Advanced

Image Processing and Stochastic Modeling

TP2: Spatial Filtering

Prof. Sviatoslav Voloshynovskiy,
Olga Taran <olga.taran@unige.ch>,
Maurits Diephuis.

Stochastic Information Processing Group

March 15, 2017

UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

SIP stochastic information processing

Submission

Please archive your report and codes in "Name_Surname.zip" (replace "Name" and "Surname" with your real name), and upload to "`Assignments/TP1: Basic Image Processing`" on `https://chamilo.unige.ch` before Thursday, April 5 2017, 23:59 PM. Note, **the assessment is mainly based on your report, which should include your answers to all questions and the experimental results**.

1 On Spatial Filtering

In image processing some neighborhood operations work with the values of the source image, which can be a single value or the values from a certain region of interest (ROI), together with the values defined in a subimage or matrix. (Figure 2. This subimage is commonly called a *filter* or *kernel*. The values in the *kernel* are usually referred to as the *coefficients*.

For now, we will concentrate on filtering operations that are performed directly on the image values: Filtering in the Spacial domain.

Convolution

Convolution of functions f and g is by definition:

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

The discrete convolution of f and g is given by:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

The result of a convolution between two functions, is thus a new function. Let there be two functions:

$$f_1(n) = \delta(n-1) + 2\delta(n-2) + \delta(n-3) + 2\delta(n-4) \quad (1)$$

$$f_2(n) = \delta(n-1) + 3\delta(n-2) + \delta(n-3) + 2\delta(n-4) \quad (2)$$

The result $(f_1 * f_2)(n)$ can be seen in Figure 1.

Exercise

1. Let $x_1 = [1, 0, 0]$ and $x_2 = [0, 1, 0]$. Determine $x_1 * x_2$ manually and check your answer with Matlab `conv` and `stem`.

2. Let $x_1 = [0, 1, 0, 0, 0, 0]$ and $x_2 = [1, 1, 1, 1, 1, 1]$. Determine $x_1 * x_2$ manually and check your answer with Matlab `conv` and `stem`.

