Master in computer VIsion and roBOTics

Report on

# Simple Object Recognition Scheme

Visual Perception Lab number 4

Made by:

Viktor Stefanovski

Chalikonda Prabhu Kumar (Chakon)

Mentors:

Rafael Garcia

Joaquin Mas Salvi

5/26/2013

# Introduction

The objective of this coursework is developing object recognition scheme from a given dataset of images of various building. For this purpose features are extracted using a descriptor, matching of two entries from the dataset is performed which will give the number of features matched, and on a higher level the most similar image, or the image with highest number of correctly matched points is retrieved from the proposed dataset when given a desired input image.

In the first part the Difference of Gaussians (DoG) as detector and Scale Invariant Feature Transform (SIFT) descriptor are used for extracting local descriptor from given image. The descriptor consists of the most salient points also called key points. The main advantages of SIFT, making it frontrunner in the field is as the name states the invariance to changes in scale, translation, rotation, local geometric distortion and furthermore to noise and different illumination. The majority of the points that SIFT extracts as salient points lie in high-contrast areas, such as object edges. As result when using the SIFT descriptor feature vectors are obtained. These vectors have different size for every image due to the different image dimensions. The constant value is the length of this vector, 128 bits used for describing each key point.

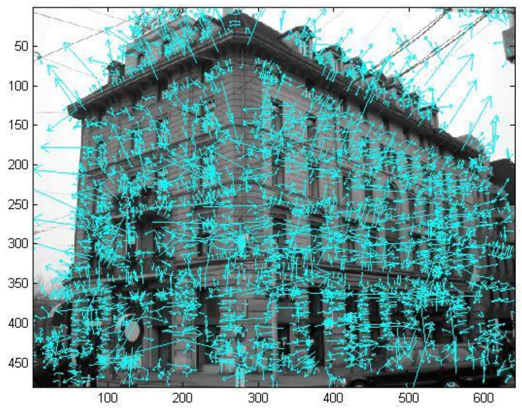
# 1. Step 1

## 1.1. DoG and SIFT

The implementation of SIFT used for completing this task is the original implementation of the founder of this descriptor David Lowe. Also a matching solution is proposed for matching two images by using the features extracted by their descriptors.

## 1.2. SIFT descriptor extraction

When extracting the features using SIFT for the images object0016.view01.png and object0016.view02.png, two images of same object with different scale factor, the number of key points changes. The feature vectors obtained are similar but have some differences, like, position, orientation and magnitude of the features. This is to be expected, because SIFT defines the key points on the maximum of the Difference of Gaussians computed at different scale factor, assuring this way the orientation and scale invariance. The feature vector of the first image is 3282x128 and the one of the second image is 3398x128, meaning that 3282 and 3398 key points are extracted respectively. The conclusion drawn is that with increasing the scale the number of features extracted also increases. The reason for this is the smaller background (sky) coverage in the scaled image. Note SIFT extracts does not extract great number of features from uniformly distributed areas. Here is the visual representation of the feature extraction in these two images.



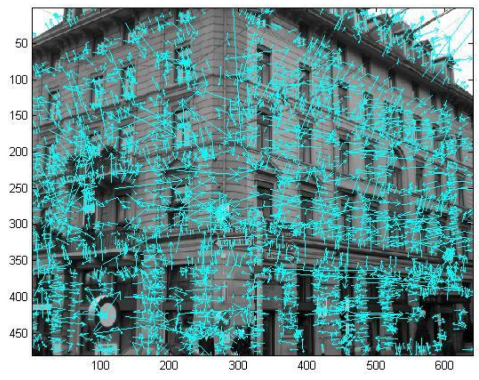


**Figure 1 SIFT descriptor feature points image object0016.view01.png (left) and object0016.view02.png (right)**

When observing the result in Figure 1 we can conclude that the most of the features are similar, but there are some new descriptors that appear in the second image. They can be visualized more effectively in the images of Figure 2.

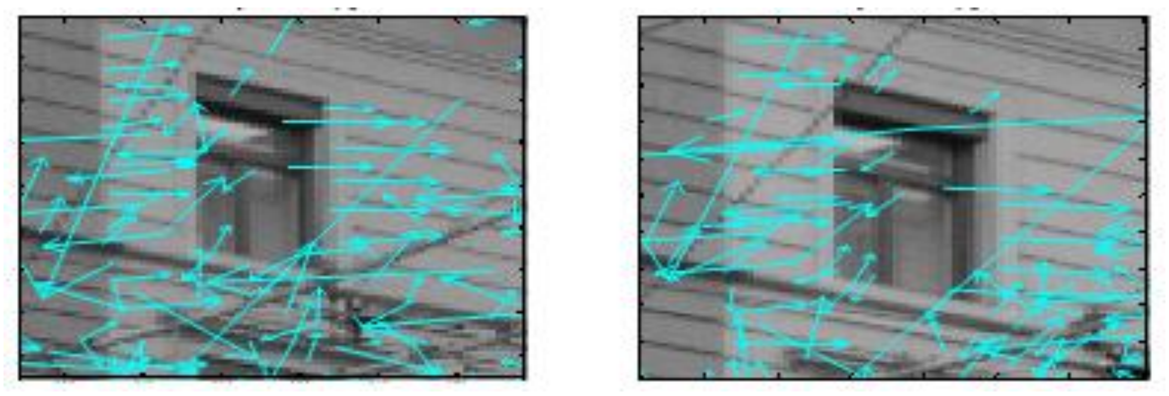


**Figure 2 SIFT features zoomed area of image object0016.view01.png (left) and object0016.view02.png (right)**

In all of the images the arrows represent the magnitude and the orientation of the feature points extracted with the descriptor, and the length of the arrow contains the gradient information.

When we have change in the viewpoint, as in the case of object0016.view01.png and object0016.view04.png, also the number of key points changes from 3282 in the first image to 2853 key points in the second image. Furthermore the descriptors are very different for almost all of the feature points (except for 2, see next step figure). This means that SIFT is not robust to view point changes because it does not take into account affine transformations. Here is the output of these trials. For best solving of the correspondence between feature vector length the Dense SIFT descriptor can be used.

