HW3: Automatic Panoramic Image Stitching

tags: `computer vision`

資科工碩　309551178　秦紫頤
電資學士班　610021　鄭伯俞
清華　H092505　翁紹育

Introduction

Image stitching or photo stitching is the process of combining multiple photographic images with overlapping fields of view to produce a segmented panorama or high-resolution image. Commonly perform through the use of computer software, most approaches to image stitching require nearly exact overlaps between images and identical exposures to produce seamless results. However, some stitching algorithms benefit from differently exposed images by doing high-dynamic-range imaging in regions of overlap. Some digital cameras can stitch their photos too! In this homework, we implement a simple panorama stitching using the computer vision techniques we have been taught in class.

Implement Procedure

Crop the image

Because some image borders may affect the image after stitching, like white or black border, we decide to crop the four borders of the image. We first get the original height and width of the image. Then we choose the percentage to cut the x% of the width and height. In the implementation, we choose 3% as the cutting percentage.

Detect interest points

We use opencv to detect our interest points. First, we define feature type to **sift** by `cv2.SIFT_create()`. Then we detect our interest points and the sift descriptors by `detectAndCompute()`

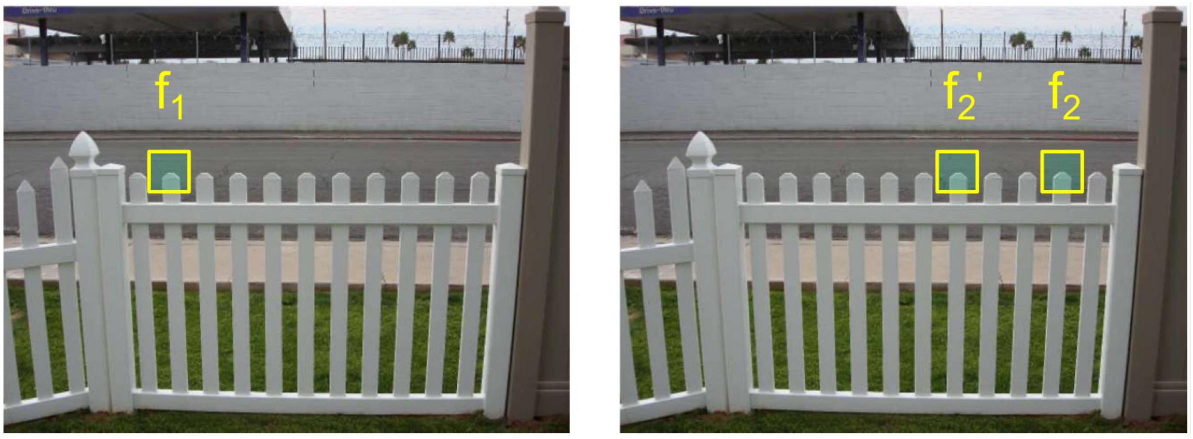
Feature matching

Get the initial matches

First, we calculate the euclidean distance between all the features in the first image and the second image. Then, for each feature descriptor in the first image, we choose the top two matching feature descriptors in the second image (by min distance). Then we save the distance, image1 feature descriptor index, and image2 feature descriptor index in the data structure we create.

Get the good matches

We use a ratio test to decide suitable matches. For all the matches (original each image1 feature descriptors matches two feature descriptors in the second image), we need to decide whether they are suitable matches by distance; that is, the distance of the first match must be smaller than the second match times a threshold. We need to know whether the first match is significant enough (there is only one match on both images). The figure below explains the reason why we need this ratio test.



In the figure, the top two matches for f1 are f2 and f2'. However, they all look too similar, so we will decide to throw away this feature.

Build the correspondence list and draw matching

Since we do not need descriptors for the rest of the homework, we decide to save only the matching points. ex:[x1,y1,x2,y2] (this is the correspondence list, (x1,y1) is the coordinate of the feature points in image1 and (x2,y2) is the coordinate of the feature points in image2). Before moving on to the next step, we draw the feature matching results. First, we concatenate the two images so that they lie side by side. Then, we draw a circle on the corresponding feature points of image1 and image2 and draw a line between them to show that they match. For each corresponding feature, we use the same color to differentiate from the other points. We choose the color randomly for each correspondence.

