Lecture 06: Feature Detection II

1 Feature Detection

1.1 Scale Changes

How can we match image patches corresponding to the same feature but belonging to images taken at different scales? A possible solution is to rescale the patch, i.e. bring it to the canonical scale. The problem by scale search is that it is time consuming: we need to do it **individually** for all patches in one image. In fact, the complexity would be $(NM)^2$ (assuming N features per image and M scale levels for each image). A possible solution to this problem is to assign each feature its own scale (the size).

1.1.1 Automatic Scale Selection

The process of automatic scale selection can be described through:

- Design a function on the image patch, which is scale invariant, i.e., which has the same value for corresponding regions, even if they are at different scales.
- For a point in one image, we can consider it as a function of region size (patch width).

Approach

We take a local maximum of the function: the region size for which the maximum is achieved, should be invariant to image scale. **This scale invariant region size is found in each image independently**. When the right scale is found, the patch must be **normalized**. A good function consists in single and sharp peaks. If there are multiple peaks, we need to assign **more** region sizes to have a unique feature. In this case, sharp, local intensity changes are good regions to monitor in order to identify the scale. Blobs and corners are the ideal locations.

Function

The idea beind the function for determining the scale is to convolve the image with the kernel to identify sharp discontinuities:

$$f = \text{Kernel} * \text{Image}. \tag{1.1}$$

It has been shown that the Laplacian of Gaussian kernel is optimal under certain assumptions

$$\text{LoG} = \nabla^2 G(x,y) = \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2}. \tag{1.2}$$

Then, the correct scale is found as local maxima across consecutive smoothed images. This should be done for **several**s region sizes.
Note that an efficient implementation of multisale detection uses the so called scale-space pyramid: instead of varying the window size of the feature detector, the idea is to generate upsampled (enlarge the image, interpolating) or downsampled versions of the same image.

2 Feature Descriptors

We already know how to detect points, but how can we describe them for matching? We present here two methods:

- **Simplest Descriptor**: **Intensity** values within a squared patch or gradient histogram.
- **Census transform** or **Histograms of Oriented Gradients**.

Once the descriptors are generated, descriptor matching can be done using **Hamming Distance (Census)** or **(Z)SSD,(Z)SAD**, **(Z)NCC**. We would like to find the same features regardless of the **transformation** that is applied to them. In fact, most feature methods are designed to be invariant to

- 2D translation,
- 2D rotation,
- scale.

Some of them can also handle

- Small view point invariance (e.g. SIFT works up to about 60 degrees).
- Linear illumination changes.

2.1 How to achieve Invariance?

2.1.1 Re-scaling and De-rotation

The approach reads:

- Find the **correct scale** using the LoG operator.
- **Rescale** the patch to a default size (e.g. 8×8 pixels).
- Find the **local orientation**, i.e. the dominant direction of gradients for the image patch (Harris eigenvectors).
- **De-rotate** the patch.

In order to de-rotate the patch, one uses **patch-warping**.

Patch Warping

1. Start with an **empty** canonical patch (all pixels set to 0).
2. For each (x, y) in the empty patch, apply the **warping function** $W(x, y)$ to compute the corresponding position in the detected image. It will be in floating point and will fall between the image pixels.
3. Interpolate the intensity values of the 4 closest pixels in the detected image with
   - Nearest neighbor or

- Bilinear interpolation

Example 1. Rotational Warping.
Counterclockwise rotation:

$$\begin{aligned} x' &= x\cos(\theta) - y\sin(\theta) \\ y' &= x\sin(\theta) + y\cos(\theta) \end{aligned} \tag{2.1}$$

Bilinear Interpolation

It is an extension of the linear interpolation, for interpolating functions of two variables on a rectilinear 2D grid. The key idea is to perform linear interpolation in one direction and then, again, in the other direction. Although each step is linear in the sampled values and in the position, the interpolation as a whole is not linear but rather quadratic in the sample location. We have that

$$I(x,y) = I(0,0)\cdot(1-x)\cdot(1-y) + I(0,1)\cdot(1-x)\cdot y + I(1,0)\cdot x\cdot(1-y) + I(1,1)\cdot xy \tag{2.2}$$

Example 2. Affine Warping.
To achieve **slight view-point invariance**:

- The second moment matrix M can be used to identify the two directions of fastest and slowest change of intensity around the feature.
- Out of these two directions, an elliptic patch is extracted at the scale computed with the LoG operator.
- The region inside the ellipse is normalized to a circular one.

