Key point matching and optical flow

3D Computer Vision for Medical Applications

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

Contents

1 INTRODUCTION 1
2 KEY POINT DETECTION 1
2.1 CORNER DETECTION 1
2.2 KEY POINTS BASED ON BIT PATTERNS 3
2.3 KEY POINTS BASED ON SCALE SPACE 3
2.3.1 OTHER KEY POINTS 4
3 OPTICAL FLOW 5
3.1 OPTICAL FLOW EQUATION 5
3.2 LUCAS-KANADE 6
3.2.1 FARNEBÄCK 8
3.2.2 OTHER METHODS 8
4 FEATURE VECTORS 9
4.1 FEATURES FROM HISTOGRAMS OF GRADIENT DIRECTIONS 9
4.2 OTHER FEATURES 10
5 FEATURE MATCHING 10
6 AN APPLICATION: OBJECT RECOGNITION 11
APPENDICES 14
A. SIFT KEY POINTS 14
B. MATLAB IMPLEMENTATIONS 15
REFERENCES 16

Key point matching and optical flow

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.

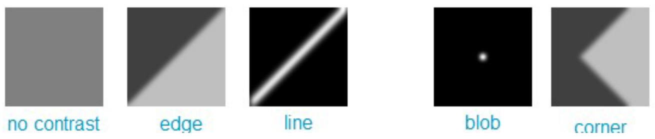
The organization of the syllabus is as follows. Section 2 deals with the question how to detect so-called corners in an image. In a dynamic scene, a video of that scene shows moving objects. The key points will move as well. From frame to frame, each key point has an associated displacement. The collection of displacements vectors of all key points is called the *optical flow*. This will be discussed in Section 3. Another class of interest points are point features. These are introduced in Section4. 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 5. An example of an application, object recognition, is provided in Section 6.

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:

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

$w(n,m)$ is a window function. (n_0,m_0) is the position of a candidate key point. The expression $w(n-n_0,m-m_0)$ select points in the neighborhood of (n_0,m_0). If $w(n-n_0,m-m_0)=1$, then a pixel at $(n-n_0,m-m_0)$ is fully taken into account. If $w(n-n_0,m-m_0)=0$, then that pixel is fully ignored. 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. In that case, $(p,q)=(+1,+1)$, $(-1,+1)$, $(+1,-1)$, 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:

$$f(n+p,m+q)\approx f(n,m)+p\,f_x(n,m)+q\,f_y(n,m) \qquad (2)$$

which leads to the following expression for $E(p,q)$:

$$E(p,q)=\begin{bmatrix}p & q\end{bmatrix}\mathbf{M}\begin{bmatrix}p\\ q\end{bmatrix} \qquad (3)$$

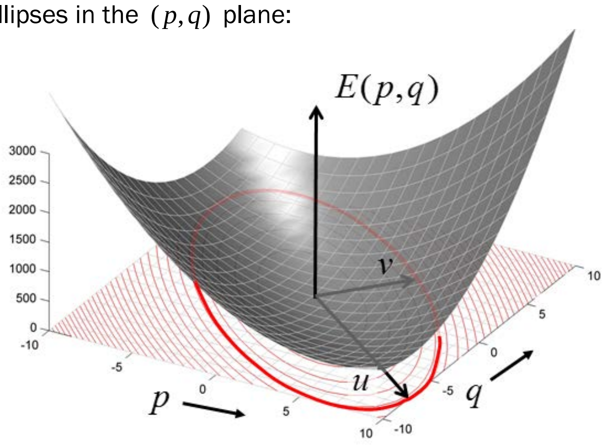
The matrix $\mathbf{M}$ turns out to be:

$$\mathbf{M}(n_0,m_0)=\sum_{n,m} w(n-n_0,m-m_0)\begin{bmatrix} f_x^2(n,m) & f_x(n,m)\,f_y(n,m)\\ f_x(n,m)\,f_y(n,m) & f_y^2(n,m)\end{bmatrix} \qquad (4)$$

They also suggested to replace the binary window function by a Gaussian:

$$w(n_0-n,m_0-m)=\exp\left(-\frac{(n-n_0)^2+(m-m_0)^2}{2\sigma^2}\right) \qquad (5)$$

Equation (3) defines an ellipsoid. 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 ellipsoid is described by:

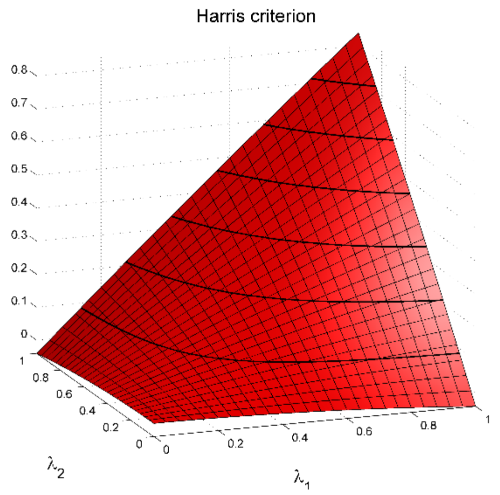
$$E(p,q) \sim \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \lambda_1 u^2 + \lambda_2 v^2 \qquad (6)$$

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

Roughly speaking, the ellipsoid 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.



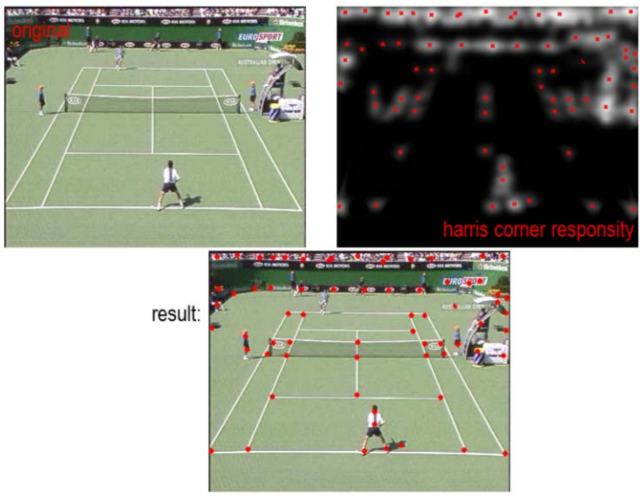
The motivation for this definition of corner responsity as a criterion is that it can also be expressed in terms of the derivatives of images:

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

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

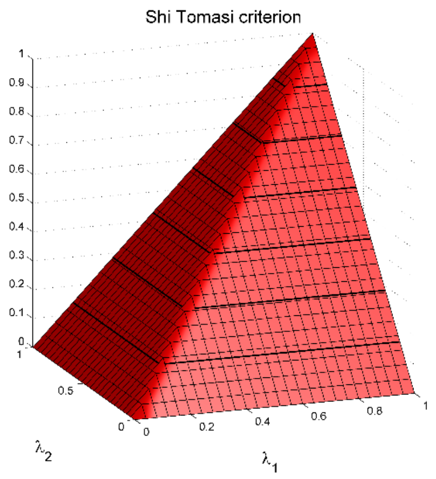
An example:



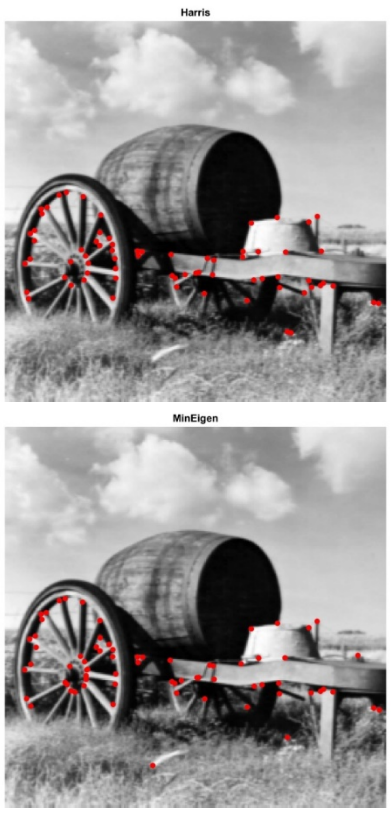
Shi and Tomasi (1994) proposed a slight modification of the Harris criterion, namely the minimum of the two eigenvalues:

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

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

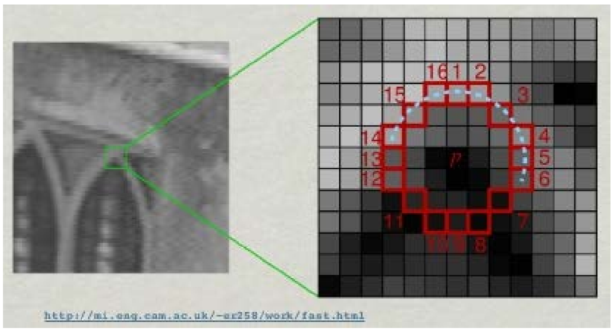


This criterion, in Matlab called the '*minimum eigenvalue criterion*', is claimed to have better performance than Harris. However, comparison of the two graphs shows that the difference will not be large. This is confirmed in the example shown below.

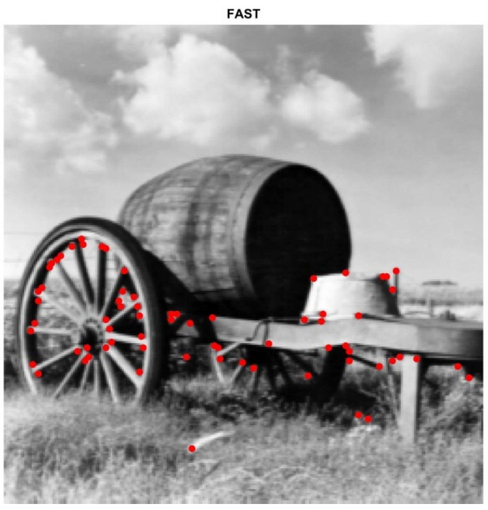


2.2 Key points based on bit patterns

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.



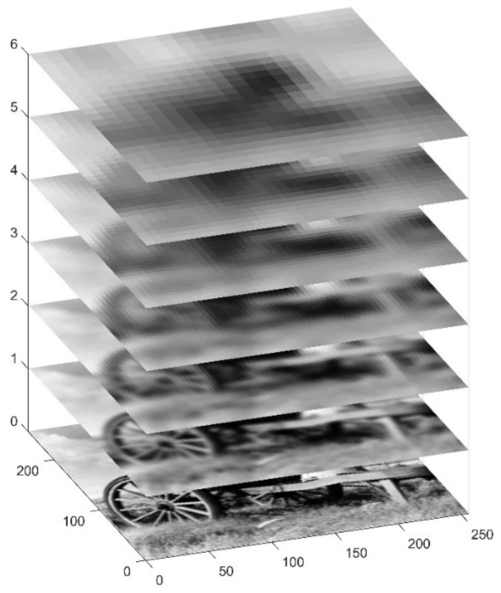
2.3 Key points based on scale space

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, this is not guaranteed. Scale spaces can take care of scale invariance.

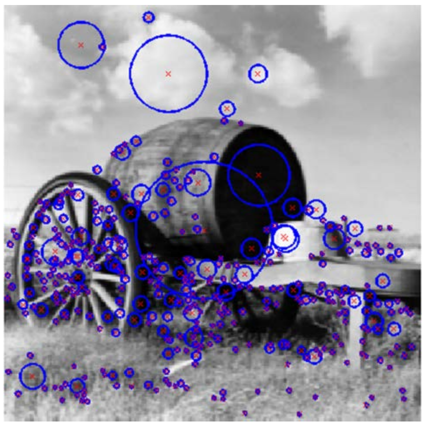
Scale space theory provides the means to transform a 2D grey level image into a 3D image data structure called the *scale space*. This enables *scale invariant* key point detection. An example of a scale space is shown below. The supplemented third dimension is the *scale* s.

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



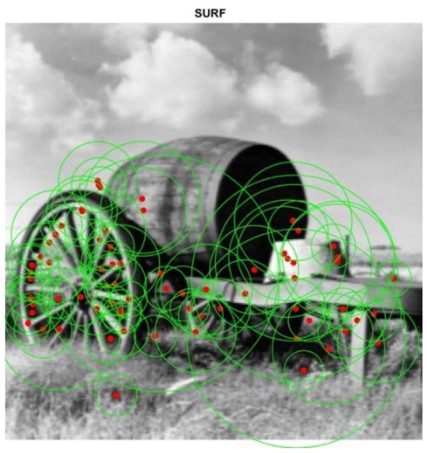
An example of SIFT key points is given below. The scale at which the key points are found is indicated by the circle.



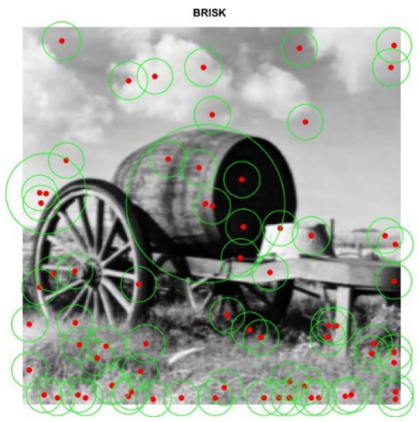
2.3.1 Other key points

After Lowe's publication on SIFT, in 1999, several other algorithms are 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 $g(x, y, t) = \left(f(x, y) > t \right)$. 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.

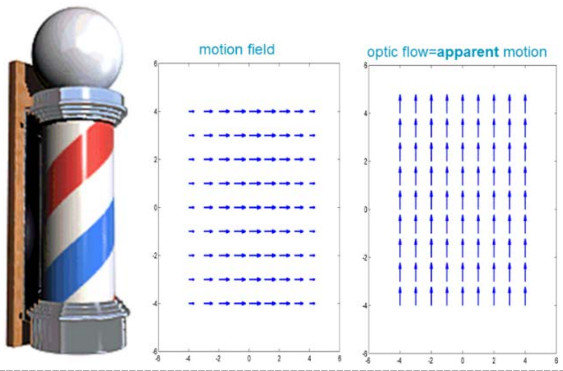
3 Optical flow

A human observer is quite capable to 'see' in a movie how objects move in the image plane. However, they use cognitive knowledge. It is not easy to have computers observing these movements at the same level of abstraction. Computer vision works usually at a much lower level: it detects key points, like blobs, spots, and corners. This section addresses the estimation of the displacement vectors of these points. A field of displacement vectors in an image is called the *optical flow*.