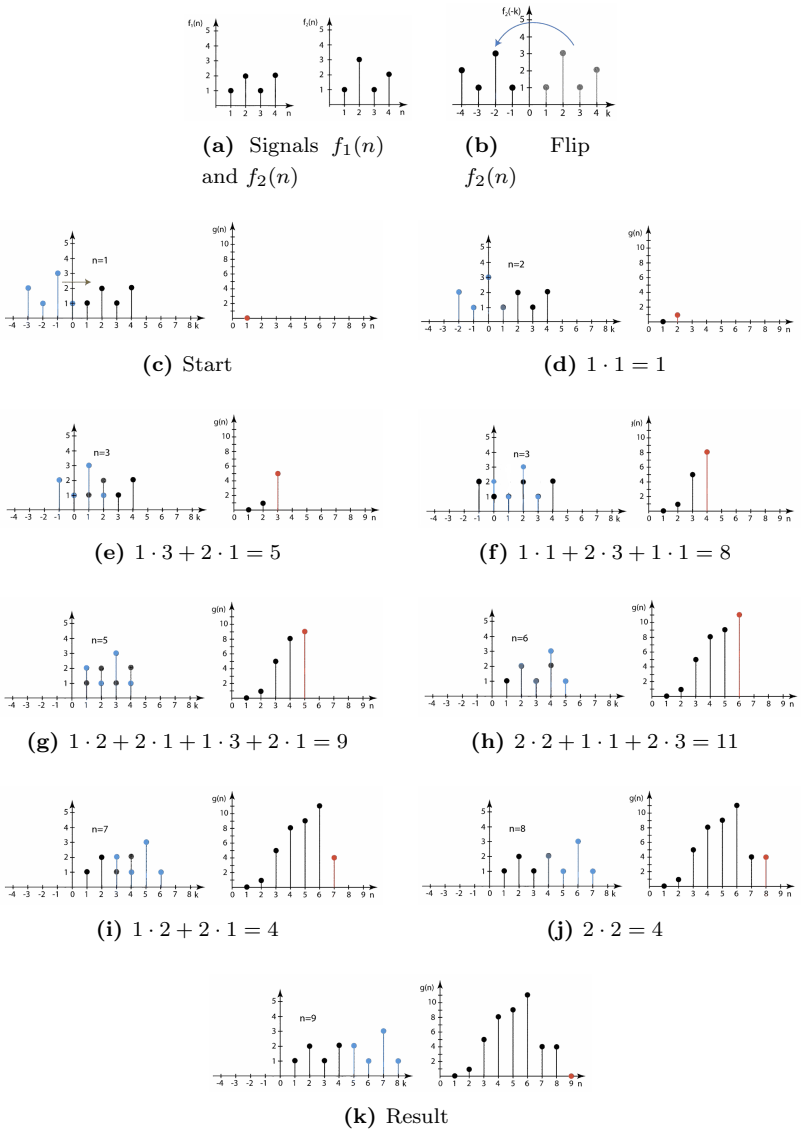
Exercise

Let there be the functions $h(n)$ and $x(n)$:

$$h(n) = \delta(n) + 2\delta(n-1) + 3\delta(n-2) \tag{3}$$

$$x(n) = \delta(n+1) + \delta(n) + \delta(n-1) + \delta(n-2) \tag{4}$$

1. convolve $h(n)$ with $x(n)$, **without** Matlab `conv`.



(a) Signals $f_1(n)$ and $f_2(n)$ **(b)** Flip $f_2(n)$

(c) Start **(d)** $1 \cdot 1 = 1$

(e) $1 \cdot 3 + 2 \cdot 1 = 5$ **(f)** $1 \cdot 1 + 2 \cdot 3 + 1 \cdot 1 = 8$

(g) $1 \cdot 2 + 2 \cdot 1 + 1 \cdot 3 + 2 \cdot 1 = 9$ **(h)** $2 \cdot 2 + 1 \cdot 1 + 2 \cdot 3 = 11$

(i) $1 \cdot 2 + 2 \cdot 1 = 4$ **(j)** $2 \cdot 2 = 4$

(k) Result

Figure 1 – Discrete Convolution Example

2 Spatial Filtering

The basic working of spacial filtering is illustrated in Figure 2. It simply consists of moving the filter mask from point to point in the image. At each point, the *response* of the filter is calculated. For linear spatial filtering, the response is the sum of the products of the filter coefficients and the corresponding image pixels currently under the filter mask. For the 3×3 filter in Figure 2, the response r is thus:

$$\begin{aligned} r =& w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y)+ \\ & ... + w(1,1)f(x+1,y+1) \end{aligned}$$

This notation, for a $m \times n$ sized mask is often informally denoted as

$$\begin{aligned} r =& w_1 f_1 + w_2 f_2 + ... + w_{mn} f_{mn} \\ r =& \sum_{i=1}^{mn} w_i f_i \end{aligned}$$

Which is the sum of the products of the filter coefficients against the pixels directly under the mask. Note that in this particular example the filter is centered at (x, y). The operation is called *space invariant* or *shift invariant* if the filter operation does not depend on the pixel position. Further more, if the operation is linear, it is in fact identical to *discrete convolution*.

An important consideration is what to do at the border of an image, when part of the filter mask falls outside the image:

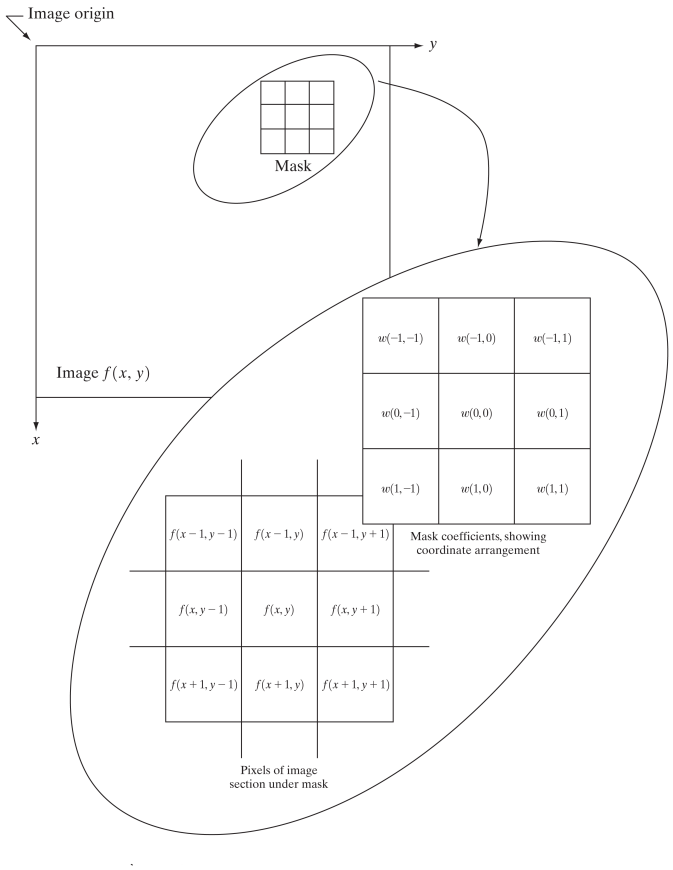
- One can simply discard the border pixels, in that case, the output image will be smaller than the input.
- If the output image is to be the same size as the input image, one can pad the border either with 1's or 0's.
- One can replicate the image values, so that the image becomes larger. After discarding the border pixels, the output image will be of identical size as the input image. This last method is known as *circular convolution*.

