NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

AI6121 COMPUTER VISION

ASSIGNMENT 2 IMAGE STITCHING

School of Computer Science and Engineering

Programme: MSAI

Date: 1 November 2023

Authored By: Tan Jie Heng Alfred (G2304193L)

Siew Tze Kang, Julian (G2303643F)

Contents

1. Introduction

Being able to take panoramic photos has become commonplace, with smartphones being able to do it. This allows users to capture wider "field of vision" in an artificial manner. This is done by stitching multiple images using homography. This report will cover the theory behind the algorithm, the implementation of it within the assignment's boundaries and a discussion of the results.

In order to create a panoramic image, we go through the following three phases:

1. Detecting and matching of features
2. Recovery of the homography matrix
3. Image stitching

In the following sections, we will describe and explain the steps in each phase.

2. Feature Points Detection and Matching

2.1 Harris corner detector

For detecting features, Harris corner detector is an option. As the name suggests, this detector can detect corners and also edges within the image. This is done by a shifting window (i.e., kernel). When the window shifts in a direction, there would likely be a change in pixel intensity. By studying how the pixel intensity changes with respect to the direction of shift, we could detect different kinds of feature points. The figure below shows the result when Harris corner detector is applied to two images that we wish to stitch.

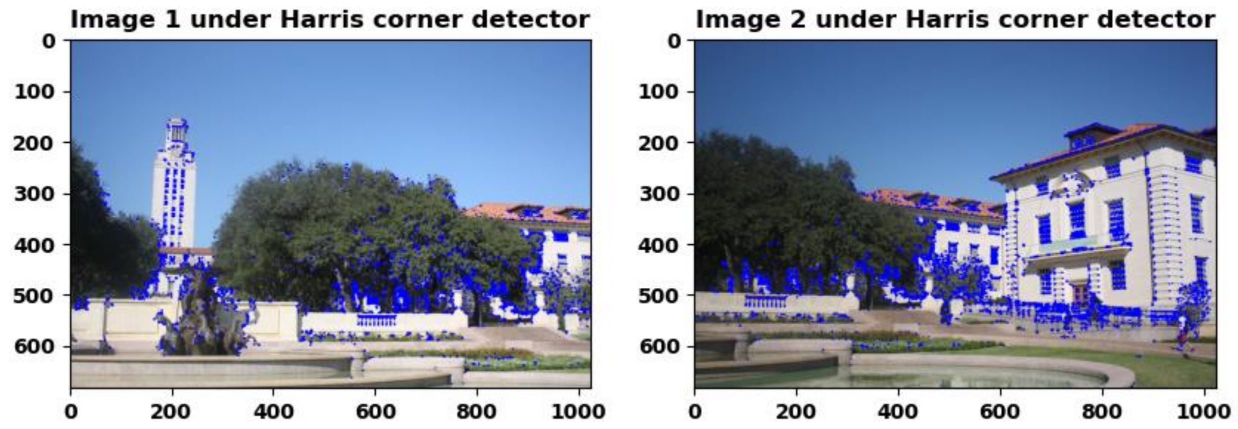
Image 1 under Harris corner detector
Image 2 under Harris corner detector
0
100
200
300
400
500
600
0
200
400
600
800
1000

Fig. 1. Harris corner detector applied to two images. Blue lines are the feature points detected.

From the figure, we can see that there are many edges and corners detected within the images. There are even similar feature points detected within the two images, and if we could match them using an algorithm, we would be able to stitch the two images at this common point. Unfortunately, we are unable to describe the feature points qualitatively using Harris corner detector. This prevents us from easily comparing and matching two feature points if they are representing the same object in the physical world. Because of this, we would have to choose another feature point detector.

2.2 Scale-Invariant Feature Transform (SIFT)

Feature Point Detection

In order to perform image stitching, we decided to use scale-invariant feature transform (SIFT) instead of Harris corner detector. On top of being able to describe feature points, the also former performs better if the images are of different scale.

SIFT is a method to detect and describe feature points, such as edges and corners, based on the image gradient. These feature points can be detected using Laplacian of Gaussian (convolution), by locating the zero-crossing of the Laplacian on the image-space (S. Tabbone, 1994). As it turns out, from an equation in heat diffusion, the Laplacian of Gaussian could be approximated as a difference in Gaussians:

$$\sigma\nabla^2 G(x,y) = \frac{\partial G(x,y)}{\partial\sigma} \approx \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{k\sigma - \sigma},$$

where σ and $k\sigma$ are the standard deviations of the two Gaussians. In other words, we have,

$$(k-1)\sigma^2\nabla^2 G \approx G(x,y,k\sigma) - G(x,y,\sigma)$$

Because finding the Laplacian of Gaussian, $\nabla^2 G$, is computationally more expensive, we will approximate it using difference of Gaussians (DoG). Note that $(k-1)\sigma^2$ are constants and does not affect the calculation to find the zero-crossing of $\nabla^2 G$.

SIFT applies this concept and constructs a Gaussian image pyramid. In this pyramid, there are several octaves (i.e., levels), each with multiple images of the same size and scale. Within an octave, the images are smoothed using Gaussian blur (kernel), with different standard deviations. Between two consecutive images in the same octave, the standard deviation increase by a multiple of k across the octave. Mathematically, say we have two consecutive smoothed images, call them L_1 and L_2, in the first octave, then they can be computed as,

$$L_1(x,y,\sigma) = G(x,y,\sigma) * I_{(h_1,w_1)}(x,y)$$
$$L_2(x,y,k\sigma) = G(x,y,k\sigma) * I_{(h_1,w_1)}(x,y)$$

where $I_{(h_1,w_1)}(x,y)$ is the original image, with size $h \times w$ (i.e., $h_1 = h$ and $w_1 = w$) , and $*$ is the convolution operation.

Now, moving from octave to the next, the image is being downsampled by 50%. Using the above image in the first octave, the corresponding blurred images, L_1' and L_2' are,

$$L_1'(x,y,\sigma) = G(x,y,\sigma) * I_{(h_2,w_2)}(x,y)$$

$$L_2'(x,y,k\sigma) = G(x,y,k\sigma) * I_{(h_2,w_2)}(x,y)$$

where the image size is now $h_2 \times w_2$, and $h_2 = \frac{h_1}{2} = \frac{h}{2}$ and $w_2 = \frac{w_1}{2} = \frac{w}{2}$. In general, the images in the k^{th} octave will be of the size $h_k = \frac{h}{2^{k-1}}$ and $w_k = \frac{w}{2^{k-1}}$.