Optical flow estimation is based on two assumptions:

- The *constant brightness assumption*: the observed colors of an image 3D surface patch do not change when this surface patch moves.
- The *continuity assumption*: the motions of two neighboring surface patches are almost the same.

As said before, optical flow (or optic flow) is the *apparent motion* of pixels in an image sequence. Ideally, it is the 2D projection of the 3D motion of the points on the objects. However, there are few (unwanted) exceptions in which the optical flow differs from this projected motion. An example is shown below.



When rotating the barber's pool, the colored helixes appears to move upward, whereas a fixed 3D surface patch on the pool just rotates around the axis of the pool. Hence the true motion field differs much from the apparent motion in the image.

Another example in which the optical flow differs from the true motion field is a glossy, shiny spots. The example of the barber's pool, shown above, has an elongated glossy spot in front of the pool. The apparent motion of this spot is zero, yet the pool is rotating.

In many applications, these exceptions are of minor concern, optical flow and motion fields are considered to be the same.

3.1 Optical flow equation

Suppose that a moving 3D surface patch is imaged in the image plane at a position $[x(t), y(t)]$ where t is the continuous time. Then the velocity of this patch, projected on the image plane, i.e. the motion field, is:

$$\mathbf{v} = \begin{bmatrix} \frac{dx}{dt} & \frac{dy}{dt} \end{bmatrix}^T \tag{9}$$

or in shorthand $\mathbf{v} \overset{def}{=} [u(t) \quad v(t)]^T$ with $u(t) = dx/dt$ and $v(t) = dy/dt$.

The constant brightness assumption entails that the grey levels (or colors) in the image do not change. Hence:

$$f\left(x(t), y(t), t\right) = \text{constant} \tag{10}$$

For now, we assume that the grey levels $f()$ is given in the continuous domain: $(x, y, t) \in \mathbb{R}^3$ We also assume that $f()$ is a differentiable function. Equation (10) then implies that the total differential of $f()$ with respect to time must be zero:

$$\frac{d}{dt} f\left(x(t), y(t), t\right) = 0 \tag{11}$$

Applying the chain rule for differentiation allows us to express this differential in term of partial derivatives:

$$\frac{\partial f(x,y,t)}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f(x,y,t)}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial f(x,y,t)}{\partial t} = 0 \tag{12}$$

Introducing the notations: $f_x = \partial f/\partial x$, $f_y = \partial f/\partial y$, and $f_t = \partial f/\partial t$, the equation simplifies to:

$$f_x u + f_y v + f_t = 0 \tag{13}$$

This equation is known as the *optical flow equation* (OFE). For the sake of completeness, we mention that a more compact notation for the OFE is $\mathbf{v}^T \nabla f + f_t = 0$ where ∇ is the gradient operator.

The OFE by itself does not suffice to find the optical flow as it provides only 1 equation per pixel while we have 2 unknown variables u and v to solve. We need the continuity assumption "neighboring patches move similar" to solve the problem.

3.2 Lucas-Kanade

Lucas and Kanade solve the problem by assuming that the optical flow is constant within a window that surrounds the pixel of interest.

The time-discrete optical flow equation

In practice, images are only available at discrete points in time. The variable t must be regarded as a discrete time index, $t = 0,1,2,\cdots$. The sampling period, i.e. the inverse of the frame rate. Will be denoted by T.

In discrete time, the velocities become displacements. A pixel located at $\mathbf{x} = [x \quad y]^T$ at time t will be displaced to $\mathbf{x}+\mathbf{d}$ at time $t+1$. The vector $\mathbf{d}$ is the displacement vector.

$$\mathbf{d} \approx \mathbf{v}T \tag{14}$$

The constant brightness assumption becomes:

$$f(\mathbf{x}+\mathbf{d}, t+1) = f(\mathbf{x}, t) \qquad \text{with} \quad \mathbf{x} = (x, y) \tag{15}$$

To retrieve the OFE, a truncated Taylor series expansion can be applied:

$$f(\mathbf{x}+\mathbf{d}, t+1) \approx f(\mathbf{x}, t+1) + \mathbf{d}^T \, \nabla f(\mathbf{x}, t+1) + HOT \tag{16}$$

Here ∇ is the gradient operator with respect to $\mathbf{x}$. Thus:

$$\nabla f(\mathbf{x}, t) = \begin{bmatrix} f_x(\mathbf{x}, t) \\ f_y(\mathbf{x}, t) \end{bmatrix} \tag{17}$$

Neglecting the higher order terms (HOT), the OFE takes the form of:

$$\mathbf{d}^T \, \nabla f(\mathbf{x}, t+1) + f(\mathbf{x}, t+1) - f(\mathbf{x}, t) \approx 0 \tag{18}$$

Comparison with (13) shows that it has exactly the same form as the original OFE except that the time derivative has been replace with a time difference. To shorten the notation we use $\Delta f(\mathbf{x}, t) = f(\mathbf{x}, t+1) - f(\mathbf{x}, t)$.

Window functions

Suppose that the pixel, for which we want to calculate the displacement vector, has coordinates (n, m) which we concisely denote by the vector $\mathbf{n} = [n \quad m]^T$. A window function is a function that determines whether pixels in the vicinity of $\mathbf{n}$ are considered close to $\mathbf{n}$. The window can be binary, i.e. a pixel is considered either close or not close to the central pixel $\mathbf{n}$. The window function can also be more subtle by assigning weights. The weight is 1 if a pixel is very close to the central pixel. The weight decreases when a pixel is further away, and it decays to 0 when a pixel is considered far away from the central pixel. A popular window function is the Gaussian function, which is given by:

$$W(\mathbf{n}-\mathbf{x}) = \exp\left(-\frac{\|\mathbf{n}-\mathbf{x}\|^2}{2\sigma^2} \right) \tag{19}$$

This window softly decays from one to zero when the distance between $\mathbf{x}$ and $\mathbf{n}$ becomes larger than the width parameter σ.

The optimization criterion

Lucas and Kanade define a quadratic error measure to express the closeness of a solution $\mathbf{d}$ given the image data:

$$J(\mathbf{d}) = \sum_{\mathbf{x}} W(\mathbf{n}-\mathbf{x}) \left(\mathbf{d}^T \nabla f(\mathbf{x}, t+1) + \Delta f(\mathbf{x}, t) \right)^2 \tag{20}$$

The inherent assumption here is that $\mathbf{d}$ is constant for each position $\mathbf{n}-\mathbf{x}$ within the window function.

