# VFX Homework 2 - Image Stitching

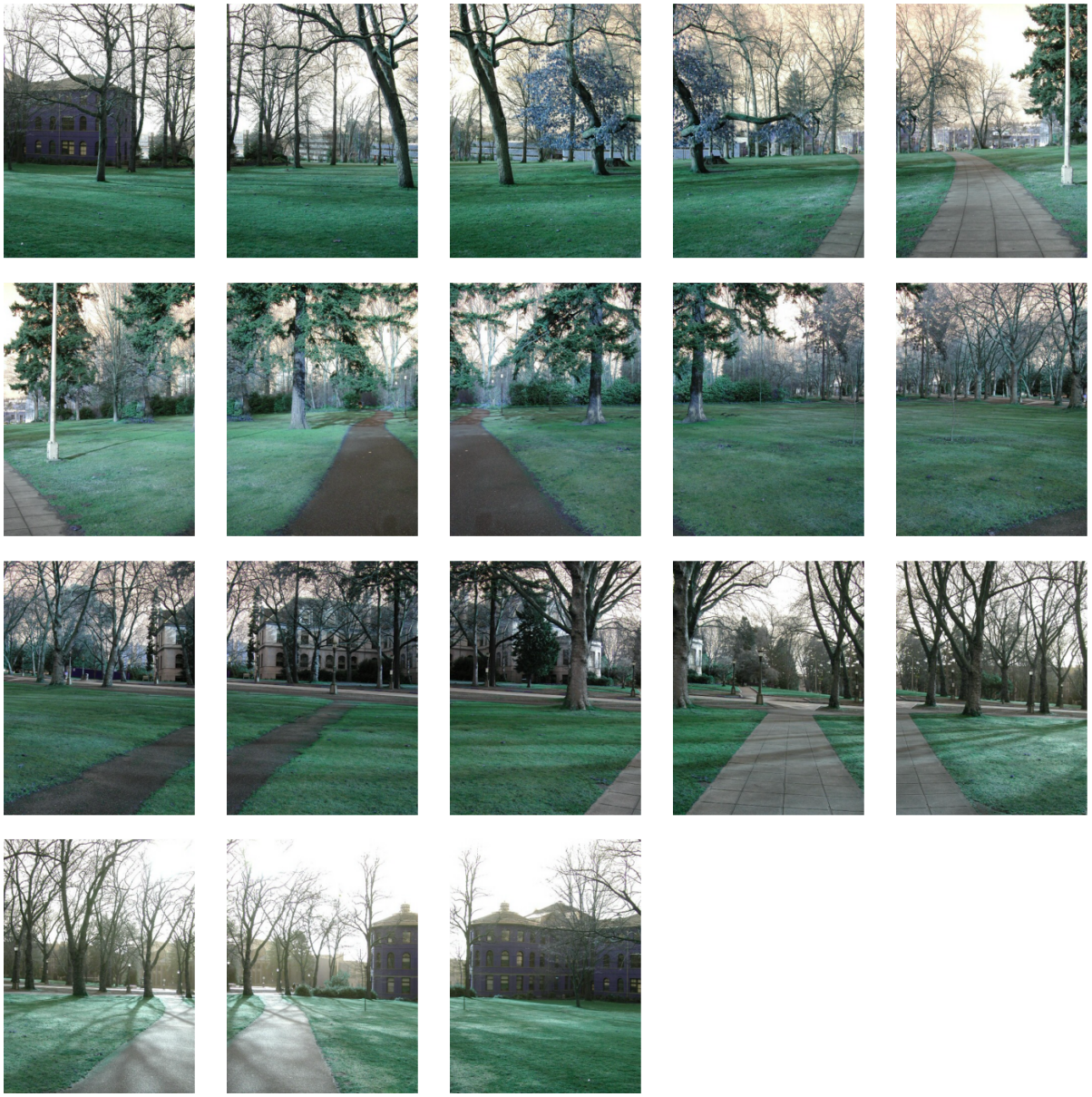
R10942152 游家權

---

## Algorithm Explain:

1. Warp input images to Cylindrical coordinate.
2. Utilize **Harris Corner** to find feature locations and use **SIFT** to describe features.
3. Use a **brutal force** scheme to match features and use the **SIFT matching algorithm** to determine if it's a match or not.
4. Use **RANSAC** to refine matches between two images and calculate dx, dy accordingly.
5. Use dx,dy to stitch all images together, also a **linear image blending algorithm** is applied.
6. To eliminate vertical drifting, I **evenly distribute vertical error** in the whole panorama.