With these, SIFT could construct a DoG pyramid, by computing DoG for each pair of neighbouring images within each octave. For example, the first level pyramid would contain $L_2 - L_1$, which are two smoothed images from the Gaussian pyramid. Mathematically, it can be proven that,

$$\begin{aligned} L_2 - L_1 &= G(x, y, k\sigma) * I_{(h_1,w_1)}(x, y) - G(x, y, \sigma) * I_{(h_1,w_1)}(x, y) \\ &= [G(x, y, k\sigma) - G(x, y, \sigma)] * I_{(h_1,w_1)}(x, y) \end{aligned}$$

Now, knowing that DoG is approximately the Laplacian of Gaussian, we have,

$$L_2 - L_1 \approx (k-1)\sigma^2 \nabla^2 G * I_{(h_1,w_1)}(x, y)$$

In other words, the DoG pyramid contains information about the Laplacian of Gaussian, across the scale-space (i.e., differently scaled Gaussian convolutions, determined by the standard deviations). This allows us to use the DoG pyramid to find the feature points in the image.

To detect feature points, SIFT checks whether the point of interest is larger than 8 neighbours within the same DoG map, as well as 9 neightbours of the previous and next neighbouring DoG maps (i.e., across scales). If that is the case, then the point is flagged as a key point (i.e., feature point). This search for extremas of the DoG pyramid across scale-space means that the feature points detected to be largely independent of the scale of the image. This is beneficial over calculating the Laplacian of Gaussian explicitly, as the Laplacian itself is not scale-invariant due to the σ^2 term within the Gaussian function. Hence, SIFT is an efficient method to detect feature points that tend to be scale invariant.

The figure below shows the key features detected by SIFT.

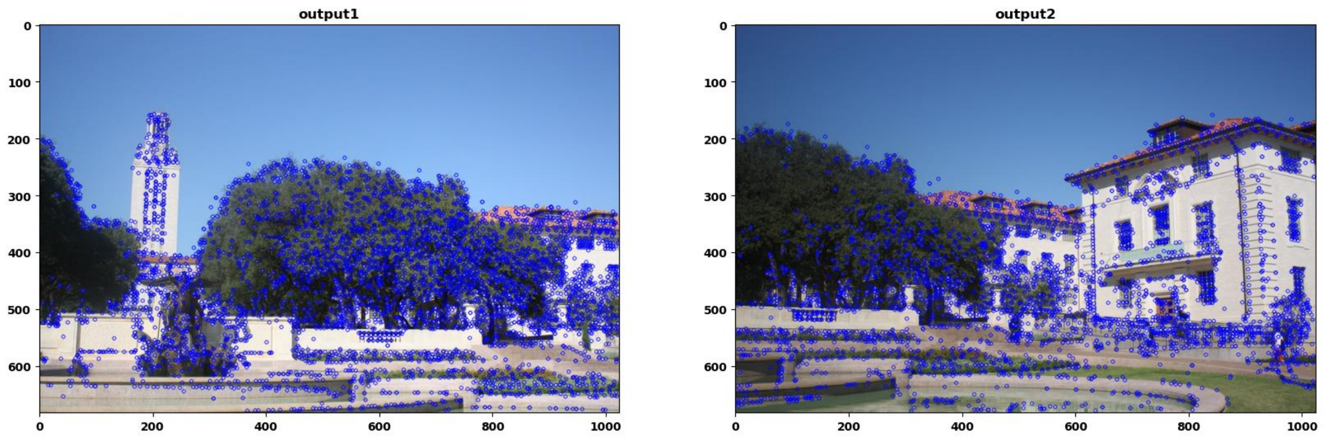

Fig. 2. SIFT detector applied to two images. Blue circles are the feature points detected.

2.3 Feature Point Description

After finding the feature points, we hope to describe these points so that we could compare them across the two images. For a given feature point, its feature descriptors should ideally be the invariant to different settings. These include changes in illumination and geometric transformations such as rotations and scaling. Therefore, it may not be ideal to describe the feature points using pixel intensity or pixel gradient, as these descriptors are dependent on the settings the image is taken. Fortunately, SIFT provides a way to describe these feature points that is largely invariant to the aforementioned issues.

In SIFT, we investigate the local neighbourhood of the feature points. Typically, this is a 16×16 patch centred at the feature point. To deal with rotational invariance, the image patch of the feature point must be warped with a dominant orientation. The SIFT function in the `cv2` library helps to automatically identify the canonical scale and dominant orientation (or canonical orientation) of the patch. In this case, we simply warp the region around the feature point to the canonical orientation (i.e., point in the canonical orientation), and scale and resize the region to 16×16 pixels. By always pointing in the canonical orientation, the feature points are rotational invariant.

If the canonical orientation is unavailable, we would have to examine the surrounding pixels, using them as an orientation collection region, to sample the dominant orientation. The size of the region in which we sample the orientation depends on the keypoint scale, calculated as 1.5σ. In this collection step, each neighbouring pixel in the region will have its gradient amplitude and orientation (i.e., gradient angle) calculated. Using finite difference, we have $\frac{\partial x}{\partial L} \approx \frac{L(x+1,y)-L(x-1,y\,)}{2}$ and $\frac{\partial y}{\partial L} \approx \frac{L(x,y+1)-L(x,y-1)}{2}$, where $L(x,y)$ refers to the pixel intensity at point (x,y). Therefore, the gradient amplitude and orientation can be approximated using the following formulae respectively.

$$m(x,y) = \sqrt{[L(x+1,y) - L(x-1,y)]^2 + [L(x,y+1) - L(x,y-1)]^2}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{\big(L(x,y+1) - L(x,y-1)\big)}{\big(L(x+1,y) - L(x-1,y)\big)}\right)$$

The gradient angle is then binned to a count histogram, with uniform interval, and scaled by its amplitude to obtain the dominant orientation as the peak angle (i.e., most count). This will be the dominant orientation. With this, we can again warp the region to the dominant orientation, rescale and resize the region to obtain the 16×16 image patch. This image patch will now be our feature descriptor. (Gillevicv, 2013)

Within the 16×16 patch, we compute the gradient orientation and magnitude in the same way outlined in the formulae above. Then, we divide the patch into 16, 4×4 sub-patches. Within each sub-patch, we compute the gradient direction (similarly, scaled by its magnitude) of each pixel, bin them into 8 directions (i.e., histogram with 8 uniform interval). Finally, we concatenate 16 of the gradient direction histogram, each from each sub-patch, to obtain a 128-bin histogram.

