Project 2 Report

- 電機四 B05502012

Implementation

Step 1. Image Warping

- Getting the focal length: I got the focal length of my image with the autostitch app.
- Image warping:
  - For each pixel in the image, compute the coordinate of that pixel on the cylindrical image with the following formula:

$$x' = focal * tan(\frac{x - xc}{focal})$$
$$y' = (y - yc)\frac{\sqrt{(x - xc)^2 + focal^2}}{focal}$$

    - (x, y) is the coordinate on the original image
    - (xc, yc) is the center of the original image
    - (x', y') is the new coordinate
- Implementation detail
  - The canvas to store the warped image is of black background
  - The warped image will be cropped to a rectangle with no black background.

Step 2. Harris Corner Detection

- The warped image is input into the `get_feature_pts()` function to find feature points.
- Steps
  1. Conver the image into grayscale.
  2. Compute the gradient in x and y direction of the grayscale image
  3. Compute matrix M:

$$M = \begin{bmatrix} I_x^2 & I_{xy} \\ I_{xy} & I_y^2 \end{bmatrix}$$

  4. Compute the R score with the following formula:

$$R = detM - k(traceM)^2$$

- I choose the top 2000 points for calculation of descriptors

Step 3. Constructing Descriptors for Each Patch

- Originally, I did compute the orientation for each feature point. However, I found out that the performance is low due to the rotation (We can have holes in the patch if the coordinates do not round to an integer). Therefore, I decided not to rotate the image and compute the descriptors directly from the 16*16 neighborhood of the feature points.
- To compute the descriptor, we have to first convert the image into a grayscale image. Secondly, the grayscale image should be blurred with a Gaussian kernel.

```
blur_kernel = cv2.getGaussianKernel(17, 5, cv2.CV_64F)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
L = cv2.sepFilter2D(gray, -1, blur_kernel, blur_kernel, cv2.BORDER_REFLECT)
```

- I found out that the size of the Gaussian kernel used for the calculation of magnitude and gradient can affect the performance greatly. It is better to use a larger kernel, such as 17*17.

Step 4. Feature Matching

- For each image, there will be two set of descriptors, one set is on the left hand side of the image, the other set is on the right hand side of the image. When two images are being matched, the descriptors on the right hand side of the left image is used to match with the descriptors on the left hand side of the right image. The main purpose for doing this is to reduce the time it takes to compute distance between descriptors. This method also prevents incorrect matching.
- The Euclidean distances between descriptors will be calculated. A dictionary is used to store the matched points between two images:

```
matched_pt[(x1, y1)] = (x2, y2, dist)
```

  - Where `(x1, y1)` is the coordinate of a feature point in the left image and `(x2, y2)` is the coordinate of a feature point in the right image. `dist` is the distance between the two points. The information is stored because we have to update the point that `(x1, y1)` is matched to with the distance. That is, if we find another `(x2, y2)` that gives a smaller distance, we should update the feature point that `(x1, y1)` matches to with this point.
  - Descriptors with distance < 0.07 will be considered as a matching pair.
- The matched pair of points will be the input of RANSAC

Step5. RANSAC

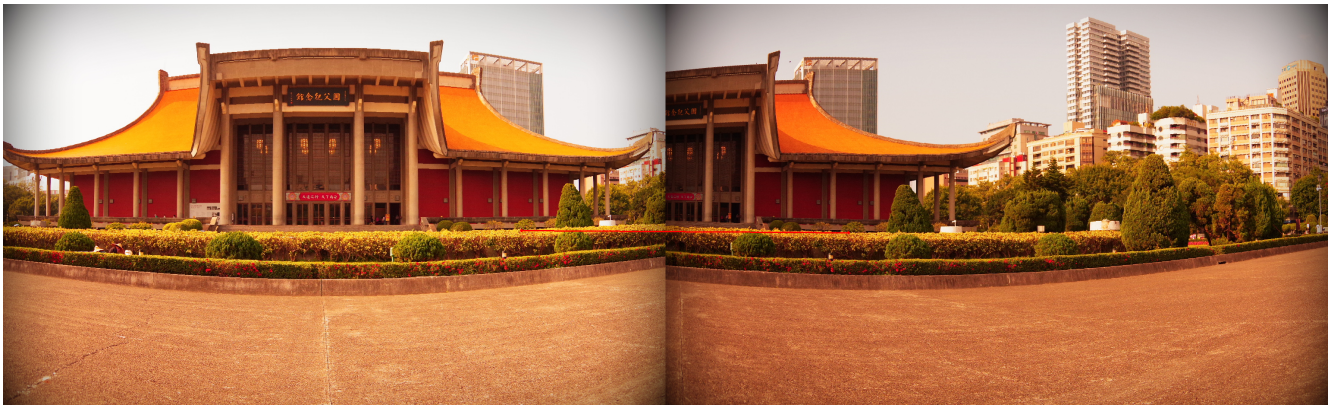
- Parameters for RANSAC

```
n = 4 # number of points sampled per round
P = 0.99999 # the precision we wish to reach
p = 0.5 # probability that a point is an inlier
thresh = 15 # the threshold of distance for a point to be considered as an inlier
k = int(log(1-P)/log(1-p**n)) # number of iterations
```

- Distances between m1 and m2 will be computed for all the points that are not sampled. A dictionary is used to store the number of inliers for a pair of m1, m2. Another dictionary is used to store the inliers of `(m1, m2)`

```
match_cnt[(m1, m2)] = # of inliers
inlier_dict[(m1, m2)] = [(x1, y1), (x2, y2), ...]
```

- The inlier list of the `(m1, m2)` pair with the most inliers will be used to compute the offsets for image stitching.
- After RANSAC, the points that are of the smallest distance will be used to compute the offset of the images for stitching. (The red line two matched points are connected through the red line. It is a little bit hard to see. But it is there...)



