CSCI-B657
Assignment 2: Image Warping, Matching and Stitching Basics

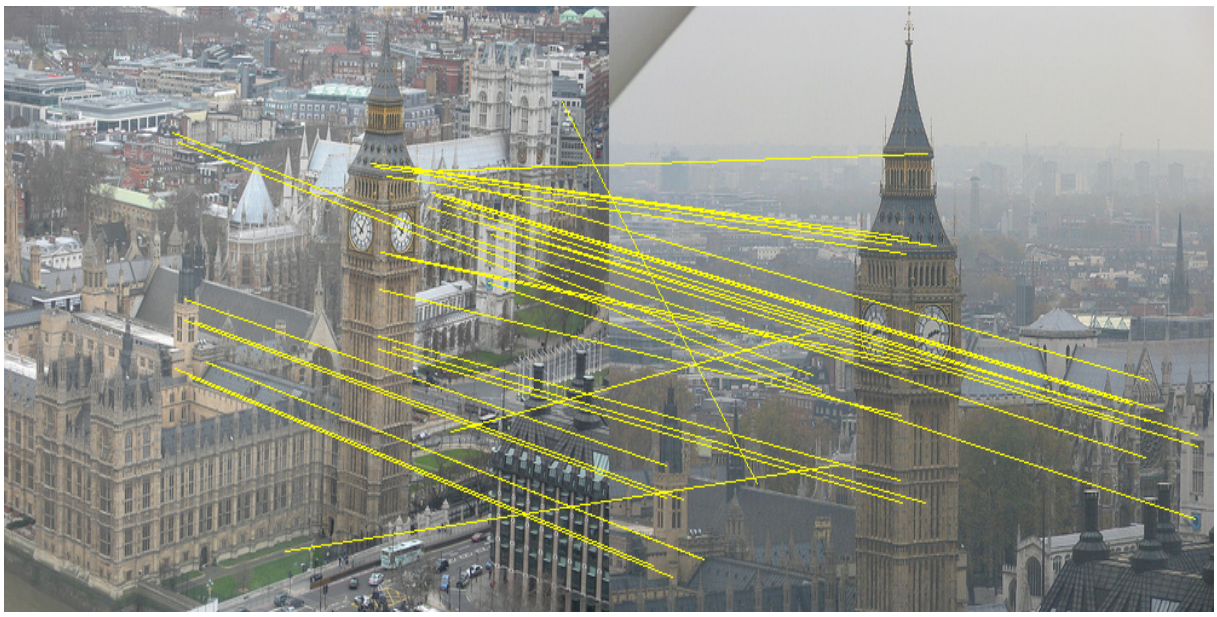
Prakash Rajagopal
Sagar Bhandare
Suhas Jagadish

February 28, 2016

1. Write a function that takes two images, applies SIFT on each one, finds matching SIFT descriptors across the two images, and creates an image.

   Solution: The steps taken to perform the above task are -
   - First step is to convert both the query image and target image to grayscale.
   - Sift descriptors are then computed for both the grayscale images.
   - We then iterate over all the descriptor values of the target image for each value of query image to find the minimum euclidean distance.
   - The closest and second-closest match is computed. If the ratio of closest to second-closest is less than the threshold, then we treat the point as the best point. All the best points are stored in a vector.
   - The query and target images are merged together and a yellow line is drawn between the matching points.

   The result of above steps is depicted in the below image, considering "bigben_2" as query image and "bigben_3" as target image -



2. Use your functions above to implement a simple image retrieval program, that searches an image

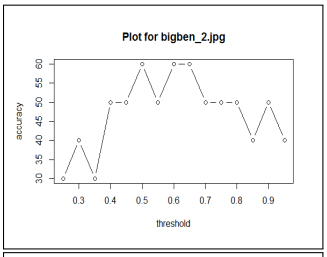
library to find the best matches for any given query image. The program should then output the list of remaining images, sorted in decreasing order of number of matched features.

