Theerut
Amornkasemwong

Automatic image stitching using SIFT and projective transformations

PERSONAL VIEWS

In today's world, it has become clear how integral technology has become. In an effort to automate human tasks and push for a more seamless integration of technology into our everyday lives, computer vision was born in hopes of endowing machines with the ability to mimic the human visual system. Today, computer vision has found its way into countless fields, ranging from the processing of visual stimuli in neurobiology to the development of self-driving cars in the automotive industry.

This course has taught me the fundamental building blocks that spurred the growth of computer vision during its early years. Through the eyes of computer vision's early pioneers, I came to understand the thought processes these individuals went through that led to their eventual discoveries. For example, this course has challenged me to put myself in the shoes of John F. Canny, the man who developed the renowned Canny edge detector, and ponder ways to improve what was considered to be the state-of-the-art method at the time.

With the rise in popularity of augmented and virtual reality, computer vision—specifically the study of 3D tracking and scene reconstruction—is more relevant than ever. It is these applications that have stimulated my interest and are the fields that I hope to be able to continue my studies in.

PROJECT DESCRIPTION

Given my interest in scene reconstruction, I have chosen the to work on the topic of image stitching for this final project. The project involves designing an algorithm whose task is to stitch a series of images seamlessly to form one panoramic image. The images are photos of a scene with overlapping fields of view.

The challenge of this project is not so much because of the stitching process itself, but rather the need for a *seamless* stitching. Since different images may not necessarily share the same viewpoints, merely overlapping them will not guarantee a seamless stitching. Furthermore, the lighting of the two images may not be the same. This implies that some blending algorithm is also needed to seamlessly transition from one image to another.

For example, consider the images in Figure 1 and 2 below. The two images are photos of the same scene, which is a lake nearby some residential area. The scene in Figure 1 lies slightly to the left of that in Figure 2.

Figure 1: Left side of the scene

Figure 2: Right side of the scene

Observe the two beige buildings seen in both figures (on the right side of Figure 1 and left side of Figure 2). Since the two images do not share the same viewpoint, nor do they have the same lighting, simply overlapping the two buildings by overlaying them over each other will not create a seamless stitching. As evident in Figure 3, there is a clear distinctive line that separates the two photos. Furthermore, regions further away from the two buildings, such as the swimming pool, fail to even align properly.

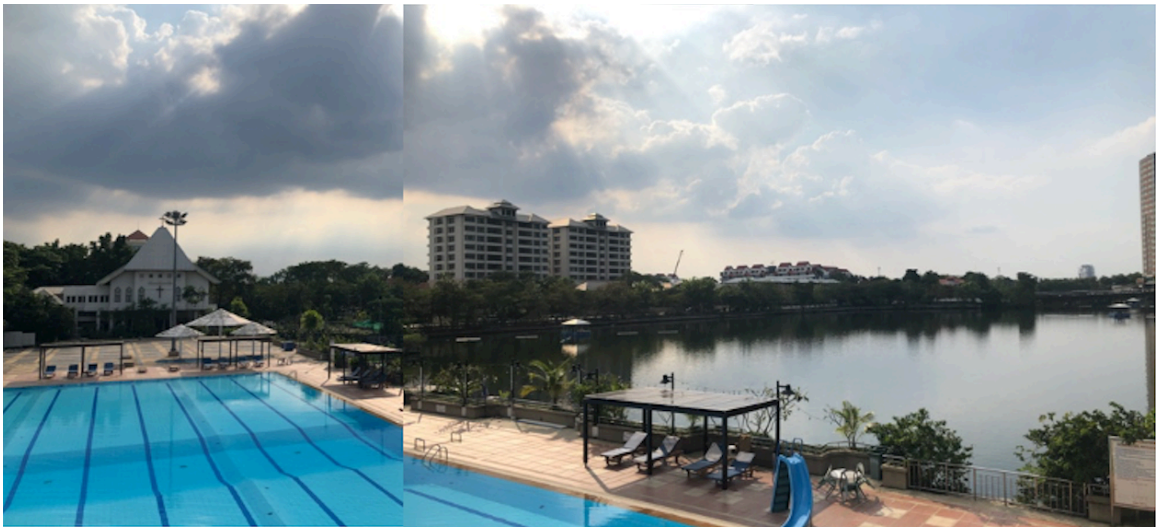

Figure 3: Overlapping of the two images in Figure 1 and Figure

DESIGN

My implementation of the stitching algorithm can be split into five general steps listed below.

