[Auto]Stitching Photo Mosaics

IMAGE WARPING and MOSAICING

The goal of this assignment is to explore different aspects of image warping through a practical and exciting application—image mosaicing. In this project, I worked with photographs and created image mosaics by performing several key operations: image registration, projective warping and blending. Along the way, I learned how to compute homographies and used them to warp and align images.

Recover Homographies

Before warping the images into alignment, we need to recover the parameters of the transformation between each pair of images. In our case, the transformation is a homography: $\mathbf{p}' = \mathbf{H}\mathbf{p}$, where $\mathbf{H}$ is a 3×3 matrix with 8 degrees of freedom (lower right corner is a scaling factor and can be set to 1), that is

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

We could convert the equation above into th form of $\mathbf{A}\mathbf{h} = \mathbf{b}$, where $\mathbf{h} = [a, b, c, d, e, f, g, h]^\top$. Then, for each pair of corresponding points, we have

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i x_i' & -y_i x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -x_i y_i' & -y_i y_i' \end{bmatrix} \cdot \mathbf{h} = \begin{bmatrix} x_i' \\ y_i' \end{bmatrix}$$

To solve this, at least 4 pairs of corresponding points should be chosen. However, with only four points, the homography recovery will be very unstable and prone to noise. Therefore more than 4 correspondences should be provided producing an overdetermined system which should be solved using least-squares. Then, we could solve this using least-squares,

$$\mathbf{h} = \arg\min_{\mathbf{h}} \|\mathbf{A}\mathbf{h} - \mathbf{b}\|^2 = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{b}$$

Warp the Images

To ensure that the warped image will not go beyond the canvas, I first add borders with 0, i.e. black borders, to them and modify the position of the corresponding points as well, which should be related to the size of border added. With the homography matrix $\mathbf{H}$, I could warp the source image to match the view direction of the target image.

I choose to perform inverse warping. I compute the corresponding coordinates of pixels in the source image of each pixels in the destination image, which could be calculated according to the homography matrix $\mathbf{H}$. Then, I use `scipy.interpolate.griddata` to perform interpolation.

Image Rectification

Here are two examples of image rectification.

Original Phone **Rectifies Phone** **Original Shoe Box** **Rectified Shoe Box**

Blend the images into a mosaic

To blend the images into a mosaic, I choose the central image as a target for the other images to warp to. Once they are warpped, build alpha mask for them following the logic below:

$$\alpha = \text{logical}(\text{dtrans1} > \text{dtrans2})$$

$$\alpha = \text{blurred}$$

That is, set the part where transformed distance of image 1 is larger as 1, and then blur the mask.

Then I could blend all image together with Laplacian stacks to make a mosaic. I also tried Laplacian pyramids as well, but the results of stack are better. I use alpha mask to blend images and the target image. To blend the bended images together, I just use the trivial mask that divide the image into left and right part since the target is set to center.

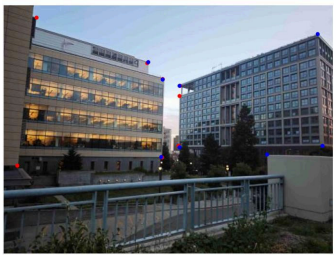
The results are shown below.

Li Ka Shing Center

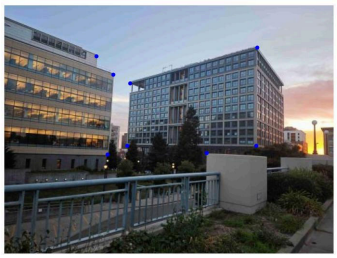
These are images taken near the Li Ka Shing Center.

Left Most Image

Central (Target) Image

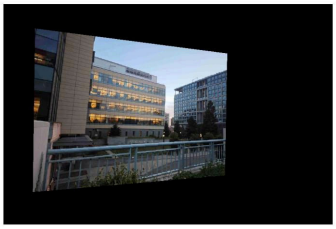


Right Most Image

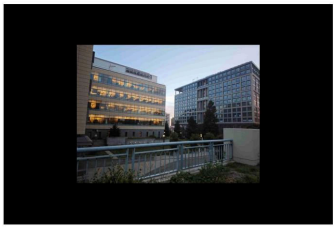


Warp them into the direction of central image:

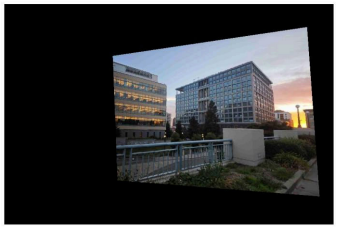
Left Most Image



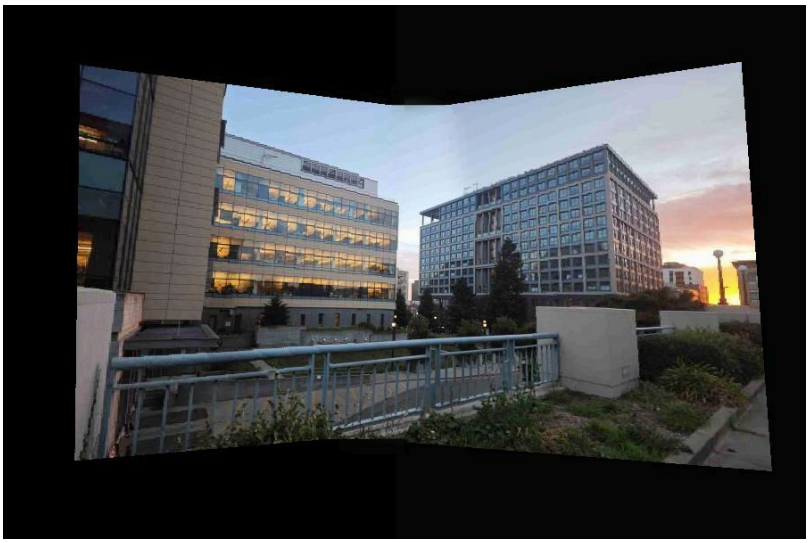
Central (Target) Image



Right Most Image



Blend them together,



Street

These are images of a student housing at Hearst Street.

Left Most Image

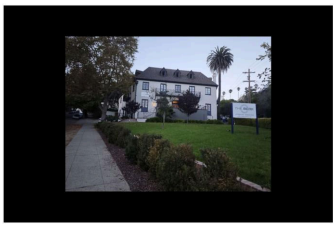
Central (Target) Image

Right Most Image

Warp them into the direction of central image:

Left Most Image

Central (Target) Image



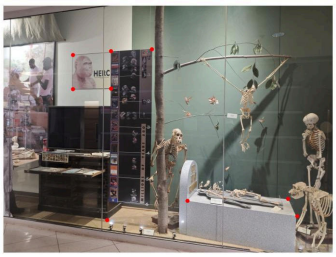
Right Most Image

Blend them together,

Valley Life Sciences Building

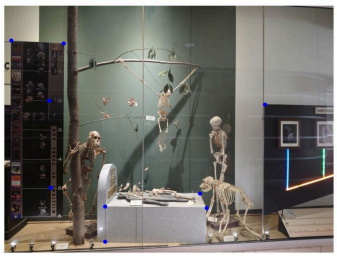
I took these cool images at Valley Life Sciences Building.

Left Most Image



Central (Target) Image

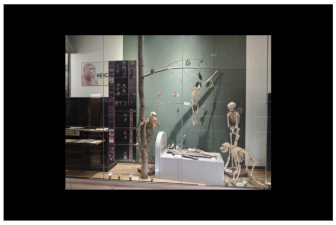
Right Most Image



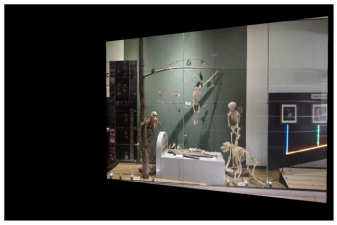
Warp them into the direction of central image:

Left Most Image

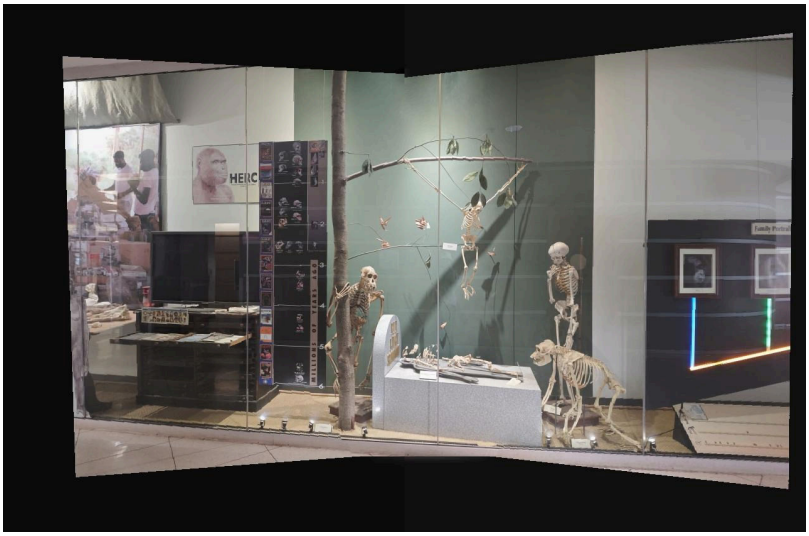
Central (Target) Image



Right Most Image



Blend them together,



Feature Matching and Autostitching

I manually selected the corresponding points in the previous part. In this section, I developed a system to automatically identify matching points between a pair of images and then use these key points to warp and create a mosaic, similar to the previous method. This will consist of the following steps:

- Detecting corner features in an image.
- Extracting a Feature Descriptor for each feature point.
- Matching these feature descriptors between two images.
- Use a robust method (RANSAC) to compute a homography.
- Produce a mosaic.

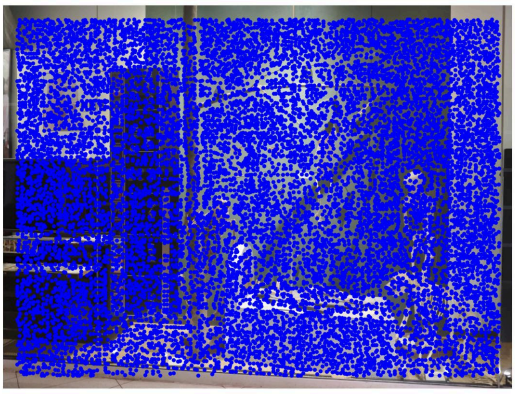
I used the images taken at Valley Life Sciences Building as an example.

Start with Harris Interest Point Detector

For this part, I just used the sample code for implementing Harris corners detector. The `get_harris_corners` function is based on the Harris corner detection algorithm, which identifies corners by analyzing how the intensity values in a local neighborhood vary. The key idea is to detect points in an image where small shifts in any direction result in significant intensity changes.

The Original Image

Harris Corners Overlaid



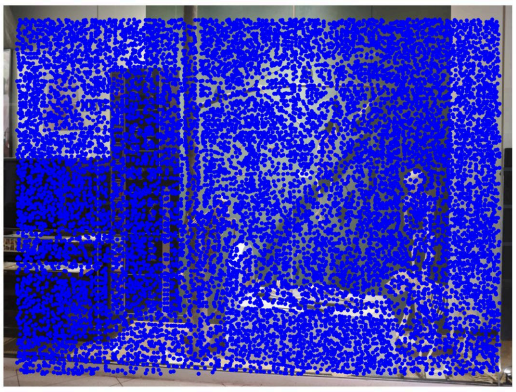
Implement Adaptive Non-Maximal Suppression

Once we have identified the Harris corners, we can apply Adaptive Non-Maximal Suppression to filter the potential keypoints, retaining only those that correspond to strong corners and are approximately evenly distributed across the image. The algorithm assigns every key point an r score as follows, where the function f corresponds to the Harris response:

$$r_i = \min_j |\mathbf{x}_i - \mathbf{x}_j|, \text{s.t.} f(\mathbf{x}_i) < c_{\text{robust}} f(\mathbf{x}_j), \mathbf{x}_j \in \mathcal{I}$$

where $\mathbf{x}_i$ is a 2D interest point image location, and $\mathcal{I}$ is the set of all interest point locations.

Harris Corners Overlaid

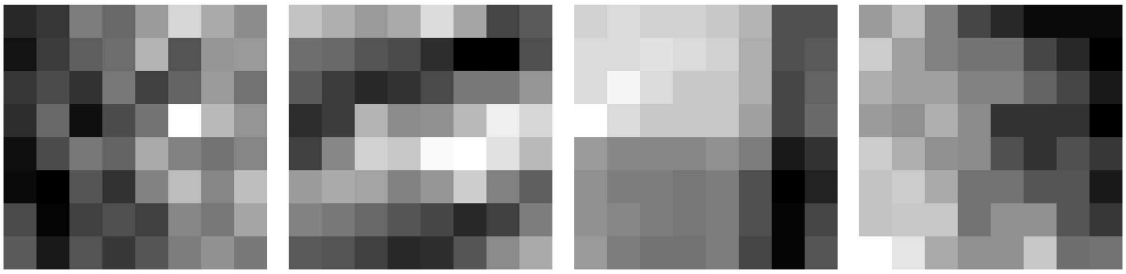


Harris Corners Selected by ANMS Overlaid

Implement Feature Descriptor Extraction

To effectively match feature points across images, it's essential to gather more than just single pixel information. This requires the use of feature descriptors that encapsulate local details about the image, ensuring consistency across various images of the same underlying scene. To achieve this, I extracted a 40x40 window centered on each feature point, which was then downsampled into an 8x8 patch. This process included normalization and mean subtraction to enhance robustness.