**Input Images(From Example)**

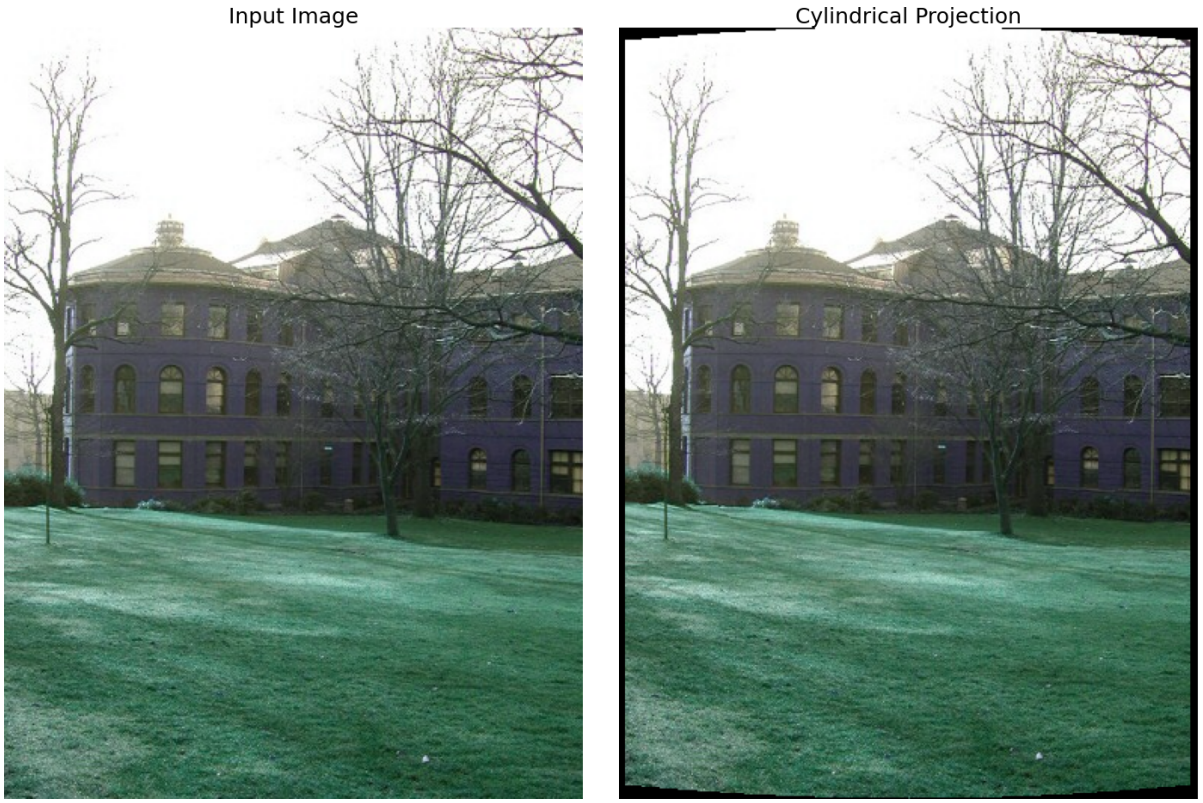


**Step1: Cylindrical Projection**

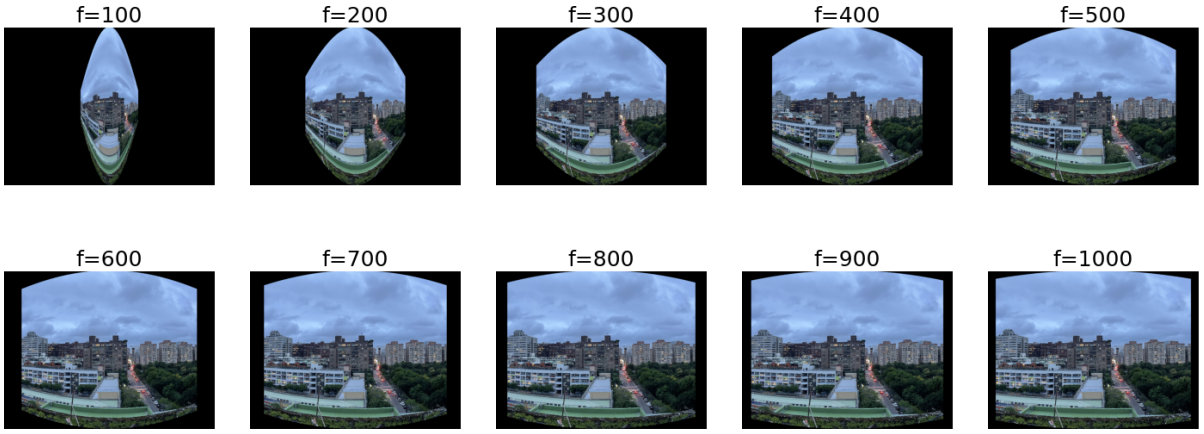
I project every input image to cylindrical coordinates, making two consecutive images stitch together directly without distortion. In this step, focal length needs to be specified. I used the AutoStitch program to measure focal length for each image. Through experiments, I found a shorter focal length would result in a more twisted cylindrical projection. This experiments is shown below.

$$x = x_i - \frac{width}{2}$$
$$y = -y_i + \frac{height}{2}$$
$$x' = ftan^{-1}\frac{x}{f}$$
$$y' = f\frac{y}{\sqrt{x^2+f^2}}, \text{ where f is focal length, width and height is the dimension of input image}$$
$$x_i' = x' + \frac{width}{2}$$
$$y_i' = -y' + \frac{height}{2}$$

An example of cylindrical projection for an image.




