Master MVA
Object Recognition and Computer Vision

Assignment 1: Instance Level Recognition

Victor Busa
victor.busa@ens-paris-saclay.fr

October 6, 2017

I Sparse features for matching specific objects in images

I.A SIFT features detections

1. It is important to obtain similarity co-variant features for matching because, in real life, images undergo various transformations (scaling, rotation, translation, shear, lighting, ...) depending on the point of view from which the picture was taken. Hence, we need a model that does not depend upon these transformations.

2.

Figure I.1: detected features for three different values of the peakThreshold

3. The density of detections across images change with the *peakThreshold*. Indeed, the *peakThreshold* parameter allows rejecting the potential key-points locations found if the intensity of those points is less then the *peakThreshold* value. Hence, the matching won't work as well when the two images have different intensity (luminosity). We can fix the drawback of this technique by preprocessing the images in order to bring them to the same intensity (luminosity).

Figure I.2: The number of detected features depends upon the intensity (luminosity) of the image. here **peakThreshold = 0.007**

I.B SIFT features descriptors and matching between images

1. The fact that the descriptors are computed over a much larger region than the detection is a good idea in the sense that the matching will occur on a broader space and hence the detection would likely be more accurate.

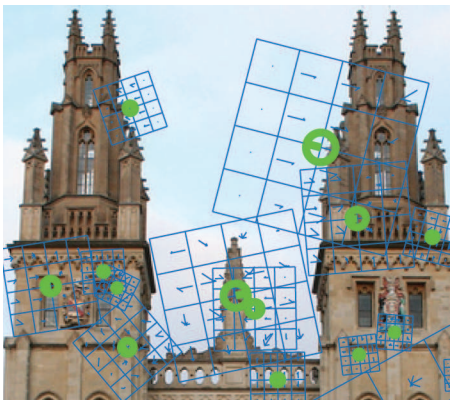


Figure I.3: SIFT descriptors

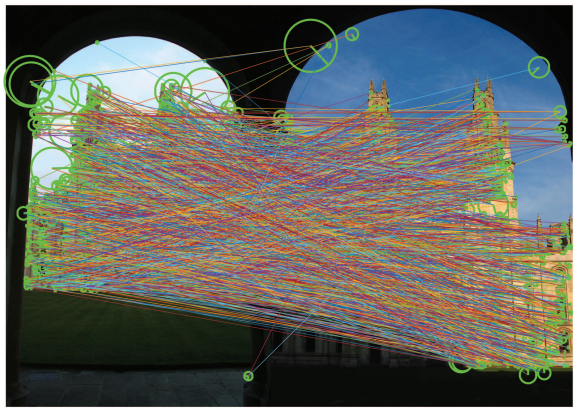


Figure I.4: Nearest neighbour matches

2. As we can see in Figure I.2, mismatches are lighting dependent. Actually, SIFT doesn't work well when there are either a large illumination change or nonrigid deformations. To reduce the amount of mismatch, one can use a cluster of at least 3 features that agree with the geometrical position of an object. These clusters are more likely to be correct than if we only focus on individual feature matches.