The function $W(\mathbf{n}-\mathbf{x})$ weights the pixels in the neighborhood of $\mathbf{n}$. The summation takes place over all pixels $\mathbf{x}$ in the image. In practice, the window function is almost zero if $\|\mathbf{n}-\mathbf{x}\| > 4\sigma$. Thus, the tail of the Gaussian can be truncated. The notation is shortened by introducing the bar operator:

$$\overline{\left(\mathbf{d}^T \nabla f + \Delta f \right)^2} \overset{def}{=} \sum_{\mathbf{x}} W(\mathbf{n}-\mathbf{x}) \left(\mathbf{d}^T \nabla f(\mathbf{x}, t+1) + \Delta f(\mathbf{x}, t) \right)^2 \tag{21}$$

In this concise notation, the bar denotes summation over the window. In fact, such a summation can be regarded as a *convolution*. The image $(\mathbf{d}^T\nabla f(\mathbf{x},t+1)+\Delta f(\mathbf{x},t))^2$, with fixed $\mathbf{d}$, is convolved by the point spread function $W(\mathbf{x})$.

Optimization

If a $\mathbf{d}$ exists such that $J(\mathbf{d})=0$, then the OFE is satisfied, and we have a solution. Hopefully, this solution is unique. However, the presence of noise, and the violations of the two assumptions will make sure that such a $\mathbf{d}$ does not exists. Therefore, our objective is to find $\mathbf{d}$ such that $J(\mathbf{d})$ is minimized. The condition for the minimum is that the gradient[1] of $J(\mathbf{d})$ is zero; $\nabla J(\mathbf{d})=\mathbf{0}$. Defining $\mathbf{d}=[u \quad v]^T$ and rewriting (20) and (21) yield:

$$\begin{aligned} J(\mathbf{d}) &= \overline{\left(u\ f_x + v\ f_y + \Delta f\right)^2} \\ &= \overline{u^2 f_x^2 + v^2 f_y^2 + (\Delta f)^2 + 2uv f_x f_y + 2(uf_x + vf_y)\Delta f} \end{aligned} \tag{22}$$

Differentiation with respect to u and v, and equating the results to zero give:

$$\begin{aligned} J_u &= 2u\,\overline{f_x^2} + 2v\,\overline{f_x f_y} + 2\,\overline{f_x \Delta f} = 0 \\ J_v &= 2u\,\overline{f_x f_y} + 2v\,\overline{f_y^2} + 2\,\overline{f_y \Delta f} = 0 \end{aligned} \tag{23}$$

Introduce the following matrix and vectors:

$$\mathbf{M} \overset{def}{=} \begin{bmatrix} \overline{f_x^2} & \overline{f_x f_y} \\ \overline{f_x f_y} & \overline{f_y^2} \end{bmatrix} \qquad \mathbf{t} \overset{def}{=} -\begin{bmatrix} \overline{f_x f_t} \\ \overline{f_y f_t} \end{bmatrix} \tag{24}$$

With these definitions, (23) is rewritten into $\mathbf{Md}=\mathbf{t}$ for which the solution – if it exists – is:

$$\hat{\mathbf{d}} = \mathbf{M}^{-1}\mathbf{t} \tag{25}$$

Whether a solution exists, or not, follows from the properties of the matrix $\mathbf{M}$. These properties depends on the image data within the window. Below some typical examples of image patches are shown for which $\mathbf{M}$ is invertible or not.

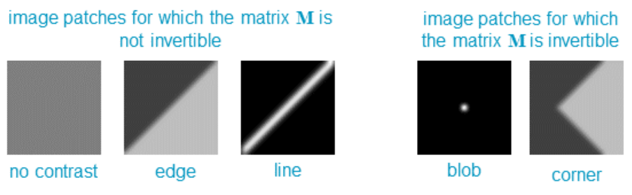
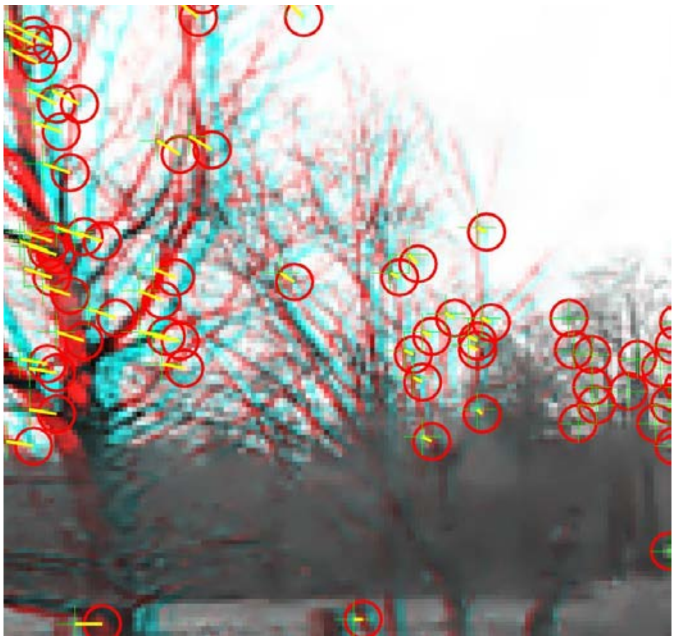


image patches for which the matrix M is not invertible
image patches for which the matrix M is invertible
no contrast
edge
line
blob
corner

Note that $\mathbf{M}$ the same matrix is as was used in the Harris corner detector; see equation (4). Key points that are detected by Harris should also be well tractable by Lucas-Kanade.

[1] The gradient with respect to $\mathbf{d}$.

The image below is an example. It shows two frames from a movie. The first frame is shown in cyan. The second one is red. Harris corners are detected in the first image and indicated by a red circle. The displacement vectors, obtained with Lucas-Kanade, are shown by yellow line segments.

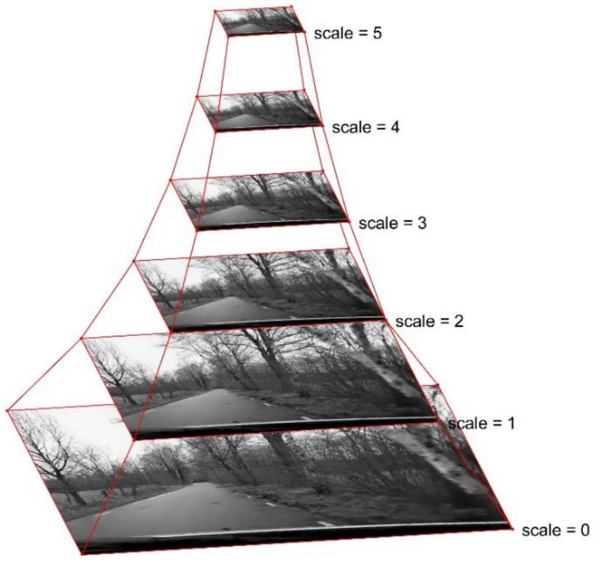


The Lucas-Kanade method works well only under strict conditions. An obvious condition is that the matrix $\mathbf{M}$ must be invertible (non-singular). In image regions without contrast and texture, e.g. the road and the sky, the matrix $\mathbf{M}$ is not invertible, or near singular. That is, one or more of the eigenvalues of $\mathbf{M}$ are small. Consequently, the Lucas-Kanade method cannot reliably produce results there. Another condition is that the truncated Taylor series approximation in (16) must be valid. This will not be the case if the displacement is large, or if the grey levels vary too wildly within the window.