This final histogram can be thought of as a 128-dimension vector and will be used to describe our feature point.

The images below show two of the feature descriptor vectors, visualized as histograms. The left histogram is a keypoint from the first image and the second is a keypoint from the second image.

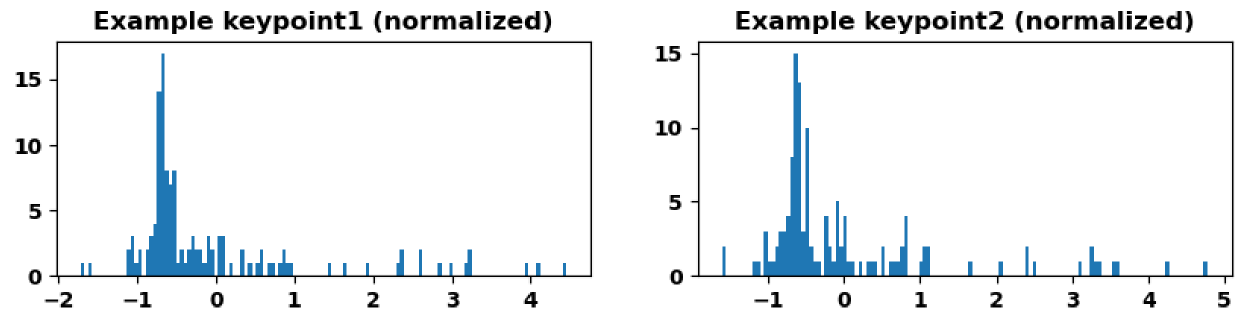

Fig. 3. Keypoints description visualized as histograms.

From the visualization above, it seems that the two keypoints are very similar. Indeed, we are now able to compare feature points using their descriptor vectors, and hence define a concept of similarity between two keypoints in the two images.

2.4 Feature Points Comparison

Before we can effectively compare the feature points, we first apply normalization. This is done to ensure that all the feature point descriptors have a mean of 0 and a unit standard deviation. Now, we will compare each of the feature points in the first image to the feature points in the second image. To do that, we will find the sum of squared distance (i.e., Euclidean distance squared) between the two feature points.

$$SSD(p_1, p_2) = \|p_1 - p_2\|^2,$$

where $p_1, p_2 \in \mathbb{R}^{128}$ are the normalized descriptor vectors of some feature points in the first and second image respectively. Intuitively, the smaller the value of $SSD(p_1, p_2)$, the likelier they are to be similar. Hence, we might consider thresholding the SSD to be lower than specified value d. However, note that if p_1 is a very common feature in the physical world (e.g., a leaf on a huge tree) , there could be multiple choices for p_2 (many similar looking leaf), such that $SSD(p_1, p_2) < d$. But this does not mean that p_1 and p_2 are the same object in the physical space. Therefore, they may not be a good match and this thresholding method might produce false matches.

Instead of thresholding the SSD value, we will look for another threshold – the ratio of two $SSDs$. Let p_2 best matches to some p_1, and p_2' be the second best match to the same p_1, i.e., $SSD(p_1, p_2) \leq SSD(p_1, p_2') \leq SSD(p_1, p_2'')$, for any feature points p_2'' in the second image, such that $p_2'' \neq p_2$. Then, we shall threshold against the ratio of $SSD(p_1, p_2)$ to $SSD(p_1, p_2')$,

$$\text{If } \frac{SSD(p_1, p_2)}{SSD(p_1, p_2')} < d', \text{then } (\mathrm{p}_1, \mathrm{p}_2) \text{ are a good match.}$$

Here, good match means that we regard p_1 and p_2 to be referring to the same object in the physical space. Also, d' is a user-defined threshold.

Intuitively, the smaller the ratio, the more distinct p_2 and p_2' (compared against p_1) are in the physical world. Hence, p_1 and p_2 are more likely to be distinctive object in the physical space, and a good match if their $SSD(p_1, p_2)$ is low (which is usually the case). In our experiment, we used $d' = 0.3$ and the equivalent condition of $SSD(p_1, p_2) < d' \times SSD(p_1, p_2')$. The choice of d' is so that we have sufficient pairs of good matches (between the key points from the two images) for the subsequent step. In particular, we require at least 4 points for the subsequent step of homography matrix recovery.

The figures below show the results of this step of the algorithm. The two images below show the selected key points from each image.

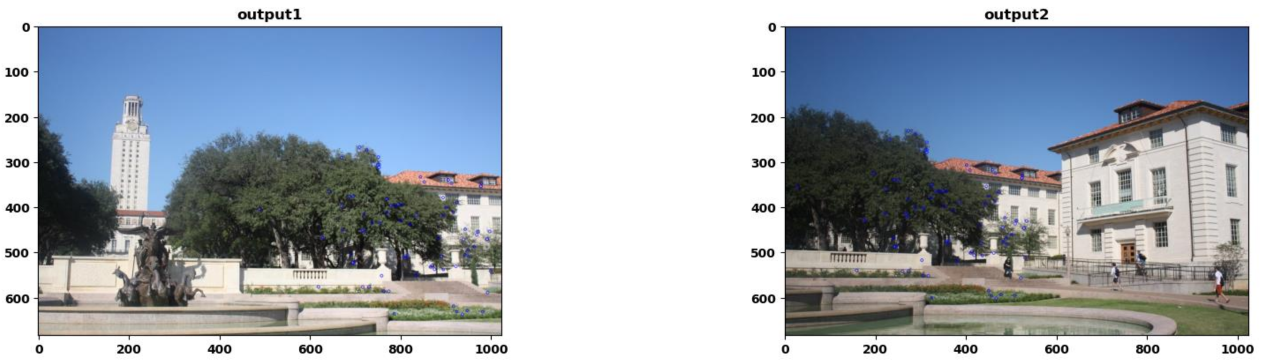

Fig. 4. Feature point pair selection in both images denoted by blue circles.

The figure below further shows the matching between the two key points.

