Assignment - 2
CSCI - B657 Computer Vision
Prof. David J Crandall

Prashanth Kumar Murali (prmurali) , Rohit Nair (ronair)

The program runs for each question differently. The program should be run as:

Part 1:

for question 1:
$ ./a2 part1 q1 <image1> <image2>

for question 2:
$ ./a2 part1 q2 <image1> <image2> <image3> ….

for question 3:
$ ./a2 part1 q3

for question 4:
$ ./a2 part1 q4

Part 2:

for question 1:
$ ./a2 part2 q1 <image1>

for question 2:
$ ./a2 part2 q2 <image1> <image2>

for question 3:
$ ./a2 part2 q3 <path>seq1/*.jpg

PART 1:

1. Takes in 2 images computes the SIFT vectors for both, finds matches and connects the best matches with lines. We checked a lot of thresholds for proper matching and drawing lines. The best was achieved with distances < 1/40th element of the distances vector. The distances are calculated as follows:

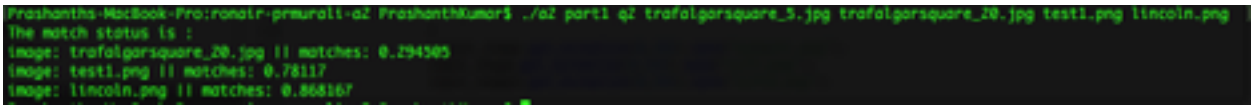
Euclidean distances are calculated between each vector of the input image and each vector of the query image. For one vector in the input image we calculate the least and next least distant vector in the query image, then take a ratio of these distances and save it in the distances vector. 1/20th vector gave the best possible result for us. This was calculated after a lot of trials from high values to low values.

The below attached image is the one calculated between two pics of trafalgarsquare. The output is stored in a file called output1.png.

Notice that only the best matched pairs are marked.

2. Takes in an input image <image1> and a set of other images matches and gives the decreasing order of matched images.

Below is an image of the list of matches for a sample set:



3. The set of images for part 1 is present in the folder part1_images. We pick an image at random and compare it with all the other images. The distance is measured in the same way as above. A complete list of images with their 10 matches and the distances to with the precision is given in the text file output3.txt. The precision goes from as low as 0.1 to as high as 0.7. But the precision values are generally lower considering the time taken to run.

4. We form quantized x vectors of size k by taking random samples between 0-1 of length 128 (same as a descriptor of input image) and doing a dot product with each vector of input and output image. We divide that dot product sum with a W, which is the degree of quantization. We set the k value to be 10 and the W value to be 500 as these values gave us the best results. The records processed were not too much and accuracy increased with 3 sample sets.

The table below gives the comparison between the images and the accuracy achieved with each method (3 and 4):

Image Name	Max Precision by Q3	Max Precision by Q4
bigben_10.jpg	0.1	0.3
bigben_12.jpg	0.1	0.2
bigben_13.jpg	0.1	0.2
bigben_14.jpg	0.1	0.3

Image Name	Max Precision by Q3	Max Precision by Q4
bigben_16.jpg	0.1	0.3
bigben_2.jpg	0.2	0.6
bigben_3.jpg	0.1	0.3
bigben_6.jpg	0.2	0.4
bigben_7.jpg	0.1	0.3
bigben_8.jpg	0.2	0.5
colosseum_11.jpg	0.1	0.2
colosseum_12.jpg	0.1	0.1
colosseum_13.jpg	0.1	0.1
colosseum_15.jpg	0.1	0.1
colosseum_18.jpg	0.1	0.1
colosseum_3.jpg	0.1	0.2
colosseum_4.jpg	0.2	0.2
colosseum_5.jpg	0.2	0.1
colosseum_6.jpg	0.1	0.1
colosseum_8.jpg	0.1	0.2
eiffel_1.jpg	0.1	0.2
eiffel_15.jpg	0.1	0.1
eiffel_18.jpg	0.2	0.3
eiffel_19.jpg	0.2	0.3
eiffel_2.jpg	0.2	0.3
eiffel_22.jpg	0.2	0.4
eiffel_3.jpg	0.2	0.4
eiffel_5.jpg	0.1	0.3
eiffel_6.jpg	0.2	0.4
eiffel_7.jpg	0.2	0.3
empirestate_10.jpg	0.1	0.4
empirestate_12.jpg	0.1	0.4

