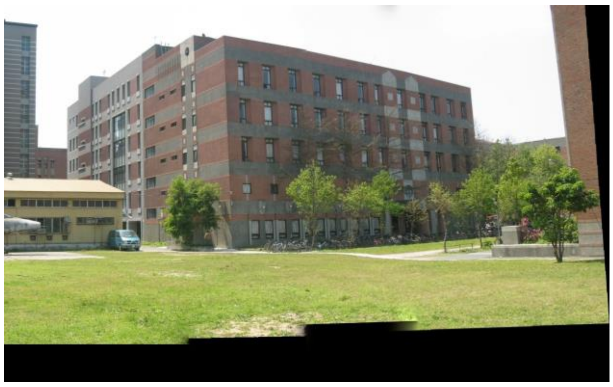

(e)

Fig. 19: Result.

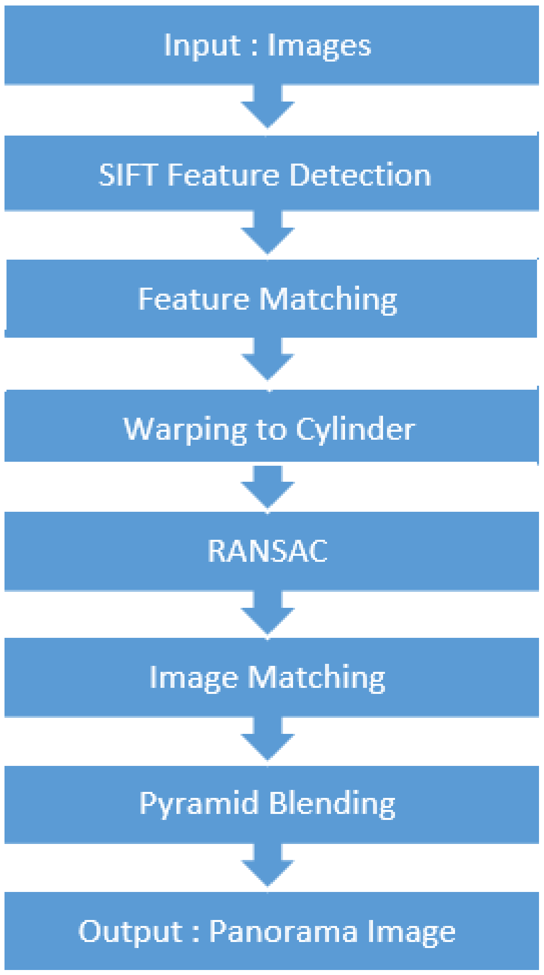


Fig. 20: The flow chart of this paper.

5. CONCLUSION AND FUTURE WORK

With multiple input images taken in panorama style, we can successfully produce an excellent panorama image. Scale-invariant feature points are detected by SIFT to ensure the quality of the feature points, if the quality of feature points is low, the method we use afterward will be in vain. We also use SIFT to generate 128-dimensional feature descriptors for each feature point. Feature matching is easily achieved by choosing the matches with the shortest Euclidean distance. Since there are too many matches in the images, which would slow down the execution speed, we choose to refine the matching pairs by discarding those whose Euclidean distance of second best matches is close to the distance of the best matches. With successful discard, we can still ensure the matching quality while saving much time. Finally, we use RANSAC to get the robust transform matrix to paste images together according to the feature matching pairs and we use pyramid blending to solve the problem of conspicuous boundaries in the edge of overlapping of two images.

As for the future work, we want to implement the recognizing of panoramas, which could still output the finished panorama while the order of input set is random. Finally, we want to achieve the goal to generate many different output panoramas while the input images are from different kinds of images and randomly ordered. We think this will make the process of panorama stitching more automated.

6. REFERENCES

[1] M. Brown, and D. G. Lowe, “Automatic Panoramic Image Stitching using Invariant Features,” *International Journal of Computer Vision*, Volume 74, Issue 1, pp. 59–73, 2007.

[2] Wikipedia, “Scale-Invariant Feature Transform,” https://en.wikipedia.org/wiki/Scale-Invariant_feature_transform, 2019.

[3] Wikipedia, “Random sample consensus,” https://en.wikipedia.org/wiki/Random_sample_consensus, 2019.

[4] CSDN, “Feature Detection and Matching,” https://blog.csdn.net/tuwenqi2013/article/details/83119988, 2013.

[5] Wikipedia, “image stitching,” https://en.wikipedia.org/wiki/Image_stitching, 2019.

[6] Wikipedia, “Curvature,” https://zh.wikipedia.org/wiki/%E4%B8%BB%E6%9B%B2%E7%8E%87, 2019.

[7] Wikipedia, “Second Fundamental Form,” https://zh.wikipedia.org/wiki/%E7%AC%AC%E4%BA%8C%E5%9F%BA%E6%9C%AC%E5%BD%A2%E5%BC%8F, 2019.

[8] CSDN, “SIFT,” https://blog.csdn.net/abcjennifer/article/details/7639681?fbclid=IwAR28oBbY0lPnXCJI-q1EihTSfLV5v3EvZB92WYMlF5Yuxb0zsc2mgXEQCqE, 2012.

[9] OpenCV-Python Tutorials, “Introduction to SIFT,” https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_

sift_intro/py_sift_intro.html?fbclid=IwAR3lFcVKDwEn4FnYH8rU5K2Cnxf9y3Ajwkr9RLGcn4y6yTOc09CxHItef6c, 2013.

[10] Seedmanee, "Image stitching," https://seedmanee.blogspot.com/2010/08/digivfx-hw2-image-stitching.html?fbclid=IwAR0fNKhO78kBDbED5A5egtpPhlsU-NEGIIcCtvpZNABasdz8Rv81Q40Th5E, 2010.

[11] David G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, *International Journal of Computer Vision*, 2004.

[12] CSDN, "Laplacian Pyramid Blending," https://blog.csdn.net/u014485485/article/details/89516028, 2019.

[13] Pyimagesearch, "OpenCV panorama stitching," https://www.pyimagesearch.com/2016/01/11/opencv-panorama-stitching/, 2016.

[14] Winbiz, "360 Degree Panorama Photo Stitching Services," http://winbizsolutionsindia.com/photo-editing-services/panoramic-photo-stitching-services/, 2019.

[15] Yung-Yu Chuang, "Image stitching", https://www.csie.ntu.edu.tw/~cyy/courses/vfx/13spring/lectures/handouts/lec07_stitching.pdf, 2013.

[16] CSDN, "Multi-band Blending", https://blog.csdn.net/real_myth/article/details/52343612, 2016.

[17] Chaman Singh Verma, and Mon-Ju, "Panorama Image Mosaic", http://pages.cs.wisc.edu/~csverma/CS766_09/ImageMosaic/imagemosaic.html, 2009.

[18] Adobe Open Source Datasets, "Adobe Panoramas Dataset", https://sourceforge.net/adobe/wiki/Home/, 2010.

[19] Brandt, J. "Transform coding for fast approximate nearest neighbor search in high dimensions", *IEEE Conf. on Computer Vision and Pattern Recognition*, 2010.

[20] Yuxin's Blog, "SIFT and Image Stitching", http://ppwwyyxx.com/2013/SIFT-and-Image-Stiching/, 2013.