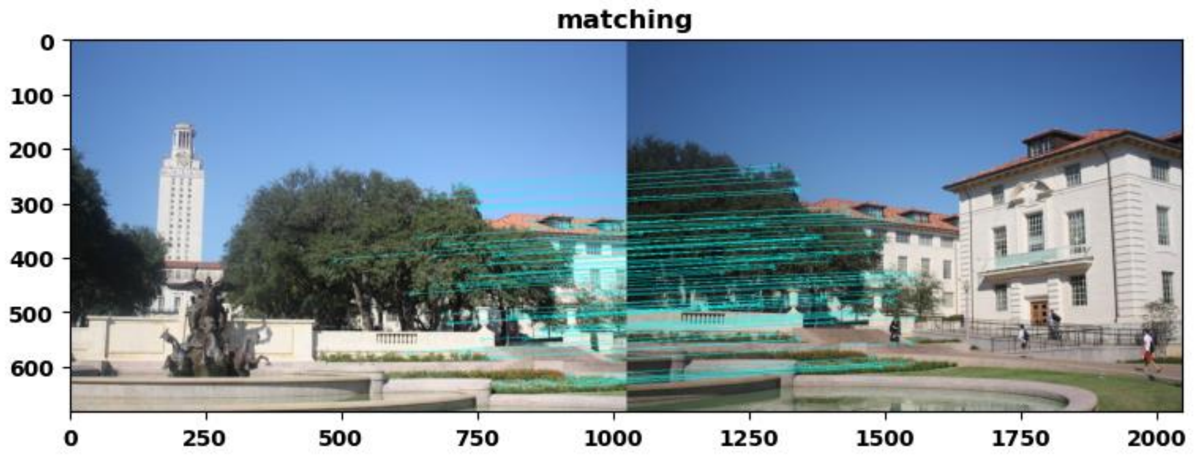

Fig. 5. The relationship of the feature point pairs illustrated by the cyan line joining the points of a corresponding pair.

3. Recovery of Homography Matrix

After finding the feature points in the two images that match with each other, we would have to transform one of the images onto the other, such that these matching feature points align with each other. This transformation is called a homography and is determined by a homography matrix, H. In this section, we will describe the process of recovering this matrix.

A homography matrix, H, is a 3×3 matrix that describes the following transformation: If $\mathbf{a} = (x, y)$ is a point in the first image and $\mathbf{a}' = (x', y')$ is the corresponding point in the second image, then

$$\mathbf{a}'_h = H\mathbf{a}_h,$$

where $\mathbf{a}_h$ and $\mathbf{a}'_h$ are some homogeneous coordinates of $\mathbf{a}$ and $\mathbf{a}'$ respectively. We can write

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}, \mathbf{a}_h = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \text{and } \mathbf{a}'_h = \begin{bmatrix} ux' \\ uy' \\ u \end{bmatrix}$$

It can be proven that H and sH, where s is a non-zero scalar, define the same transformation. Indeed, since the homogenous coordinates of a point is not unique, the scaling factor of $\mathbf{a}'_h$ in the two cases will just differ by a factor of s as well. A corollary of this is that, instead of solving for 9 unknowns, we can take $s = \frac{1}{h_{33}}$ such that the bottom right element is a constant 1. Therefore, we only need to solve for 8 unknowns to determine H. Now, each pair of corresponding feature points, $\mathbf{a}$ and $\mathbf{a}'$, we can obtain 2 equations in terms of h_{ij}, for $1 \leq i, j \leq 3, (i, j) \neq (3,3)$. Therefore, we need at least 4 pairs of $(\mathbf{a}, \mathbf{a}')$ to fully determine H. However, since our feature points may be noisy, we have to instead solve the following least square problem to obtain $\mathbf{h}'$:

$$\mathbf{h}' = \arg\min_{\mathbf{h}} |A\mathbf{h} - \mathbf{b}|^2,$$

where $\mathbf{h}$ is the vector obtained by unrolling H, and the entries of A and $\mathbf{b}$ are in terms of $x, y, x'$and y', coming from the k pairs of corresponding feature points, where $A \in \mathbb{R}^{2k \times 8}$ and $\mathbf{b} \in \mathbb{R}^{2k}$, for $k \geq 4$.

Since we may have more than 4 pairs of matching feature points, some of which may be outliers, we can use random sample consensus (RANSAC) to obtain the best H. RANSAC is an iterative method used for estimating the parameters of a mathematical model from a set of observed data which may contain outliers. In our case, the model is the homography matrix H and the observed data are the matched feature points. RANSAC works in the following steps:

1. Randomly select 4 pairs of matched feature points to compute the homography matrix H. This is done by solving the least square problem outlined above with $k = 4$.
2. Apply the computed H to the other matched feature points from the first image, and measure the error between the projected feature points and the actual corresponding feature points in the second image. The error metric can be Euclidean distance (or other geometric error). Matches for which the error is below a specified threshold are then considered inliers.
3. Count the number of inliers associated to this H and store it.

4. Repeat step 1 to 3 for n iterations (n is a hyperparameter). Keep the H with the most number of inliers (and least number of outliers).

This process is handled automatically by OpenCV's `cv2.findHomography()` function, with the parameter `method = cv2.RANSAC`. The function above directly generates the homography matrix H, given two arrays of at least 4 matching feature points, using the RANSAC method.

4. Image Stitching

After obtaining the best homography matrix H, we can apply the matrix to the first image, I_1, to get a transformed image, $I_1' = HI_1$. The figure below depicts an original image and its transformed image.

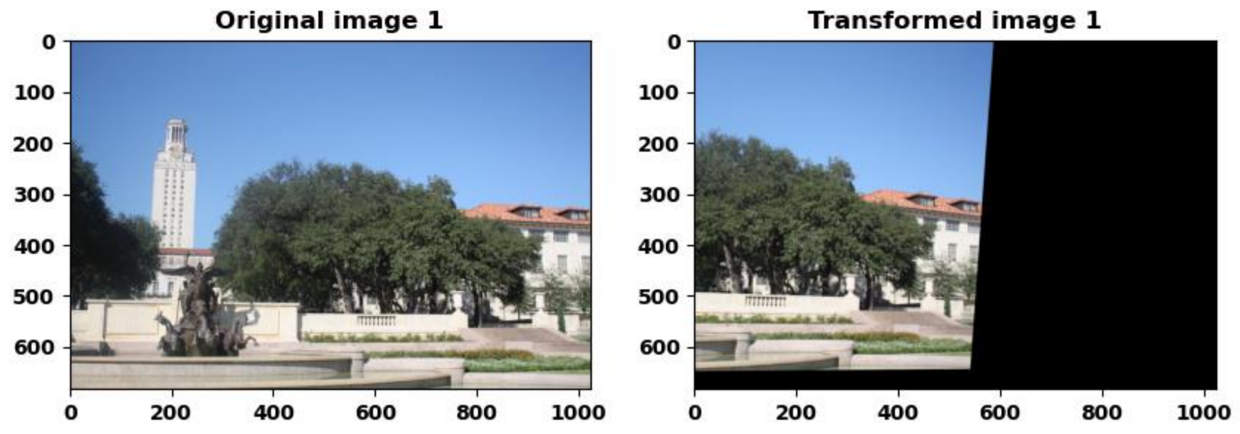

Fig. 6. Image 1, I_1, and its transformed image 1, I_1'

Clearly, I_1' has been cut off, as the transformation includes some sort of translation of the image to the left. In other to bring the image back, we have to find a translation matrix, T. Note that a translation matrix is a special type of homography matrix, with fewer unknowns. A translation matrix is of the form,

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix},$$

where t_x and t_y are the translation in the x and y directions respectively.