Another consideration is the size of the window. If the window is small, the influence of noise will be large. But if the window is too large, the continuity assumption breaks down.

Iterated multi-scale approach

One way to overcome these limitations is by using a resolution pyramid of the image:



In this multi-scale approach, the images are initially processed at a large (coarse) scale, and the processing iteratively descends to a small (detailed) scale. Large displacements at the original scale (scale=0) become relatively small displacements at the largest scale. So, Lucas-Kanade estimation is first applied at this largest scale, and then propagated to the image at one scale lower. At this level the second image is warped according to these displacements after which Lucas-Kanade is applied again. This principle is iterated to the lowest level.

3.2.1 Farnebäck

Farnebäck et al proposed a quadratic approximation of the image data around the pixel of interest, i.e. a truncated Taylor series expansion with three terms:

$$f(\mathbf{n}+\mathbf{x},t) \approx b + \mathbf{b}^T\mathbf{x} + \mathbf{x}^T\mathbf{B}\,\mathbf{x} \tag{26}$$

The coefficients are a scalar b, a vector $\mathbf{b}$, and a symmetric matrix $\mathbf{B}$ that are estimated with a least squares data fit within the vicinity of $\mathbf{n}$ and weighted by a window function. Of course, the image data in the next frame can be approximated similar:

$$f(\mathbf{n}+\mathbf{x},t+1) \approx c + \mathbf{c}^T\mathbf{x} + \mathbf{x}^T\mathbf{C}\,\mathbf{x} \tag{27}$$

According to (26), the image data in $f(\mathbf{x},t)$ around the displaced pixel $\mathbf{n}-\mathbf{d}$ will be:

$$\begin{aligned} f(\mathbf{n}-\mathbf{d}+\mathbf{x},t) &\approx b + \mathbf{b}^T(\mathbf{x}-\mathbf{d}) + (\mathbf{x}-\mathbf{d})^T\mathbf{B}(\mathbf{x}-\mathbf{d}) \\ &= b - \mathbf{b}^T\mathbf{d} + \mathbf{d}^T\mathbf{B}\mathbf{d} + \mathbf{b}^T\mathbf{x} - 2\mathbf{d}^T\mathbf{B}\mathbf{x} + \mathbf{x}^T\mathbf{B}\mathbf{x} \end{aligned} \tag{28}$$

The contrast brightness assumption $f(\mathbf{x}-\mathbf{d},t) = f(\mathbf{x},t+1)$, stated in (15), allows us to equate (27) and (28). This yields:

$$c = b - \mathbf{b}^T\mathbf{d} + \mathbf{d}^T\mathbf{B}\mathbf{d} \tag{29}$$

$$\mathbf{c}^T = \mathbf{b}^T - 2\mathbf{d}^T\mathbf{B} \tag{30}$$

$$\mathbf{C} = \mathbf{B} \tag{31}$$

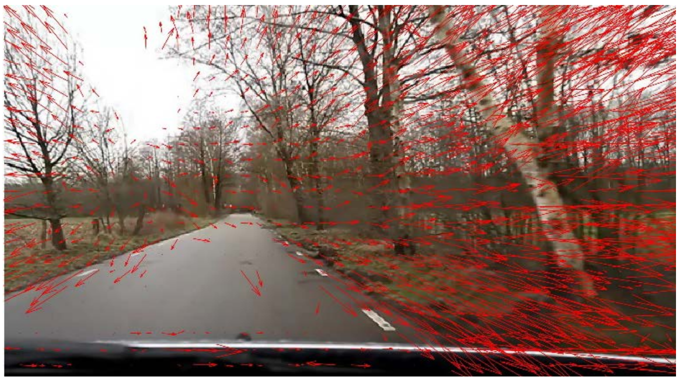
The key observation is in (30). From this equation, and provided that $\mathbf{B}$ is invertible, it follows that:

$$\hat{\mathbf{d}} = \tfrac{1}{2}\mathbf{B}^{-1}(\mathbf{b}-\mathbf{c}) \tag{32}$$

Another improvement of Farnebäck is that the constant displacement assumption of Lucas-Kanade is relaxed by assuming that the field within the window function is varying linearly (or even quadraticly), i.e.

$$\mathbf{d}(\mathbf{n}-\mathbf{x}) = \mathbf{d}_0(\mathbf{n}) + \mathbf{D}_n(\mathbf{n}-\mathbf{x}) \tag{33}$$

An example of Farnebäck optical flow estimation is given below.



3.2.2 Other methods

Many alternatives exists for optical flow estimation. The alternative to the window approach of the Lucas-Kanade, and others, is the assumption that the optical flow can only change gradually over the image plane. This principle is first proposed by Horn and Schunck, and later adjusted and extended by others.

Suppose that we have an optical flow field given by the vector $\mathbf{u}(x,y) = [u(x,y) \quad v(x,y)]^T$. Then a measure for the global smoothness of this field is:

$$J_{smooth} = \sum_{\text{all } x,y} (u_x^2 + u_y^2) + (v_x^2 + v_y^2) \tag{34}$$

in which u_x, u_y, v_x, and v_y are the derivatives of $u(x,y)$ and $v(x,y)$. So, $(u_x^2+u_y^2)+(v_x^2+v_y^2)$ measures locally the change of the optical flow field. This measured is summed over the whole image area.

The tightness of the solution $\mathbf{u}(x,y)$ to the OFE is measured by:

$$J_{OFE} = \sum_{\text{all } x,y} (uf_x + vf_y + f_t)^2 \qquad (35)$$

Horn-Schunck defines the optimal solution as the optical flow field that minimizes

$$J_{total} = J_{OFE} + \alpha J_{smooth} \qquad (36)$$

Thus, they try to minimize both factors. The design parameter α defines the balance between the two measures. The optimal flow field is found by an iterative algorithm in which at each iteration J_{total} decreases. Hopefully, this iterative process converges to lowest J_{total} possible.

4 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.