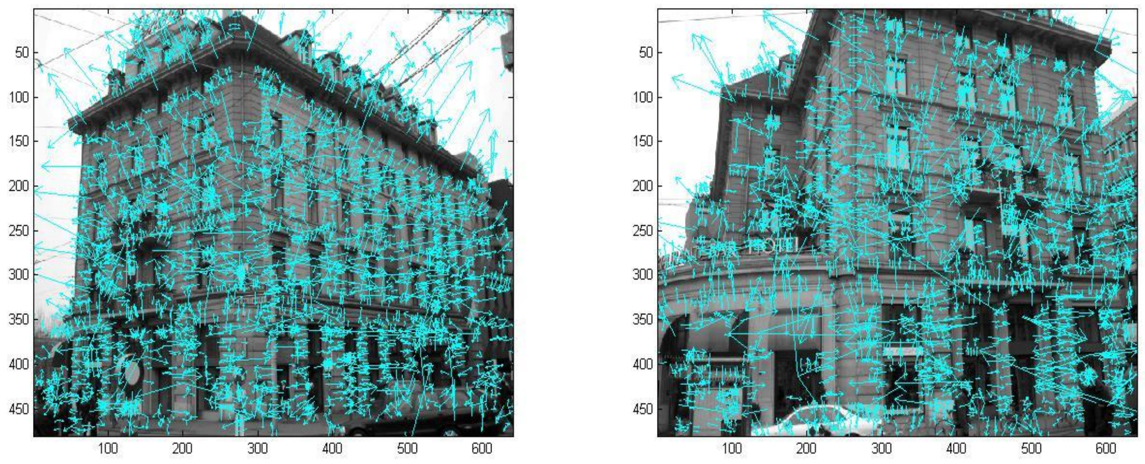


**Figure 3 SIFT descriptor feature points image object0016.view01.png (left) and object0016.view02.png (right)**

As we can observe in the next figure, when taking the same window from the images obtained from different viewpoints the descriptors are very different from each other.

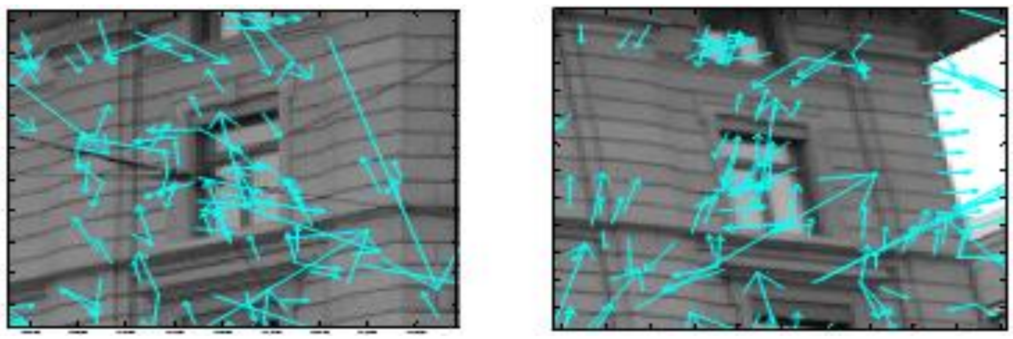


**Figure 4 SIFT features zoomed area of image object0016.view01.png (left) and object0016.view04.png (right)**

## 1.3. Matching

Matching procedure was performed with the previously presented image set, yielding the following results. When object0016.view01.png and object0016.view02.png were matched the number of correct matches found was 1064, and in the case of object0016.view01.png and object0016.view04.png only 2 correct matches were found. Results are shown in the two following figures.

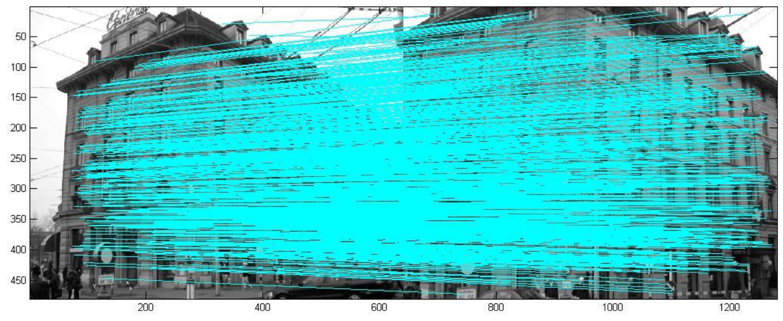


**Figure 5 Matching between object0016.view01.png and object0016.view02.png**

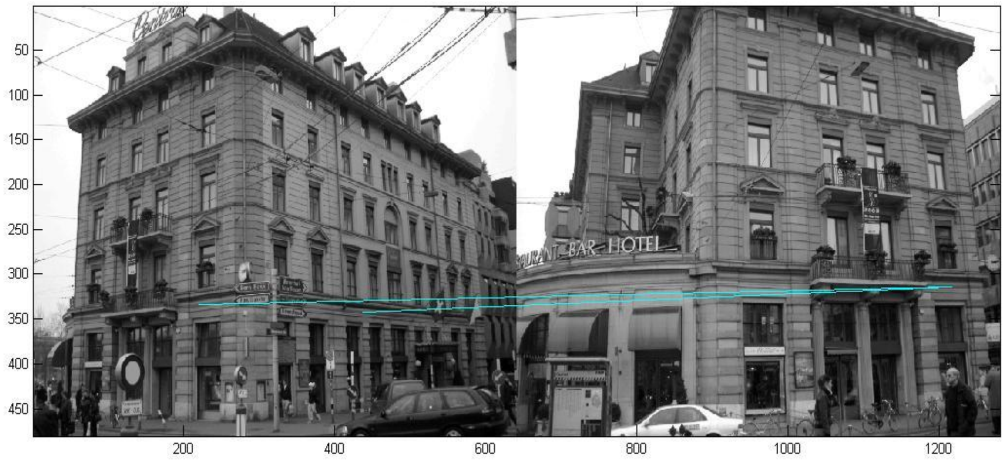


Figure 6 Matching between object0016.view01.png and object0016.view04.png

When matching images of different objects (buildings) for example object0004.view04.png and object0007.view04.png a small number of correct matches (actually mismatches) is found (1 in this case) which represent marginal error. In some cases there are no matches found (which is correct), for example in the case of matchingobject0006.view04.png and object0007.view04.png. SIFT descriptor is not robust when matching images which contain different objects.

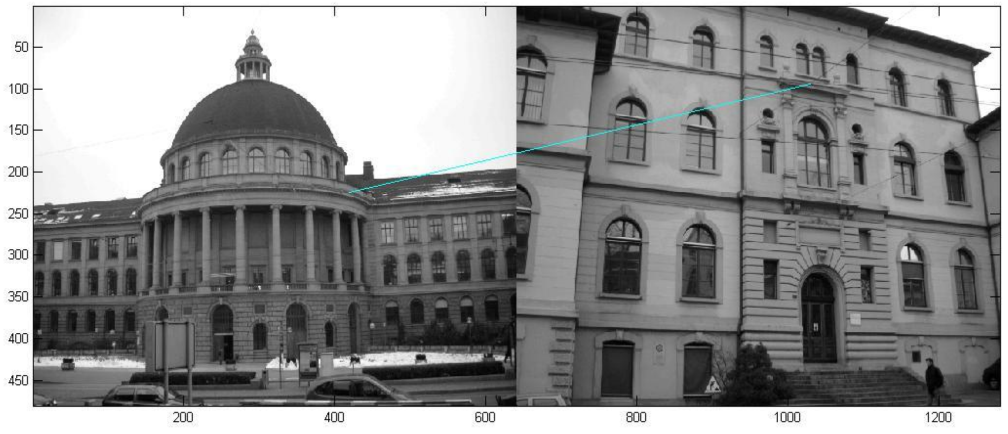


Figure 7 Matching between object0004.view04.png and object0007.view04.png

The main parameter for maximizing the number of correct matches and minimizing the number of incorrect ones, or optimizing the number of matches is the distance ratio parameter. It represents a threshold value keeping only the matches where the ration of the nearest and second nearest neighbor is below the specified value. The predefined value of 0.6 gives the optimal results. When we increase the value, the number of matches increases, but it is difficult for a conclusion to be drawn based on the visualization as it can be seen in Figure (*distRatio*=0.9, matches found 1801). Also when we decrease this value we lose some of the correct matches (*distRatio*=0.3, matches found 418). The experimenting is done upon the same images as previously (object0016.view01.png and object0016.view02.png).

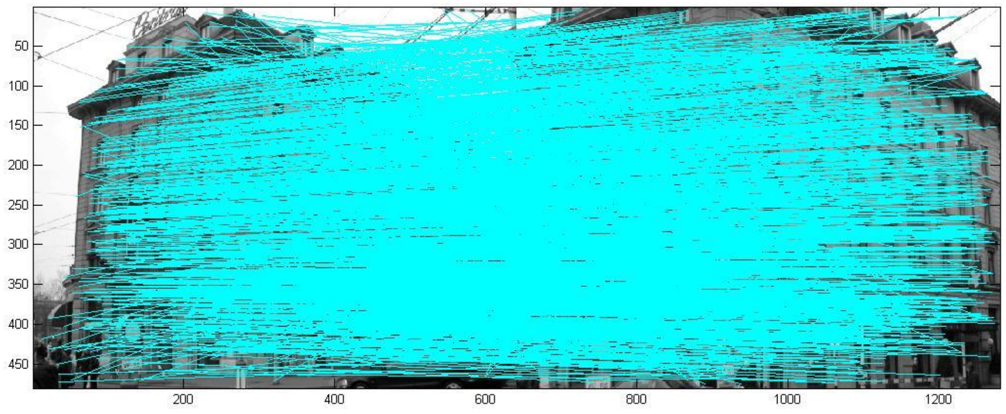


**Figure 8 Matching between object0016.view01.png and object0016.view02.png with distance ratio 0.9**

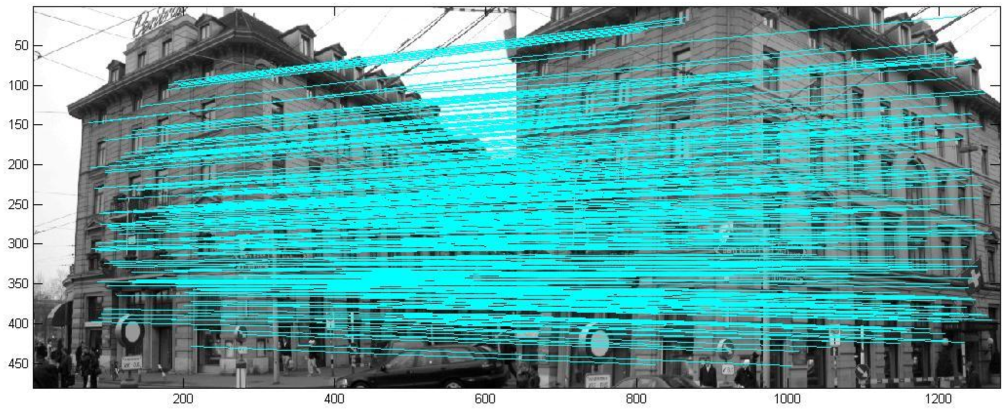


**Figure 9 Matching between object0016.view01.png and object0016.view02.png with distance ratio 0.3**

In the next table we can see the results on more experiments along with their computational time:

| Matching between | *distRatio* | | | Number of matches | | | Computational time (sec) | | |
|---|---|---|---|---|---|---|---|---|---|
| object0016.view01.png object0016.view02.png | 0.3 | 0.6 | 0.9 | 418 | 1064 | 1801 | 20.20 | 20.11 | 19.92 |
| object0002.view02.png object0002.view03.png | 0.3 | 0.6 | 0.9 | 1 | 54 | 596 | 13.51 | 12.96 | 13.32 |
| object0007.view03.png object0007.view05.png | 0.3 | 0.6 | 0.9 | 0 | 33 | 490 | 12.88 | 13.34 | 13.12 |
| object0005.view04.png object0005.view01.png | 0.3 | 0.6 | 0.9 | 28 | 271 | 733 | 12.36 | 11.98 | 11.72 |

**Table 1 Comparison of number of matches achieved using different ratio of distance between first and second neighbor and the computational time consumption of the task**

In the next figure experimental proof of the rotation invariance of the SIFT descriptor can be seen. The *distRatio* parameter is set to 0.3, producing only the 90 "strongest" matches.

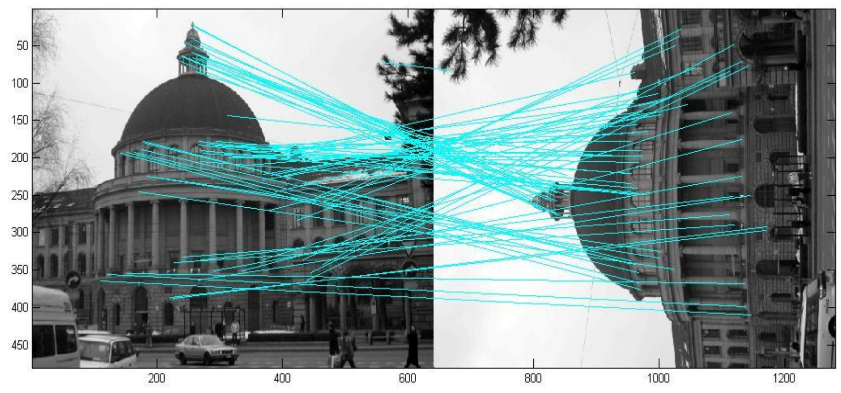


**Figure 10 Rotational invariance of SIFT descriptor**

The following graphical information show the comparison of number of matches used for the image set with change in the *distRatio* parameter:

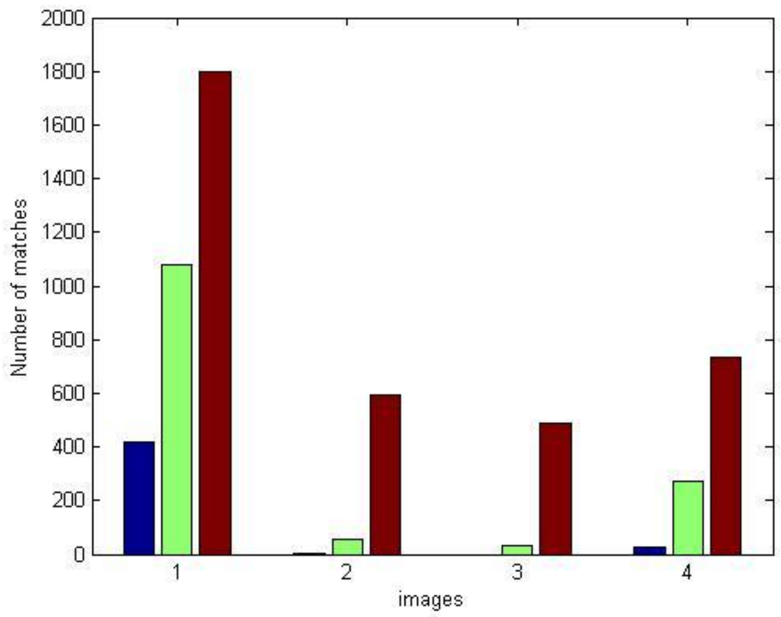


**Figure 11 Number of matches on same image set using different *distRatio*, blue=0.3, green=0.6 and red=0.9**

## 1.4. Euclidean Distance VS Dot Product

In the proposed solution angle between the feature vectors in the vector space is used. If the feature vectors are same the angle difference will be zero, and in the case when they are very similar it will be very small. Due to this the angle between feature vectors is considered to be effective measure for comparing feature vectors.

For computing this angle first the dot product of the first descriptor and the inverse of the second descriptor is used. After that, the arc cosine function of the dot product is obtained, instead of dividing the dot product with the module of each feature vector, since both feature vectors have been previously normalized to unit length values. In a way, this can be considered as approximation of the Euclidean distance. When the angle between the feature vectors is small the cosines of that angle converges to 1, and by this the Euclidean distance will converge to zero.

After changing the code to computer the Euclidean distance between the feature vectors, we observed that the number of matches increased. Yet, the dot product computation approach is considered better because of the less computational cost. For comparison the time of computation using dot product is 20.55 seconds and the time of computation using Euclidean distance is 35.19 seconds. Here is a graph comparing this values for different matches made:

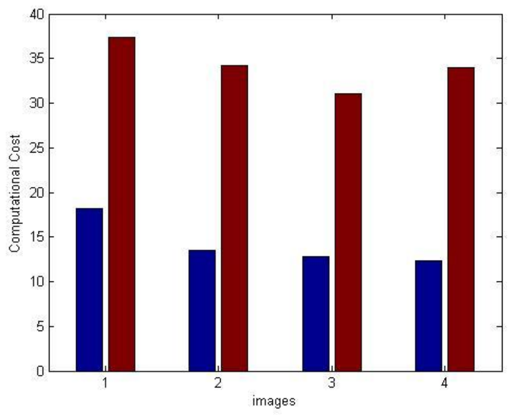


**Figure 12 Computational time using dot product (blue) and Euclidean distance (red) on different matches**

Regarding the efficiency of the algorithms, there is a big difference. For example if we use *distRatio* of 0.3 we obtain 966 features which is close to the value when using *distRatio* of 0.6 and dot product approach. Additionally, when we use 0.6 and 0.9 for *distRatio* we obtain 1367 and 2236 matches respectively, which are hard to visualize. The computational time varies around 35 seconds for both the abovementioned trials.

By reducing the *distRatio* to 0.1 and 0.01 we get only the strongest 450 and 14 feature points respectively. The time of execution is same as for the previous cases.

On Figure 11 we can observe one example of using the Euclidean distance approach with big *distRatio* (0.9) – the one that is hard to be visualized and the case when we use 0.01 *distRatio* where we can clearly observe that the “strongest” feature points are correctly matched.

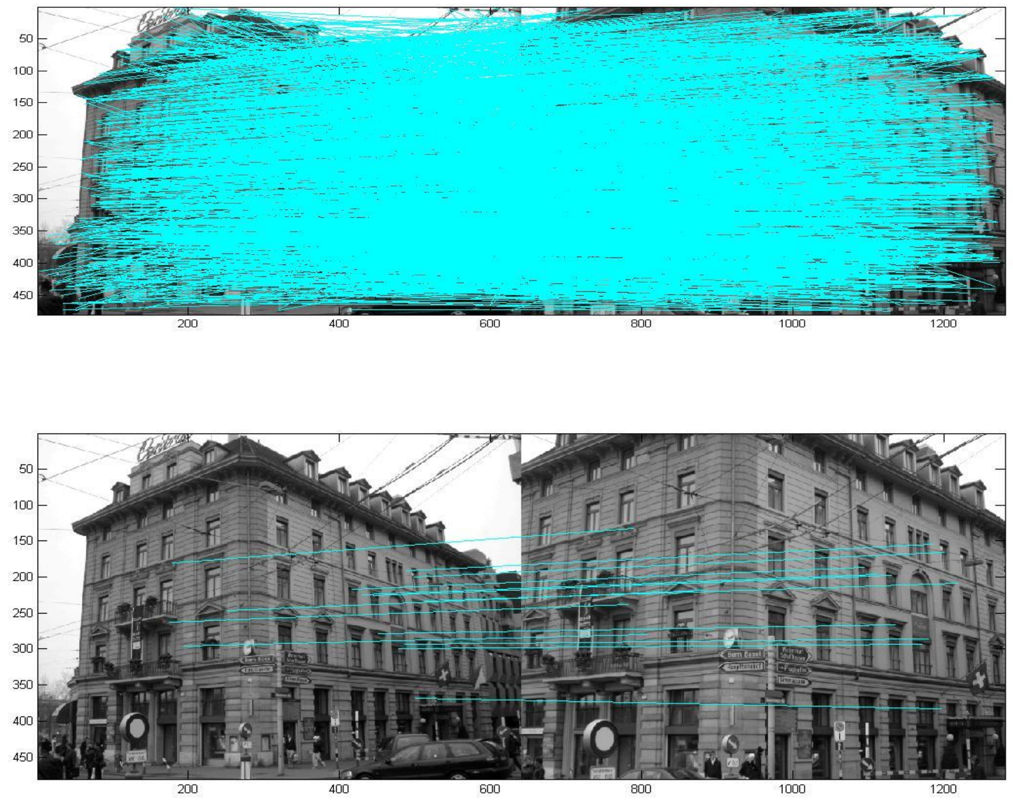


Figure 13 Matching with using *distRatio* of 0.9 (top) and 0.01 (bottom)

## 1.5. Thresholding on nearest neighbor distance

Initially, the code is using the ratio between the first and second nearest neighbor. After changing the code to consider a thresholded value of the nearest neighbor, either using a dot product of Euclidean distance, the conclusion drawn is that the ratio between first and second neighbor is better choise because correct threshold value is very difficult to be obtained, and may cause loss of correct matches or gain of incorrect matches if it is not properly set. For example when using a threshold value of 0.2 to the angular approximation only 710 matches are found between the previously described images (object0016.view01.png and object0016.view02.png) (Figure 12 top).

When threshold value of 0.05 is used along with Euclidean distance 842 key points are matched (Figure 12 center). And lastly, when threshold value of 0.005 is used to the Euclidean distance approach 57 features are extracted. The computational time is varying around 35 seconds for all three attempts.

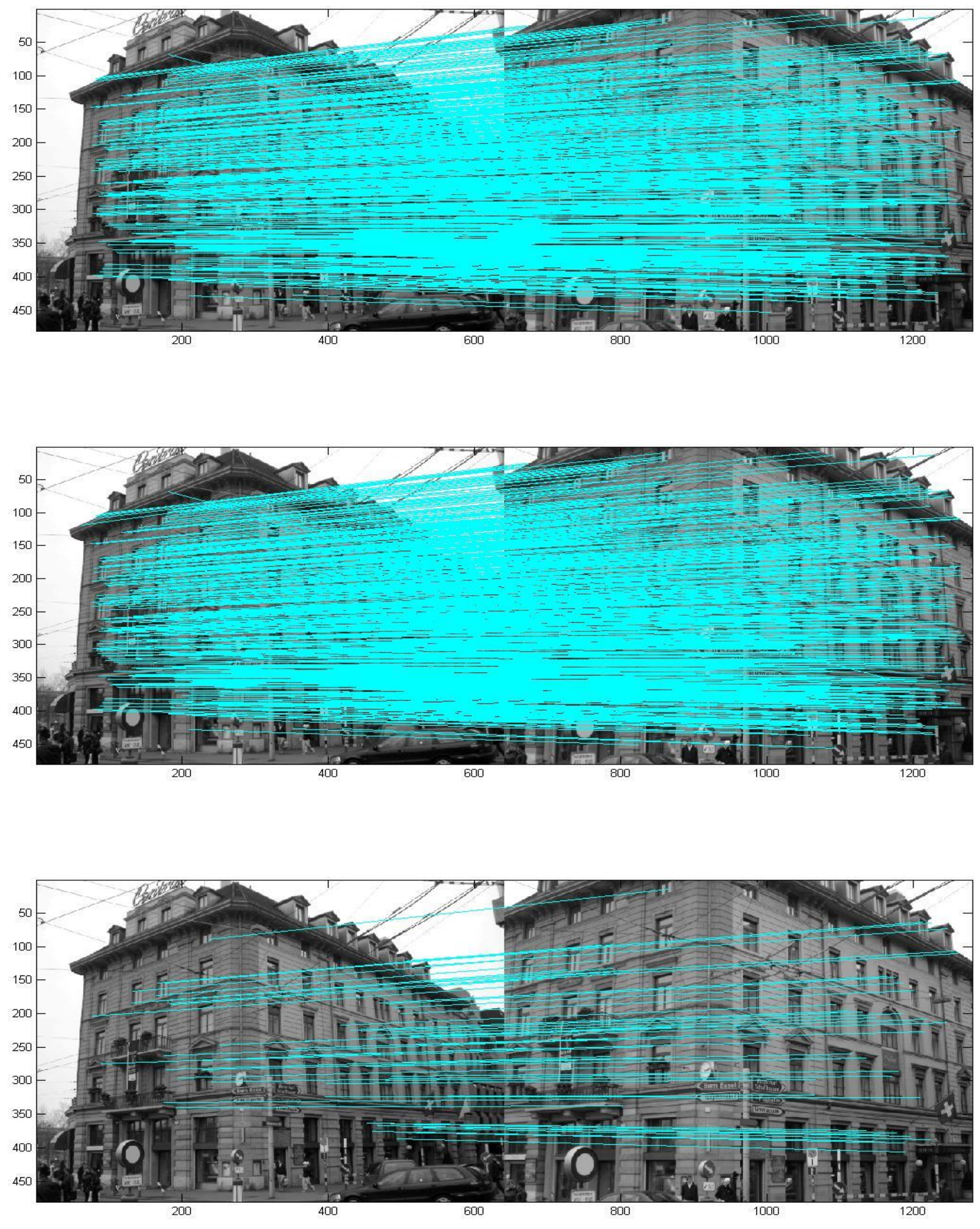


**Figure 14 Matching of images with different threshold – 0.2 (top), 0.05 (center) and 0.05 (bottom)**

# 2. Step 2

In this step the Harris, Hessian and the Harris/Hessian/Laplace detectors will be used for matching the key points. Difference in the output produced will be noticeable when observing, namely instead of arrows elliptical regions will be used to represent the affine invariance. These detectors should contribute to better matching process rather than the Difference of Gaussians (DoG) detector used in the first step.

## 2.1. Extract features

The extract features executable performs feature extraction using different detectors (previously mentioned). It has significantly different way of work, it can be only called using the system command and can have various inputs, such as type of detector, type of descriptor (sift or extended sift), input and output image, also it can use PCA for projecting the descriptor into lower dimensional basis etc.

## 2.2. Feature extraction

Default threshold value of 500 is used in the extract features executable. It can be used to output a great number of features which can be used for processing but are not very good for visualizing.

The script *display_features.m* is responsible for demonstrating the output of the extract features executable. The number of features extracted is not the same because different detector is used. The proof to the previously mentioned lack of effective drawing (plotting) capabilities of the features extracted with this threshold value for the image set (object0016.view01.png and object0016.view02.png) is given in the following figure. The different number and position of extracted features is sufficient prerequisite for them being complementary in Harris/Hessian detector, shown in the following examples.

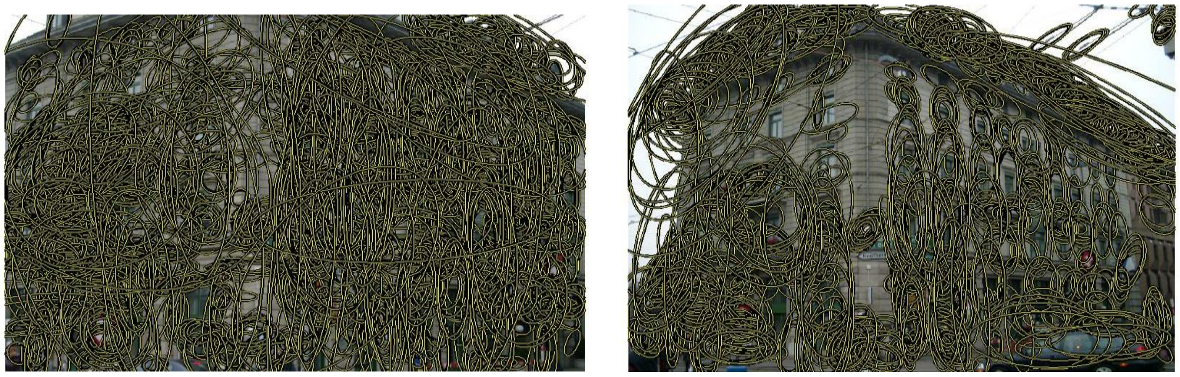


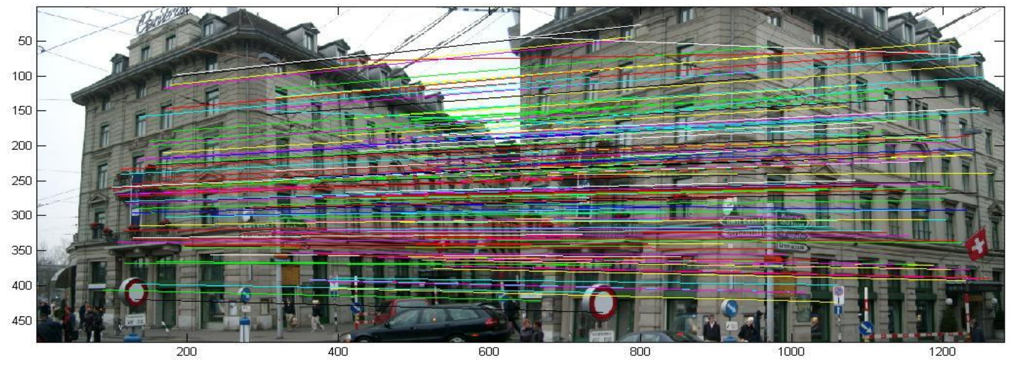
Figure 15 Extracting features of images object0016.view01.png using Harris detector (left) and object0016.view02.png using Hessian detector (right)

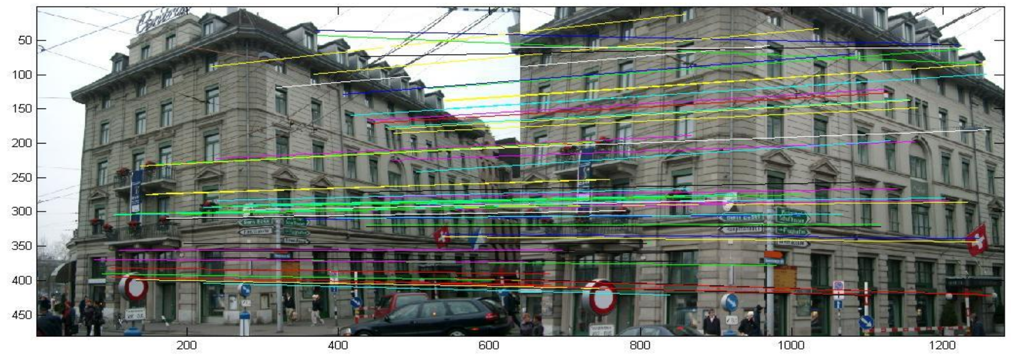
## 2.3. Code adaption

The code adapting of the code of step 1 in that way so that it is compatible with the new features and descriptors obtained with the feature extraction executable is coded in the *Lab4VP.m file*.

## 2.4. Matching using affine invariant detectors

After several experiments with the affine transformation the best matching results when considering the images with difference in scale are the following: using *distRatio* of 0.3 and Harris detector, 213 matches are made, Figure 14 (top), when using *distRatio* of 0.33 and Hessian detector, 81 matches are made, Figure 14 (middle) and when using *distRatio* of 0.2 and Harris/Hessian/Laplace detector, 19 matches are made, Figure 14 (bottom).





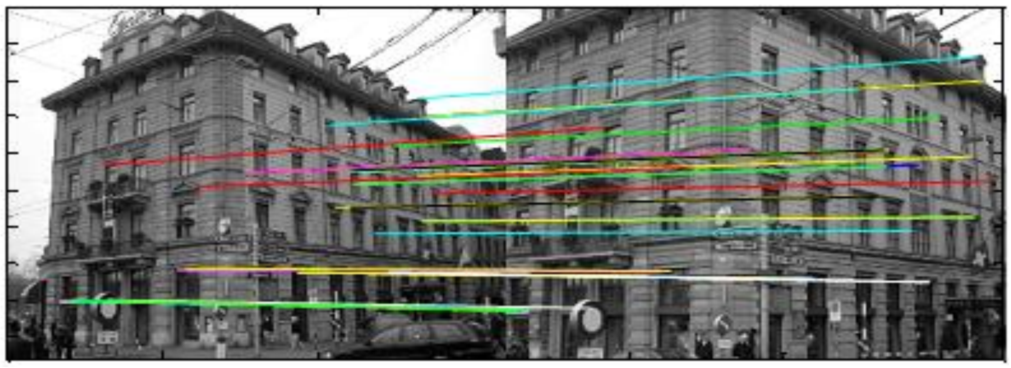


Figure 16 Matching using affine invariant detectors

Consequently, matching experiments on images taken from a different viewpoint was made. During this experimental part the *distRatio* parameter was set to 0.5, while using the Harris affine detector and two image sets were examined. The conclusion drawn is that the Harris + SIFT approach outperforms the DoG + SIFT, where we had only a few or none matches when there was change in viewpoint. Here the number of matches is considerably higher.

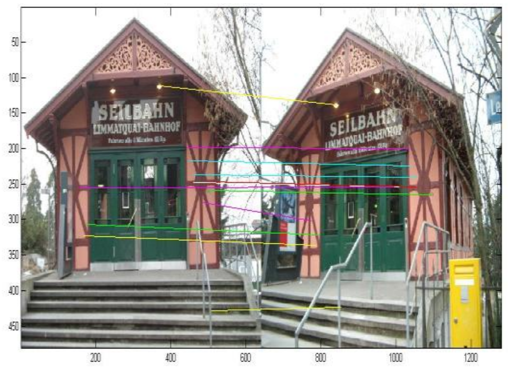
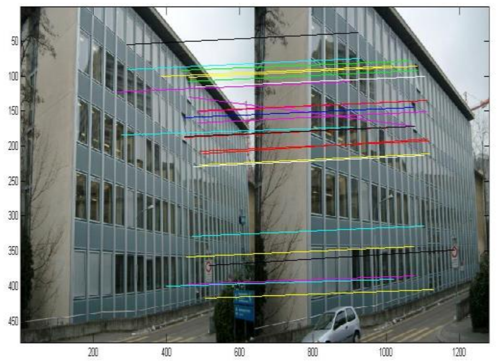



**Figure 17 Matching using Harris detector using *distRatio*=0.5 on image sets with change in viewpoint on object0006.png (left) where 16 matches were found and image object0002.png where 31 match was found**

Lastly, using the affine detectors a matching on rotated image set (object0004.png) was made. The Harris, Hessian and the combination Harris/Hessian/Laplace were used along with *distRatio*=0.35 and the obtained results are represented in the following figure.