Apply RANSAC to find the homography matrix

Every four correspondence can find a homography matrix, so we iterate 1000 times and each time randomly choose four correspondence to calculate the homography.

Find homography matrix

$$PH = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x_1' & y_1x_1' & x_1' \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y_1' & y_1y_1' & y_1' \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x_2' & y_2x_2' & x_2' \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2y_2' & y_2y_2' & y_2' \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3x_3' & y_3x_3' & x_3' \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3y_3' & y_3y_3' & y_3' \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x_4' & y_4x_4' & x_4' \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4y_4' & y_4y_4' & y_4' \end{bmatrix} \begin{bmatrix} h1 \\ h2 \\ h3 \\ h4 \\ h5 \\ h6 \\ h7 \\ h8 \\ h9 \end{bmatrix} = 0\$$$

The formula above is the way we solve the homography matrix h1,h2,...h9. And (xi',yi') is our (x2,y2) .We use SVD to solve the null space of the formula above. The smallest eigenvalue's corresponding eigenvector is our homography. After that, we reshape $h_1...h_9$ to a 3×3 matrix.

Count inliers

Every time we got a homography matrix, we will need to evaluate this homography matrix. We do this by using every correspondence point and get the transformation points from image1 to image2 by homography matrix. We calculate the distance between $(x*{1transform},y*{1transform})$ and x_2,y_2. The distance we use here is the L1 distance (take the sum of the absolute difference). If the distance is smaller than 1, then we count it as inlier. The homography that has the most inliers is recognized as the best homography.

Image warping

In this part, we will move both image1 and image2 to the output image. We will first move image2 to the output image. Then, we will get the inverse of the best homography matrix, and this is because we want to use each pixel of the output image and transform it back to image1's coordinate. The above procedure ensures that all the output image pixels have a corresponding value.

Decide the output image size

Because the output image will be slightly larger than the original two images, we decide to first get the size of the output image at the convenience of the rest of the procedures. We decide to subject to image2, so the coordinates of image1 will be different. Since the extreme values occurs in the 4 corners, so we first identify the four corner of image1 ($(0,0,1),(0,h,1),(w,0,1),(w,h,1)$) Then, we use the best homography matrix to transform it to image2 coordinates. Then we will get the min of x, max of x, min of y, and max of y. These will be the new four corners of the output image. Also, we will need to define the offset of x and offset of y, and this is how much we need to shift the original image so that they are aligned to the output image coordinates. The offset is the $-min_value$. Also the output image size will be $width=max_x+offset_x, height=max_y+offset_y$.

Move image2 to output image

We first move image2 to the output image because this is relatively simple compare to image1. We only need to shift the original pixels in image2 by offset_x and offset_y.

Move image1 to output image

We do this in a backward way. For every pixel in the output image, we find the corresponding pixel in image1 and paste that to the output image. First, we get the inverse of the homography matrix because the homography matrix transforms image1 to image2, but we want to do it in a reverse manner. Next, we shift that by (-offset_x,-offset_y) for every pixel in the output image because we need first to transform the point to image2 coordinate. Then, we will get the corresponding point in image1 by multiplying with the inverse homography matrix we compute beforehand. The result should divide by the last value of the output vector because that is the scaling vector; it should be one at last.

Bilinear interpolation

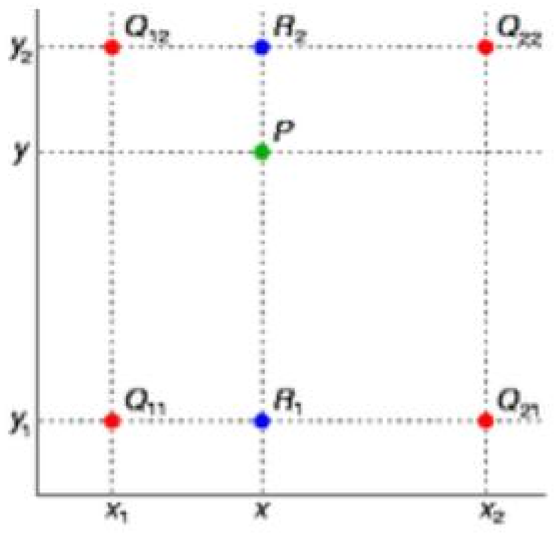
The point we get has two scenarios: either exist in the image1 or outside the image1 boundary (negative or >h or >w). Even if it is a valid point, it may not be an integer, so we will need to interpolate to get the correct pixel color. We choose bilinear interpolation because it is accurate. If the point is on edge, then we use linear interpolation.
On the horizontal edge:

$$\frac{rgb(x_2,y)(x-x_1)+rgb(x_1,y)}{x_2-x}$$

On the vertical edge:

$$\frac{rgb(x,y_2)(y-y_1)+rgb(x,y_1)(y_2-y)}{y_2-y_1}$$

If the point is inside image1 then we will use bilinear interpolation to get the pixel color.



$$f(x,y) \approx \frac{f(Q_{11})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y_2-y) + \frac{f(Q_{21})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y_2-y) + \frac{f(Q_{12})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y-y_1) + \frac{f(Q_{22})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y-y_1).$$

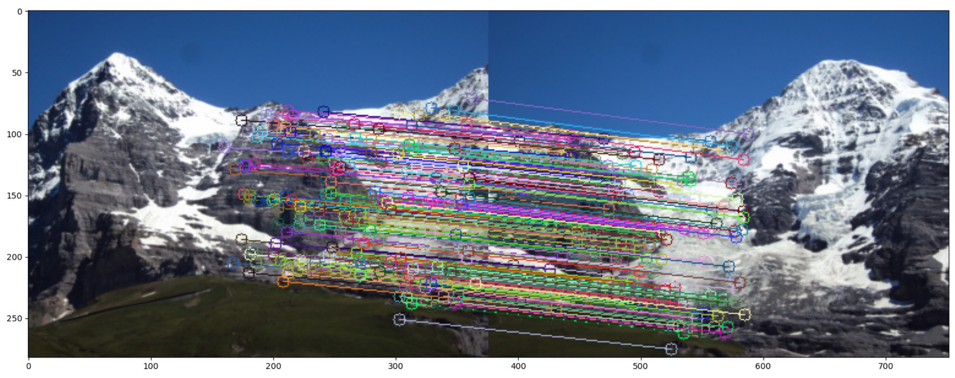
Image morphing (optional)

Here we are trying to blend the two images correctly in the overlapping part, so we decided to do image morphing. Since all our image is warping horizontally, so we use only the x-axis of the point. If the x-axis is closer to image2, then the ratio of image1 color will be higher, vice versa. We regard this step as an optional step because sometimes the morphing effect is not significant. If the output image color looks weird, we will ignore this step. We will cover the pixel that has image1 onto the output image.

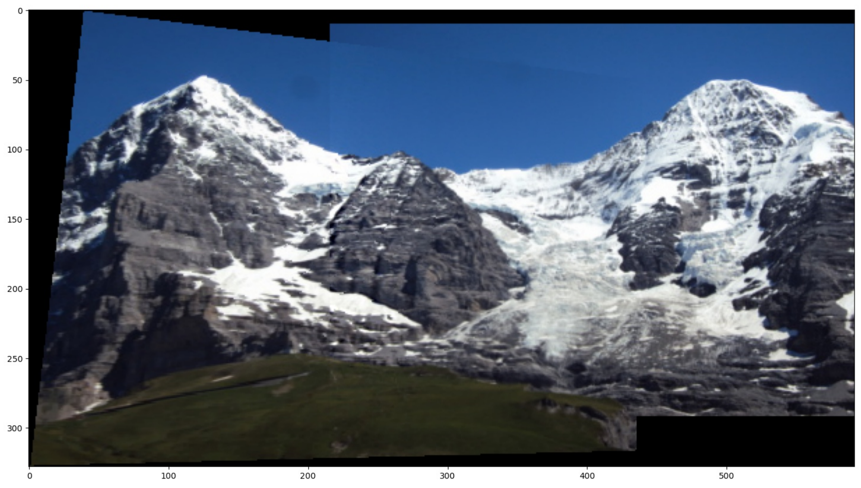
Output the image

Since we are using matplotlib to show and save our output image, we will need to post-processing. There are only two steps. First, since our output image pixel values are all between [0,255], we will need to convert to [0,1], so we divide the pixel values by 255. Second, we use OpenCV to open our image, and OpenCV is in BGR form, we need to convert the image to RGB by `cv2.cvtColor(output_image,cv2.COLOR_BGR2RGB)`.