1. Keypoints detection
2. Homography Matrix Computation
3. Image Warping
4. Image Blending
5. Image Cropping

For the remainder of this report, the image whose viewpoint we wish to use as the final perspective will be referred to as the reference image. The image whose viewpoint will be warped to match that of the reference image will be referred to as the test image.

Generally, the image stitching algorithm works by transforming the viewpoint of all images to that of the reference image. This is done via projective transformation, which uses an 8-DoF homography matrix that is retrieved by solving a least-squares problem. By applying this matrix to the test, we warp the test image to the perspective of the reference image. This allows the same object in both images to appear identical. These two images are then blended and then cropped to form one panoramic photo. We will delve into the technical details of each step in this section.

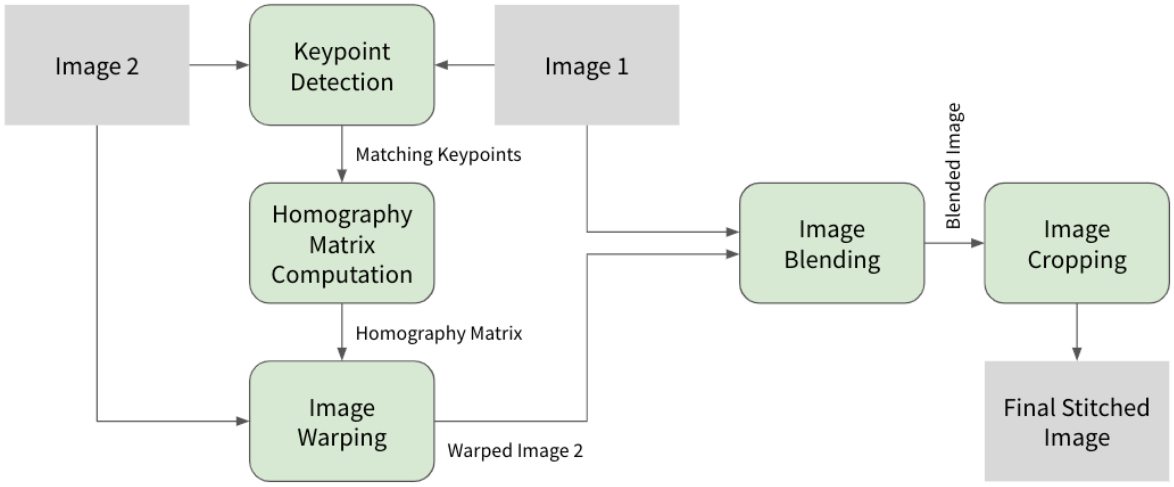

Figure 4: Data pipeline of the stitching algorithm

The data pipeline associated with this stitching algorithm is illustrated in Figure 4 above. The pipeline originates from the blocks *Image 1* and *Image 2* on the top right corner and ends with

the *Final Stitched Image* block on the bottom left corner. *Image 1* is the reference image and *Image 2* is the test image here.

Keypoints Detection

To compute the homography matrix needed for projective transformation, a set of keypoints have to be identified. The specific details as to why these keypoints are needed are elaborated in the next part when we talk about homography matrix computation. For now, we will concern ourselves with the algorithm used to detect these keypoints. The reason we need to automatic keypoints detection is because we wish to automate the entire stitching process—we do not want to have to tell the machine which pair of points in the two images have to line up, but have the machine identify these points automatically on its own.

The main algorithm used for keypoint detection is the scale-invariant feature transform. SIFT is a feature detection algorithm developed by David Lowe in 1999. SIFT works by first extracting a set of points in the reference image. These points are obtained through a process known as scale-space peak selection, which attempts to find potential keypoint candidates via difference of gaussians. However, since scale-space peak selection usually returns too many insignificant keypoints, we apply keypoint localization to filter out less useful ones. These could be points that are along the edges (non-corner points) or points that do not have enough contrast. Next, we perform orientation assignment, which is the determination of a point's orientation. A local gradient orientation histogram is first computed; this is a histogram that counts the frequency of neighboring gradient vectors. The keypoint is then assigned the gradient vector that appears most frequently in the histogram. Finally, we compute the keypoint descriptor, which is

a set of orientation histograms around the keypoint. The idea behind the descriptor is each keypoint can be described by a set of gradient vectors around the keypoint. Pairs of keypoints (the same objects in different images) are expected to share similar neighboring gradient vectors. Given the keypoint's orientation, we can normalize the descriptor to allow it to be compared with descriptors from other images.