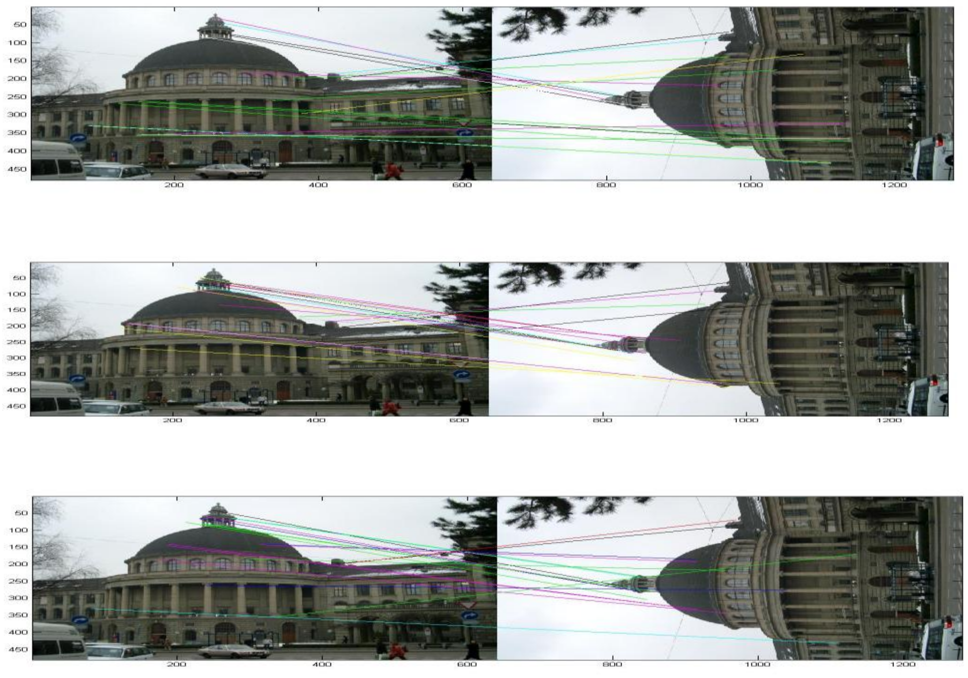


**Figure 18 *distRatio*=0.35, Harris (top) 25 matches, Hessian (center) 24 matches and Harris/Hessian/Laplace (bottom) 23 matches**

## 2.5. Parameters selection

Here, as in the previous section the distraction parameter plays main role in the number of matches produced. As it can be seen from the previous results, in most of the cases the *distRatio* has values of in the range (0.2-0.5) with values around 0.3 found as most suiting in the majority of cases.

## 2.6. Detectors comparison

From the previously presented results it is easily noticeable that the affine detectors outperform the Difference of Gaussians (DoG) detector, especially when there is a change in viewpoint of the images which are supposed to be matched. Otherwise, the matches in the first part (DoG + SIFT) are greater, but that does not mean that they are of higher quality as often the number of false positives can be big which depends on the different approaches (Euclidean distance, thresholding of first neighbor etc.) and the *distRatio* parameter.

In the next figure we can observe graphical comparison of the affine detectors applied when matching images with change in scale and viewpoint.

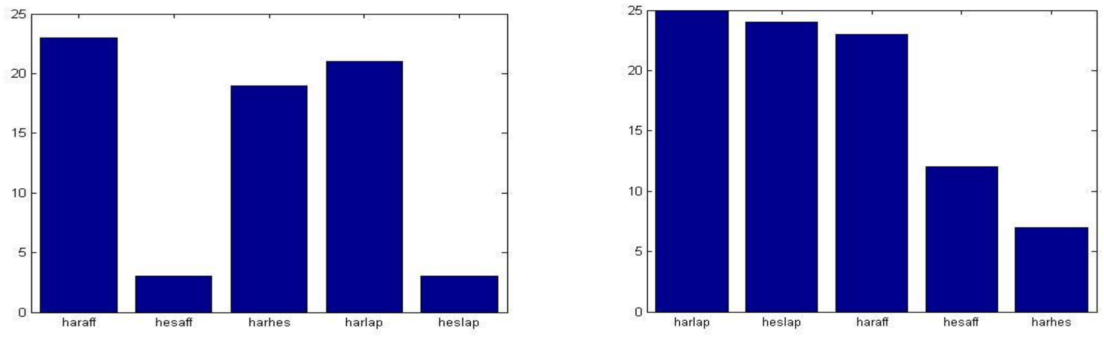


**Figure 19 Detectors comparison when matching images with difference in scale (left) and difference in viewpoint (right)**

From the extensive experiments made the conclusion drawn is that in the majority of cases (not always) the detector which yield best matching results is the Harris affine detector.

# 3. Step 3

Using the features obtained in the previous section a simple building recognition scheme was developed for recognizing images of buildings from the local dataset. The task of this scheme is to check whether one of the buildings is present in the new test image, and give the ranking of the first four images where the building is most likely to appear.

## 3.1. Downloading

Firstly, the data set of images, creating train and test subsets was downloaded.

## 3.2. Offline component

The features are extracted in the offline component. This means that if we have sample data set with the image names we have to extract the features using the names of the images.

For the purpose of performing feature extraction we developed a matlab script where the file names in the training data and testing data were used for selecting the images.

During the experiments where the feature extraction process we used various detectors followed by sift descriptors. If needed or wanted a change to any descriptor of interest can be done in order for the features to be obtained.

## 3.3. Online component

In this part the task was writing the script for extracting features in online component. This case resembles when we have bunch of data with different file names. For completing this task we did a code which extracts the features without using the file name of the images. We have trained the data to find the matches.

After this training we have to extract the features based on the output format. In the next step, we used the function called *getparam* to get descriptors and location of the features which are necessary for matching. In this part, firstly, we are checking whether the image is grayscale and if it is not we are converting it to grayscale. Then we are obtaining the descriptors and the locations of the feature. Here one should remember format of the output extracted from extract_features.exe. Some time the descriptors start from 6th column and sometimes from 13th column which completely depends on the features format we obtained. Then we have to normalize the descriptors.

Once we obtain the descriptors with the location the training of the data set is finished. We can use them for matching against the testing.

In the testing part we are obtaining the results for the best four strongest matches of the images. After getting the number of matches for each image we are taking only four of them for displaying.

## 3.4. Results

In the next figure we can see some outputs produced from the simple object recognition scheme, as well as the four most probable matches for every query image.

In all of the cases except one of them, the retrieved image was the correct match between the query image and the training image.