2.1 Low Pass Filtering

An operator that cuts of high frequencies while passing through low-frequencies, is a *low-pass filter*. Examples of low-pass filters are the *averaging* operator and the *Gaussian* filter.

An averaging filter can be implemented by convolving the following mask with an image:

$$\mathbf{h} = \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$



(a)

Figure 2 – Example of linear filtering with 3×3 Region of Interest and a 3×3 kernel

The Gaussian filter can be approximated by:

$$\mathbf{h} = \frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

Exercise

- Implement and test the averaging filter for any image.
- Implement and test the Gaussian filter for any image.
- How could these matrices be modified to increase the blurring effect, i.e. remove more high frequency components.

2.2 Order-Statistics Filtering

Order-statistics filters are non-linear filters whose response is based on the ranking (or ordering) of the pixels currently under the filter mask. The central pixel is then replaced by the result of the ranking operation. The best known example is the so called *median* filter. This filter replaces the central pixel by the median value of the region around it. Median filters are particularly capable in dealing with *salt & pepper* noise.

Exercise

- Read in an image and corrupt it with 5-10 various degrees of *salt & pepper* noise.
- Try to remove the *salt & pepper* noise with an averaging filter.
- Try to remove the *salt & pepper* noise with a median filter.
- Test the median filter for various neighbourhood sizes.
- Report all results.

2.3 High Pass Filtering

High pass filtering cuts of low frequency and only passes on high frequency content. Informally, you have seen applications of such filters in *edge detection* and in *sharpening* algorithms. Before jumping into application we will have a look at some of the fundamental properties of discrete derivatives and digital imaging.

The basic definition for a discrete 1 dimensional first and second order derivative are:

$$\begin{aligned} \frac{\partial f}{\partial x} &= f(x+1) - f(x) \\ \frac{\partial^2 f}{\partial x^2} &= f(x+1) + f(x-1) - 2f(x) \end{aligned} \tag{5}$$

Exercise

Given the following discrete signal: $x = [5, 7, 6, 5, 4, 3, 2, 1, 0, 5, 4, 3, 2, 1, 0, 0, 0, 6, 0, 0, 0, 0, 1, 3, 1, 0, 0, 0, 0, 7, 7, 7, 7]$:

1. Determine the first and second order derivative
2. Plot the original and the derivatives
3. What difference can you see between the first and second order derivative

2.4 The Laplacian

The second order derivative is known as the *Laplacian* and it is used extensively in image enhancement. Formally it is defined as follows:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{6}$$

As derivatives of any kind are linear operations, the Laplacian is also a linear operator. In order to be useful for digital image processing, we need to express the Laplacian in the discrete form. In x-direction:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \tag{7}$$

and in y-direction:

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y) \tag{8}$$

The final digital form is then obtained by summing these components:

$$\nabla^2 f = (f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)) - 4f(x, y) \tag{9}$$

Exercise

- Implement and test (apply to any image) a Laplacian filter based on Equation 9 and a 3×3 neighbourhood.

Other matrices that implements a Laplacian filters are:

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Exercise

- Implement and test (apply to any image) the other Laplacian Filter implementations.

The Laplacian can be used for basic image enhancement in the following way:

$$g(x, y) = \begin{cases} f(x, y) - \nabla^2 f(x, y) & \text{If mask center is negative} \\ f(x, y) + \nabla^2 f(x, y) & \text{If mask center is positive} \end{cases} \tag{10}$$

Exercise

- Implement and test (apply to any image) Equation 10

Equation 10 can be implemented in a single filter:

$$\begin{aligned} g(x, y) = f(x, y) - (&f(x+1, y) + f(x-1, y) + f(x, y+1) + \dots \\ &+ f(x, y-1)) + 4f(x, y) \end{aligned} \tag{11}$$

Exercise

- Simplify, implement and test (apply to any image) Equation 11

2.5 Unsharp masking

Unsharp masking is a process used commonly in the print industry and in digital camera's for image enhancement. Formally:

$$f_{um}(x,y) = f(x,y) - \bar{f}(x,y) \tag{12}$$

where $\bar{f}(x,y)$ is a low-pass filtered (blurred) image.