Since matching keypoints are expected to have similar feature vectors (as defined by the descriptor), we pair the keypoint in the test image whose descriptor forms the shortest Euclidean distance with the descriptor of the keypoint in the reference image. A ratio test is used to filter out noise. The ratio test works by comparing a keypoint in the reference image with the two best keypoints in the test image. If there is not enough difference between the two best keypoints in the test image, then we may assume that the two keypoints in the test image are merely noise and reject them.

Note that OpenCV's implementation of the SIFT algorithm was used in my image stitching algorithm. The matching keypoints detected when SIFT was used with images in Figure 1 and 2 are shown in Figure 5.

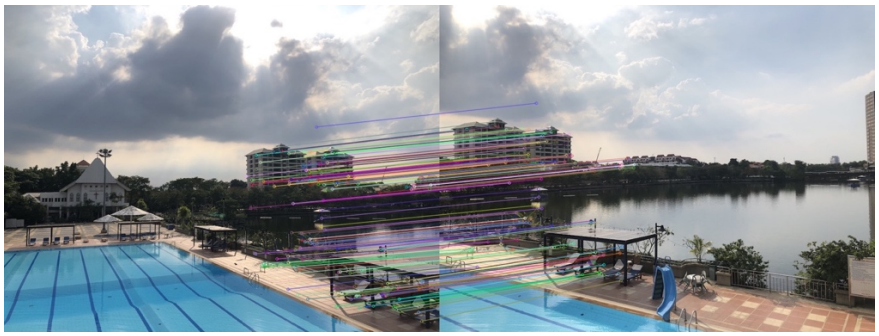

Figure 5: Detected keypoints using the SIFT algorithm

Homography Matrix Computation

Knowing the keypoints, we can now compute the homography matrix needed for projective transformation. The homography matrix is used to map some keypoint (x, y) in the reference image to the corresponding keypoint (x', y') in the test image. More specifically, this function is defined by

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = F(x, y) = \begin{bmatrix} x \\ y \end{bmatrix} + H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \qquad where\ H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}.$$

By expanding and rearranging the expression above, we can obtain the following linear system,

$$\begin{bmatrix} x' - x \\ y' - y \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ \vdots \\ h_{21} \end{bmatrix}.$$

Since the keypoint positions (x, y) and (x', y') are known from the first step, we can use them to solve for $h_{00}, h_{01}, ..., h_{21}$. Given 8 unknowns, we will need at least four pairs of keypoints in order to solve the linear system above via least squares. After the values of the homography matrix are known, we can find any point (x', y') on the test image that corresponds to point (x, y) on the reference image by solving $\begin{bmatrix} x' \\ y' \end{bmatrix} = F(x, y)$.

Image Warping

Now that we have the homography matrix, we can perform projective transformation on the test image. This would change the viewpoint of the test image to match that of the reference image.

Recall that the function $F(x, y)$ transforms images from the reference image's viewpoint to that of the test image. Since we wish to transform the test image's viewpoint to that of the reference image, it would make seem intuitive for one to use the inverse of the homography matrix, $F^{-1}(x', y')$. However, adopting this approach may lead to an unexpected problem: missing pixels. Because there is a possibility that the warped image is larger than the original test image, certain pixel (x, y) in the warped image will be skipped. Thus, to avoid the possibility of skipping pixels, we will iterate through every pixel in the warped image instead. This ensures that every pixel (x, y) in the warped image, no matter how big or small, will be matched to a corresponding pixel (x', y') in the original test image.

However, how does one determine the size of the warped image? To tackle this problem, I computed the location of the warped test image's vertices. By taking the maximum horizontal and vertical pixel positions, I am able to determine the size of the array needed to store the warped image. Note that if any of the four vertices were mapped to a negative valued position, the test and reference images are translated in a manner such that no pixels of the test or reference images have negative-valued positions.

When Figure 1 is treated as the reference image and Figure 2 as the test image, the warped image obtained via this implementation is shown in Figure 6.

Figure 6: Comparison between test image and the warped image

Image Blending

The process of image blending is implemented by using the concept of feathering. At this step, the reference image and warped test image are combined to form one image and the colors are blended to allow the two images to be stitched seamlessly.

The merging of the two images is very simple. Since the test image has been warped, keypoints in the reference and test images are aligned. For non-overlapping regions, we can directly copy the values of the warped test image onto the reference image. For the overlapping region, we have to apply feathering.