To recover this transformation matrix, we can observe where the four corners of the original images are being projected to. Suppose $\mathbf{a} = (x, y)$ and $\mathbf{a}' = (x', y')$ are the one corner of I_1 and I_1' respectively. Then, we can solve for t_x and t_y using the following equation:

$$\mathbf{a}' = T\mathbf{a}$$

Similar to the recovery of the homography matrix, the $\mathbf{a}$ and $\mathbf{a}'$ will produce two linear equations. Since we only have two unknowns, t_x and t_y, to solve for, we could theoretically use one pair of $\mathbf{a}$ and $\mathbf{a}'$.

However, in the implementation of the code, we leveraged on the fact that transformation matrix is a homography matrix, and found T using the `cv2.getPerspectiveTransform()` function. This function takes in exactly 4 points and produce a homography matrix. Since we only have 4 points (and we are assuming they are all inliers), there is no need to use the RANSAC algorithm. However, solving of the least square equation may still be warranted, since the points may not be perfect correspondence.

Finally, we have to stitch the images together. This is done by using another `numpy` array to account for the height and width of the panoramic image. Then, we overlay image 2 over the projected image 1. This will output the panoramic image as shown here in Figure 2.

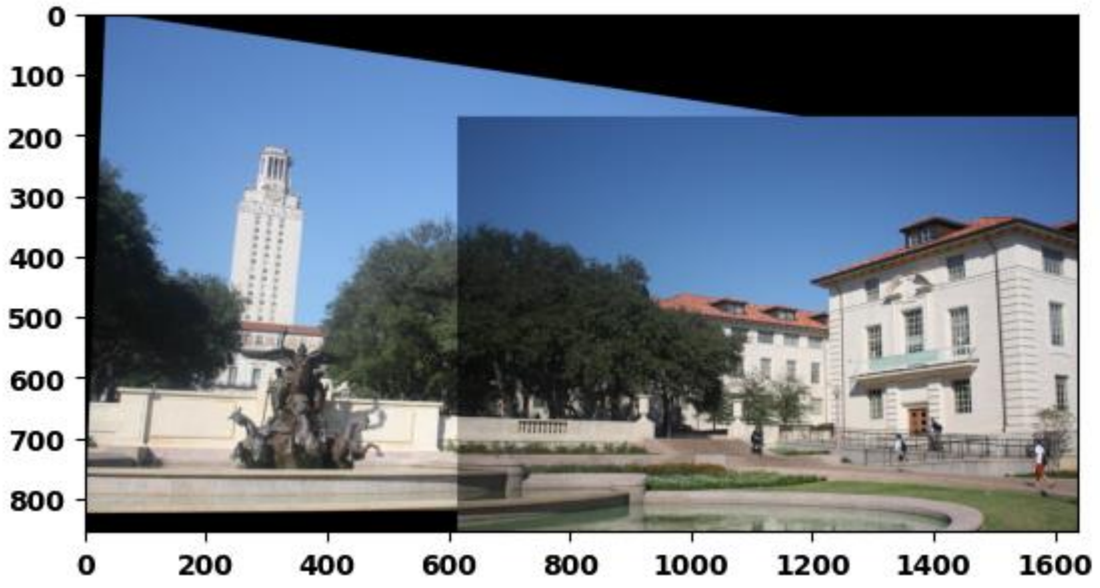

Fig. 7. Stitched up, panoramic image.

5. Results and Discussions

As observed in the stitched images above in Fig. 7, the two images have been blended to create a visually cohesive result with the elements in the image being properly aligned. The other image pairs and stitched image are shown below.

