Key point detection and matching

3D Computer Vision for Medical Applications

Ferdi van der Heijden ▸ University of Twente - RAM ▸ 11/12/2016

Contents

Key point detection and matching

3D Computer Vision for Medical Applications

1 Introduction

Key points, or interest points, are points in the image plane that are easily localized and easily identified. The latter property assures that if such a point is found in an image, the corresponding point in another, similar image of the scene can be found easily. Applications of key point detection and matching are found in stereo vision, visual navigation, panoramic image stitching, and object recognition.

The organization of the syllabus is as follows. Section 2 deals with the question how to detect so-called corners in an image. Another class of interest points are point features. These are introduced in Section 3. The association of key points found in one image can be associated with key points in another image. The process of associating is called *key point matching*. It is discussed in Section 4. An example of an application, object recognition, is provided in Section 5.

2 Key point detection

There are roughly two approaches to find key points. The oldest method is to find so called corner points. More recent approaches focus on finding blobs.

2.1 Corner detection

The name “corner detection” is misleading as the algorithms in this category not only detects corners, but also blobs. The basic principle is that a pixel is considered a key point if, and only if, in the neighborhood of that pixel the grey levels change in at least two different directions.

Considering the characteristic image patches above, the image “no contrast” does not show any changes at all. So, it doesn’t contain a key point. The images “edge” and “line” do show a change of grey level, but only in one direction. The image “blob” shows changes of grey level in any direction, whereas the image “corner” mainly shows changes in two directions. The last two images have key points in the center.

The first attempt to find key points was by Moravec (1980). The idea was to check if a shift of the image in a given direction would lead to large changes of the grey levels surrounding a candidate key point. For that, a criterion was proposed: the sum of squared differences (SSD) between the original image and the shifted version. If the shift is defined by the translation vector (p,q), then the shifted version of an image $f(n,m)$ is $f(n+p,m+q)$. The SSD (sum of squared differences) becomes:

$$E(p,q)=\sum_{n,m} w(n_0-n,m_0-m)\big(f(n,m)-f(n+p,m+q)\big)^2 \tag{1}$$

(n_0,m_0) is the position of a candidate key point. $w(n,m)$ is a window function. The expression $w(n-n_0,m-m_0)$ selects points in the neighborhood of (n_0,m_0). If a pixel at position $(n-n_0,m-m_0)$ is within the window, that is, when $w(n-n_0,m-m_0)=1$, then that pixel fully contributes to the SSD. If for that pixel $w(n-n_0,m-m_0)=0$, then that pixel is fully ignored. The summation of pixels that are within the window makes the operation less noise sensitive.

The candidate point is a key point if $E(p,q)$ is large in any shift direction (p,q). For instance, one could test whether $E(p,q)$ is large enough for each of the vertical and horizontal shifts. The four different cases $(p,q)=(+1,+1)$, $(-1,+1)$, $(+1,-1)$, and $(-1,-1)$ should be tested.

Harris and Stephens (1988) improved the ideas of Moravec by applying an approximation of (1) based on a truncated Taylor series expansion:

$$f(n+p,m+q)\approx f(n,m)+p\,f_x(n,m)+q\,f_y(n,m) \tag{2}$$

In vector-matrix notation, the squared difference becomes:

$$\big(f(n,m)-f(n+p,m+q)\big)^2\approx\begin{bmatrix}p & q\end{bmatrix}\begin{bmatrix}f_x^2 & f_xf_y\\ f_xf_y & f_y^2\end{bmatrix}\begin{bmatrix}p\\ q\end{bmatrix}$$

f_x and f_y are shorthand notations for $f_x(n,m)$ and $f_y(n,m)$. The summation within the window function is taken care of by introducing a matrix $\mathbf{M}(n_0,m_0)$:

$$\mathbf{M}(n_0,m_0)=\sum_{n,m} w(n-n_0,m-m_0)\begin{bmatrix}f_x^2(n,m) & f_x(n,m)f_y(n,m)\\ f_x(n,m)f_y(n,m) & f_y^2(n,m)\end{bmatrix} \tag{3}$$

Harris and Stephens suggested to replace the binary window function by a Gaussian:

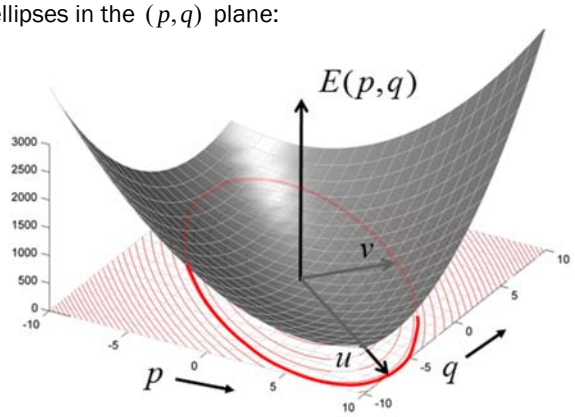
$$w(n_0-n,m_0-m)=\exp\left(-\frac{(n-n_0)^2+(m-m_0)^2}{2\sigma^2}\right) \tag{4}$$

so that pixels that are at a larger distance from (n_0, m_0) are counted less.

With the introduction of $\mathbf{M}(n_0, m_0)$, the SSD becomes:

$$E(p,q) = \begin{bmatrix} p & q \end{bmatrix} \mathbf{M}(n_0, m_0) \begin{bmatrix} p \\ q \end{bmatrix} \qquad (5)$$

Equation (5) defines a paraboloid. Its contour lines are ellipses in the (p,q) plane:



These ellipses have two principal axes which can be used to define a local coordinate system (u,v). In this local coordinate system the paraboloid is described by:

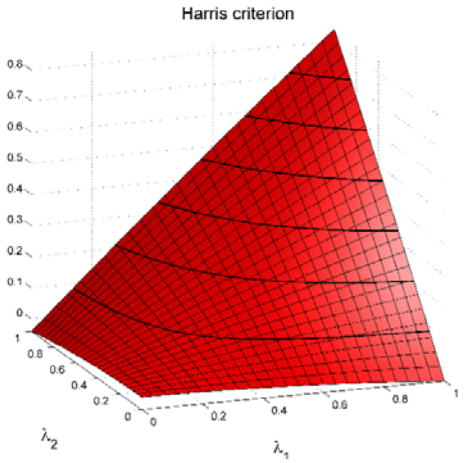
$$E(p,q) \sim \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \lambda_1 u^2 + \lambda_2 v^2 \qquad (6)$$

λ_1 and λ_2 are the eigenvalues of the matrix $\mathbf{M}$.

Roughly speaking, the paraboloid can have 3 different shapes:

- Very blunt, i.e. almost flat. This happens if both eigenvalues are small. This represents the situation that the neighborhood of (n_0, m_0) doesn't show much contrast.
- A valley. This happens if one eigenvalue is very small, and the other is large. In that case there is one direction in which the grey levels don't change much, whereas in the orthogonal direction it does change. This represents elongated structures such as edges and lines.
- A high-pitched shape. This happens if both eigenvalues are large. In that case, the grey levels change much in any direction. This represents structures like corners and spots.