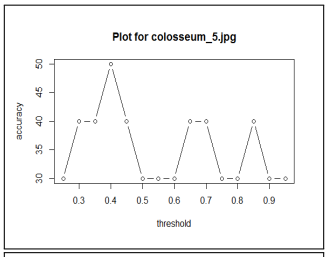
Solution: The steps taken to perform the above task are -
- The query image is run against a set of target images.
- All the target images are stored in a vector called "compare_images". Then for each image in this vector, we calculate the sift descriptors and match the key points with our query image using the technique described in problem 1.
- A priority queue can be used in this scenario where the top of the queue always contains the greatest element. We pass the number of matching points for the comparator and everytime we insert a new matching point value, it ensures that the one with the highest number of matching points stays at the top.
- All the image names are displayed starting from the top of the queue which gives us the list of target images, sorted in decreasing order of number of matched features.

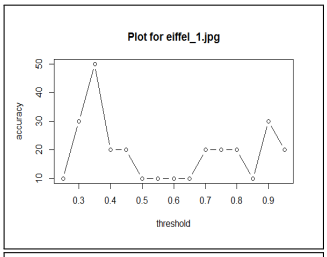
```
[jagadiss@tank sagabhan-jagadiss-prakraja-a2]$ make
g++ a2.cpp -o a2 -lX11 -lpthread -I. -Isiftpp -O3 siftpp/sift.cpp
[jagadiss@tank sagabhan-jagadiss-prakraja-a2]$ ./a2 part1.2 bigben_2.jpg bigben_
*.jpg
matching query image with each from the argument the list
Matching image: bigben_10.jpg
Matching image: bigben_12.jpg
Matching image: bigben_13.jpg
Matching image: bigben_14.jpg
Matching image: bigben_16.jpg
Matching image: bigben_2.jpg
Matching image: bigben_3.jpg
Matching image: bigben_6.jpg
Matching image: bigben_7.jpg
Matching image: bigben_8.jpg
Part 1, task 2. Printing matched images by size of matched features
bigben_2.jpg
bigben_3.jpg
bigben_14.jpg
bigben_6.jpg
bigben_8.jpg
bigben_12.jpg
bigben_10.jpg
bigben_7.jpg
bigben_16.jpg
bigben_13.jpg
[jagadiss@tank sagabhan-jagadiss-prakraja-a2]$
```

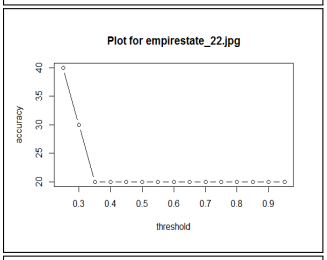
3. To measure the performance of your retrieval algorithm on this task, conduct the following experiment. For each of the 10 attractions, choose 1 image of that attraction at random to use as the query image, and then pass all 100 images as the others. Look at the top 10 ranked images returned by your program, and calculate the percentage of images that are from the correct attraction. This is called the precision of your system. Repeat this process for each of the remaining attractions, and present the results in your report. Which attractions seem to be the easiest to recognize, and which are most difficult?

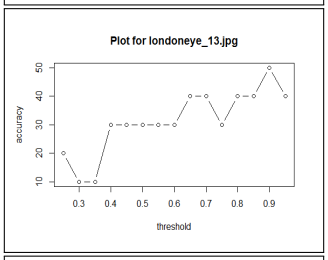
   Solution: The steps taken to perform the above task are -
   - The directory containing all the 100 images and the query image are passed as arguments.
   - Function "get_files" lists all the files in the specified directory and the result is stored in a vector.
   - Sift descriptors are calculated for each image in the vector and key points are matched with our query image using the technique described in problem 1.
   - For the top 10 ranked images, we calculate the precision by matching the filenames of the query image and matched image.
   - The above step is repeated for each of the 10 attractions and the results are tabulated.
   - One thing to notice in this problem is that, accuracies of each attractions varied as we varied the threshold. We tried to vary the threshold for each of the 100 images to yield the best precision. Plots of threshold vs accuracy for each of the best picked 10 attractions is depicted below.
   - The results are then tabulated with best accuracy for each of the 10 attractions as represented in Table 1.