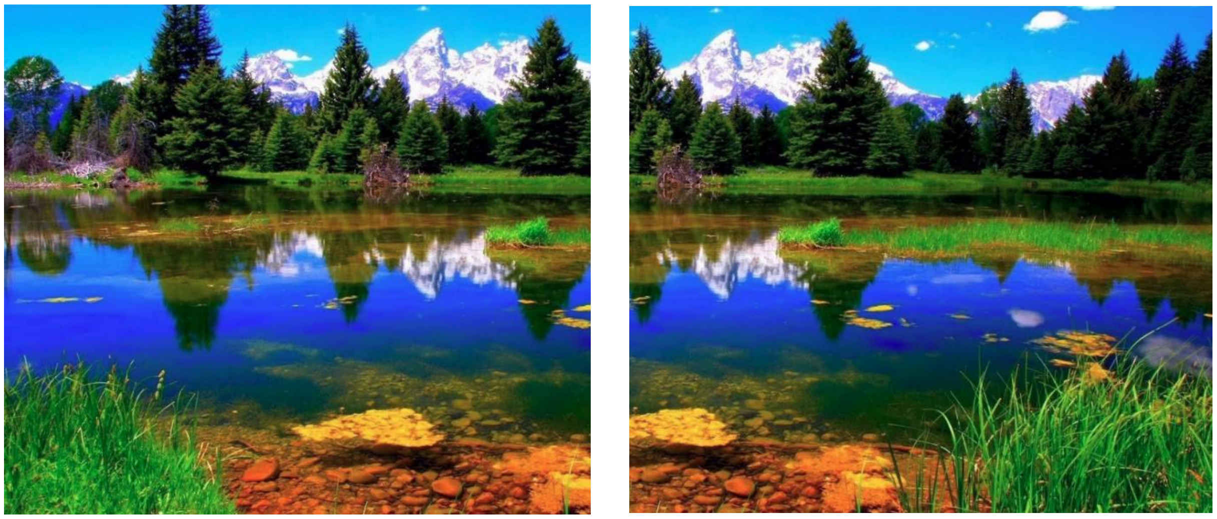

Fig. 8. Input image pair 1

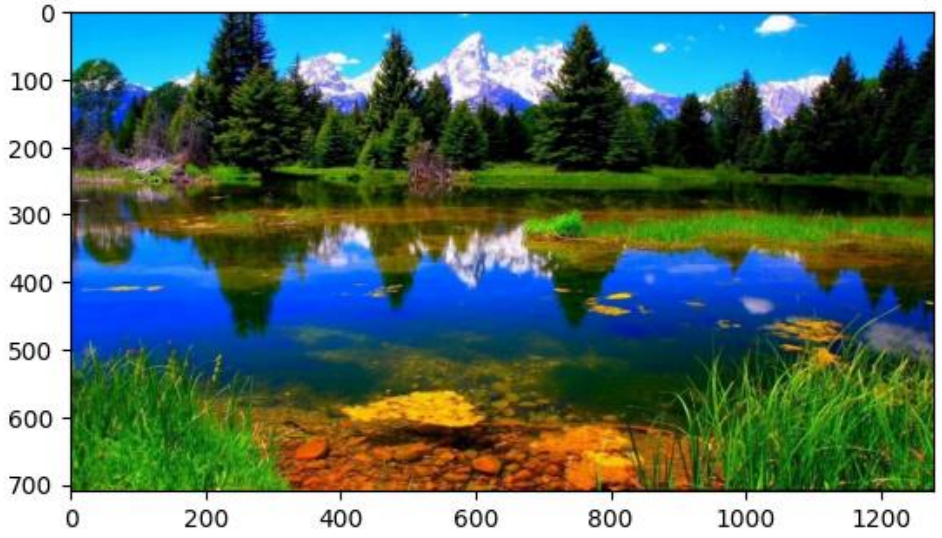

Fig. 9. Stitched Image 1

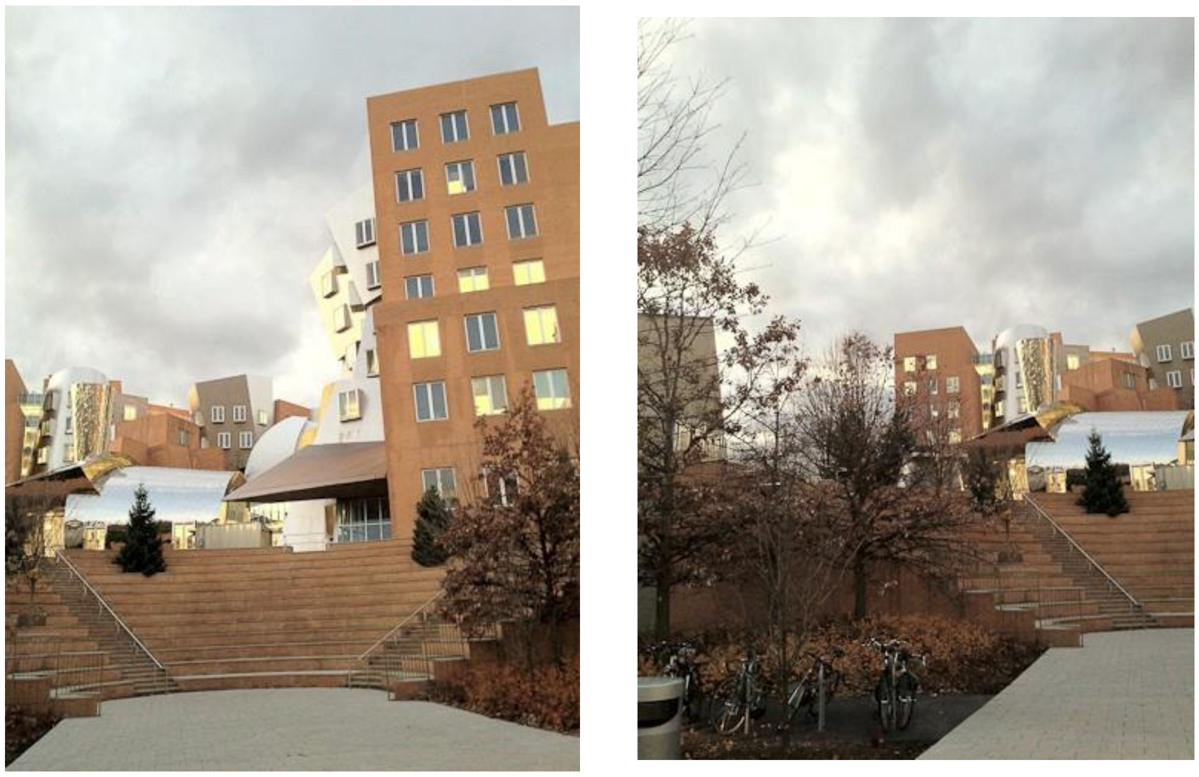
Fig. 10. Input image pair 2

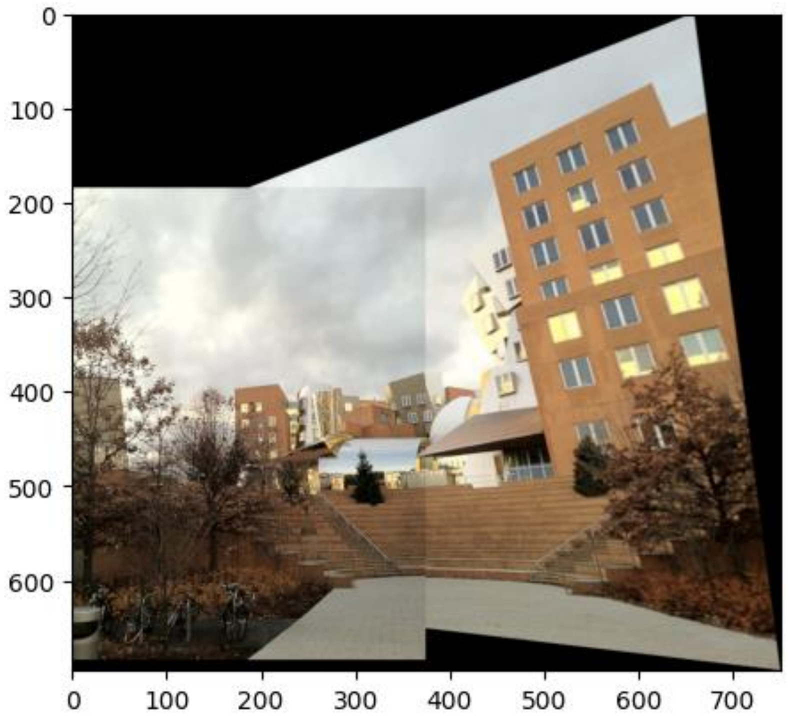

Fig. 11. Stitched Image 2

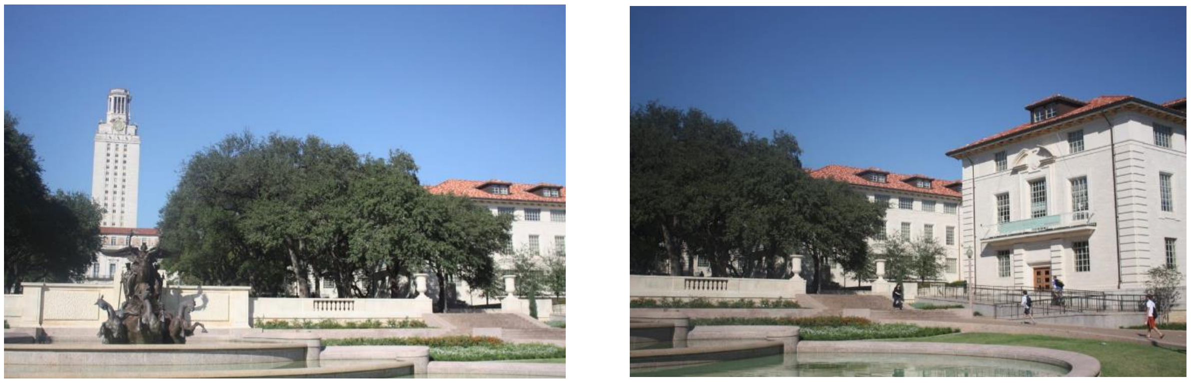
Fig. 12. Input image pair 3 (used as example in the report)