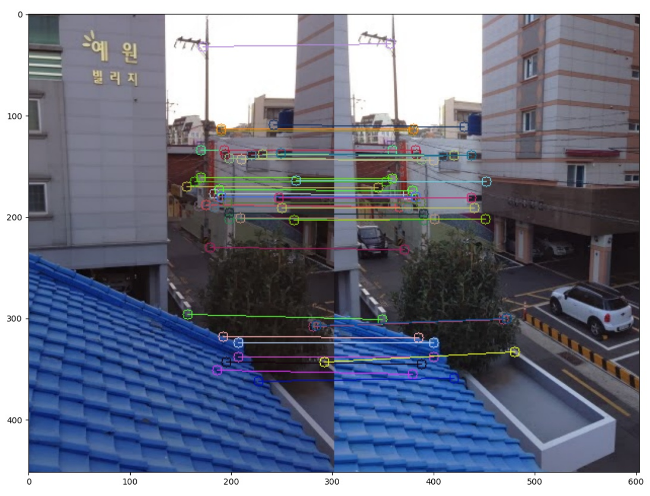
Experimental Results



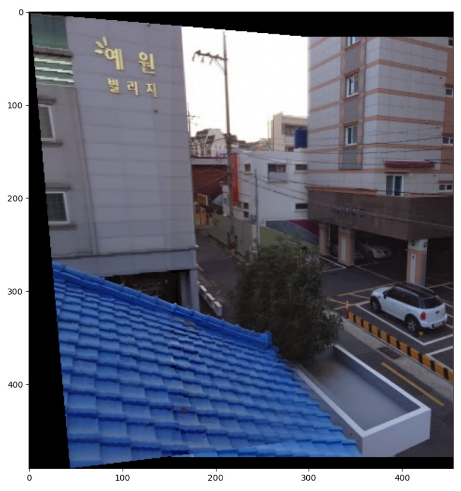
0
50
100
150
200
250
0
100
200
300
400
500
600
700



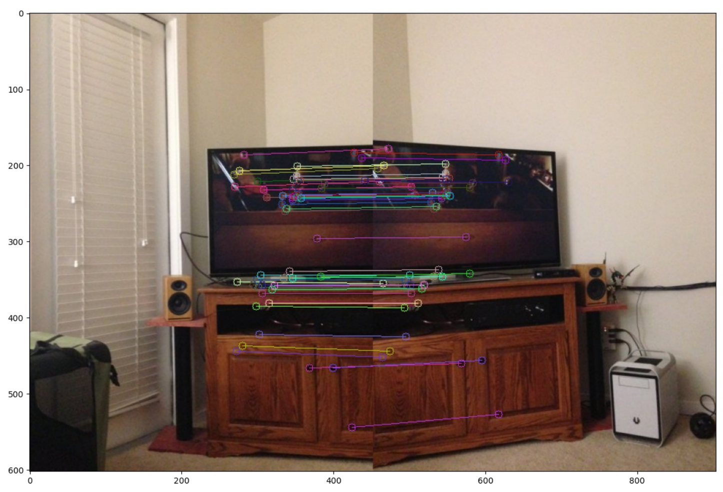
0
50
100
150
200
250
300
0
100
200
300
400
500



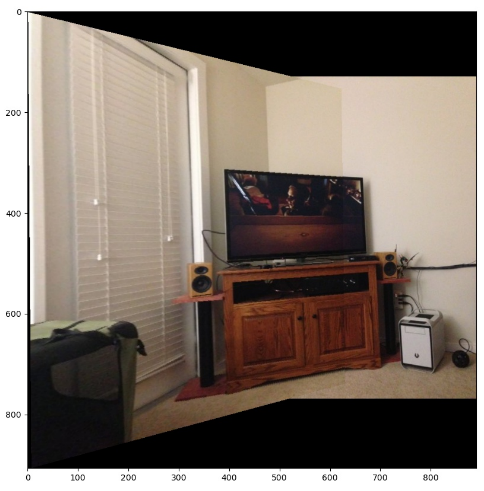
예원
빌리지
0
100
200
300
400
0
100
200
300
400
500
600