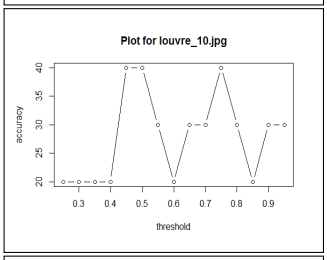


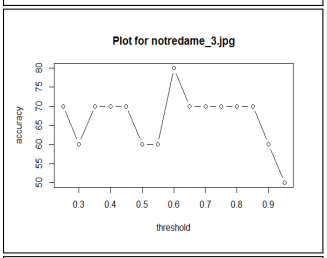


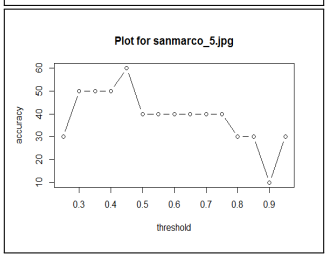


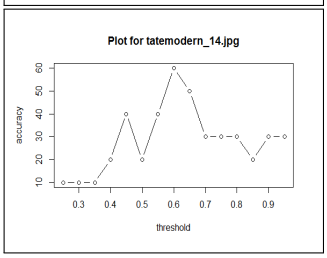












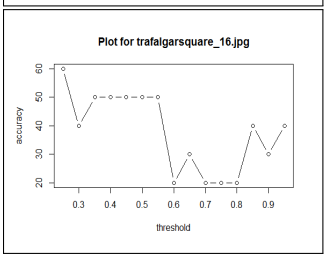
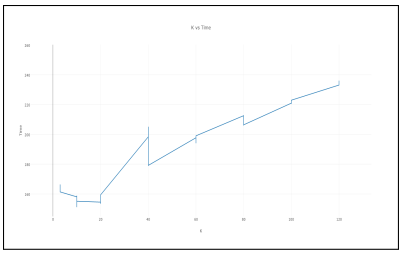
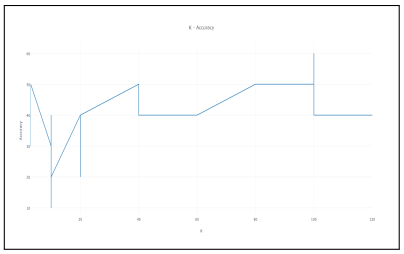
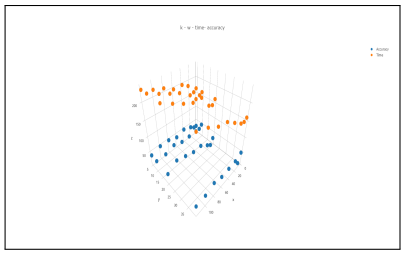


Table 1: Results of precision

	Attractions	Threshold	Accuracy
1	bigben_2.jpg	0.500	60%
2	colosseum_5.jpg	0.400	50%
3	eiffel_1.jpg	0.350	50%
4	empirestate_22.jpg	0.250	40%
5	londoneye_13.jpg	0.900	50%
6	louvre_10.jpg	0.500	40%
7	notredame_3.jpg	0.600	80%
8	sanmarco_5.jpg	0.450	60%
9	tatemodern_14.jpg	0.600	60%
10	trafalgarsquare_16.jpg	0.250	60%

- Number of distinct or unique features in a image is an important factor while comparing two images. If we look at "bigben", "notredame" and "tatemodern", they all have lot of matching points because they have many distinct features and all the images are clicked from somewhat similar projection. Hence they are quiet easy to recognize. But images such as "louvre" and "colosseum" have pictures taken from different angles, pictures with outer and inner layers and relatively less distinct points to match. Hence we found these little difficult to recognize.

4. Implement the quantized projection function for the image matching. Does the algorithm speed up image retrieval, and if so, by how much? Does the approximation seem to affect the quality of results? Give quantitative answers to these questions if possible. What is the best trade-off you can achieve between running time and accuracy, by adjusting w, k, and the number of trials?

   Solution: The steps taken to perform the above task are -
   - Finalize value of K (reduced dimensions), W (step size) and generate K sample vectors of size 128 each from uniform distribution of variable x between [0,1].
   - For all the descriptors of two images, img1 and img2 calculate summary vectors by taking dot product of sift descriptor with sample vectors.
   - This step takes time O(m + n) where m and n denotes number of key-points in img1 and img2.
   - Now, compare both newly generated summary vectors for each summary vector in img1 to find similar summary vectors in img2. Keep top 3 matches with least distance.
   - This step takes time O(m*n*k).
   - From top 3 matches calculate actual distance using euclidean distance function and pick key-point with least distance as best match.
   - Repeat this process multiple time and choose most occurring point.
   Observations:
   - The algorithm does speed up the image retrieval. The speed up is proportional to the size of K chosen. It changes time complexity from m*n*128 to m*n*k. Figure k-time plots K vs Time required values. The speed up is almost linear.
   - The reduced execution time comes with cost of quality of answer. With increased value of K the accuracy increases. Being said this there is no linear relation between value of k and accuracy as there is some randomness involved. Typically small and high values of K has given better values. We got accuracy above 50- We performed series of experiment to understand the trade off between k,w, and number of trials. The results are plotted in the graphs displayed at the top of the page.