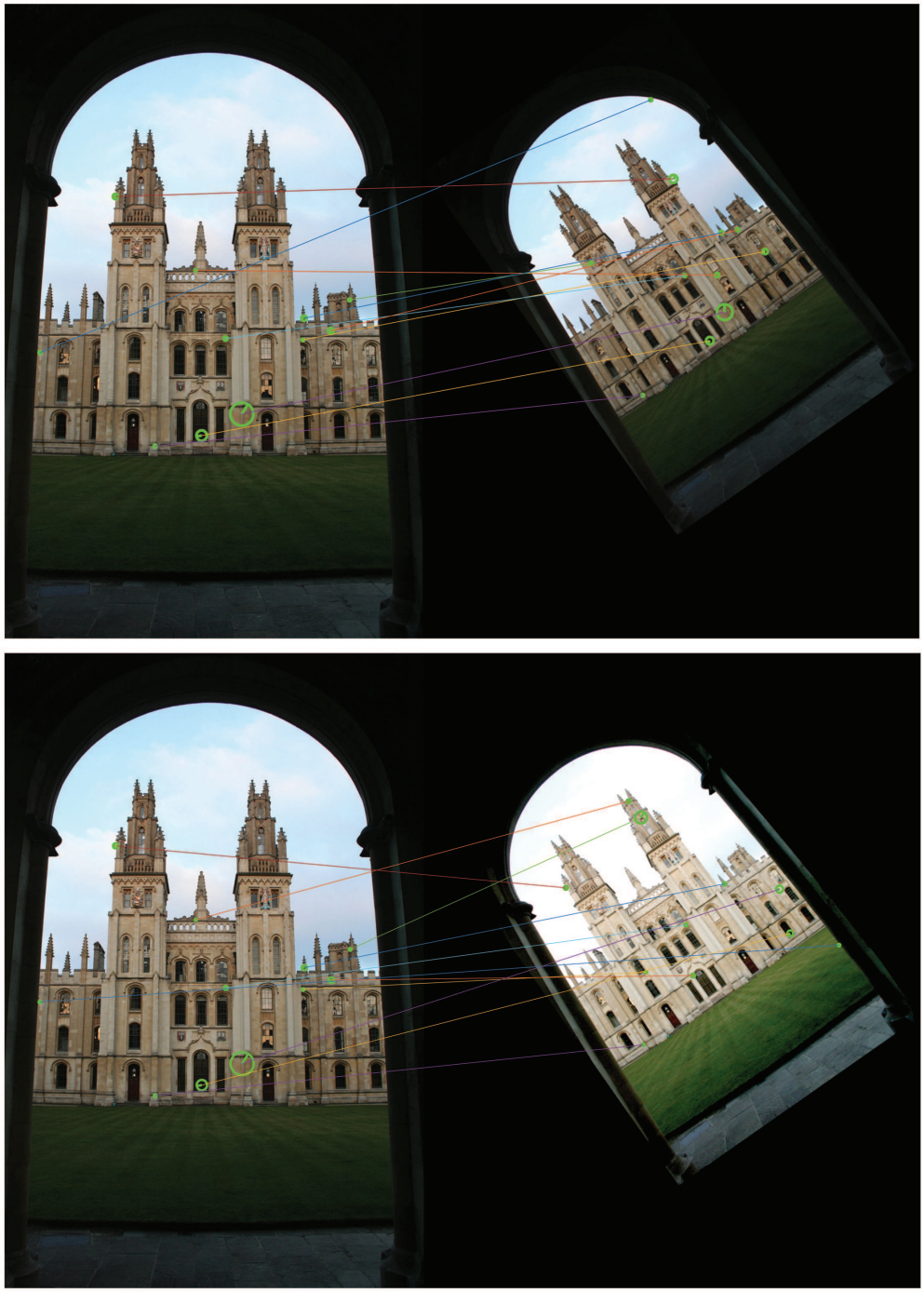
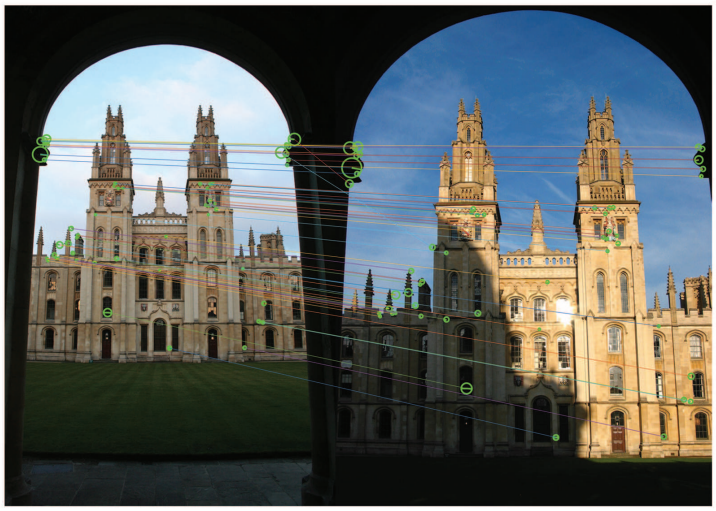
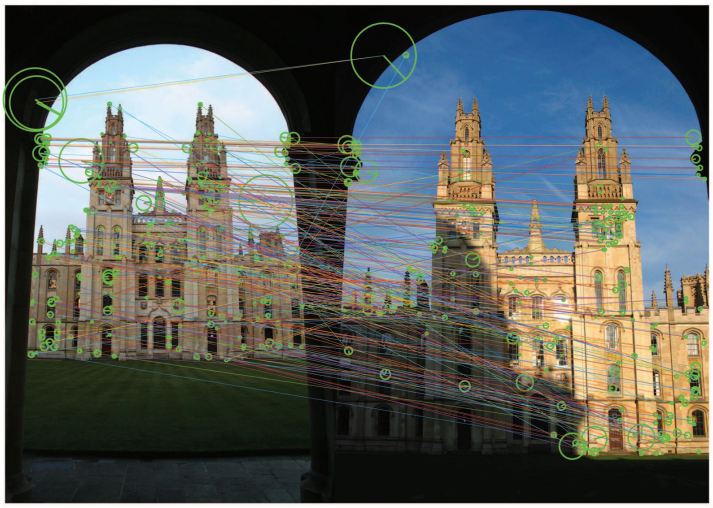