Clearly, the latter case is what we are looking for. To identify this case, Harris used the criterion that $\lambda_1\lambda_2 - 0.04(\lambda_1 + \lambda_2)^2$ should be large enough. This criterion is called the *corner responsity* or the *cornerness* of the image. The figure below shows indeed, that if λ_1 and λ_2 are both large, then this corner responsity is large, whereas if λ_1 or λ_2 or both are small, then the corner responsity is small.



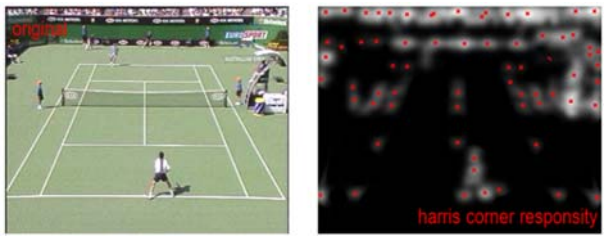
The motivation for this definition of corner responsity as a criterion is that it can also be expressed in terms of the derivatives of images:

$$\begin{aligned} C_{HARRIS} &= \lambda_1\lambda_2 - 0.04(\lambda_1 + \lambda_2)^2 \\ &= \det(\mathbf{M}) - 0.04\,\mathrm{trace}^2(\mathbf{M}) \\ &= \overline{f_x^2}\,\overline{f_y^2} - \left(\overline{f_x\, f_y}\right)^2 - 0.04\left(\overline{f_x^2} + \overline{f_y^2}\right)^2 \end{aligned} \qquad (7)$$

where the bar, e.g. $\overline{f_x^2}$, stands for the Gaussian smoothing of f_x^2. Harris algorithm to calculate the corner responsity is:

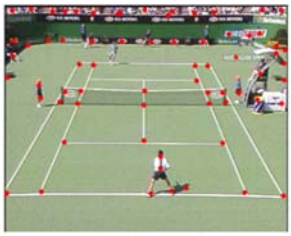
1. calculate the derivatives f_x and f_y of the image
   (by using Gaussian filtering)
2. calculate the elements of the matrix $\mathbf{M}$:
   $$M_{1,1} = f_x^2 * gauss(n,m)$$
   $$M_{2,2} = f_y^2 * gauss(n,m)$$
   $$M_{1,2} = M_{2,1} = f_x f_y * gauss(n,m)$$
3. calculate the corner responsity : $R = \left(M_{1,1}M_{2,2} - M_{1,2}^2\right) - 0.04\left(M_{1,1} + M_{2,2}\right)^2$
4. Corners are detected at pixels in R where:
   - R exceeds a threshold
   - R is a local maximum

An example:



The white blobs represents the responsity. The red dots are local maxima after thresholding. In the original image these points are indeed situated near corner-like pixels:

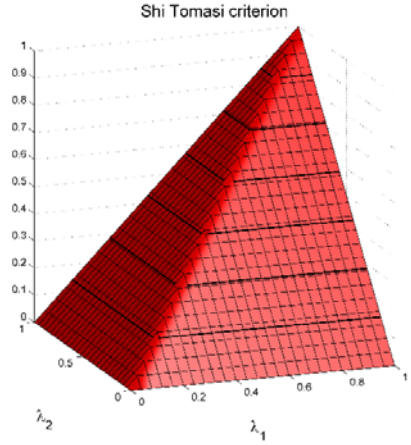
result:



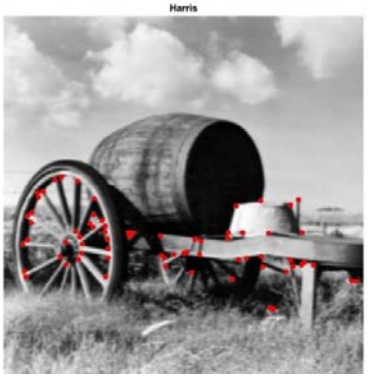
Shi and Tomasi (1994) proposed a slight modification of the Harris criterion, namely the minimum of the two eigenvalues:

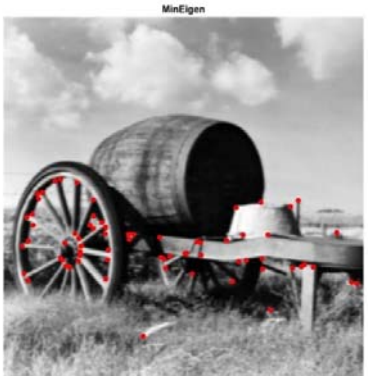
$$C_{SHI-TOMASI} = \min(|\lambda_1|, |\lambda_2|) \quad (8)$$

The justification for this is that if the minimum eigenvalues is large, then both eigenvalues are large. The criterion is shown below:



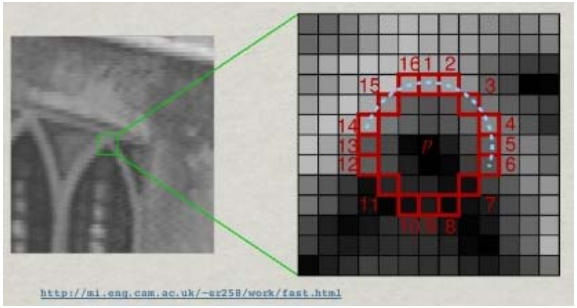
This criterion, in Matlab called the '*minimum eigenvalue criterion*', is claimed to have better performance than Harris. However, comparison of the two graphs shows that the difference will not be large. This is confirmed in the example shown below.





2.2 Key points based on bit patterns

Another approach to see whether a pixel is a key point is to consider a selection of pixels surrounding this central pixel. In the case of the so-called FAST algorithm, this selection consists of 16 pixel chosen on a circle with a radius of 3.



The bit pattern associated with the central pixel is constructed by checking whether the grey levels on the circle deviate more than a threshold t from the central pixel. This yields a binary pattern of 16 bits. These bits are used to decide whether the central pixel is a key point or not. The decision is based on a decision tree that has been trained with machine learning algorithms. An example of detected FAST key points is shown below.

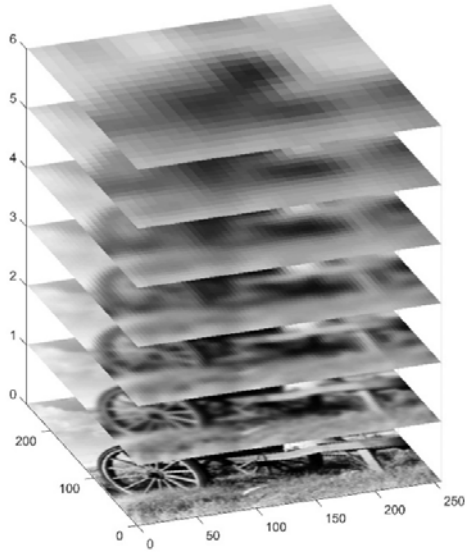
2.3 Key points based on scale space

The key point detectors, discussed so far, are approximately rotation invariant, but they are not scale invariant. That is, if the image is rotated, the same key points will be found, but if the image is zoomed in or zoomed out, this is not guaranteed. Scale spaces can take care of scale invariance.