The idea behind feathering is to smoothly transition from the color of one image to another by gradually shifting the color via weighted combinations. Imagine moving from the reference image to the warped test image: as we move towards the test image, we want to progressively become more similar to the test image. To do so, we decrease the weight of the

colors in reference image and increase the weight of the colors in the test image. Eventually, the weight of the colors in reference image reduce to 0 at the edge of the overlapping region, and the weight of the colors in the test image is 1.

The resulting image when the reference image in Figure 1 and warped image in Figure 6 and combined and blended is shown in Figure 7. The top image shows the stitched image before feathering is applied; observe the slight discontinuity in colors at the stitching boundaries. The bottom image shows the stitched image after feathering is applied. The stitched image appears much smoother and seamless.

Figure 7: Stitched image before (top) and after blending (bottom)

Image Cropping

The final step is to crop the image. The purpose of this step is solely for presentational purposes. We wish to present the final panoramic image as a rectangular box as it is more appealing. Thus, in this step, we fit the largest the rectangle on the stitched image and then crop out all regions that lie outside this rectangle. The resulting cropped image is shown in Figure 8.

Figure 8: Final seamlessly stitched image

RESULT AND ANALYSIS

To test the algorithm's robustness, it is tested on multiple sets of images. First, we tried switching the reference image and test image from the previous example, making Figure 1 the test image and Figure 2 the reference image. The resulting stitched image is shown in Figure 9 on the next page.

Figure 9: Seamlessly stitched image after switching reference and test images

Images of the Northwestern Campus are also tested with the algorithm. The purpose of this test is to see whether the algorithm is capable of stitching more than two images. Algorithmically, multiple images are stitched by simply performing the 2-image stitching algorithm iteratively. The image neighboring our reference image is chosen as the test image. The two images are stitched, and the resulting stitched image becomes our new reference image. We then pick the next neighboring image as the test image and repeat until all images have been stitched. Stitched photos of different parts of the Northwestern campus are shown in Figure 10, 11, 12 and 13.