Figure I.5: Effect of lighting on the SIFT descriptors

Matches filtered by the second nearest neighbour test: nnThreshold = 0.3



Matches filtered by the second nearest neighbour test: nnThreshold = 0.5



Matches filtered by the second nearest neighbour test: nnThreshold = 0.8

Figure I.6: Effect of the second nearest neighbor threshold on the number of correct and incorrect matches

I.C Improving SIFT matching using Loweś second nearest neighbor test

1. We can considerably reduce the number of mismatches while discarding very few correct matches by using the second nearest neighbor test. The idea behind this technique relies on the fact that:

- correct matches need to have the first nearest neighbor closer than the closest incorrect match (second nearest neighbor)
- false matches will have the first and second nearest neighbor within a similar distance. Hence false matches will have a ratio: $\frac{1NN}{2NN} \approx 1$

We can see in Figure I.6 that the higher the *nnThreshold* is, the more matches we have. Also, a good trade-off is around 0.8. Above 0.8 we will keep too many mismatches. Below 0.8 we will have too few matches.

I.D Improving SIFT matching using a geometric transformation

1. TO DO

2. For each tentative correspondence, we compute the similarity transformation. Then we apply the same step as in the RANSAC algorithm. At the end, after testing several affine transformations, we select the one with the highest count of inliers.
We can see in Figure I.7 that there are fewer matches than when using the second nearest neighbor test with a threshold of 0.8. Yet, we can clearly see that these matches are very accurate (we can quickly see that all the lines between matches are parallel).