Here is an example feature descriptor:



Implement Feature Matching

To match feature descriptors, I employ ***Lowe's trick***. First, I compute the closest (1-NN) and second-closest (2-NN) matches for each feature patch. Then, the ratio between the distances of the 1-NN and 2-NN is calculated. If this ratio falls below a specified threshold, the 1-NN is considered a valid match.

We could observe that the result is not good enough in this example since there are still some bad matches. Thus, consider Random Sample Consensus (RANSAC) for robust homography estimation.

4-point Random Sample Consensus (RANSAC)

Random Sample Consensus (RANSAC) helps minimize the influence of outliers on the resulting model. I followed the steps mentioned in course slides, here is a RANSAC loop:

1. Select four feature pairs (at random).
2. Compute homography H (exact).
3. Compute *inliers* where $dist(p_i', Hp_i) < \varepsilon$.
4. Keep largest set of inliers.
5. Re-compute least-squares H estimate on all of the inliers.

Here is an example of applying RANSAC for automatically stitched results. We could notice that the algorithm successfully removes the poor matches.

Then we could create the mosaic:

Automatically Stitched Result **Manually Stitched Result**

Comparing the manually and automatically stitched results, we could observe that the automatically one performs better on some details, such as the border of the display cabinet.

More Examples

I also considered some interesting samples I did not show you in the previous part.

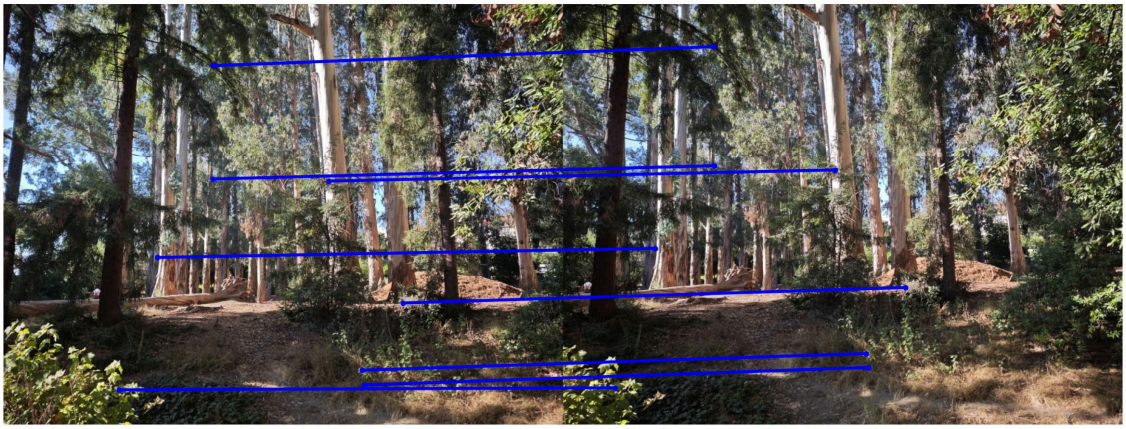
Halloween Decoration

Here are some sonderful Halloween decoration! Both manually and automatically stitched results perform well.

Automatically Stitched Result **Manually Stitched Result**

Forest

This is the most amazing example! For these images of forest, it is difficult for me to manually select the corresponding points. However, the automatical method could easily figure out them even with a tiny threshold!



Once again, both manually and automatically stitched results perform well.

Automatically Stitched Result **Manually Stitched Result**

Cool Stuff I Learnt

My favorite parts in this project were the Image Rectification and Autostitching sections. I was thrilled to see how homography could smoothly transform images with significant distortion back into the correct alignment. Autostitching made it incredibly easy to find corresponding points between two images, which was a great experience for me. At the same time, I applied knowledge from previous projects, like the Laplacian pyramid algorithm, which made me realize that I'm systematically building my understanding of computer vision. Last but not least, I discovered just how interesting and crucial linear algebra is in these applications.