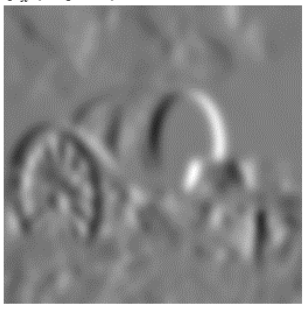
4.1 Features from histograms of gradient directions

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

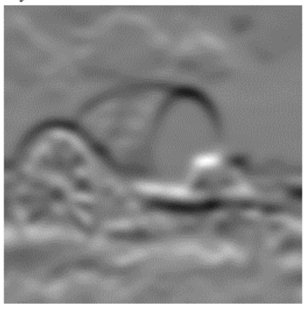
The approach of SIFT to get a feature is to first estimate the gradients $f_x(x,y,\sigma)$ and $f_y(x,y)$ of the image at the same scale σ at which the key point was found. Below an example:

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

$f_x(x,y,\sigma)$

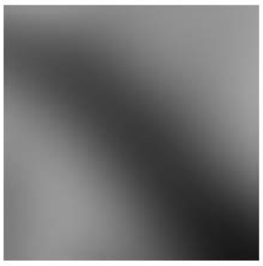


$f_y(x,y,\sigma)$



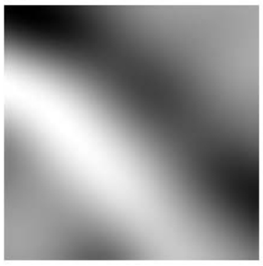
This gradient images provide information about the local orientation of the data. Suppose that we have a key point on the top right side of the wheel at $\sigma = 4.5$, then locally the images look like:

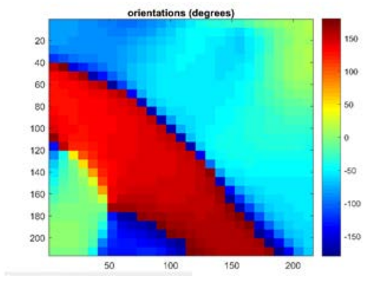
$f(x,y,\sigma)$ with σ=4.5



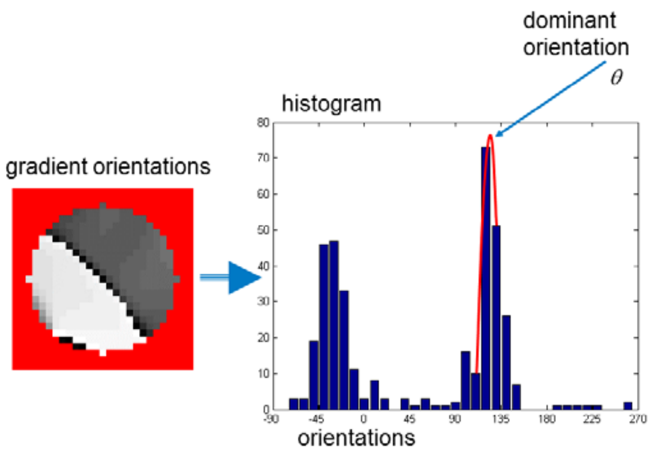
$f_x(x,y,\sigma)$

$f_y(x,y,\sigma)$



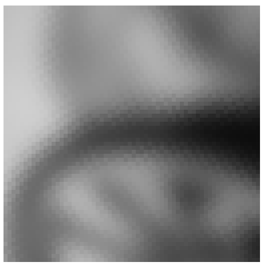


The first step will be to extract the dominant orientations around the key point. The can be found by calculating a histogram of the orientations of the gradient vectors within a circular neighborhood of the key point.

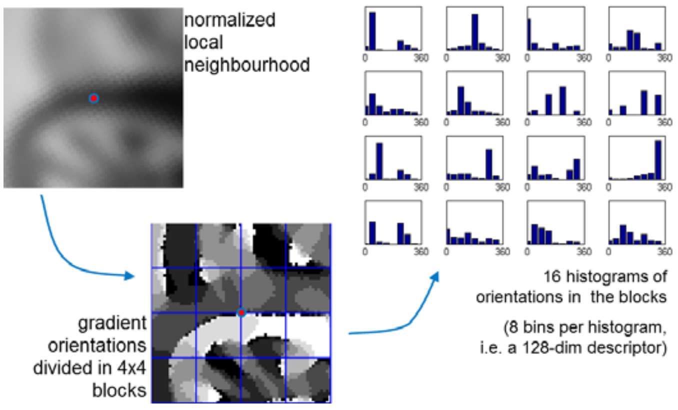


The example here has a dominant orientation of about 130 degrees. Together with the scale, the found orientation is associated with the key point as two attributes.

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.



normalized local neighbourhood
gradient orientations divided in 4x4 blocks
16 histograms of orientations in the blocks
(8 bins per histogram, i.e. a 128-dim descriptor)

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 $Af(x,y)+B$.

4.2 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.

5 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:

$$\begin{array}{lll} \hat{m}(): & 1,\cdots,N & \rightarrow & 0,\cdots,M \\ \hat{n}(): & 1,\cdots,M & \rightarrow & 0,\cdots,N \end{array} \qquad (37)$$

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:

$$\hat{n}(\hat{m}(n)) = n \quad \text{and} \quad \hat{m}(\hat{n}(m)) = m \tag{38}$$

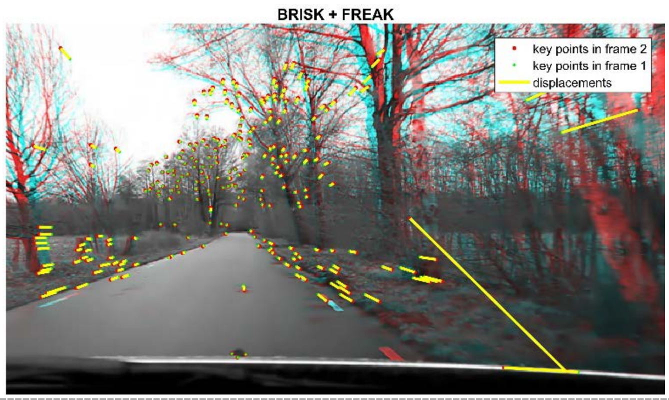
The problem can be formulated as a graph matching problem whose optimization can be quite complicated. A greedy algorithm is as follows:

$$\begin{aligned} \hat{m}(n) &= \underset{m=1,2,\cdots}{\arg\min}\left\{d\left(\mathbf{z}_1(n),\mathbf{z}_2(m)\right)\right\} \\ \hat{n}(m) &= \underset{n=1,2,\cdots}{\arg\min}\left\{d\left(\mathbf{z}_1(n),\mathbf{z}_2(m)\right)\right\} \end{aligned} \tag{39}$$

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 (39) does not necessarily yield a full consistent solution that satisfies (38). So, in a second step, the requirements as stated in (38) 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:

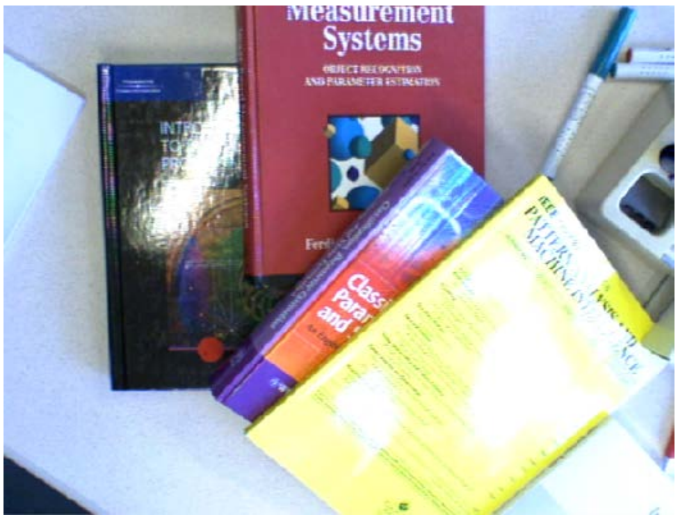


Key points have been detected with the Shi and Tomasi corner detector. The number of key points in frame 1 and frame 2 are 2273 and 2294, respectively. Next, for each key point, BRISK and FREAK features have been extracted. This resulted in 223 BRISK matches and 82 FREAK matches. Some of the BRISK matches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