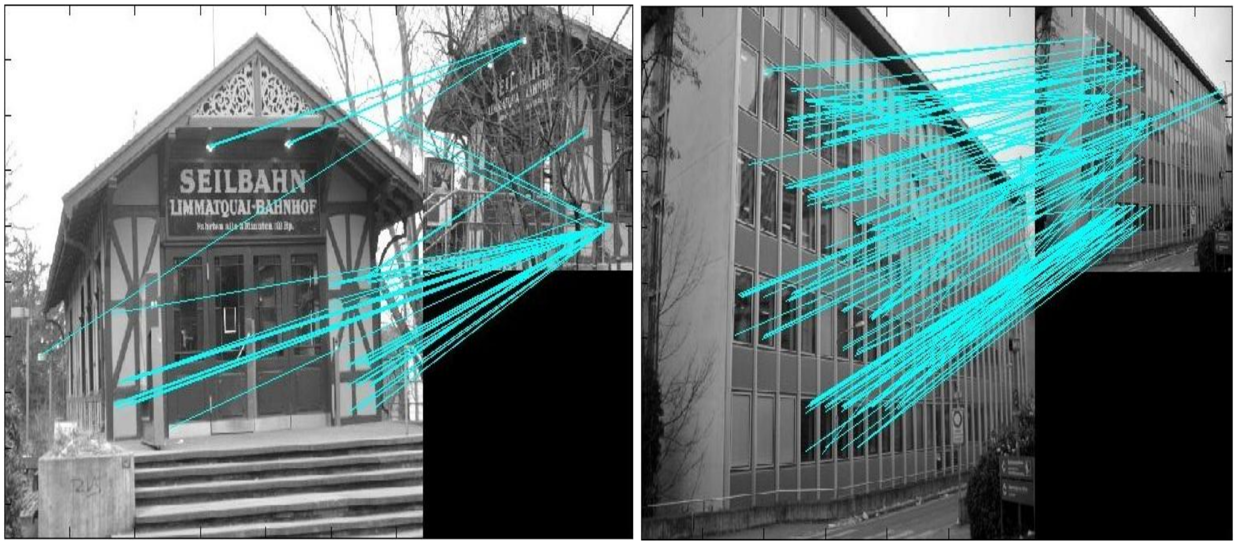


Figure 20 Simple object recognition scheme output

```
For qimg0044.png:
        1: object0002.view01.png with 214 matches
        2: object0002.view02.png with 195 matches
        3: object0005.view05.png with 174 matches
        4: object0002.view03.png with 119 matches
For qimg0045.png:
        1: object0002.view03.png with 252 matches
        2: object0002.view02.png with 182 matches
        3: object0002.view01.png with 144 matches
        4: object0002.view04.png with 131 matches
For qimg0046.png:
        1: object0002.view03.png with 87 matches
        2: object0002.view04.png with 70 matches
        3: object0002.view02.png with 41 matches
        4: object0002.view01.png with 22 matches
For qimg0047.png:
        1: object0003.view02.png with 86 matches
        2: object0003.view01.png with 80 matches
        3: object0006.view01.png with 37 matches
        4: object0006.view04.png with 36 matches
```

```
For qimg0048.png:
        1: object0003.view03.png with 54 matches
        2: object0003.view02.png with 32 matches
        3: object0003.view04.png with 29 matches
        4: object0006.view04.png with 14 matches
For qimg0049.png:
        1: object0005.view05.png with 280 matches
        2: object0005.view01.png with 178 matches
        3: object0005.view03.png with 155 matches
        4: object0005.view04.png with 143 matches
For qimg0050.png:
        1: object0004.view04.png with 293 matches
        2: object0004.view05.png with 265 matches
        3: object0004.view01.png with 179 matches
        4: object0004.view02.png with 176 matches
For qimg0051.png:
        1: object0004.view05.png with 159 matches
        2: object0004.view04.png with 131 matches
        3: object0004.view01.png with 85 matches
        4: object0004.view03.png with 78 matches
```

```
For qimg0052.png:
        1: object0005.view05.png with 228 matches
        2: object0006.view03.png with 153 matches
        3: object0005.view01.png with 130 matches
        4: object0006.view01.png with 121 matches
For qimg0053.png:
        1: object0006.view01.png with 132 matches
        2: object0006.view02.png with 120 matches
        3: object0006.view03.png with 94 matches
        4: object0006.view04.png with 91 matches
For qimg0057.png:
        1: object0007.view03.png with 397 matches
        2: object0005.view05.png with 149 matches
        3: object0005.view01.png with 76 matches
        4: object0005.view02.png with 75 matches
```

When we were doing training we observed that there is some probability of matching with the features in the testing. This is because even though we have lot of features there may be chance of having mismatches. In order to avoid this problem we can use the dense SIFT instead of original SIFT. This will give the better result because by using dense SIFT we can compute same number of features over the data.

There are some weaker features which will be problem of cornerness to get good corners we have to keep the features which have better results due to their strength and smaller chance of being outliers. Once getting the features we can use the Ransac algorithm to avoid the least probable matches.

# 5. Conclusion and Future Work

In the first part, the feature extraction and the key points matching procedures were performed using the original code from David Lowe where the Difference of Gaussians (DoG) as detector and SIFT (Scale Invariant Feature Transform) were used.

One of the most important if not the most important parameter responsible for the difference in results is the *distRatio* parameter, who we are planning to expose to additional experimentation in order retrieving best value for a specific purpose of different shifts in images.

Also, as a future work we would like to implement the dense SIFT descriptor because of its advantages over original SIFT such as same number of features in images of same size.

The comparison between using Euclidean distance and angle approximation approaches was presented, stating that Euclidean distance should be avoided because of the enormous time consumption property. Also the comparison between using ratio between first and second nearest neighbor and thresholding of nearest neighbor approaches was given, stating that effective thresholding is hard to be achieved and it is better to be avoided.

Affine invariant feature extraction is made with the help of affine detectors, such as Harris, Hessian, combination of Harris/Hessian and Laplace. Harris affine detector yields most often the best results from the abovementioned, but there are many other options which should be additionally explored in near future.

And lastly, simple object recognition scheme was developed presenting the image with highest number of matches to a given image in a given database, as well as list of four image sorted in descending order from most probable match to the input image.

**NOTE:** The scripts Lab4VP.m contains the first two steps along with the offline component of step 3 while step3onlinecomp.m contain the implementation of the online component part. The additional attached scripts are the functions called for extracting SIFT features, matching, obtaining parameters, loading features and displaying features.

## 6. References

[1] http://www.cs.ubc.ca/~lowe/keypoints/

[2] http://www.robots.ox.ac.uk/~vgg/research/affine.

[3] www.cygwin.com

[4] http://en.wikipedia.org/wiki/Scale-invariant_feature_transform

[5] http://en.wikipedia.org/wiki/Harris_affine_region_detector

[6] http://en.wikipedia.org/wiki/Hessian_affine_region_detector