Exercise

1. Perform unsharp masking using the Gaussian and average filter from Section 2.1.
2. Given the following two Laplacian matrices:

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad \begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Compare the results from unsharp filtering to the results of directly filtering with a Laplacian filter.

3 The Covariance Matrix and the Eigenvectors

In this introduction we will have a look at how the *covariance* matrix characterizes a dataset, and how linear operations applied to that dataset effect it.

Exercise

1. Generate 2x100 realisations using `randn` with zero mean and unit variance. These values, $\mathbf{x}$, are realisations of two random variables X_1 and X_2 that are Gaussian distributed, i.e $X_i \sim \mathcal{N}(0,1)$. Plot them.
2. What is the expectation of X_i, $E[X_i]$ (or $\mu_{\mathbf{X_i}}$).
3. What is the value of $var[X_1]$ and $var[X_2]$.

The covariance between two random variables is a measure how much the two numbers covariate (to covariate: if one variable fluctuates, the other variable will fluctuate proportionally, and vice verse).

$$\begin{aligned} cov(X_1, X_2) &= E(X_1X_2) - E(X_1)E(X_2) \\ &= E(X_1X_2) - \mu_{X_1}\mu_{X_2} \end{aligned} \tag{13}$$

As the two elements in the vector $\mathbf{x}$ are independently generated their covariance is per definition **zero**:

$$cov(X_1, X_2) = 0 \tag{14}$$

Let $Y = (X_1; X_2)$. Then the covariance matrix of Y is defined as:

$$\mathbf{C}_Y = \begin{bmatrix} var[X_1] & cov[X_1, X_2] \\ cov[X_1, X_2] & var[X_2] \end{bmatrix} \tag{15}$$

Associated with a covariance matrix and an expectation vector is an ellipsoid (in two dimensions: an ellipse) that can be regarded as a region of uncertainty around the expectation.

Exercise

1. Determine $\mathbf{C}_Y$ empirically.
2. Use the function `plotcov` to draw the ellipse associated with $\boldsymbol{\mu}_Y$ and $\mathbf{C}_Y$. When plotting, use `axis equal` to correctly plot circles.

If realisations $\mathbf{y}$ from random vector Y with covariance matrix $\mathbf{C}_Y$ are linearly transformed:

$$\mathbf{y}' = \mathbf{A}\mathbf{y} \tag{16}$$

Then the covariance matrix and expectation for the resulting vector $\mathbf{y}'$ are:

$$\mathbf{C}_{Y'} = \mathbf{A}\mathbf{C}_Y\mathbf{A}^T \tag{17}$$

$$\boldsymbol{\mu}_{Y'} = \mathbf{A}\boldsymbol{\mu}_Y \tag{18}$$

Exercise

1. Let linear operator $\mathbf{A}$ be defined as:

$$\mathbf{A} = \begin{pmatrix} \sqrt{2} & 0 \\ 0 & 1/\sqrt{3} \end{pmatrix}$$

   Apply this operator to the dataset generated by Y to form realizations $\mathbf{g}$
2. Calculate the covariance matrix $\mathbf{C}_G$ and $\boldsymbol{\mu}_G$.
3. Plot the resulting new dataset $\mathbf{g}$ together with the associated ellipsoid. Explain its shape.
4. Define a new linear operator $\mathbf{A}$:

$$\mathbf{A} = \begin{pmatrix} \cos(\pi * 30/180) & -\sin(\pi * 30/180) \\ \sin(\pi * 30/180) & \cos(\pi * 30/180) \end{pmatrix}$$

   And apply $\mathbf{A}$ to the dataset $\mathbf{y}$ generated by Y to form realizations $\mathbf{e}$.