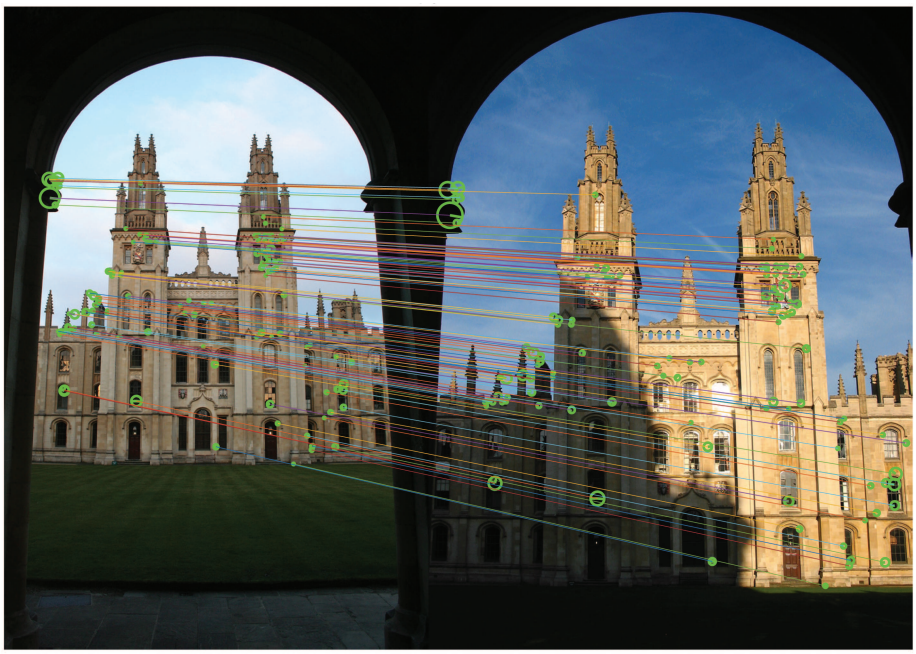


Figure I.7: Matches after geometric verification

II Affine co-variant detectors

1. At first, there are more similarity detector matches than affine because similarity transformations are invariants through rotation, translation, and zoom and the first 2 images undergo only simple variations similar to these transformations. On the other hand, when the initial image undergoes a more complex transformation like shear, affine co-variant detectors tend to be more effective as affine transformation can take into account a perspective deformation (a square is mapped to any parallelogram through any affine transformation).
Note: Affine transformation has 6 degrees of freedom while similarity has only 4 degrees of freedom.

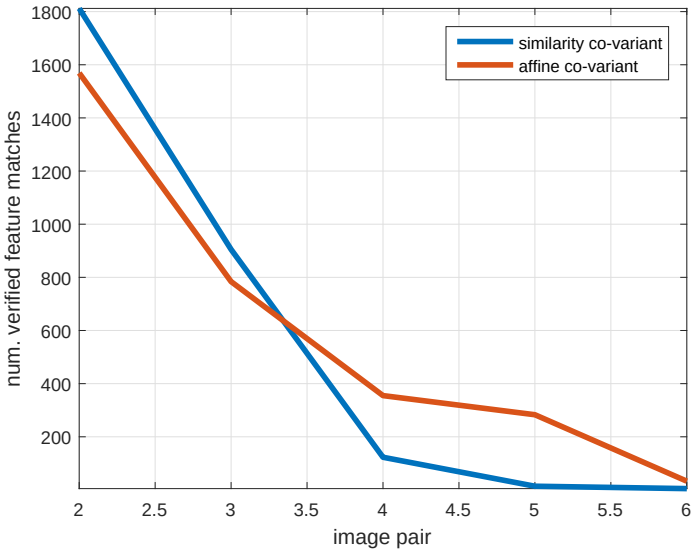


Figure II.1: Number of verified matches with changing viewpoint

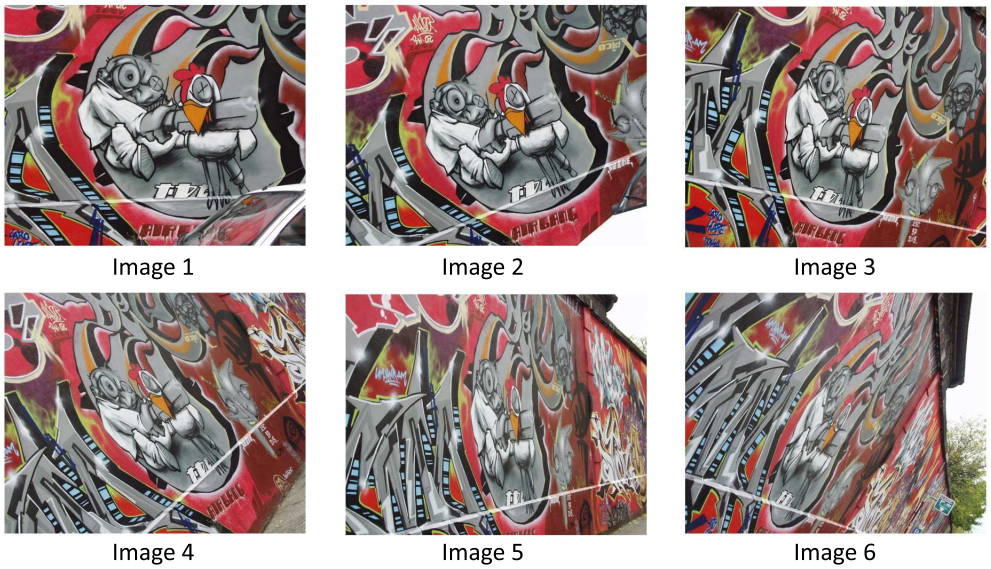


Figure II.2: The first 2 transformations of image 1 (image 2 and image 3) are approximately similar to a rotation, while the others transformations are more complex and involve changes in perspective.