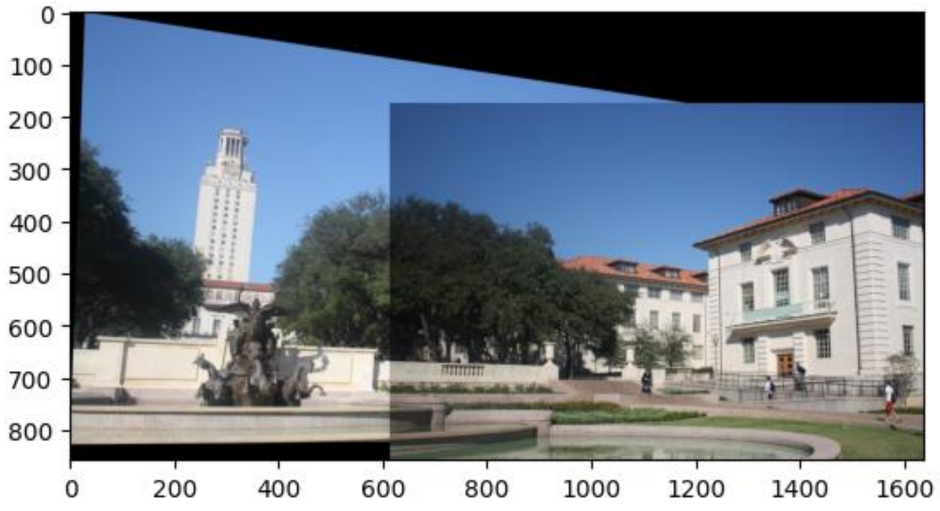

Fig. 13. Stitched Image 3

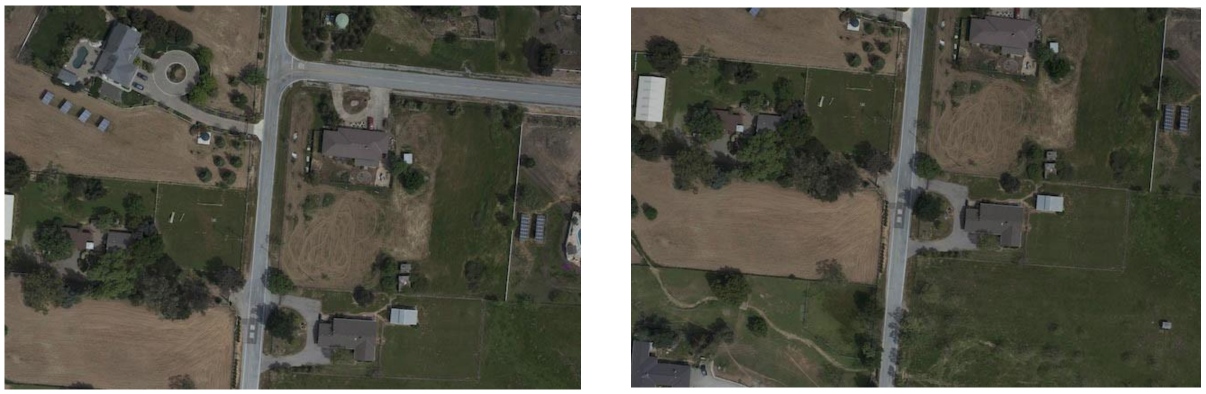
Fig. 14. Input image pair 4

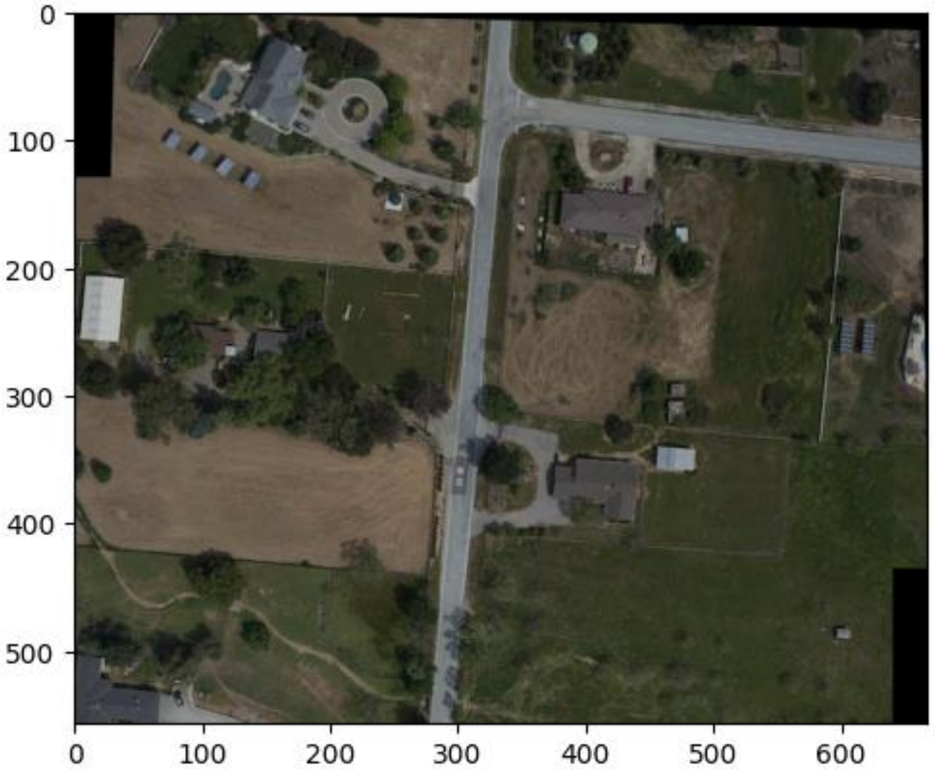

Fig. 15. Stitched Image 4

While the stitching algorithm seems to work well on all the images to produce a cohesive panoramic image, there are obvious seams and difference in contrast seen in the stitched images of Figures 11 and 13. This is due to the fact that the two initial images are of different contrast and brightness because of the difference in illumination. While the exact mechanism is beyond the scope of this project, a solution to this is the use of gain compensation and multi-band blending (Brown & Lowe 2007). Gain compensation adjusts the pixel intensity to compensate for differences in lighting conditions between the two initial images, while multi-band blending helps to create a seamless composite image with reduced artifacts and improved visual quality.

The two functions above can be implemented easily with the `cv2.Stitcher` object. The figure below is the resulting image from stitching the same two images from Figure 12.

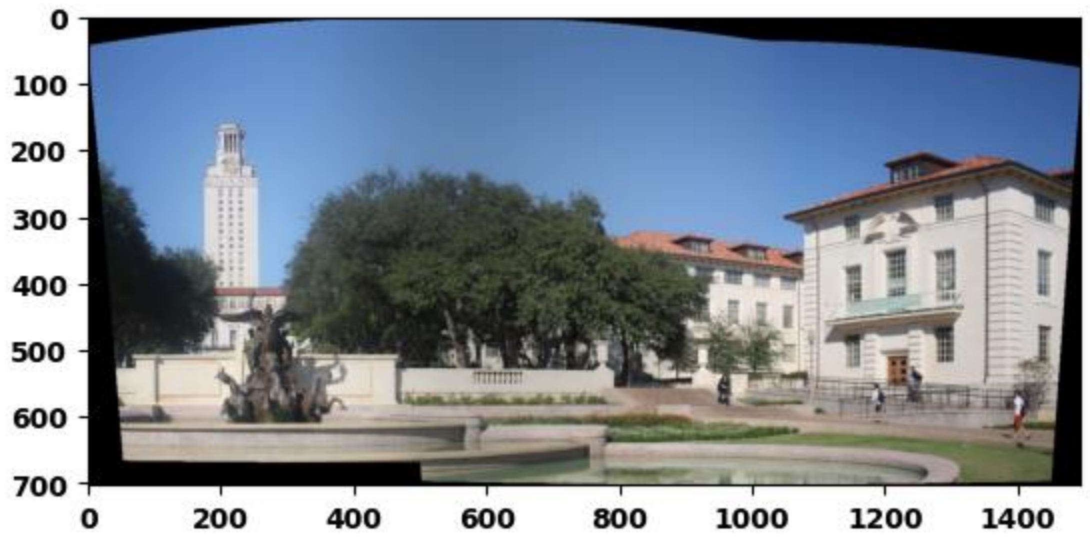

Fig. 16. Stitched image from `cv2.Stitcher`. Multiband blending "removes" the seam while gain compensation ensures the contrast and brightness of the two images are balanced well.

From here, with appropriate cropping, we will get a panoramic image, like the ones we can obtain using our smartphones!

Apart from not accounting for differences in contrast and brightness in the initial images, we will discuss some potential limitations of this technique. The basis of this technique relies on the use of homography; should the method of recovering the homography matrix not perform well, the stitched image may look unsatisfactory. There are two potential causes for this – insufficient overlap and image artefacts.