- Other findings:
  - The randomize process makes the result of the stitching different. Sometime, the matching is horrible, therefore it is necessary to set the threshold to a lower value. Increasing the number of iterations helps too.

Image stitching

- I used alpha blending for image stitching.
- For the overlaped region of the image on the left. The right side of it will be weighted with a kernel that goes from 1 to 0 linearly. For the overlaped regoin of the image on the right. The left side of it will be weighted with a kernel that goes from 0 to 1 linearly. The two overlaped part will be added together.
  - Ex. The left most image weighted with a weighting kernel that goes from 1 to 0.

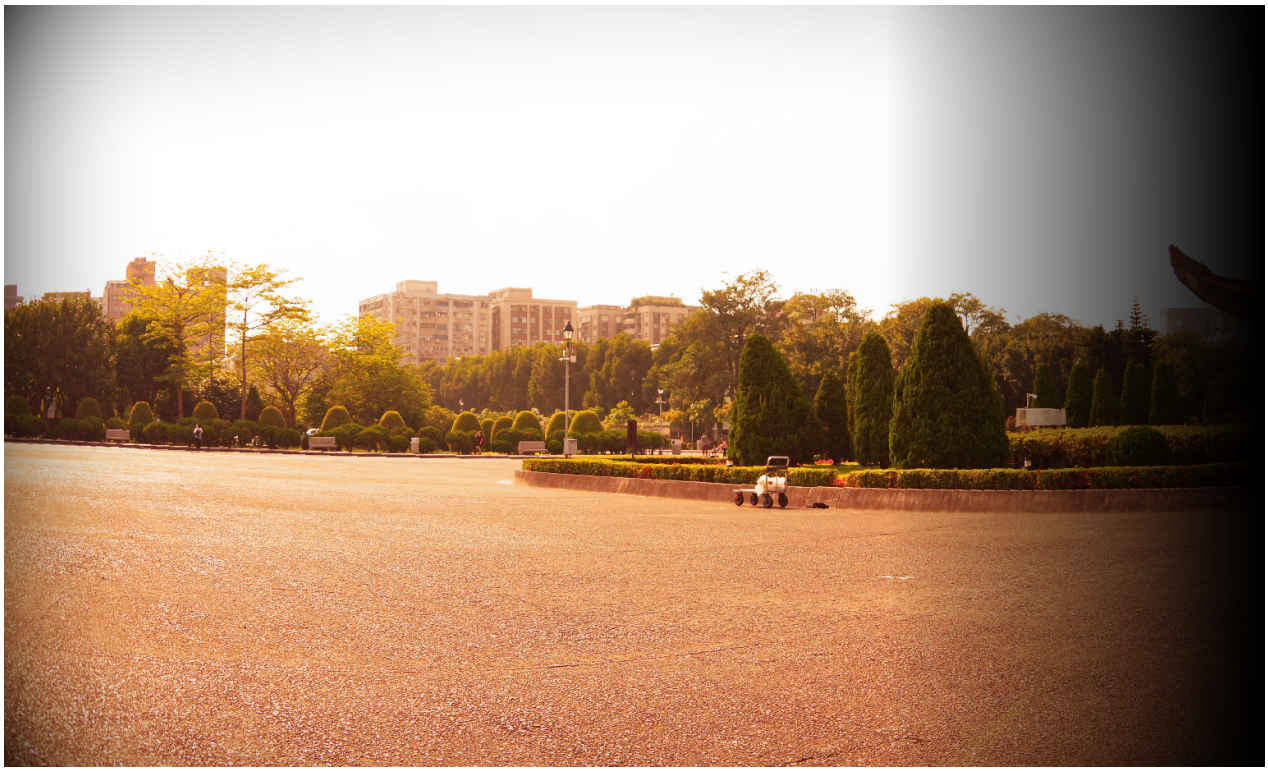


Image cropping

- The black part of the image will be cropped with the largest offset of the height (the y-coordinate where the lowest image in the panorama starts). The bottom part will be cropped with the image's bottom that is the highest in the panorama.

How to run the code

- In the directory:

```
.
+-- code
|   |   utils.py
|   |   main.py
|
+-- data
    |   metadata.txt
    |   01.jpg ~
```

  - Note the images are labeled from left to right. i.e, 01.png is the leftmost image.
- The `metadata.txt` file is in the images folder with the rest of the images. It contains information such as the focal lenght and the file name of the images. In `metadata.txt`:

```
../images
988.23
01.jpg
02.jpg
...
```

    - The first line is the path to the image
    - The second line is the focal length that I got from autostitch
    - The rest of the lines are the name of the images in path `images/`
- To run the code, run the following command in `code/:

```
$ python3 main.py metadata.txt $output_filename
```

    - `output_filename` is the file name of the resulting image, remember to specify the file extensions (`.png`, or `.jpg`)

Result