III Towards large scale retrieval

III.A Accelerating descriptor matching with visual words

1. In practice, we precompute the visual words of each descriptor, so that we don't need to account for the time required to convert the descriptors into visual words.

2. I've run the code for all the images in the folder (662 files) and I've recovered the times elapsed by the two methods for a different fixed number of images.
We can notice that, for a large database, visual words are around 50 times faster than comparing descriptors between the query image and the images in the database.

Table 1: Speed comparison (in second) of Raw SIFT descriptors and Visual Words methods on different database size

Database size	**1**	**5**	**10**	**50**	**100**	**250**	**500**	**600**
Raw SIFT Time	0.038	0.137	0.243	1.242	2.399	2.399	11.305	13.754
Visual Words Time	0.0016	0.0039	0.0067	0.0297	0.0574	0.1329	0.2684	0.3250

III.B Searching with an inverted index

1. The first image has a score of one because it is both the query image and the target image. As the similarity between a query image and a target image is calculate as the inner product of the normalized histograms of theses images, that explains why the score of the first image is 1 ($< u, u >= \|u\|^2 = 1$)

2. There are 16 erroneous images among the 25 top results. We can notice that the erroneous images are quite similar (Roman architecture). You can see the erroneous images in Figure III.1

Figure III.1: Erroneous images are highlighted in red

III.C Searching with an inverted index

1. We are now re-scoring the images based on the number of inlier matches after a geometric verification step. Hence, the measure is not normalized anymore and that explains why the score is much higher than 1.

2. Using geometric verification allows improving the recognition a lot. Indeed, the erroneous images appear after the correct one whereas when we were searching with an inverted index the correct and incorrect images were mixed. Also, we can notice that there is a gap in term of geometric scoring between the last correct image and the first erroneous image: respectively 37 and 10. See Figure III.2

Figure III.2: Erroneous images are highlighted in red

IV Large scale retrieval

1. the painting database actually stores an inverted file index with 100000 visual words for each image and there are 1734 images. Also, we can notice that each word is weighted by its inverse document frequency.

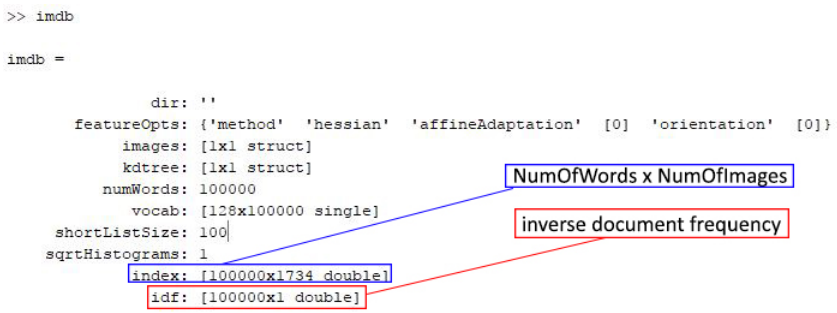


Figure IV.1: image database structure

2. the image database takes $\mathcal{O}(NumOfWords \times NumOfImages)$ Memory.
Here we have: **NumOfWords** = 100000 and **NumOfImages** = 1734.

3. Stage of the search:

- We compute the histogram of the visual words in the query image and in the database images
- We score each image of the database using an inverted file index. This can be done quite efficiently using Matlab's built-in sparse matrix engine.
- We re-score the top-ranked images from the previous step using a geometric verifications step.

Table 2: Search Times (in second) for each mystery images

Image	**The Starry Night**	**Wheat Field with Cypresses**	**Sedie**
Feature time	0.235	0.424	1.770
Index time	0.028	0.032	0.034
Geometric verification	0.372	0.541	1.967