Insufficient overlap refers to having sufficient number of matching feature points. Recall that to recover the homography matrix, we require at least 4 pairs well-matched feature points from the two images. If there are fewer than 4, the system of equations will be indeterminate, and we will not be able to recover a unique homography matrix. Along the same vein, image artefacts such as noise, lens distortion, and lens flares may result in noisier description for the feature points. This in turn interferes with the matching process and hence the matrix recovery, leading to incorrect homography estimation.

We should note also that our project only looks at the stitching of two distinct images of the same scene. However, if we want to stitch multiple images into a panoramic image, the general principles are still identical:

1. Start with two images with sufficient overlap, perform the steps in the three phases (features detection and matching, recovery of homography, and stitching) outlined above to produce a panoramic image.

2. Repeat step 1, but having the panoramic image obtained from the previous step as one of the input image.

6. Conclusion

In this report, we detailed the theory of using homography matrix used to stitch pairs of images into one panoramic image. Although the images produced are well stitched, there are still some areas for improvements when it comes to smoothing of the seam and balancing of the contrast and brightness. Also, care should be taken to ensure that that a satisfactory homography matrix can be recovered.

7. References

Brown, M., Lowe, D.G. Automatic Panoramic Image Stitching using Invariant Features. *Int J Computer Vision* **74**, 59–73 (2007). https://doi.org/10.1007/s11263-006-0002-3

Gillevicv. (2013, October 4). *A Short introduction to descriptors*. Gil's CV Blog. https://gilscvblog.com/2013/08/18/a-short-introduction-to-descriptors/#more-3

S. Tabbone, "Detecting junctions using properties of the Laplacian of Gaussian detector," Proceedings of 12th International Conference on Pattern Recognition, Jerusalem, Israel, 1994, pp. 52-56 vol.1, doi: 10.1109/ICPR.1994.576225.