However, there exist some disadvantages in using patches as descriptors:

1. If patches are not warped, very small errors in rotation, scale and view-point will affect the matching score significantly.
2. The procedure is computationally expensive (we need to unwarp every patch).

A better solution nowadays are Histrograms of Oriented Gradients HOGs.

Histogram of Oriented Gradients

- Compute a histogram of orientations of intensity gradients.
- Peaks in the histogram represent dominant orientations.
- **Keypoint orientation= histogram peak**. If there are multiple candidate peaks, construct a different keypoint for each such orientation.
- **Rotate patch** according to this angle: this puts the patches into a canonical form.

2.2 Scale Invariant Feature Transform (SIFT) Descriptor

The uniqueness of SIFT is that these features are extremely distinctive and can be successfully matched between images with very different illumination, rotation, viewpoint, and scale-changes. The SIFT algorithm performs:

- Identification of keypoint **location and scale**
- Orientation assignment
- Generation of keypoint descriptor

Descriptor computation is performed as follows:

1. Divide the patch into 4×4 sub-patches=16 cells.
2. Compute HOG (8 bins, i.e. 8 directions) for all pixels inside each sub-patch.
3. Concatenate all HOGs into a single 1D vector. This is the resulting SIFT descriptor: $4 \times 4 \times 8 = 128$ values.
4. Descriptor matching: SSD (euclidean-distance).

2.2.1 Intensity Normalization

The descriptor vector v is then normalized such that its l_2 norm is 1:

$$\bar{v} = \frac{v}{\sqrt{\sum_i^n v_i^2}}. \tag{2.3}$$

Remark. This guarantees that the descriptor is invariant to linear illumination changes. This was already invariant to additive illumination because it is based on gradients.

2.2.2 SIFT Matching Robustness

- Can handle changes in viewpoint (up to 60 degree out-of-plane rotation).
- Can handle significant changes in illumination (low to bright scenes).
- Expensive: 10fps.

In order to reduce the computational cost, one can use difference of Gaussian instead of Lapiacian:

$$\text{LOG} \approx \text{DOG} = G_{k\sigma}(x, y) - G_\sigma(x, y) \tag{2.4}$$

2.2.3 SIFT Detector

SIFT keypoints are local extrema (maxima and minima) in both **space and scale** of the DoG images:

- Detect maxima and minima of difference-of-Gaussian in scale space.
- Each point is compared to its 8 neighbors in the current image and 9 neighbors each in the scales above and below.

- For each maxima and minima found, the output is its location and the scale: this is a candidate keypoint.

Is this similar to Harris? While in Harris (keypoint location) the keypoint is identified in the image plane as local maximum of the corner function, in SIFT the keypoint is a local minimum or maximum of the DoG image in both position **and** scale.

Implementation:

1. The initial image is **incrementally convolved with Gaussians** $G(k\sigma)$ to produce images separated by a constant factor k in scale space.
   (a) The initial Gaussian $G(\sigma)$ has $\sigma = 1.6$.
   (b) k is chosen such that $k = 2^{\frac{1}{s}}$, where s in an integer (typically $s = 3$).
   (c) For efficiency reasons, when k reaches 2, the image is downsampled by a factor of 2 and then the procedure is repeated up to 4 or 6 octaves (pyramid levels).
2. Adjacent image scales are then subtracted to produce the difference-of-Gaussian (DoG) images.

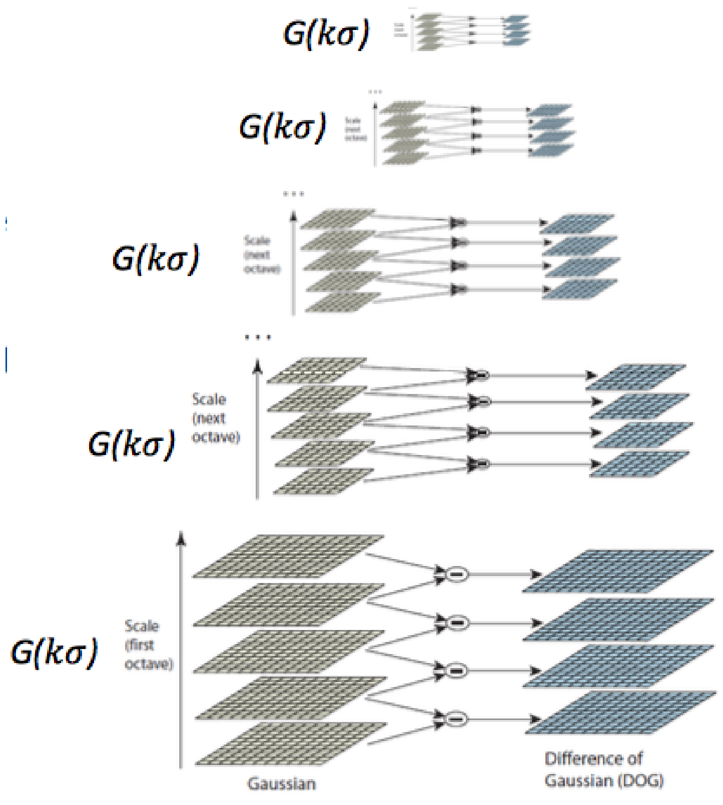


Figure 1: Difference of Gaussian.

Summary

- An approach to detect and describe regions of interest in an image.
- SIFT detector = DoG detector.
- SIFT features are reasonably invariant to changes in rotation, scaling, and changes in viewpoint (up to 60deg) and illumination.
- Real time but still **slow** (10Hz on an i7 laptop).

The **repeatability** can be expressed as

$$\frac{\text{number of correspondences detected}}{\text{number correspondences present}}. \tag{2.5}$$

The highest repeatability is obtained when sampling 3 scales per octave.
Influence of Number of Orientation and Number of Sub-Patches: Single orientation histogram is poor at discriminating, but the results continue to improve up to a 4x4 array of histograms with 8 orientations.

- **Descriptor**: 4x4x8 = 128-element 1D vector.
- **Location**: 2D vector.
- **Scale** of the patch. 1 scalar value.
- **Orientation** (angle of the patch). 1 scalar value.

SIFT for object recognition

Can be simply implemented by returning as best object match the one with the largest number of correspondences with the template (object to detect). 4 or 5 point RANSAC can be used to remove outliers.

2.3 Feature Matching

Given a feature I_1, how to find the best match in I_2?

1. Define distance function that compares two descriptors (Z)SSD,SAD,NCC, or Hamming distance for binary descriptors (e.g. Census, BRIEF, BRISK).
2. **Brute-force matching**:
   - Test all the features in I_2.
   - Take the one at minimal distance.

2.3.1 Issues with Closest Descriptor

Can give good scores to very ambiguous (bad) matches (curse of dimensionality). A better approach would be to compute the ratio of distances between the *first* and the *second* match:

$$\frac{d(f_1)}{d(f_2)} < \text{Threshold (usually 0.8)}, \tag{2.6}$$

where

$$\begin{aligned} &d(f_1) \text{ is the distance of the closest neighbor} \\ &d(f_2) \text{ is the distance of the second closest neighbor.} \end{aligned} \tag{2.7}$$

But why is the distance ratio relevant? In SIFT, the nerest neighbor is defined as the keypoint with minimum euclidean distance. However, many features from a first image may not have any correct match in a second image because they arise from background clutter or were *not detected at all* in the first image. An effective measure is obtained by comparing the distance of the closest **neighbor** to that of the second closest neighbor. Why? Correct matches need to have the closest neighbor significantly closer than the closest incorrect match, to achieve reliable matching. Moreover, for **false matches**, there will likely be a number of other false matches within similar distances due to the high dimensionality of the feature space (also known as **curse of dimensionality**). We can think of the second closest match as the one providing an estimate of the density of false matches within this portion of the feature space, and at the same time identifying specific instances of feature ambiguity. Furthermore, the factor 0.8 has been selected because:

- Eliminates 90% of the false matches,
- Discards less than 5% of the correct matches.

2.4 Other Detectors and Descriptors

2.4.1 SURF (Speeded Up Robust Features)

- Based on ideas similar to SIFT.
- Approximated computation for detection and descriptor using box filters.
- Results are comparable with SIFT but
  - Faster computation and
  - Shorter descriptors.

2.4.2 FAST detector (Features from Accelerated Segment Test)

- Studies intensity of pixels around candidate pixel C.
- C is FAST corner **if** a set on N contiguous pixels on circle are
  - all brighter than $intensity(C) + threshold$ or
  - all darker than $intensity(C) + threshold$.
- Typically tests for 9 contiguous pixels in a 16 pixel circumference.
- **Very fast detector** (100 Mega pixel/second).

2.4.3 BRIEF descriptor (Binary Robust Independent Elementary Features)

- Goal: high speed.
- **Binary** descriptor formation
  - Smooth image,
  - For each detected keypoint (e.g. with FAST) **sample** 256 intensity pairs (p_1^i, p_2^i), $i = 1 - 256$ within a squared patch around keypoint.
  - Create an empty 256-element descriptor.
  - For each i-th pair:
    - if $I_{p_1^i} < I_{p_2^i}$, then set i-th bit of descriptor to 1.
    - else to 0.
- The **pattern is generated randomly** (or by ML) only once: then, same pattern is used for all patches.
- Pros: **Binary Descriptor** allows very fast Hamming distance matching: count the number of bits that are different in the descriptors matched.
- Cons: **Not scale/rotation invariant**.

2.4.4 ORB descriptor (Oriented FAST and Rotated BRIEF)

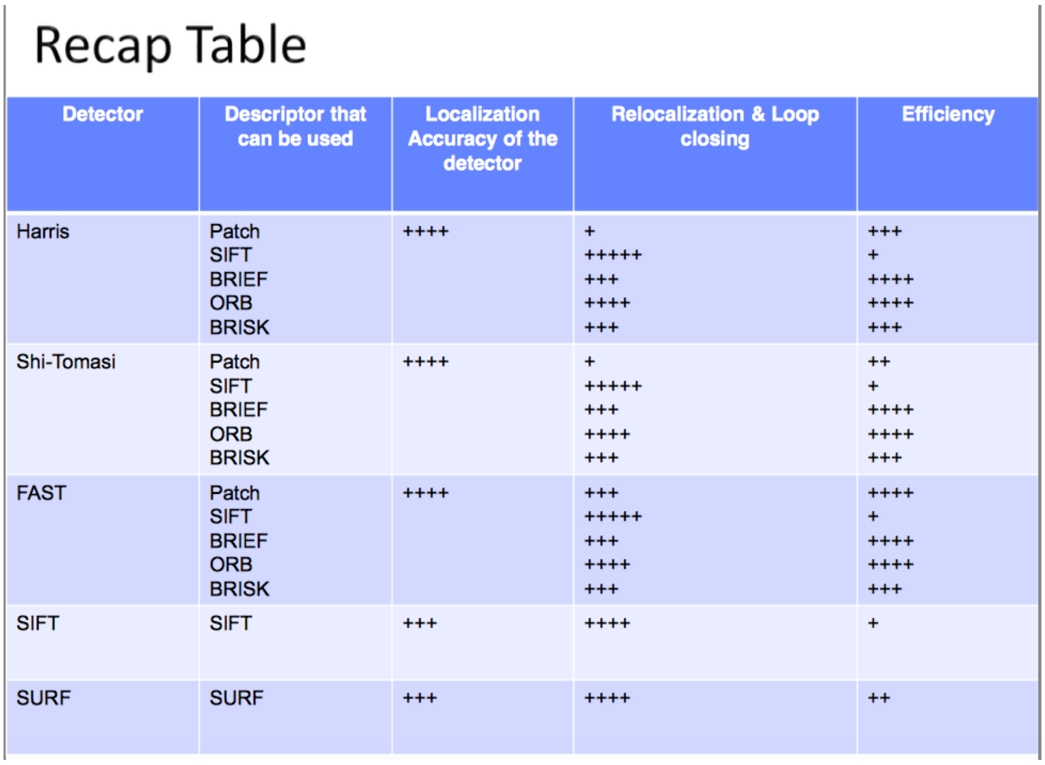
- Keypoint detector based on FAST.
- BRIEF descriptors are steered(?) according to keypoint orientation (to provide rotation invariance).
- Good binary features are learned by minimizing the correlation on a set of training patches.

2.4.5 BRISK descriptor (Binary Robust Invariant Scalable Keypoints)

- **Binary**: formed by pairwise intensity comparisons.
- **Pattern** defines intensity comparisons in the keypoint neighborhood.
- **Red circles**: size of the smoothing kernel applied.
- **Blue circles**: smoothed pixel values used.
- Compare short- and long-distance pairs for orientation assignment and descriptor formation.
- Detector and descriptor speed: circa **10 times faster than SURF**.
- Slower than BRIEF, but scale- and rotation- invariant.

2.4.6 Recap

Finally, the different introduced methods are compared in Figure **??** and Table 1



Recap Table

Detector	Descriptor that can be used	Localization Accuracy of the detector	Relocalization & Loop closing	Efficiency
Harris	Patch SIFT BRIEF ORB BRISK	++++	+ +++++ +++ ++++ +++	+++ + ++++ ++++ +++
Shi-Tomasi	Patch SIFT BRIEF ORB BRISK	++++	+ +++++ +++ ++++ +++	++ + ++++ ++++ +++
FAST	Patch SIFT BRIEF ORB BRISK	++++	+++ +++++ +++ ++++ +++	++++ + ++++ ++++ +++
SIFT	SIFT	+++	++++	+
SURF	SURF	+++	++++	++

Figure 2: Recap for detectors and descriptors.

Detector/Descriptor	Brief Overview	Pros	Cons
Harris detector	corner	Rotation invariant	No blob detection, not scale and affine invariant
SIFT both	Blob	Rotation, scale and affine invariant	no corner detection, inefficient
SURF both	Speeded Up Robust Features. Based on ideas similar to SIFT. Blob detector. Approximated computation for detection and descriptor using box filters.	Rotation, scale and affine invariant, faster computation, shorter descriptor	No corner detection, inefficient
FAST detector	Feastures from Accelerated Segment Test. Studies intensity of pixels around candidate pixel C. C is FAST corner **if** a set on N contiguous pixels on circle are all brighter than $intensity(C) + threshold$ or all darker than $intensity(C) + threshold$.	Very fast detector	no blob detection, not scale and affine invariant
BRIEF descriptor	Binary Robust Independent Elementary Features. For **binary** see summary. The **pattern is generated randomly** (or by ML) only once: then, same pattern is used for all patches.	Very fast Hamming distance matching: count the number of bits that are different in the descriptors matched.	Not scale or rotation invariant
ORB descriptor	Oriented FAST and Rotated BRIEF. Keypoint detector based on FAST. BRIEF descriptors are steered according to keypoint orientation	Binary features are learned by minimizing the correlation on a set of training pathces	
BRISK descriptor	Binary Robust Invariant Scalable Keypoints. Detect corners in scale-space using FAST. compare short and long distance pairs for orientation assignment and descriptor formation	Rotation and scale invariant, 10 times faster than SURF	

Table 1: Recap for detectors and descriptors

2.5 Understanding Check

Are you able to answer the following questions?

- *How does automatic scale selection work?*
- *What are the good and the bad properties that a function for automatic scale selection should have or not have?*
- *How can we implement scale invariant detection efficiently? (show that we can do this by resampling the image vs rescaling the kernel).*
- *What is the Harris Laplacian and what is its repeatability after a rescaling of 2?*
- *What is a feature descriptor? (patch of intensity value vs histogram of oriented gradients). How do we match descriptors?*
- *How is the keypoint detection done in SIFT and how does this differ from Harris Laplacian?*
- *How does SIFT achieve orientation invariance?*
- *How is SIFT descriptor built?*
- *What is the repeatability of the SIFT detector after a rescaling of 2? And for a 50 degrees viewpoint change?*
- *Illustrate the 1st to 2nd closest ratio of SIFT detection: whats the intuitive reasoning behind it? Where does the 0.8 factor come from?*