EE5175 Lab 3

Subhankar Chakraborty

February 2021

1 Introduction

In this assignment we implement image mosaicing. Image mosaicing is the alignment and stitching of a collection of images having overlapping regions into a single image. We have been given three images which were captured by panning the scene left to right. These images (`img1.png`, `img2.png`, and `img3.png`) capture overlapping regions of the same scene from different viewpoints. The task is to determine the homographies between these images and stitch them into a single image. Apart from this, we also implement image mosaicing for three images captured by us. The images given and those clicked by myself can be seen in Figures 1-3. Please keep in mind that the displayed sizes of the images is not representative. I had to make the captured images smaller so that all three for each set could fit into one page while maintaining the same aspect ratio. Derivation of the equations and pseudo codes already present in the problem statement PDF are skipped for the sake of brevity.

2 Implementation and Running the Code

The code has been implemented and tested to work on MATLABR2020b. Each function has a file of its own. The images and support functions are assumed to be present in the same directory as the code. The files `main.m`, `main_own_images_1.m`, `main_own_images_2.m` should be run to generate the mosaiced images from the given images, the first set of captured images, and the second set of captured images respectively. To run them, open a MATLAB window and navigate to the directory containing those three files. Run them from the MATLAB command window or open the respective files and press f5.

2.1 Description of the support functions

A brief description of the support function is provided in this section.

2.1.1 A_matrix.m

- Inputs : n Points and their corresponding points

- Output : The A matrix of size 2nx9 as described in section 2.2 of the problem statement

In our case as we are using RANSAC, we always call this function with 4 point correspondences. Hence the A matrix returned is always of the shape 8x9.

2.1.2 bilinear_interpolate.m

- Inputs : source image zero padded with one array of zeroes on each side, xs, ys. xs and ys are the source coordinates for which the value is needed while doing target to source mapping. xs and ys need not be integer
- Output : The estimated value at xs, ys using bilinear interpolation

If the xs, ys values lie outside of the original image coordinates, we just return a zero. This has the effect of showing a black patch of pixels in the final image if a bunch of coordinates are requested which lie outside the canvas of the source image.

2.1.3 bilinear_interpolate_modified.m

The inputs are essentially the same as the previous function. There is just one extra output, a within_canvas binary variable which is set to 1 if xs, ys are within the canvas of the source image and 0 if not. This is useful when we blend images during mosaicing. We perform averaging for only those values whose coordinates lie within the canvas of their original image as mentioned in point 4 of section 2 of the problem statement.

2.1.4 compute_homography.m

- Input : A matrix
- Output : The homography matrix H

In this function we calculate the homography matrix, H, from the A matrix. We know that we can construct the A matrix from the point correspondences as mentioned in section 2.2 of the problem statement. Once we have A, we can solve for $\bar{h}$ as done in class. A can be decomposed as given below using SVD.

$$\mathrm{A} = U\Sigma V^T$$

U, Σ, V^T have the standard definitions as used in SVD. We know from our discussion in class that the solution for $\bar{h}$ is the vector corresponding to the smallest singular value of A. From the decomposition mentioned above, this is the last row of V^T when the singular values are in decreasing order. We then arrange the values appropriately to get the H matrix.

2.1.5 corresponding_points_2D.m

- Inputs : H matrix, vector X which is [x; y; 1]
- Outputs : [x_, y_], the non homogenous coordinates corresponding to X under the matrix H

This file is essentially the implementation of calculation of corresponding coordinates for a point when the homography matrix is H. In case the corresponding coordinates in the homogenous notation have the last coordinates as zero, [x_, y_] are set to [0, 0] as such a point does NOT have a non homogenous representation.

2.1.6 corresponding_points_2D_multiple.m

- Inputs : H matrix, vector points which is of the shape (n,2) with points(i,:) = [x(i), y(i)]
- Outputs : vector corresponding_points which is of the shape (n,2) with corresponding_points(i,:) = [x_prime(i), y_prime(i)], the point corresponding to [x(i), y(i)] under homography matrix H.

This is a vectorised implementation to calculate the corresponding points for multiple points without a for loop. This helps with calculating the consensus set during RANSAC efficiently.