Scale space theory provides the means to transform a 2D grey level image into a 3D image data structure called the *scale space*. This enables *scale invariant* key point detection. An example of a scale space is shown below. The supplemented third dimension is the *scale* s.

Key points are detected as a 3D point (x, y, s) somewhere within this 3D data structure. Moving the camera along the optical axis backwards from the scene has the same effect to the images as moving upwards along the scale axis. Therefore, changing the distance to the scene effects in a shift of the key point along the scale axis; the pixel position (x, y) remains the same. Therefore, this method is *scale invariant*.

The original algorithm that used this principle is SIFT (Scale Invariant Feature Transform), proposed by Lowe (1999). SIFT uses the Laplacian of the image, i.e. $f_{xx} + f_{yy}$, calculated at the various scales of the input image using Gaussian filtering. A key point is detected if in this Laplacian scale space a pixel is larger than all his 26 neighbors, or smaller than all of these neighbors. Details are provided in Appendix 1A



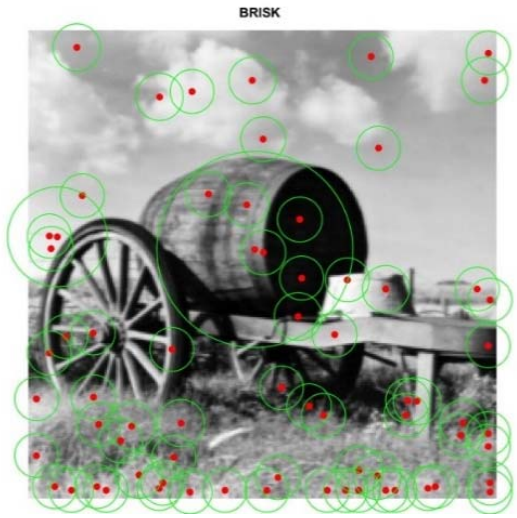
An example of SIFT key points is given below. The scale at which the key points are found is indicated by the circle.

2.3.1 Other key points

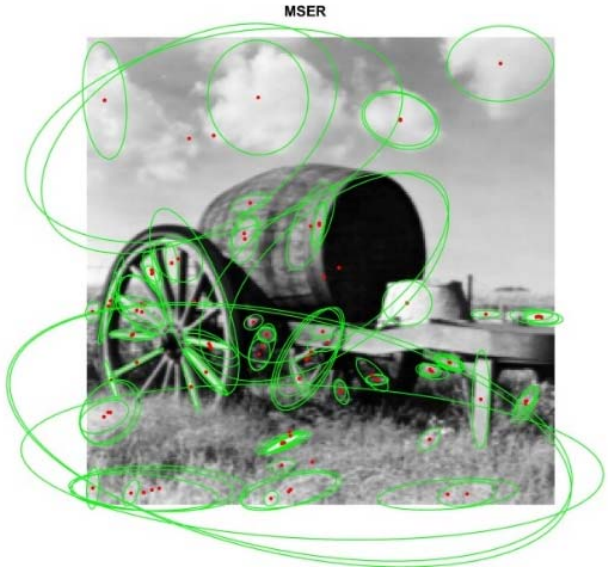
After Lowe's publication on SIFT, in 1999, several other algorithms were published to find key points. The motivation for this is to speed up the calculations of key points, and perhaps also to bypass the patent that prohibited the commercial use of SIFT.

One of the more well-known algorithms is SURF (Speeded Up Robust Features). The main difference with SIFT is that it uses (an approximation of) the determinant of the Hessian matrix, i.e. $f_{xx} f_{yy} - f_{xy}^2$ rather than the Laplacian $f_{xx} + f_{yy}$. In addition, it uses Box filters (local average filters) instead of Gaussians to speed up the calculations.

Another algorithm is BRISK (Binary Robust Invariant Scalable Keypoints). This algorithm is more or less a multi-scale implementation of the FAST algorithm.



Finally, we mention the MSER (maximally stable extremal regions) algorithm. Here, so-called extremal regions are found in an image $f(x,y)$ by comparing the grey levels with a threshold t. This results in a binary image that, of course, depends on t. We write $g(x,y,t)=\big(f(x,y)>t\big)$. With a given t, the binary image has a number of connected components which are called here *extremal regions*. By varying t, these regions shrink, expand, vanish, or merge with other regions. A maximally stable extremal region is a region found at a threshold t where the growth of this region is, when varying t, is minimal. This region is then approximated by the best fitting ellipse. The procedure can be repeated in a Gaussian scale space to obtain multiple scale detection.



3 Feature vectors

A *feature vector*, also called a *descriptor*, is a vector that serves as a signature of the associated key point.

Descriptors are formed by extraction of properties of grey levels in the vicinity of the key points. Desirably, these properties are unique for a given key point, so that the key point can be uniquely identified. The properties should not be influenced too much by noise since then the identifiability of the key point is lost. In addition, the feature should not be influence by a change of orientation and scale (and perhaps affine transform and projective transform).

There are different ways to get a feature. Examples are SIFT, SURF, BRISK, etc. Most algorithms for detecting key points have also their own features. So we have SIFT features, SURF features, and so on. However, the detection algorithm can be crossed-fertilized with other feature types. A key point that is found with, for instance, SURF can be associated with, for instance, a BRISK feature.

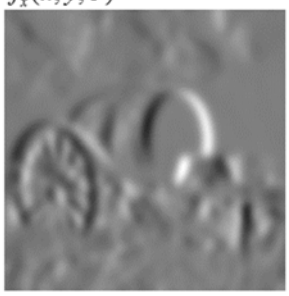
3.1 Features from histograms of gradient directions

The seminal work of Lowe, with his SIFT key points, used features that were derived from histograms of gradient orientation.

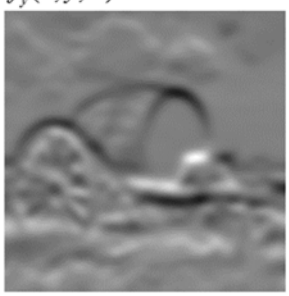
The approach of SIFT to get a feature is to first estimate the gradients $f_x(x,y,\sigma)$ and $f_y(x,y)$ of the image at the same scale σ at which the key point was found. Below an example:

original image at scale σ=4.5
i.e. $f(x,y,\sigma)$

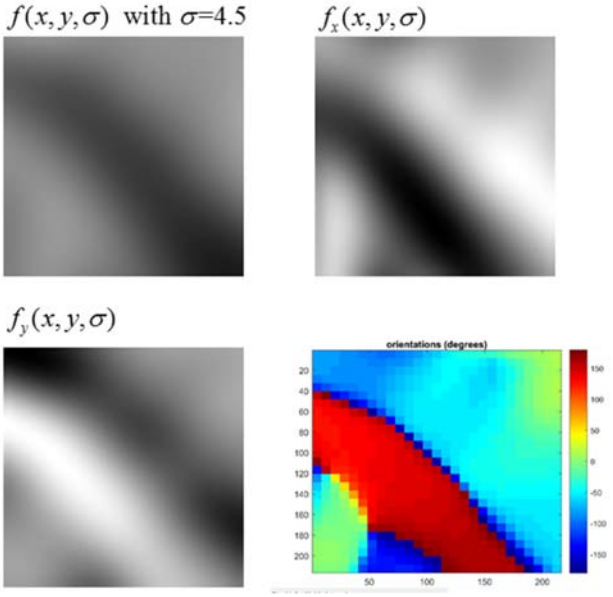
$f_x(x,y,\sigma)$



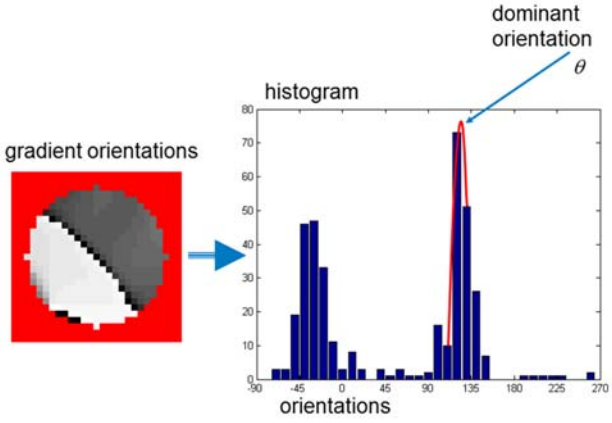
$f_y(x,y,\sigma)$



This gradient images provide information about the local orientation of the data. Suppose that we have a key point on the top right side of the wheel at $\sigma=4.5$, then locally the images look like:

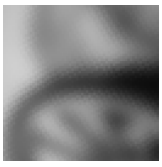


The first step will be to extract the dominant orientations around the key point. The can be found by calculating a histogram of the orientations of the gradient vectors within a circular neighborhood of the key point.

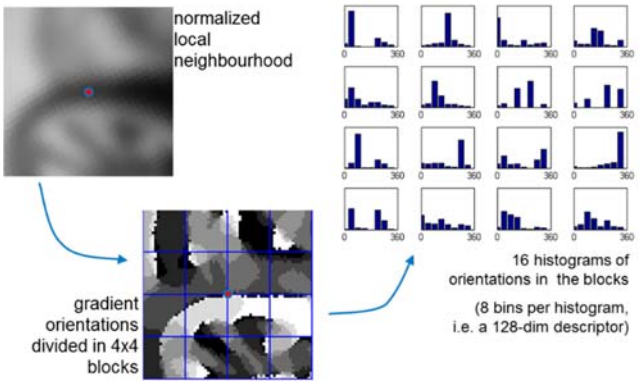


The example here has a dominant orientation of about 130 degrees. Together with the scale, the found orientation is associated with the key point as two attributes.

The next step is to normalize the image with respect to the orientation, so that the resulting feature vector becomes rotational invariant. This is done by rotating the image around the key point with an angle that is opposite to the dominant orientation:



If the camera would have been rotated along the optical axis, the resulting image, after normalization, would have been the same. Hence, the rotational invariance.



The final step is the construction of the feature vector. For that purpose, the image with orientations is also normalized with respect to orientation. This image is divided into 4x4 blocks. Within each block a histogram is calculated with 8 bins. Hence each bin covers an interval of 45 degrees. There are 16 histogram with each 8 bins. In full there are 128 counts. These counts form a 128 dimensional vector which is the feature vector.

Note that by using gradient directions, the descriptor is invariant for the contrast and brightness of the image. That is, the gradient directions are not affected by the operation $Af(x,y)+B$.

3.2 Other features

The feature extraction, described above, is more or less the procedure of the SIFT features. In the last decade a number of alternatives have been proposed. Some are variations of the SIFT features, e.g. the SURF features. The alternative approach is to calculate bit patterns by comparing the grey levels of pairs of pixels in the neighborhood. This is the approach of a variety of algorithms: BRIEF, ORB, BRISK and FREAK. They differ in computational complexity and invariance for rotations.

4 Feature matching

Suppose that we have two consecutive frames of a video, and that from both frames sets of key points have been detected along with their features. Denote the key points of the first frame by $\mathbf{x}_1(n)$ with $n=1,\cdots,N$, and the key points from the second frame by $\mathbf{x}_2(m)$ with $m=1,\cdots,M$. The corresponding feature vectors are $\mathbf{z}_1(n)$ for the first frame, and $\mathbf{z}_2(m)$ for the second frame. Note that some key point types, such as SIFT and SURF, also returns the orientations

$\theta_1(n)$ and $\theta_2(m)$, and the scales $\sigma_1(n)$ and $\sigma_2(m)$ of the images.

The question is: which key point from the first frame corresponds to which key point from the second frame? In other words, which m can be associated with a given n, and, vice versa, which n can be associated with a given m? The answers can be tabulated in two look-up tables, say $\hat{m}(n)$ which maps the integers $1,\cdots,N$ to $0,\cdots,M$, and $\hat{n}(m)$ mapping $1,\cdots,M$ to $0,\cdots,N$. More mathematically:

$$\begin{aligned} \hat{m}(): \quad & 1,\cdots,N \quad \rightarrow \quad 0,\cdots,M \\ \hat{n}(): \quad & 1,\cdots,M \quad \rightarrow \quad 0,\cdots,N \end{aligned} \tag{9}$$

Note that the mapping to nil, e.g. $\hat{m}(3)=0$, means that the key point in the first image, in this example $\mathbf{x}_1(3)$, could not be associated with any key point in the second image.

Of course, when a key point in image 1, say $\mathbf{x}_1(3)$, is matched to a key point in image 2, say $\mathbf{x}_2(5)$, then the reverse should also be true: if $\hat{m}(3)=5$, then $\hat{n}(5)=3$. For a full consistent solution, we must have:

$$\hat{n}(\hat{m}(n))=n \quad \text{and} \quad \hat{m}(\hat{n}(m))=m \tag{10}$$

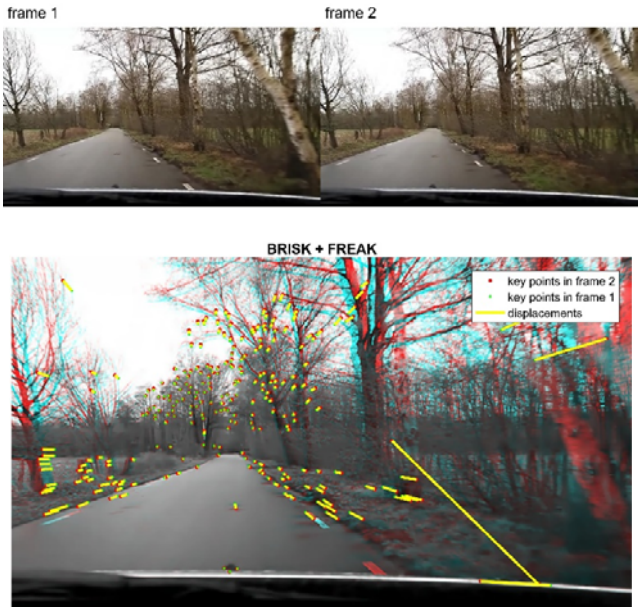
The problem can be formulated as a graph matching problem whose optimization can be quite complicated. A greedy algorithm is as follows:

$$\begin{aligned} \hat{m}(n) &= \arg\min_{m=1,2,\cdots} \left\{ d\left(\mathbf{z}_1(n),\mathbf{z}_2(m)\right) \right\} \\ \hat{n}(m) &= \arg\min_{n=1,2,\cdots} \left\{ d\left(\mathbf{z}_1(n),\mathbf{z}_2(m)\right) \right\} \end{aligned} \tag{11}$$

where $d\left(\mathbf{z}_1(n),\mathbf{z}_2(m)\right)$ is a distance measure between the two descriptor vectors. This could be the squared Euclidean distance $\left\|\mathbf{z}_1(n)-\mathbf{z}_2(m)\right\|^2$. Alternatively, the inner product $\mathbf{z}_1^T(n)\mathbf{z}_2(m)$ is used, which is interpreted as the cosine of the angle between the vectors. This angle should be as small as possible. Hence, the cosine of it should be maximized rather than minimized. For bit pattern features, the Hausdorff distance [1] can be used.

Equation (11) does not necessarily yield a full consistent solution that satisfies (10). So, in a second step, the requirements as stated in (10) should be checked. If key points do not comply with these requirements, they are removed from the sets. This usually reduces the set of corresponding key points considerably.

An example of two images with matching key points is shown below:



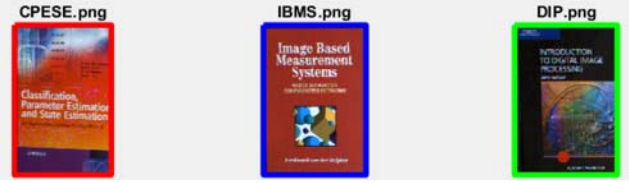
Key points have been detected with the Shi and Tomasi corner detector. The number of key points in frame 1 and frame 2 are 2273 and 2294, respectively. Next, for each key point, BRISK and FREAK features have been extracted. This resulted in 223 BRISK matches and 82 FREAK matches. Some of the BRISK matches coincide with the FREAK matches. So, the effective number of matches is less than 223+81.

It can be seen that a few matches are in error. All other matches show a pattern that is consistent with a motion of the camera towards the end of the lane.

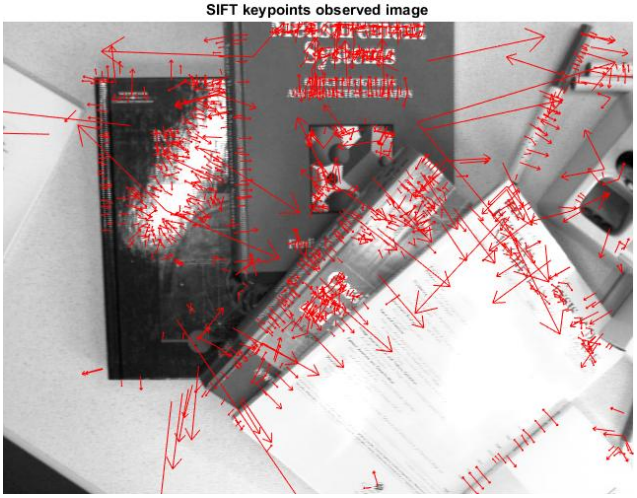
5 An application: object recognition

As an example, an object recognition application will be given showing the ability of SIFT key points to recognize objects in a scene. This example shows an image of a pile of books and magazines, and other stuff, on a desk top:

As can be seen, the pile is not nicely arranged. The books are partly occluded. They are not fronto-parallel to the image plane. Also, glossy spots disturb the image. The goal is to recognize and to locate the front sides of the three books. For that purpose, a *library* is available that contains high quality, fronto-parallel images of the covers of the books:



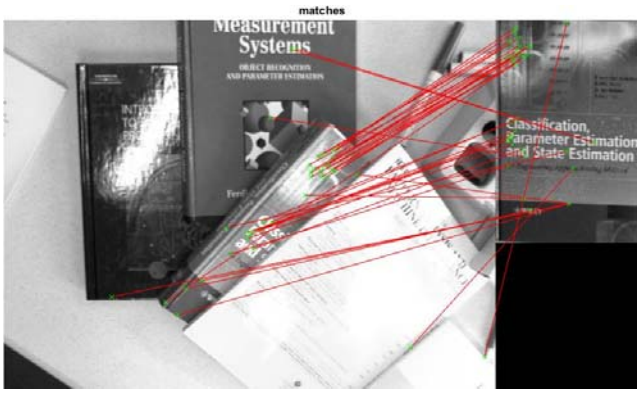
To recognize the objects, first, key points are detected in the target image, and in the library images. Application of SIFT yields key points that are shown below.



Key points are indicated by arrows. Their positions in the image are denoted by $\mathbf{x}_1(n)$ .The length of an arrow represents the scale $\sigma_1(n)$ at which a key point has been detected. The direction $\theta_1(n)$ of an arrow corresponds to the orientation of the key point.

An example of a library image with key points is shown below. The position, scale, and orientation of the key points are denoted by $\mathbf{x}_2(m)$, $\sigma_2(m)$, and $\theta_2(m)$.

The next step is to match key points from the target image with key points of library images using feature vectors $\mathbf{z}_1(n)$ and $\mathbf{z}_2(m)$ resulting in a mapping $\hat{m}(n)$. The figure below, showing the matches, indicates that there are about 22 correct matches, and about 7 incorrect matches.



These wrong matches are called *outliers*. We can detect them based on the principle that a similarity transform should exist between the library image and the observed image: the book is rotated, scaled and translated. The correct matches must be consistent with respect to rotation, scaling, and translation:

- If the rotation angle between library book and the book in the target image is φ, then the difference between orientation of two corresponding key points should also be φ.

$$\theta_1(n) - \theta_2(\hat{m}(n)) \approx \varphi \tag{12}$$

- The scale ratio s between the two images should also be reflected in the scale ration between two corresponding key points:

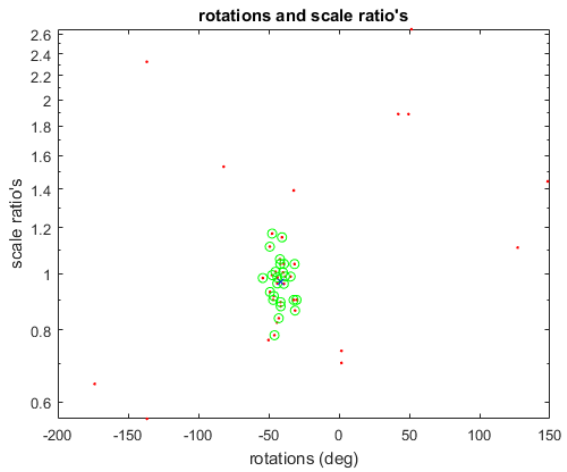
$$\sigma_2(n)/\sigma_2(\hat{m}(n)) \approx s \tag{13}$$

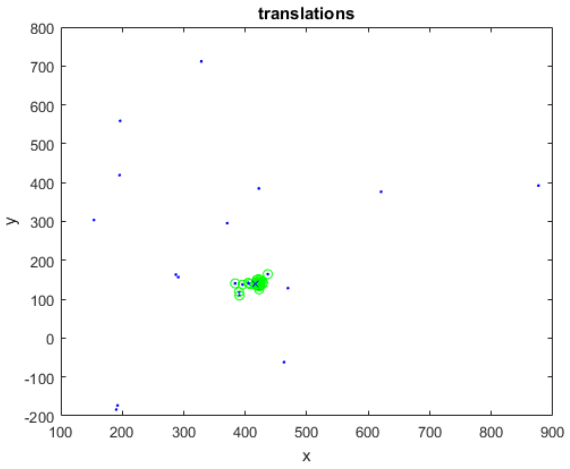
- After rotating and scaling the library image, it must be translated to the correct position in the observed image. If the translation vector is $\mathbf{t}$, then the difference between the positions of two corresponding keypoints, after that one of them has undergone the same rotation and scaling, should also be $\mathbf{t}$

$$\mathbf{x}_1(n) - s\mathbf{R}^{-1}(\varphi)\,\mathbf{x}_2(\hat{m}(n)) \approx \mathbf{t} \tag{14}$$

  Here $\mathbf{R}(\varphi)$ is the rotation matrix that corresponds to the rotation φ.

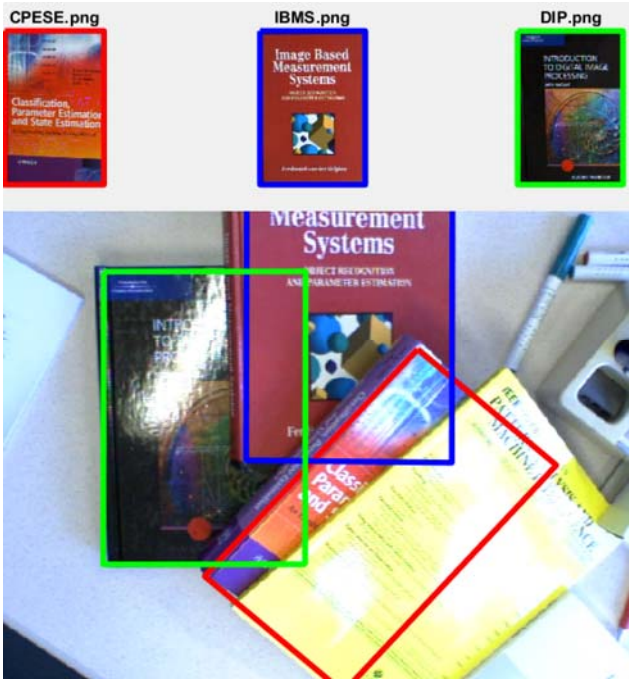
The rotations, scale ratios and translations of the matched key points are given in the figures below:





It can be seen that a majority of matched key points have a rotation of -50 degrees, a scale ration of about 1, and a translation vector (400,130). These key points form a cluster in this 4D space. All other matched key points are *outliers*. They can be detected with a *robust estimator*, such as the RANSAC algorithm. The existence of such a cluster indicates that the library book is present. The four parameters 'rotation, scale ratio, and translation vector' provide information about the location of the library book in the target image.

The method must be repeated to see whether other books are also visible. This results in 3 detections as shown below. All three books have been found, and properly located. This is despite the fact that two books are a little perspectively distorted.



6 Robust estimation with RANSAC

Ransac stands for RANdom SAmpling and Consensus. It is a parameter estimation strategy that is able to deal with outliers in a dataset. Ransac is basically an error-and-trial procedure.

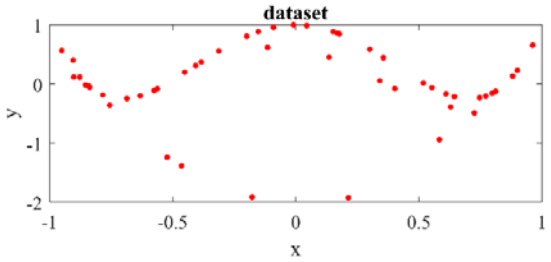
Suppose that a dataset X is available that consists of N vectors: $\mathbf{x}_n$ with $n = 1, \cdots, N$ (which are called *samples*) From this dataset, a parameter vector $\mathbf{p}$ should be estimated. For this, an estimator $\mathbf{f}(.)$ should be available which, applied to X or to a subset of X, yields an estimate of $\mathbf{p}$. That is:

$$\hat{\mathbf{p}} = \mathbf{f}(X)$$

The parameter vector could be as simple as the mean of the vectors $\mathbf{x}_n$, but could also be more complex. Examples are:

- In the previous section: the similarity transform (rotation, scaling, and translation), where each vector $\mathbf{x}_n$ consists of the attributes of a corresponding key point pair.
- Stitching images (panorama imaging): the projective transform from a set of corresponding key point pairs of overlapping images.

As a simple example, we consider the estimation of the coefficients of a polynomial using observed data points $\mathbf{x}_n = \begin{bmatrix} x_n & y_n \end{bmatrix}^{\mathrm{T}}$.



The dataset consists of N samples $\mathbf{x}_n = \begin{bmatrix} x_n & y_n \end{bmatrix}^{\mathrm{T}}$. The model that describes this data is the polynomial:

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + \text{noise}$$

The parameters to estimate are the coefficients of the polynomial: $\mathbf{p} = \begin{bmatrix} a_0 & \cdots & a_4 \end{bmatrix}^{\mathrm{T}}$. These parameters can easily be estimated by means of a least squares fitting algorithm (Matlab: `polyfit`). However, the data contains corrupted samples, i.e outliers. In the example above, there are 50 samples of which 11 are outlying.