Image Name	Max Precision by Q3	Max Precision by Q4
empirestate_14.jpg	0.1	0.3
empirestate_15.jpg	0.1	0.2
empirestate_16.jpg	0.2	0.6
empirestate_22.jpg	0.2	0.4
empirestate_23.jpg	0.1	0.2
empirestate_25.jpg	0.1	0.3
empirestate_27.jpg	0.2	0.5
empirestate_9.jpg	0.1	0.2
londoneye_12.jpg	0.3	0.3
londoneye_13.jpg	0.4	0.4
londoneye_16.jpg	0.3	0.3
londoneye_17.jpg	0.3	0.3
londoneye_2.jpg	0.3	0.4
londoneye_21.jpg	0.4	0.4
londoneye_22.jpg	0.3	0.3
londoneye_23.jpg	0.3	0.4
londoneye_8.jpg	0.3	0.4
londoneye_9.jpg	0.4	0.3
louvre_10.jpg	0.2	0.3
louvre_11.jpg	0.1	0.1
louvre_14.jpg	0.1	0.1
louvre_15.jpg	0.2	0.1
louvre_16.jpg	0.2	0.1
louvre_3.jpg	0.2	0.2
louvre_4.jpg	0.1	0.1
louvre_8.jpg	0.2	0.2
louvre_9.jpg	0.2	0.1
notredame_1.jpg	0.1	0.1

Image Name	Max Precision by Q3	Max Precision by Q4
notredame_14.jpg	0.2	0.4
notredame_19.jpg	0.2	0.2
notredame_20.jpg	0.1	0.2
notredame_24.jpg	0.2	0.2
notredame_25.jpg	0.1	0.2
notredame_3.jpg	0.4	0.5
notredame_4.jpg	0.1	0.3
notredame_5.jpg	0.4	0.5
notredame_8.jpg	0.1	0.2
sanmarco_1.jpg	0.1	0.1
sanmarco_13.jpg	0.1	0.1
sanmarco_14.jpg	0.1	0.2
sanmarco_18.jpg	0.1	0.1
sanmarco_19.jpg	0.1	0.2
sanmarco_20.jpg	0.1	0.1
sanmarco_22.jpg	0.1	0.1
sanmarco_3.jpg	0.1	0.1
sanmarco_4.jpg	0.1	0.3
sanmarco_5.jpg	0.1	0.2
tatemodern_11.jpg	0.5	0.5
tatemodern_13.jpg	0.5	0.3
tatemodern_14.jpg	0.7	0.7
tatemodern_16.jpg	0.7	0.7
tatemodern_2.jpg	0.4	0.7
tatemodern_24.jpg	0.6	0.7
tatemodern_4.jpg	0.5	0.6
tatemodern_6.jpg	0.5	0.5
tatemodern_8.jpg	0.6	0.7

Image Name	Max Precision by Q3	Max Precision by Q4
tatemodern_9.jpg	0.7	0.7
trafalgarsquare_1.jpg	0.3	0.3
trafalgarsquare_15.jpg	0.3	0.3
trafalgarsquare_16.jpg	0.2	0.4
trafalgarsquare_20.jpg	0.2	0.3
trafalgarsquare_21.jpg	0.2	0.3
trafalgarsquare_22.jpg	0.2	0.4
trafalgarsquare_25.jpg	0.1	0.3
trafalgarsquare_5.jpg	0.2	0.5
trafalgarsquare_6.jpg	0.2	0.3
trafalgarsquare_8.jpg	0.2	0.2

If you see by the above table, the accuracy has increased by 20% on an average between approach in question 3 and question 4. A full report of all the images with their match accuracies and precision values along with the matches are present in the files output3.txt and output4.text respectively. These outputs were generated by running through the whole 100 images and finding matches for each.