Focal length will determine how twisted the cylindrical coordinate are.



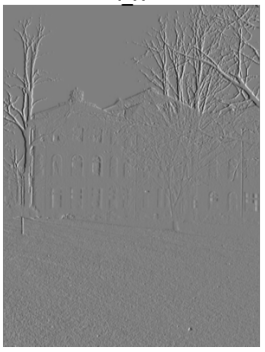

**Step2: Feature Detection - Harris Corner**

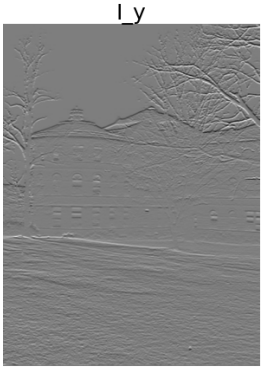
Firstly, Harris Corner algorithm finds gradients in x and y directions and detects corners by calculating the magnitude of these gradients. If gradients are big in both directions, then it'll be considered as a corner/feature. In the "I_x" and "I_y" figure shown below, white part of the images represent the right edges and lower edges in the image. And the dark part represents the left and upper edges in the image.

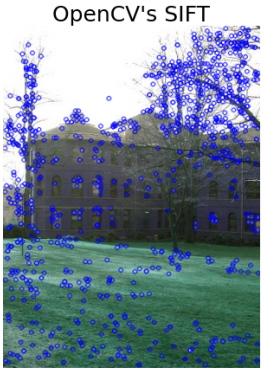
Harris Corner uses this information to evaluate the score of a pixel being a corner. The result shows in the "R(binarized)" figure. We can see that most twigs of the tree and the roof edges in the image are successfully marked as corners due to its gradient.