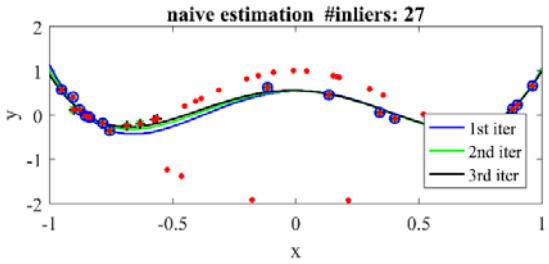
The naïve method to estimate these parameters is *iterative estimation*:

Naive iterative estimation:

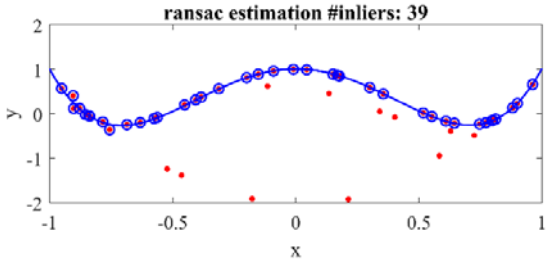
1. $X_{subset} = X$
2. repeat:
   - 2.1 $\hat{\mathbf{p}} = \mathbf{f}(X_{subset})$ `% estimate p from the subset`
   - 2.2 for all $\mathbf{x}_n$ in X:
     - $\hat{y}_n = \text{model}(x_n, \hat{\mathbf{p}})$ `% predict y from the model using the estimated p`
     - $D_n = \|y_n - \hat{y}_n\|$ `% get distance between true and predicted y`
   - 2.3 $X_{subset} = \{\mathbf{x}_n \mid D_n < threshold\}$ `% find set of inliers`

The iteration stops either after a maximum number of iterations, or after convergence. That is, after it has been detected that a new iteration does not change the subset anymore.

The result of the iteration is seen below. There is a large change that the procedure get stuck in the wrong subset. In the current example only 27 inliers have been found, but many of them are false positives. Consequently, the found parameter vector is heavily polluted.



Below the result when using the Ransac algorithm. All 39 inliers have been correctly found, and the associated estimated parameter vector will not be polluted.



Ransac starts by selecting randomly a subset with which the parameter vector is estimated. The hope is that this subset is without outliers, so that the estimated parameter is not polluted. With this estimated parameter, it checks which samples comply with the model. These samples are regarded as inliers. Then it checks whether there is consensus amongst these inliers about the correctness of this estimated parameter. This can be done, for instance, by counting the number of inliers. If there is not sufficient support (not enough number of inliers), a new trial is started.

The algorithm is as follows:

Ransac estimation:

try repeatedly:

1. Generate randomly: $X_{subset} \subset X$ `% select randomly a subset from X`
2. $\hat{\mathbf{p}} = \mathbf{f}(X_{subset})$ `% estimate p from the subset`
3. for all $\mathbf{x}_n$ in X:
   - $\hat{y}_n = \text{model}(x_n, \hat{\mathbf{p}})$ `% predict y from the model using the estimated p`
   - $D_n = \|y_n - \hat{y}_n\|$ `% get distance between true and predicted y`
4. $X_{inliers} = \{\mathbf{x}_n \mid D_n < threshold\}$ `% find set of inliers`
5. if consensus within $X_{inliers}$ `% e.g. are there enough inliers?`
   - $\hat{\mathbf{p}} = \mathbf{f}(X_{inliers})$ `% re-estimate with all inliers`
   - terminate `% and stop trying`

Once consensus has been obtained, the algorithm re-estimate the parameter once again, but this time with all inliers, so as to have maximal accuracy in the estimate.

Appendices

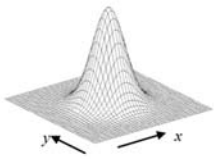
A. SIFT key points

SIFT stands for scale invariant feature transform. This method to find key points was proposed by Lowe in 1999. SIFT key points are found in a 3D "Laplacian of Gaussian" scale space. As such, it is a so-called 'multiscale approach'. Features are searched at various scales.

Remember that a scale space is formed by convolving the input image with a Gaussian PSF. The width of the Gaussian is defined by the parameter σ. The scale of the filtered image is defined as σ^2, or in some literature just as σ.

Reminders:

$$gauss(x,y,\sigma) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

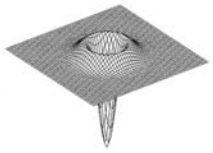


Laplacian: $f_{xx}(x,y)+f_{yy}(x,y)$

Laplacian of a Gauss:

$$LoG(x,y,\sigma) = \Delta gauss(x,y,\sigma)$$

$$= \frac{x^2+y^2-2\sigma^2}{2\pi\sigma^6}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

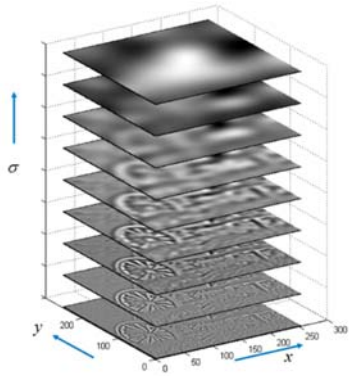


The Laplacian is the sum of the 2nd order derivatives taken into two orthogonal directions, e.g. in the x- and y-direction. The Laplacian of an image, considered at a given scale, can be computed by convolving the image with the Laplacian of the Gaussian at that scale. It is a rotational invariant operation.

By calculating this at several scales, starting with, for instance, $\sigma_1 = 0.7$, and then increasing this with very small steps, i.e. $\sigma_n = k\sigma_{n-1}$ with k being just a little bit larger than 1, a scale space of Laplacians is built:

$$\nabla f(x,y,\sigma) \overset{def}{=}$$

$$LoG(x,y,\sigma) * f(x,y)$$



In this 3D space, i.e. in the volume spanned by (x,y,σ), the Laplacian of the image is given by, say $g(x,y,\sigma)$. In this space, each point that is a local extremum is marked as a candidate key point. That is, each point (x,y,σ) for which $g(x,y,\sigma)$ is larger than all each 26 neighbors, or smaller than all each 26 neighbors is a candidate key point.