On the submitted code running q3 and q4 will pick a random image from the image samples and find matches for it. On an average q3 takes **172 - 320 seconds** (3.2 mins average) to run for one image and q4 takes **71 - 130 seconds** (1.4 mins average for one set of x vectors) for one image. I have set the repetitions to only one for q4.

PART 2:

1. Homography is calculated for the image given with the projective transformation matrix specified. The function inverseWarp houses the code for this. without interpolation, it gives correct output but a little staggered picture. Pixels are out of its grid space in the target location. The following approach was used to correct this:
    1. warped(x, y, 0, 0) = input_image(floor(out[0]), floor(out[1]), 0, 0);
    2. warped(x, y, 0, 1) = input_image(floor(out[0]), floor(out[1]), 0, 1);
    3. warped(x, y, 0, 2) = input_image(floor(out[0]), floor(out[1]), 0, 2);

The references for concept of interpolation is given in the code.

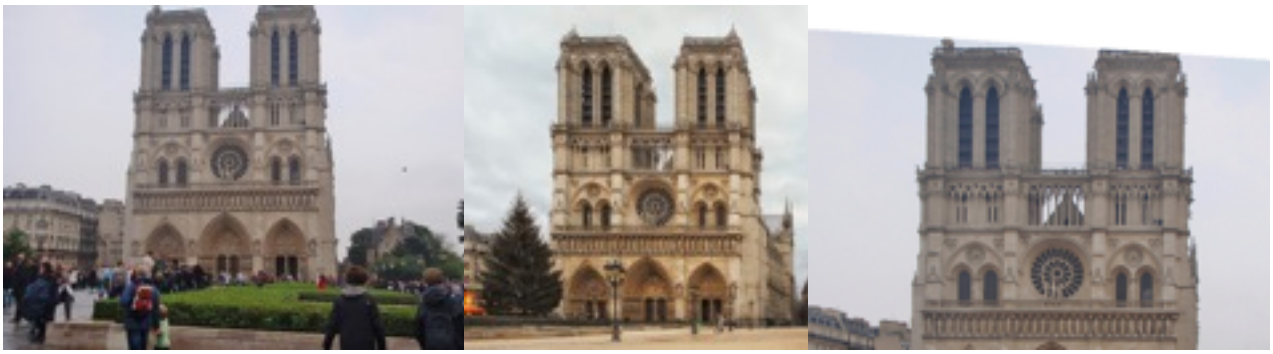
The output is as follows. The first image is the original and the next one is the warped image. Notice that the interpolation has smoothed the picture, but without loss:

2. First we calculate sift matches between the 2 images using the method described above
We take 4 random descriptors from input1 and corresponding matching descriptor points from input2
For this, we have 2 constraints
1. We select the best sift matches using the distances calculated during sift matches
2. We make sure that none of the 4 points are close to each other

Once we finalize these 4 points and the correspondences, we calculate the 8x8 matrix A and 1x8 matrix B
Now we solve AX = B and get the homography

For each descriptor , we calculate what the corresponding point in input2 will be and compare this with its corresponding sift match. Basically we are checking how many descriptors agree to this homography

We do this reps number of times, which we tried with 500 and 10000 and take the homography where maximum number of points agree
Now we apply this transformation to input1. Please find a few images below which have been warped by this method.



3. Query image is our 3rd argument (which is the first image of the sequence) and we run a loop from argument 4 till then end (all the other images).
We pair each one with the query image and do the following:
call part2q2 function as explained above. This gives us the correct Homography.
call inverseWarp which applies this Homography to the image to convert it to the query image co-ordinates.
modify the filename and save the png output.

The following series of images is the 2nd image of seq1 warped according to each image.