2.1.7 n_rows_sampler.m

- Inputs : points, corresponding_points, n_rows where n_rows is the number of point correspondences needed to be sampled randomly.
- Outputs : n_points, n_corresponding_points, remaining_points, remaining_corresponding_points

This function helps in getting 4 point correspondences by sampling without replacement from the set of all points and their corresponding points. The remaining points and corresponding points are also returned to make the calculation of the consensus set easier.

2.1.8 ransac_self.m

The inputs are

- points
- corresponding_points
- epsilon : parameter mentioned in point 2 of section 2.1

- fraction : if the consensus set is larger than fraction*(number of point correspondences), for a matrix H, return this as the homography matrix H
- n_samples : Number of samples for RANSAC. Since we are computing a homograpy matrix, this parameter is 4 always.
- n_iters : The max number of iterations to run RANSAC for if a good enough consensus set cannot be reached.

The output is H, the estimate of homography matrix between the two sets of correspondence points.

2.1.9 saving.m

A helper function to read RGB jpeg images, convert them to grayscale, resize them such that the width is less that 1000 and save it as a PNG image.

3 Results

The results of mosaicing for each of the three sets of images are presented in figures 4,5, and 6.

4 Observations and a few other points

The following points are to be noted.

- For the size of the canvas while choosing the mosaic, we choose a canvas of size (3x, 3y) and set an offset of (x, y) where each of the images is of size (x, y) . This is a very liberal allocation of the canvas size but it ensures that the mosaiced image always lies within the canvas. The canvas is finally cropped empirically to remove the black, boundary pixels.
- There is very less parallax in the mosaiced image of the images provided to us. Given that the scene is not froto-parallel, it is an indicator of the fact that the camera motion was close to pure rotation about the camera center.
- There is parallax present in both sets of images captured by me. This is much more pronounced in the second set of images over the first set of images. This is probably due to the fact that in the first set of images, the objects are much farther and we can assume them to be approximately fronto-parallel. This is not the case however in the second set of images, in which the objects are much closer and the parallax is much more pronounced.

- Another phenomenon I observed but cannot explain is that the order of choosing correspondance points affects the final results a lot. `[corresp1, corresp2] = sift_corresp('img1.png','img2.png')` gives markedly better results than `[corresp2, corresp1] = sift_corresp('img2.png','img1.png')`, especially for the set of images provided. This is probably due to how SIFT works.

(a) Image 1

(b) Image 2

(c) Image 3

Figure 1: Given Images

(a) Image 1

(b) Image 2

(c) Image 3

Figure 2: Captured Images : Set 1

(a) Image 1

(b) Image 2

(c) Image 3

Figure 3: Captured Images : Set 2

Figure 4: Mosaicing on the given images

Figure 5: Mosaicing on the first set of photos clicked

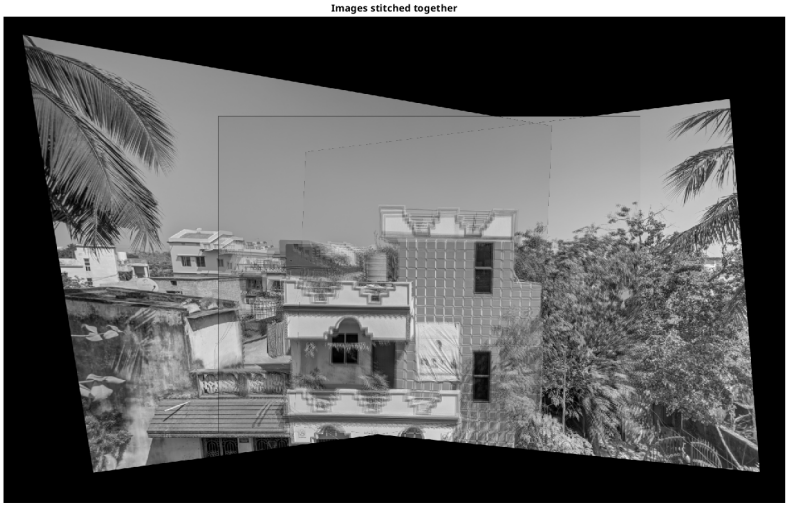

Figure 6: Mosaicing on the second set of photos clicked