The last step is non-maximum suppression(NMS). I treat every connected-component in R as an entity and try to find the maximum location to represent this component. The result

shown in the “My Harris Corner” figure. The features detected are very similar to OpenCV’s implemented Harris Corner, which means I got the algorithm right.








**Step3: Feature Description - SIFT description**

In the SIFT feature describing algorithm, it finds major orientation for every feature location, and rotates the patch to right-north, in order to make descriptors invariant to rotation. The results shown below, the major orientation of every feature point is represented by blue arrows.

After rotating the patch to correct direction, SIFT uses the feature's nighboorhood’s gradient orientation to find the descriptor. It uses a orientation histogram in nearby 8*8 patch, and output the histogram as a descriptor vector, resulting in a vector with 128 dimensions.

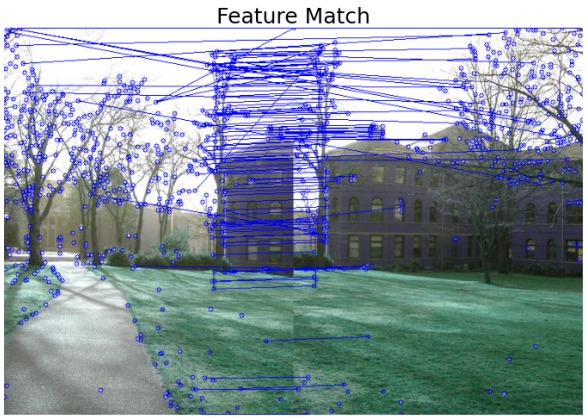

**Step4: Feature Matching**

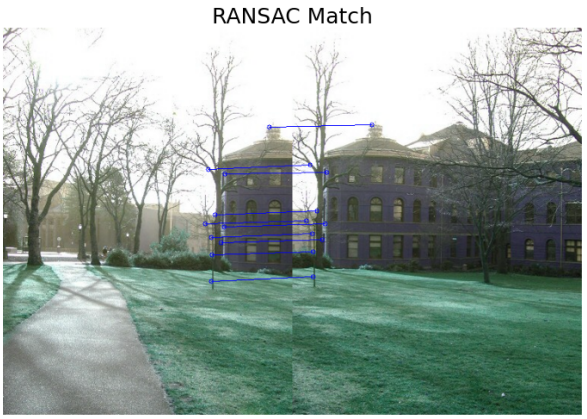
Feature matching algorithm tries to match features between two consecutive images, in hope of finding the exact same feature correspondences. I use a brutal force method to find these relationships; however, matched features are usually not 100% correct. As the “Feature Match” figure shown below, there are some matched features that are actually wrongly matched. Therefore, I use the RANSAC algorithm with 3000 iterations to find 8

matches that have the most inliners. After that, I simply calculate the average dx, dy between these 8 matches to derive the translation between two images.

$P = 1 - (1 - p^n))^k$, where p is probabilty of real inliner, n is number of sample, k is number of iteration

if set p=0.5, n=8, k=3000, then P = 99.9992%




**Step5: Image Stitching and Image Blending**

By applying dx,dy from previous steps, we can easily stitch all images together. As for areas where two images overlap, I use linear image blending to make overlapping areas look smoother. Panorama below shows the results.

**Step6: Fix Vertical Drifting**

The panorama we derived in the last step has a vertical drifting effect due to vertical error created when taking photos. This can be fixed by evenly distributing this vertical error across the whole panorama. By that, the drift would be visually eliminated. The result shown below. The drifting is eliminated and there's no artifact that could be observed with bare eyes.

**My Panorama**

As for my own panorama, I used an Iphone12 and a tripod to take 20 images recording the view of Taipei City. Note that the resolution of original Iphone images is 4032*3024, but it'll take too long to finish this algorithm due to its high resolution. Therefore, I downsampled it 4 times and makes it 1008*756. As a result, the photos have some artifacts in far objects. It's due to downsample steps not from the image stitching algorithm.

## Input Images

## Image Stitching Panorama

## Panorama after fixing drift