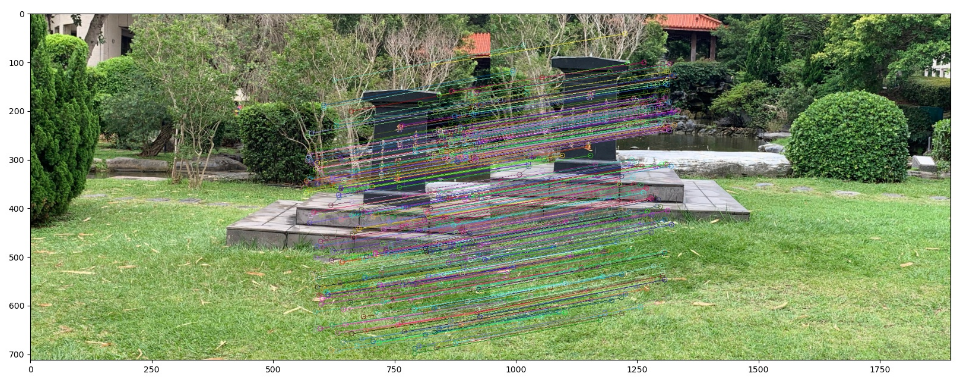
예원
빌리지
0
100
200
300
400
0
100
200
300
400

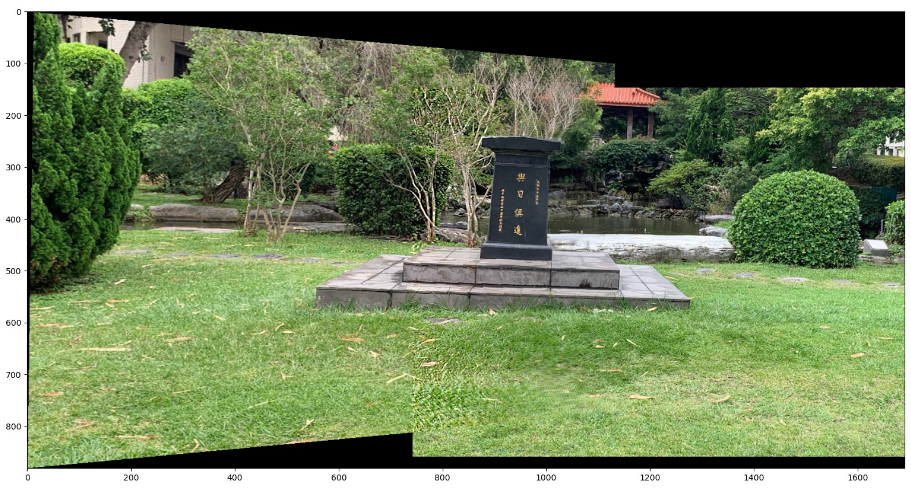


0
100
200
300
400
500
600
0
200
400
600
800



0
200
400
600
800
0
100
200
300
400
500
600
700
800





Discussion

There are two things we want to mention here. First, the distance threshold in the feature matching part significantly affects the output image. If the threshold is setting too high, we may have some false matching, and it may find a lousy homography matrix due to the noise. However, if the threshold is setting too low, then there may have too few feature matches, then we do not have enough permutations to find the homography matrix; the output still might come out bad. As for how to choose the threshold, it depends on each image. We are using try and error to test out different distance thresholds to get the best output image, and we find out that every image is different, but the threshold will be in the range [0.2,0.6]. The second thing is a problem we find out in the image morphing step. If the curvature between the two images is large, then if we do image morphing, the pixel color will be brighter than usual; this is not a good output image, so we will need to ignore morphing. However, if the curvature of the two images is slight, then image morphing did help the output image to look more natural.

Conclusion

In this homework, we make a panoramic stitching program without any software. Initially, we think this is a simple procedure, but we find out we need to take care of many little minor things when implementing it. If the math calculation of bilinear interpolation is wrong by only a one-pixel value, then the output image color will be wrong. Furthermore, if we forgot to handle the offset between the output image and the input image, the point will be in the wrong position. There is much more. We learn a lot from this homework.

Work Assignment Plan

秦紫頤: Code, Data, Report
翁紹育: Code
鄭伯俞: