# B659 Assignment 4: Image Matching and Warping

**Term:** Spring 2015

**Date:** Friday April 17, 2015

**Team Members:**

1. Gautam Poornachandra – gpoornac
2. Anirudh Ramesh - anramesh

**GitHub repository:** anramesh-gpoornac-p4

**Language used for the project:** C++

# Contents

# 1. Introduction

This project is an implementation of techniques learnt such as image descriptors, image matching, projections and transformations. This will guide through exploring the implementations, standards, tradeoffs and techniques adapted to improve the performance.

This gives a clear understanding of efficient ways of image warping, matching, and creating panoramas.

# 2. Image Warping

*Assumptions and approach:*

- First, the user enters the transformation matrix.
- Here the second approach is considered, wherein we loop over all the destination pixels and copy the appropriate pixels from the source image.
- In order to do this, we need to find the inverse of the transformation matrix and then identify the correct x, y positions. (x,y) in source is copied to the destination which is iterated over (i,j).
- The first approach doesn't provide desirable results. The warped image contains noise.

Input Image

Output Image

*Observation:* The second approach works better than the first approach since the noise gets added in the first approach.

## 3. Image Matching

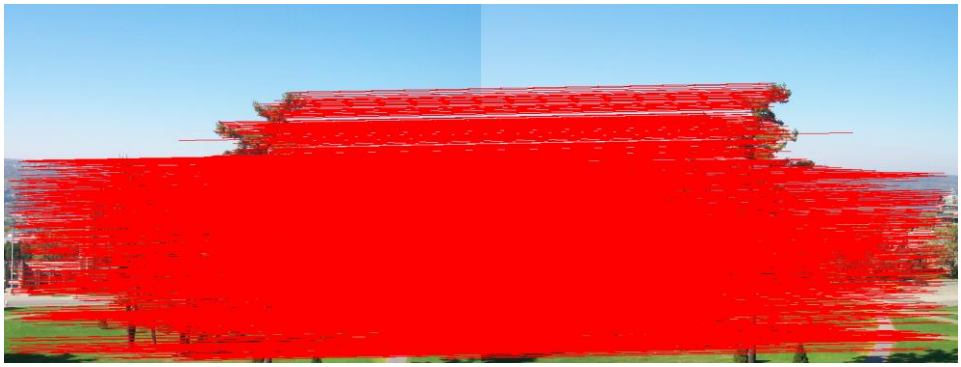
*Assumptions and approach:*

- . A circle can be specified in terms of thre parameters: the row and column coordinates of its center and its radius.
- First two images are taken, SIFT is applied to each one, finds matching SIFT descriptors across the two images

Input Image1

Input Image 2

- The inputs will generate the output image by matching the descriptors as below by computing the Euclidean distance between the two images.



Output Image

- The process is carried out for the sample set of 100 images given in the attraction_image set.
- The threshold value is set for 100.
- The number of SIFT descriptors for similar images are in the range of 4000.

*Quantitative Observations for 3.3:*

| **Name** | **No. of Matches** | **Percent (in top 10)** |
|---|---|---|
| BigBen_3 | 3 | 30% |
| Colosseum_12 | 1 | 10% |
| Eiffel_22 | 4 | 40% |
| Londoneye_2 | 4 | 40% |
| Trafalgarsquare_20 | 2 | 20% |
| Sanmarco_5 | 4 | 40% |
| Empirestate_9 | 3 | 30% |
| Louvre_4 | 2 | 20% |
| Notredame_24 | 2 | 20% |
| Tatemodern_2 | 3 | 30% |

*Note:* In the code for 3.3, the path for attraction_images folder is given.

- The quantitative analysis is done for the image set of popular attraction places. The results are mentioned above.
- We were able to get satisfactory results from the implementation.

## 4. Image Stitching

*Assumptions and approach:*

- In part 4.1, the ratio of closest to second closest distance is considered to be 0.5.
- The result is slightly better than the ones seen in part 3. We can see that there are fewer undesirable matches than in part 3.
- In part 4.2, x and y displacement values are stored in a vector and the mean for both dimensions are calculated.
- In part 4.3, random x and y values are picked for 10 times and the pair with the highest matching count is considered as the final displacement pair.
- In part 4.4, except for the pair falls_1.png & falls_2.png, stitching algorithm works accurately. In this pair, the combining area is seen.
- This might due to two reasons.
- There might be another transformation in addition to translation between the two images.
- In the displacement values approximation, there might be very few matching (or near matching values) and the randomly selected index might not have chosen the right displacement. Even if selected, since there might varied displacement value candidates, the right displacement candidate might not have obtained the highest vote. Increasing the number of such iterations could have given better results, but I observed that the speed of execution was greatly affected.

Input Image 1

Input Image 2

Output: Stitched Image

*Observation:*

- The usage of vector array was a costly operation which took more than 20 minutes to generate the output for a 120*120 image. Hence we had to stick on to these computational dependencies and settle for a very small image of size 32*32 pixels for the purpose of computation.
- After trying to use different data structures we had to go to AI Stefan regarding the costly operation to reduce it. With the given time and taking other constraints in to consideration, we had to stick on to the existing method to implement the segmentation for the very small image.

## 5. Speed ups – Implementation of LSH

*Assumptions and Approach: Mark Image*

- First, vector with components are selected at random**.**
- Dot product of the randomly generated vector and the high dimensional space is taken to reduce the high dimensional space.
- Nearby projected points are taken and put into the same bin in the Hash Map.
- The rest of the steps are done as per the LSH algorithm with implementation of the Hash.
- The paper by Slaney and Casey that outlines one way of efficiently finding approximate nearest neighbors. This is used in the implementation.

*Observations and Quantitative analysis:*

Speed up image retrieval:

Yes the algorithm speed up the image retrieval. The execution time is as follows:

| Approach | Time taken in seconds (Average value of all tests) |
|---|---|
| Brute force as in part 3 | 1200 |
| LSH implementation | 790 |

Quality of results:

- The result was not as accurate as we got in 3.3 implementation.
- There were less correct matches compared to the part 3 implementation.
- Few of the images matched even if they were taken at different location.
- *Quantitative Observations:*

| **Name** | **No. of correct matches in Brute force – Part 3** | **No. of correct matches in LSH – Part5** |
|---|---|---|
| BigBen_3 | 3 | 2 |
| Colosseum_12 | 1 | 1 |
| Eiffel_22 | 4 | 2 |
| Londoneye_2 | 4 | 2 |
| Louvre_4 | 2 | 1 |

- The performance was clearly different using the LSH algorithm.
- It took less time to execute with a tradeoff to the accuracy.
- The time taken on an average was nearly half the time taken to execute the part 3 brute force implementation.
- The accuracy was reasonable as it was able to get the correct matches.
- After discussing with the AI Stefan, was able to get a clear idea of the implementation which was hard to analyze in the beginning.

## 6. Conclusion:

- The project gave practice with the techniques feature point (corner) detection, image descriptors, image matching, and projections and transformations. It gave a clear idea on how they can be combined together to both recognize scenes and create panoramas.
- The implementation of complex algorithms like LSH gave a clear way of improving the time for the processing of images with few approximations and accuracy considerations.
- Overall it gave a clear insight and hands on experience to implement the concepts learnt in the class to match the images.
- AI’s help was really appreciable as they gave a lot of required insights to implement the complex algorithms like LSH.