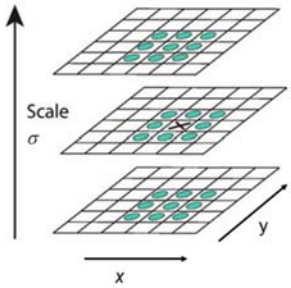
- find the extrema:

  a pixel (x,y,σ) is a candidate keypoint if:

  - it is larger than all its 26 neighbours

  or if:

  - it is smaller than all its 26 neighbours



Depending whether there is much texture in the image, this usually results in a large amount of candidate key points. Not all of them are stable. For instance, a key point that is detected on an edge may easily move along the edge due to noise, even if that noise is small. Therefore, special criterions are raised to filter out these instable key points. For instance, the extremum must be large enough, and the extremum must also be sufficiently pitched.

local quadratic approximation around a candidate keypoint $\mathbf{x}_0$:

$$\Delta f(\mathbf{x}_0+\mathbf{x}) \approx \Delta f(\mathbf{x}_0) + \mathbf{x}^T\mathbf{g}(\mathbf{x}_0) + \mathbf{x}^T\mathbf{H}(\mathbf{x}_0)\mathbf{x} \quad \text{with} \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ \sigma \end{bmatrix}$$

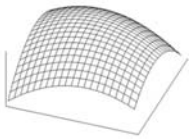
gradient vector — hessian

- the extremum must be large enough (large contrast):

  $|\Delta f(\mathbf{x}_0)|$ must be large enough

- the extremum must be highly peaked:

  eigenvalues of $\mathbf{H}$ must be large



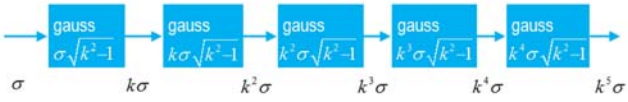
The implementation of building such a Laplacian scale space can be done very fast. A Laplacian of a Gaussian can be approximated by the difference of two Gaussians:

- approximation of LoG by differences of Gauss (DoG):

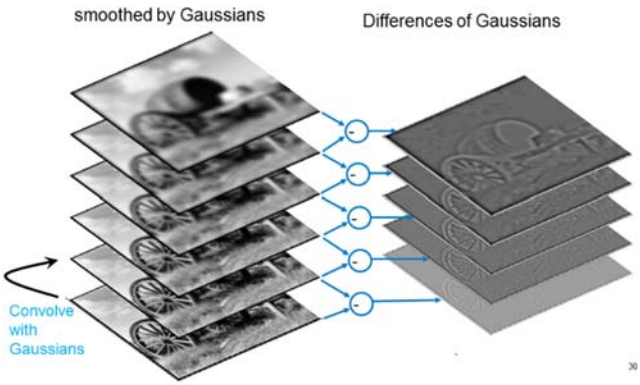
$$LoG(x,y,\sigma) \approx \frac{1}{\sigma^2(k-1)}\left(gauss(x,y,k\sigma) - gauss(x,y,\sigma)\right)$$

- cascade of Gaussians

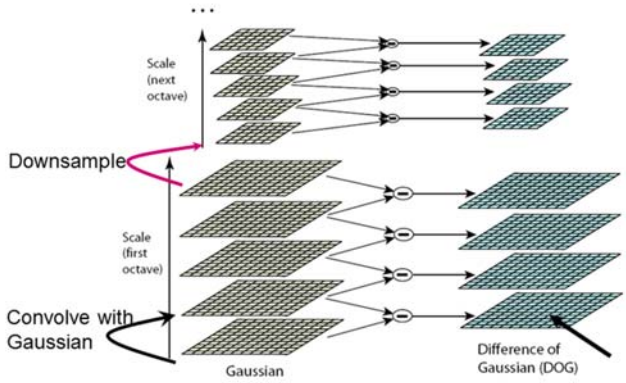
$$gauss(x,y,k\sigma) = gauss(x,y,\sigma\sqrt{k-1}) * gauss(x,y,\sigma)$$



This allows building up a Gaussian scale space first, which can be done efficiently by cascaded convolutions, and then obtaining the Laplacian scale space by subtraction:



A further computational optimization is obtained by down sampling the image as soon as the blurring is so large that a full resolution is not needed. This down sampling can be down multiple times, so that effectively the 3D scale space becomes a pyramid.



An example of detected SIFT key points are shown in Section 2.3, repeated here for convenience:

Note, that if the image is zoomed by a factor a, that is $f(x,y)$ becomes $f(ax,ay)$, then exactly the same key points will be found, but at a scale indicated by σ/a. This is because:

$$LoG(x,y,\sigma) * f(x,y) = \frac{1}{a^2} LoG\left(x,y,\frac{\sigma}{a}\right) * f(ax,ay) \qquad (15)$$

Hence, the positions at which key points are found are scale invariant, and the scale σ at which they are found is a measure of the zoom level. Note also that key points are rotation invariant in the sense that the same set will be found if the image is rotated. This is because the Laplacian of a Gaussian is rotational invariant.

B. Matlab implementations

Harris corner detection

The Computer Vision Toolbox of Matlab defines the `corner point` class for detecting, manipulating and plotting corner points. The class has the following constructors for actually creating such an object and for detecting the corners:

```
detectHarrisFeatures
detectMinEigenFeatures
detectMinFastFeatures
```

Apart from that, the Image Processing Toolbox contains the function `corner` that also detect corners using Harris' corner responsity.

Optical flow

Lucas-Kanade optical flow algorithms in Matlab's Computer Vision Toolbox are supported by the object `opticalFlow`, with constructors: functions:

- `opticalFlowLK` (Lucas-Kanade)
- `opticalFlowLKDoG` (Lucas-Kanade with Gaussian filtered time derivative)
- `opticalFlowFarneback`
- `opticalFlowHS` (Horn-Schunk)

There is also an object `vision.PointTracker` that can be used to track a number of selected points in a video. This tracker also uses the Lucas-Kanade principle.

Key point detection

Currently, there are three Matlab objects for key points are:

Class	Constructor
`SURFPoints`	`detectSURFFeatures`
`BRISKPoints`	`detectBRISKFeatures`
`MSERRegions`	`detectMSERFeatures`

These objects also contain plot facilities of the key points.

SIFT key points are not supported in the Matlab library, but can be downloaded from Matlab's file exchange, i.e. https://nl.mathworks.com/matlabcentral/fileexchange, or the VLfeat open source library: http://www.vlfeat.org.

Feature extraction in Matlab

The Computer vision toolbox of Matlab provides implementations of some descriptors. The function to get these is `extractFeatures`, the syntax of which is:

```
[features, valid_points] = ...
       extractFeatures(I, points,Name,Value);
```

Here, `I` is the input image. The variable `points` is the object that is returned by the corner finders or the key point detectors. *`Name, Value`* are options that allow to choose the descriptor type, and to add additional parameters. If *`Name`* is `'method'`, then *`Value`* can be `'SURF'`, `'BRISK'`, `'FREAK'`, or `'Block'`. These are the different types of descriptors.

The resulting descriptors are stored in `features`. A list of valid points (for some points it may not possible to calculate the descriptors; so they are removed) is provided in the object `valid_points`. This object also contains the obtained orientation.

Matlab also contains a function `extractHOGFeatures` that returns histograms of gradient orientations of a set of points in the input image.

Feature matching

Matching is implemented with the Matlab function `matchFeatures`. Visualization is accomplished with `showMatchedFeatures`.

References

[1] C. Zhao, W. Shi, and Y. Deng, "A new Hausdorff distance for image matching," *Pattern Recognition Letters,* vol. 26, pp. 581-586, 4// 2005.

<fill up>