   Trade off:
   - Number of trails increase the time required of execution by considerable amount with not much gain in accuracy. However changing values of k and W gives considerable increase in accuracy.
   - The best results are obtained at K = 100, W = 20. However K = 3 and W = 10 will give better results in very time efficient way.

   Additional efforts:
   - Along with the tasks given in the homework we performed many additional tasks, tried various new ideas, analyzed program in various ways. I am describing one of the ideas that we worked on and

implemented code for. We tried to develop better sift matcher which will not only consider the ratio between the top best matches but also considers the homography of matches.
- Following is the proposed and implemented algorithm for optimized sift matcher
- For two images img1 and img2, to be matched, calculate sift descriptors.
- Set very low threshold of 0.3 for the ratio between top 2 best matches.
- Match the two descriptors using euclidean as distance function till there are at least 30 matches, if not increase the threshold by 0.05 and repeat above steps.
- After getting 30 match points, apply RANSAC algorithm to calculate homography matrix. This will be approximate matrix.
- Restart the matching process of sift descriptors with newly generated homography matrix.
- Match each key-point descriptor to all the key-points of other image using euclidean distance and store results in priority queue.
- Remove top ten matches from the queue.
- Select a match having ratio less than threshold with its successor and matching the projected co-ordinates by homography matrix with tolerance of 20px.
- This algorithm provides a better sift matches than normal algorithm with euclidean as distance function.

Part 2: Image warping and homographies

1. Write a function that takes an input image and applies a given 3x3 coordinate transformation matrix (using homogeneous coordinates) to produce a corresponding warped image.

   Solution: The steps taken to perform the above task are -
   - An input image is chosen and a projective transform is applied to the image using the given matrix.
   - The function uses inversion technique. It takes the inverse of the matrix and then for each pixel in the new image, it calculates its corresponding position in the original image.
   - The transformed output is created in a file called "res.png".

2. Write a function that automatically estimates a homography (projective transformation) between two images. Implement this using RANSAC, as we discussed in class.

Solution: Given a set of images, our function calculates the projection matrix and prints it at first.
- SIFT matching is then applied to get a set of matching points.
- RANSAC is implemented to compute the matrix. The algorithm basically calculates the projective matrix using a set of 4 random points and then compares the quality of the result by checking if the match source corresponds to correct destination using this matrix.
- The function takes the best computed result after 100 rounds.
- This creates a warped image that is required in 2.3.

3. Combine your homography estimation and your warping code to create an image sequence warping application.

   Solution: This is similar to the previous problem but run with multiple files.
   - The function uses the matrix from the previous step and transforms the second image using that matrix. It then writes to ¡file¿_warped.jpg with an image which is in the same camera angle as the source image.
   - The results displayed in the next page shows the original image, input image 1 and its warped image, input image 2 and its warped image in 3 successive rows. The warped images appears to have been taken in the original image's camera's coordinate system.

How to run the code?

1.1) ./a2 part1.1 bigben_2.jpg bigben_3.jpg

1.2) ./a2 part1.2 bigben_2.jpg bigben_3.jpg bigben_6.jpg bigben_7.jpg bigben_8.jpg bigben_10.jpg bigben_14.jpg

1.3) ./a2 part1.3 directory(/u/jagadiss/sagabhan-jagadiss-prakraja-a2/a2-images/part1_images/) optional-src-image(/u/jagadiss/sagabhan-jagadiss-prakraja-a2/a2-images/part1_images/eiffel_18.jpg)

1.4) ./a2 part1.4 directory(a2-images/part1_images/) optional-src-img(bigben_2.jpg)

2.1) ./a2 part2 lincoln.png

2.2) ./a2 part2 lincoln.png res.png

2.3) ./a2 part2 lincoln.png f1.png f2.jpg ..

Original image