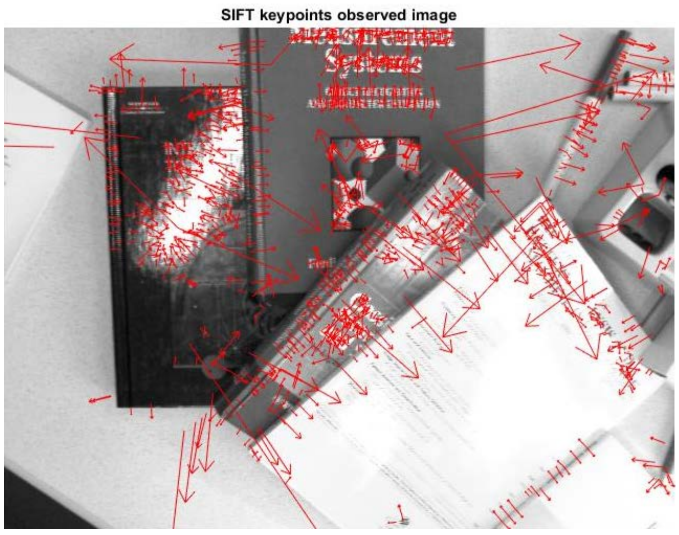
6 An application: object recognition

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:

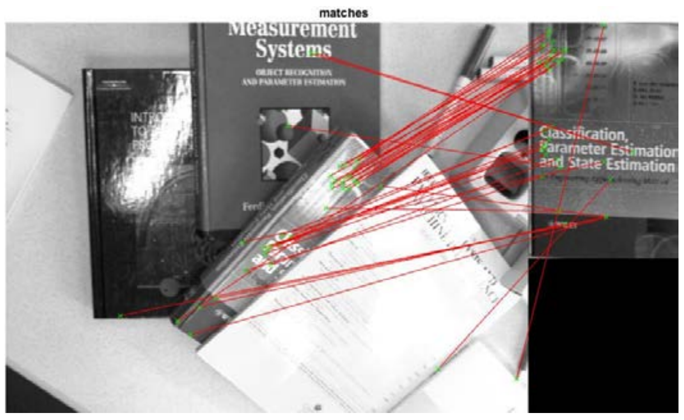
To recognize the objects, first key points are detected in the target image, and in the library images. Application of SIFT yields key points that are shown below.



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 $\theta_2(m)$.

Te 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 some found matches are erroneous.



We can detect these errors based on the principle that correct matches are consistent with respect to rotation, scaling, and translation:

- The rotation between two corresponding key points should be constant.

$$\theta_1(n) - \theta_2(\hat{m}(n)) \approx \text{constant} \qquad (40)$$

- The scale ratio between two corresponding key points should be constant:

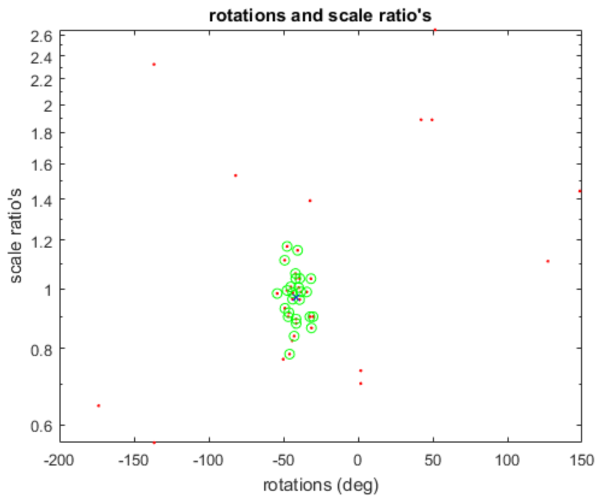
$$\sigma_2(n)/\sigma_2(\hat{m}(n)) \approx \text{constant} \qquad (41)$$

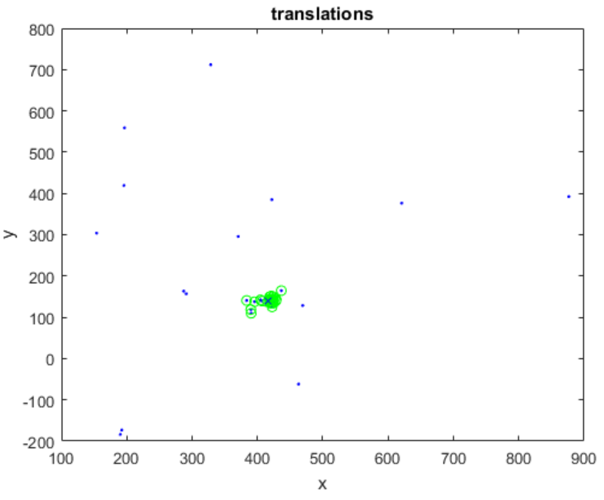
- The translation vector from a key point in the target image to the library image should be constant:

$$\mathbf{t}(n) = \mathbf{x}_1(n) - \frac{\sigma_1(n)}{\sigma_2(\hat{m}(n))}\mathbf{R}^{-1}(n, \hat{m}(n))\,\mathbf{x}_2(\hat{m}(n)) \qquad (42)$$

Here $\mathbf{R}(n, \hat{m}(n))$ is the rotation matrix that corresponds to the rotation $\theta_1(n) - \theta_2(\hat{m}(n))$.

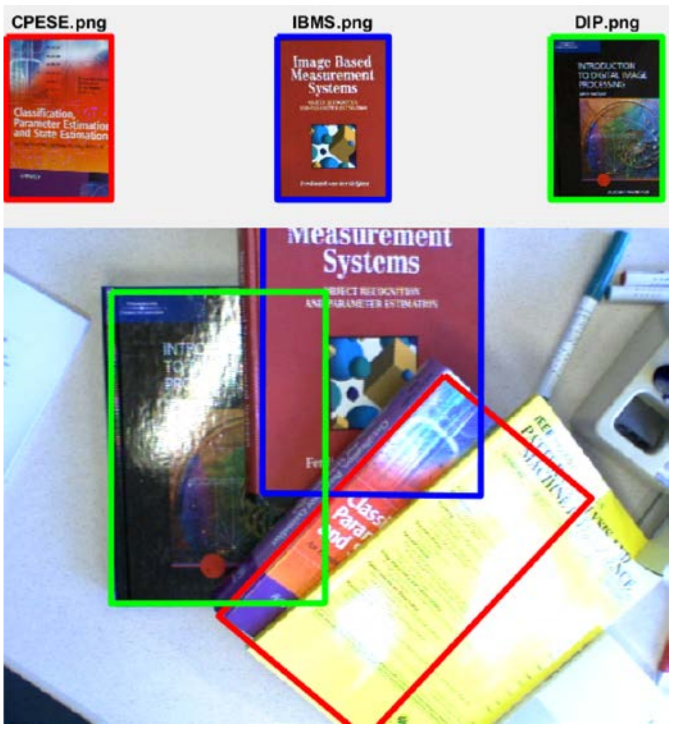
The rotations, scale ratios and translations of the matched key points are given in the figures below:





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.