Figure 10: (Top) Photos to be stitched. (Bottom) Stitched image

Figure 11: (Top) Photos to be stitched. (Bottom) Stitched image

Figure 12: (Top) Photos to be stitched. (Bottom) Stitched image

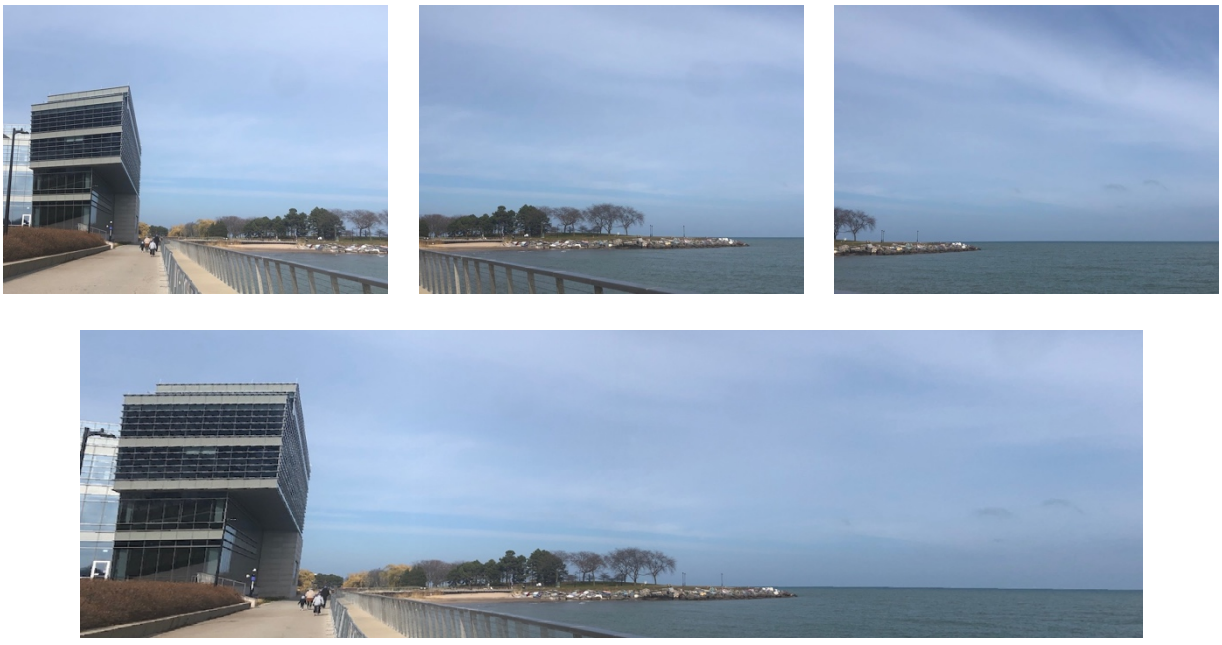

Figure 13: (Top) Photos to be stitched. (Bottom) Stitched image

One limitation of this algorithm is its inability to stitch nearby objects. When we attempt to stitched images of a scene that appears too close to the camera, the matching keypoints fail to align properly. The reason for this lies within the homography matrix used during the projection transformation. One assumption that was not mention with homography matrices is they actually transform one *plane* to another. In other words, the matching keypoints are expected to lie on the same place. In all examples we have seen so far, the images used are photos of scenes taken from a relatively far distance; this allows us to treat most objects in each image as lying on the same plane. With images where objects appear much closer, however, the same approximation cannot be made.

Consider the images in Figure 11 and Figure 12 below. These are close-up images of some street. When the stitching algorithm is applied on these two images (Figure 11 is used as the reference image and Figure 12 as the test image), the final stitched image in Figure 13 is obtained.

Looking at Figure 13, we can see the slight mismatch between the road surface marking, the golf cart, and the tree at the overlapping region. These are reflected through their blurry qualities. The feathering algorithm, nevertheless, is able to greatly improve the seamlessness by making the transition from the reference image to the test image a lot smoother.

Figure 11: Left image of a street

Figure 12: Right image of a street

Figure 13: Poorly stitched image

Another limitation of this algorithm is its great dependence on the accuracy of the homography matrix. When the keypoints used are concentrated over a small region of the image, the homography matrix obtained via least-squares becomes increasingly inaccurate. This leads to a transformation that is able to warp the images only around the keypoints but completely messes up elsewhere. An example of such is shown in Figure 14. Looking at Figure 14, you can see that the top right corner of the test image has been stretched out very far while the overlapping region seems fine.

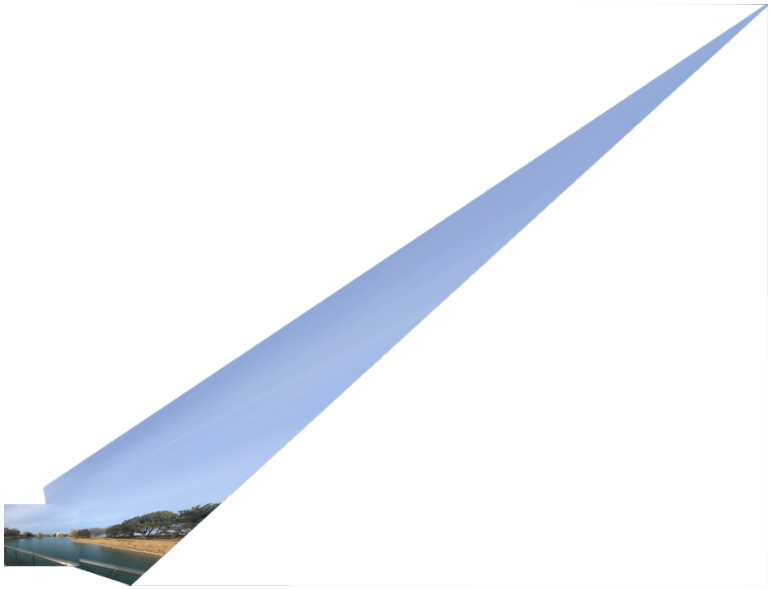

Figure 14: Poor warping of the image due to an inaccurate homography matrix

REMARKS AND FUTURE WORKS

One way we can improve the quality of the resulting panoramic image is by considering the number of matching points found through SIFT. Since SIFT returns numerous pairs of keypoints, but only four of which are actually needed to compute the homography matrix, we can vary the

Euclidean distance threshold based on the number of matched keypoints. If the number of matched keypoints is really high, then we can decrease the threshold to consider fewer, but more accurate pairs of keypoints. For example, if SIFT returns numerous keypoints for some given Euclidean distance, we can reduce this threshold so that feature vectors have to be increasingly similar in order to be considered. Assuming points with almost identical feature vectors are matching keypoints (a relatively safe assumption to make), then reducing the Euclidean distance requirement will filter out keypoints that we are less confident about. This leaves behind just the keypoints that are good matches, and thus would allow us to generate an homography matrix that accurately maps between the reference and test images.

A topic I hope to be able to study more is the different 3D reconstruction techniques and how they could be used in different situations to retrieve the most accurate representation of the actual 3D space.