Introduction to SIFT (Scale-Invariant Feature Transform)

Contents

The goal of this exercise is to learn the basic concepts of the SIFT algorithm. More specifically, you will learn to find SIFT keypoints and descriptors.

1 Preliminaries

1.1 Outline of the exercise

In this exercise, you will implement the SIFT algorithm and understand why it is better than a corner detector (implemented in the last exercise). Note that this is a simplified version of the original SIFT algorithm, enough to familiarize with the underlying concepts.

1.2 Scale-invariance

In the last exercise, we saw the Harris corner detectors. They are rotation-invariant, which means, even if the image is rotated, we can find the same corners. It is obvious because corners remain corners in rotated image also. But what about scaling? A corner may not be a corner if the image is scaled. For example, consider the simple image in Fig. 1. An image patch around a corner changes appearance when zoomed in. Hence, Harris corners are not scale invariant.

To solve this problem, Scale Invariant Feature Transform (SIFT) extracts image features from scale-invariant keypoints. Then, it describes the selected point with a history of gradient (HoG) descriptor. Given the complexity of the detection pipeline, you will implement a detector that is only scale-invariant but not rotation invariant.

2 Part 1: Keypoint detection

Since we want scale invariance, we will not only need to define the location of interest points in the image (where the descriptor is going to be calculated), but also to associate them with a scale.

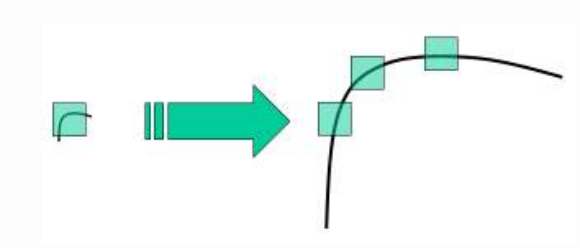

Figure 1: Harris corners are scale dependent.

As discussed in the lecture, it has been shown that the Laplacian of Gaussian kernel is optimal for this task (under certain assumptions) [Lindeberg'94]. To approximate efficiently the Laplacian of Gaussians, we use the Difference of Gaussians (DoG). DoG is obtained as the difference of Gaussian blurring of an image with two different σ; e.g. σ and $k\sigma$. The **keypoint** position and scale is found as local maxima or minima across the difference of consecutive smoothed images.

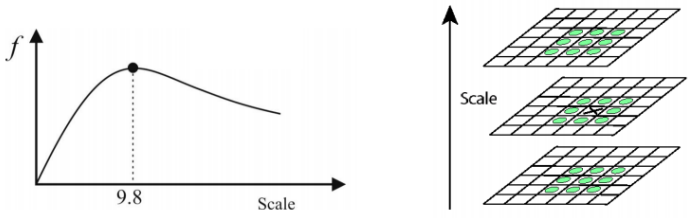

Figure 2: Interest points are at extrema of DoGs.

This process is repeated at different resolutions of the image. The result is illustrated in the following image:

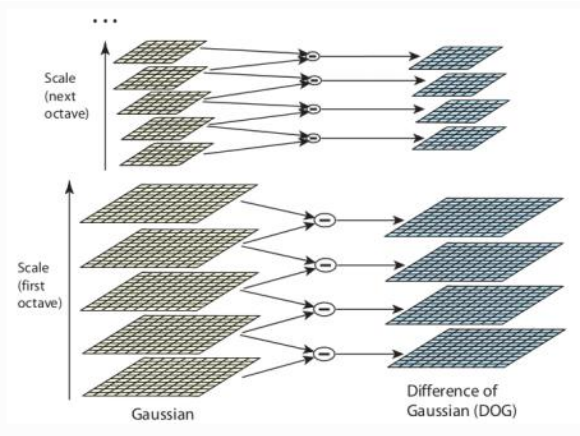

Figure 3: Different octaves of DoG filter.

Once DoGs are computed, they are searched for local extrema over scale and space. Specifically, one pixel in an layer is compared with its 8 neighbors as well as 9 pixels in next scale and 9 pixels in previous scales (Fig. 2). If it is a local extrema, it is a potential keypoint.

2.1 Generate the DoG Pyramid

A Gaussian pyramid is a set of volumes of gaussian blurred images stacked over the scale dimension (Fig. 3, in green). A DoG pyramid is formed as the difference of two subsequent layers in the

Gaussian pyramid (Fig. 3, in blue). In this exercise we will use the following parameters to generate a DoG pyramid:

1. Number of scales per octave, S=3.
2. Number of octaves, O=5.
3. Base sigma, $\sigma_o = 1.6$.

For each octave, you will generate a volume that contains 5 DoGs. What you have to do is:

1. Load the image
2. For index $s = [-1, ..., S + 1]$, blur the input image with a gaussian with standard deviation $\sigma = 2^{\frac{s}{S}}\sigma_0$ for σ_0 =1.6.
3. Compute the difference of subsequent levels (should be 5 in total), and stack them along the scale axis to produce a 3D Volume.
4. Repeat 2-3 for a total of 5 octaves. Remember that the input of octave o is the image downsampled by factor of 2^o, with $o = [0, ..., O - 1]$.

Refer to Fig. 4 for a visualization of the task. You should now have 5 3D volumes containing DoGs for all the considered octaves. We will now see how to process them to obtain interest points.

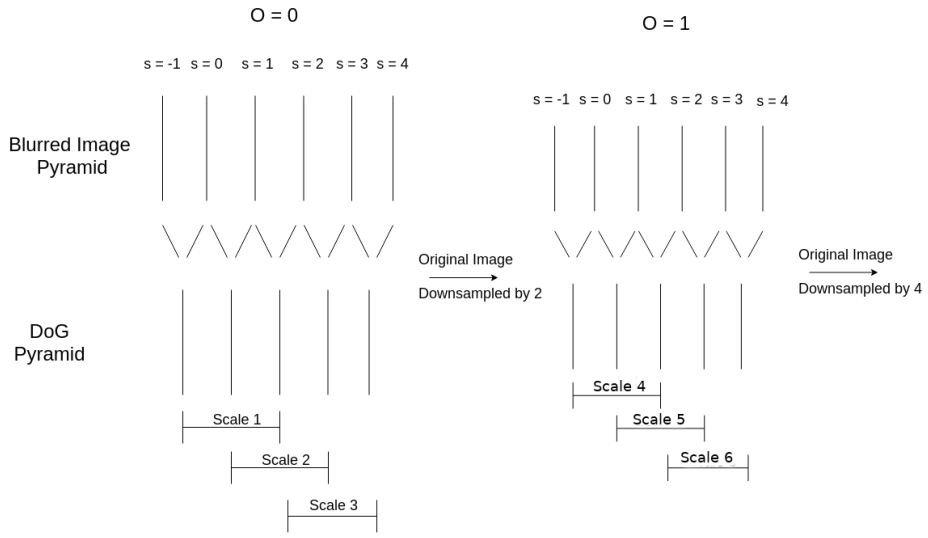

Figure 4: Generation of DoG pyramid.

2.2 Locate keypoint in scale and space

As seen in the class, a keypoint is defined as a voxel in the DoG that is higher than its neighbors in scale and space (Fig.2). However, since nearby points are likely to generate the same response to the DoG kernel we will have to first perform non-maximum suppression. As explained in the class, without non-maximum suppression we would have instability of matching, since extrema would be very dense in only few image regions. More specifically, the steps to find keypoints are (not necessarily in that order):

1. Suppress all points smaller in magnitude than a threshold $C = 0.04$, since they are likely to be generated by noise.

2. For any given point in the DoG volume, select it as keypoint only if all of the 26 (8 + 9 + 9) nearest neighbors in scale and space have a lower DoG magnitude, as depicted in Fig .2,. Discard the keypoints found in the lowest and highest layer of the DoG Pyramid. This will result in a volume of dimension $height/(o+1) \times width/(o+1) \times S$ for which the last dimension corresponds to the scale. You might want to have a look at the matlab functions *imdilate() or movmax()* to maximize the efficiency of your code.

3. Repeat the process for each octave.

3 SIFT descriptor: Histogram of Oriented Gradients

After keypoints have been localized in scale-space, a 128 dimensional descriptor is associated to each of them. This descriptor encodes the histogram of gradients in a local patch around the keypoint (Fig. 5). To implement it, you will have to do the following steps:

1. If the keypoint has been detected in octave $o \in \{0, 1, 2, 3, 4\}$ and $scale \in \{1, ..., O * 3\}$, the image used to compute gradients should be the blurred image with index $s = (scale - 1) - S * o$ from octave o.

2. Generate the norm and the orientation of the x, y-spatial gradients of the selected image. To generate them, you can use the matlab function *imgradient(I)*. Extract a 16×16 patch around the keypoint (+7 pix up-left, +8 down-right) for both the norm and orientation.

3. Scale the norm of the gradients by their distance to the keypoint center (Fig. 5). To do it, multiply elementwise the gradient norms with a gaussian centered in the keypoint and with $\sigma_w = 1.5 * 16$ (have a look at the matlab function *fspecial*).

4. Divide the 16×16 patch into 4 sub-patches of 4×4 size. For each sub-block, an 8 bin orientation histogram is created. Before creating the histogram, each gradient is weighted according to its scaled norm. Use the provided function *weightedhistc()* to generate it.

5. Concatenation of this histograms results in a 128 bin values: the **descriptor** of the keypoint. Remember to normalize the descriptor such that it has unit norm. This will make it invariant to linear illumination changes.

We recommend you to start implement this the easy way and then optimize for efficiency. **Note:** This is a simplification of the original implementation of SIFT.

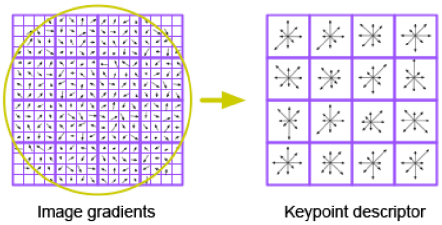

Figure 5: The SIFT descriptor is a concatenation of histogram of gradients in a patch around the keypoint.

4 Matching descriptors

After you've associated a descriptor for each keypoint, you are now ready to use it for matching keypoints in different images. There are several ways to match descriptors calculated from different images. In the last exercise, we have seen how to use an adaptive distance threshold to match descriptors from two different images. In this exercise, we will use a more robust approach based on distance comparison: *the ratio test*. Result should look like the ones in Fig. 6. Follow these steps to get matches:

1. Let us define $F_1 \in \mathbb{R}^{N\times 128}$and $F_2 \in \mathbb{R}^{M\times 128}$ the matrices containing descriptors for image 1 and image 2.
2. Use F_1 and F_2 together with the Matlab function *matchFeatures()* to generate matches. Set the argument 'MatchThreshold' to 100, 'MaxRatio' to 0.8 and 'Unique' to true.
3. Use the Matlab function *showMatchedFeatures()* to show the discovered matches.

Figure 6: Matches calculated by SIFT descriptors.

4.1 Comparison with Harris Corner Detector and Image Patch Descriptor

What are the differences between matches provided by SIFT and what you implemented last week? Are the keypoints in the same location? Compare the feature matches obtained by the two methods. What is (approximately) the ratio of true positives between the 2 methods?

4.2 (Optional) Rotational Invariance for SIFT Descriptors

Currently the SIFT descriptors are not rotation invariant. This means that key points from images with different orientation cannot be matched. Try this out by setting the parameter *rotation_img2_deg* in the main file to some nonzero value. For a relative rotation of 60° the result should look something like in Fig. 7. In this part of the exercise we will implement a method for making the descriptors rotation invariant. The main steps for this part are:

1. Compute the principal gradient direction for each feature by looking at its surrounding patch. Do this by (a) computing a histogram over orientations and (b) selecting the most common angle.
2. Compute a canonical form of the feature patch. Do this by rotating the patch so that its principal gradient direction shows to the right. You can use the provided function *derotatePatch()* which takes an image, the keypoint location, patch size (in this case 16×16) and the principal gradient direction and returns the rotated patch.

Once you have implemented this, the result should look more like in Fig. 8.

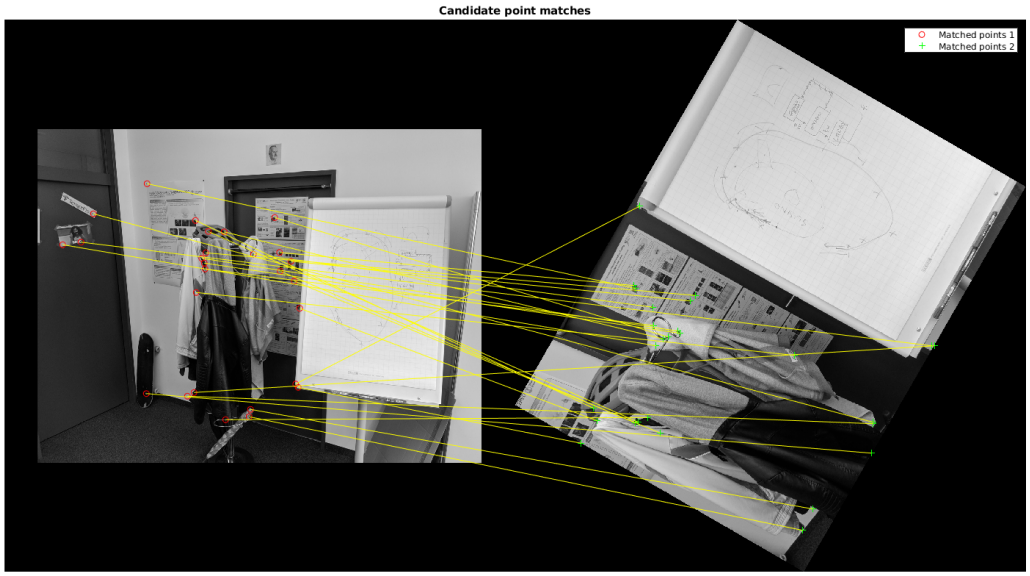

Figure 7: Wrong matches due to changing orientation.

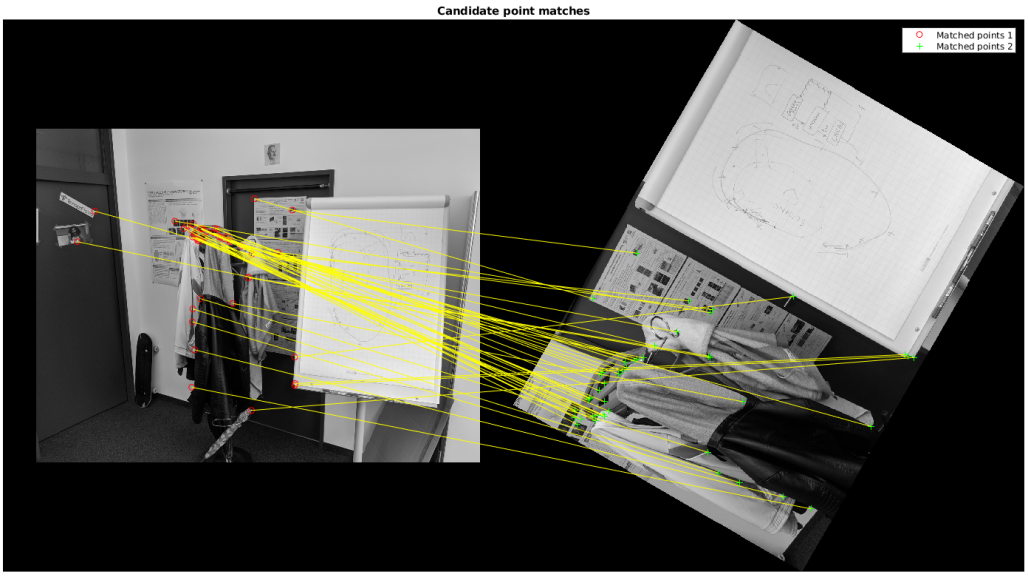

Figure 8: Correct matches due to rotation invariance.