Appendices

A. SIFT key points

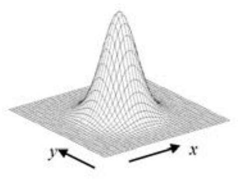
SIFT stands for scale invariant feature transform. This method to find key points was proposed by Lowe in 1999. SIFT key points are found in a 3D "Laplacian of Gaussian" scale space. As such, it is a so-called 'multiscale approach'. Features are searched at various scales.

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:

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

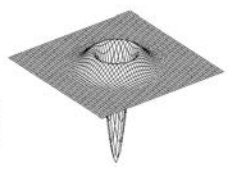
Laplacian: $f_{xx}(x,y)+f_{yy}(x,y)$



Laplacian of a Gauss:

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

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

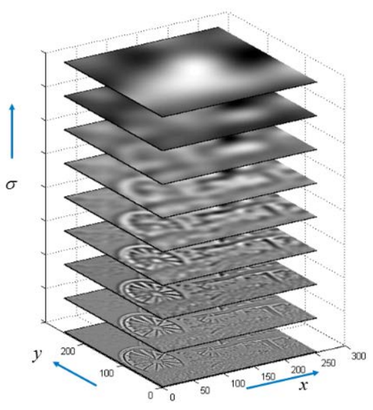


The Laplacian is the sum of the 2nd order derivatives taken into two orthogonal directions, e.g. in the x- and y-direction. The Laplacian of an image, considered at a given scale, can be computed by convolving the image with the Laplacian of the Gaussian at that scale. It is a rotational invariant operation.

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.

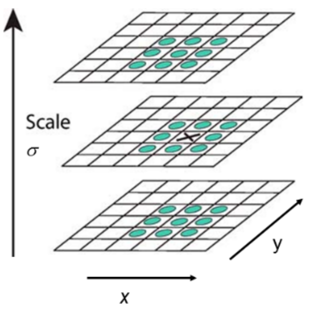
- find the extrema:

  a pixel (x,y,σ) is a <u>candidate</u> 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 $\mathbf{x}_0$:

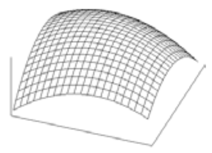
$$\Delta f(\mathbf{x}_0+\mathbf{x}) \approx \Delta f(\mathbf{x}_0) + \mathbf{x}^T \underbrace{\mathbf{g}(\mathbf{x}_0)}_{\text{gradient vector}} + \mathbf{x}^T \underbrace{\mathbf{H}(\mathbf{x}_0)}_{\text{hessian}} \mathbf{x} \quad \text{with} \quad \mathbf{x}=\begin{bmatrix} x \\ y \\ \sigma \end{bmatrix}$$

- 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



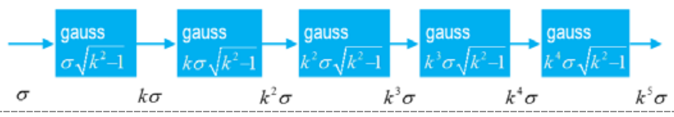
The implementation of building such a Laplacian scale space can be done very fast. A Laplacian of a Gaussian can be approximated by the difference of two Gaussians:

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

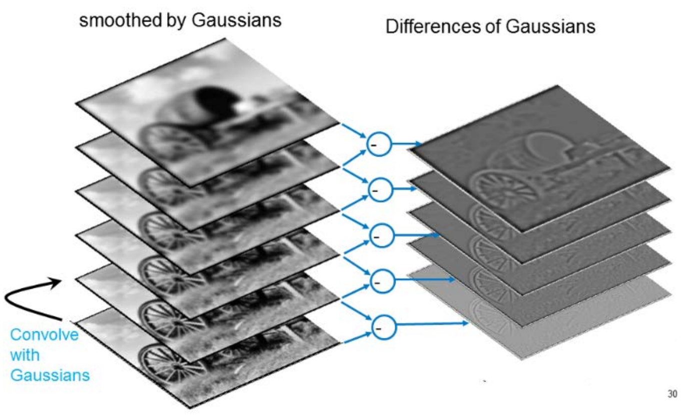
$$LoG(x,y,\sigma) \approx \frac{1}{\sigma^2(k-1)}\left(gauss(x,y,k\sigma)-gauss(x,y,\sigma)\right)$$

- cascade of Gaussians

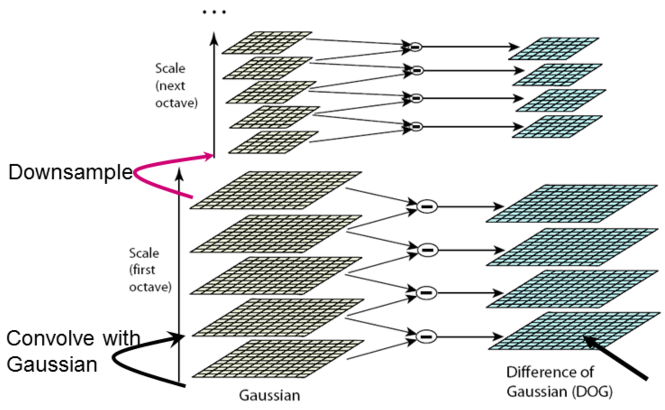
$$gauss(x,y,k\sigma)=gauss(x,y,\sigma\sqrt{k-1}) * gauss(x,y,\sigma)$$


gauss $\sigma\sqrt{k^2-1}$
gauss $k\sigma\sqrt{k^2-1}$
gauss $k^2\sigma\sqrt{k^2-1}$
gauss $k^3\sigma\sqrt{k^2-1}$
gauss $k^4\sigma\sqrt{k^2-1}$
σ
$k\sigma$
$k^2\sigma$
$k^3\sigma$
$k^4\sigma$
$k^5\sigma$

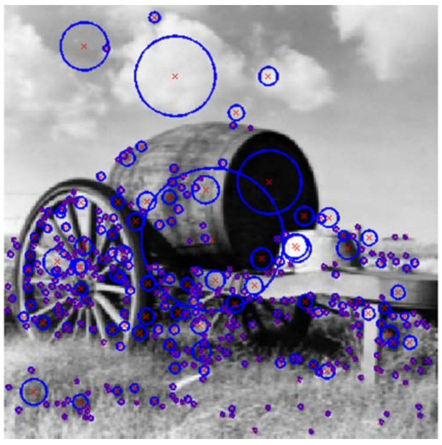
This allows building up a Gaussian scale space first, which can be done efficiently by cascaded convolutions, and then obtaining the Laplacian scale space by subtraction:



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:

$$LoG(x,y,\sigma)*f(x,y)=\frac{1}{a^2}LoG\left(x,y,\frac{\sigma}{a}\right)*f(ax,ay) \qquad (43)$$

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`.

C. Robust estimation - RANSAC

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>