5. Calculate the covariance matrix $\mathbf{C}_E$ and $\boldsymbol{\mu}_E$.

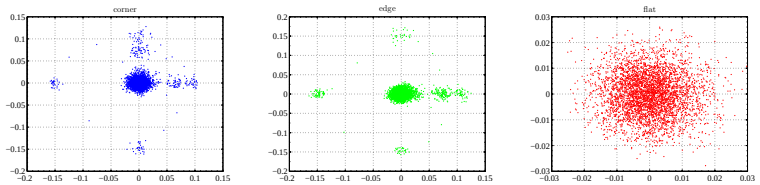
6. Plot the resulting new dataset $\mathbf{e}$ together with the associated ellipsoid.

7. Determine the *eigenvalues* and *eigenvectors* of $\mathbf{C}_E$ and $\mathbf{C}_G$. Explain their values.

4 Feature Detector

In this project, we will use spatial filtering and eigenvalues to design a so called image feature detector. This particular algorithm can find edges and corners in images. This type of feature has many applications in image domain from identifying image content to automatically stitching panorama images together. See Figure 5.

Note: In this Section, the derivatives of some function f or a matrix $\mathbf{m}$ in x direction will de denoted as: $\frac{\partial f}{\partial x} = f_x$ and $\frac{\partial^2 f}{\partial x^2} = f_{xx}$



(a) Corner **(b)** Edge **(c)** Flat

(d) **(e)** **(f)**

Figure 3 – $\mathbf{I}_x$ and $\mathbf{I}_y$ values for different image patches.

4.1 Feature Detection Algorithm

The basic algorithm is as follows:

1. Compute the image derivatives in both x and y direction. This can be done by convolving the image with a first order Gaussian derivative. The Gaussian derivatives G'^{x}_{σ} and G'^{y}_{σ} may be estimated with the 3×3 matrix kernel $\mathbf{A}_x$ for both x and y direction:

$$\mathbf{A}_x = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad \mathbf{A}_y = \mathbf{A}_x^T$$

$$\mathbf{I}_x = G_\sigma^{'x} * \mathbf{I} \tag{19}$$
$$\mathbf{I}_y = G_\sigma^{'y} * \mathbf{I} \tag{20}$$

2. For all pixels, calculate the product of the derivatives:

$$\mathbf{I}_{xx} = \mathbf{I}_x . * \mathbf{I}_x \tag{21}$$
$$\mathbf{I}_{yy} = \mathbf{I}_y . * \mathbf{I}_y \tag{22}$$
$$\mathbf{I}_{xy} = \mathbf{I}_x . * \mathbf{I}_y \tag{23}$$

3. Convolve with a Gaussian filter. This will blur the intermediate results we have so far. Generate the Gaussian filter with a size $6 * \sigma$, meaning (+/- $3 * \sigma$). σ is thus a function parameter that must be set by the user. Ensure that σ has a minimum value of 1. You may use Matlab `fspecial` to generate the filter. See **Hints**. Then convolve as follows:

$$\mathbf{L}_{xx} = G_{\sigma^w} * \mathbf{I}_{xx} \tag{24}$$
$$\mathbf{L}_{yy} = G_{\sigma^w} * \mathbf{I}_{yy} \tag{25}$$
$$\mathbf{L}_{xy} = G_{\sigma^w} * \mathbf{I}_{xy} \tag{26}$$

4. Form matrix $\mathbf{m}$ at each pixel point (x, y):

$$\mathbf{m}(x,y) = \begin{bmatrix} L_{xx}(x,y) & L_{xy}(x,y) \\ L_{xy}(x,y) & L_{yy}(x,y) \end{bmatrix} \tag{27}$$

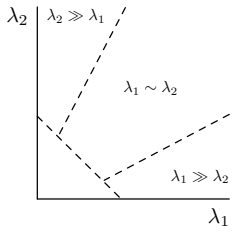
5. Matrix $\mathbf{m}$(x, y) has a number of interesting properties. The *eigen values* λ_1 and λ_2 of Matrix $\mathbf{m}$(x, y) for each point (x, y) give an indication if that particular point lies on an edge or on a corner. See Figure 4. A point on an edge is characterized by the fact that one of the eigenvalues is significantly larger than the other: $\lambda_1 \gg \lambda_2$ or $\lambda_2 \gg \lambda_1$. A corner is indicated by sufficiently large and similar eigenvalues. The 'corners-ness' measure $\mathbf{r}$ is defined as follows:

$$\mathbf{r} = \det \mathbf{m} - k(\text{trace } \mathbf{m})^2 \tag{28}$$
$$\det \mathbf{m} = \lambda_1 \lambda_2 \tag{29}$$
$$\text{trace } \mathbf{m} = \lambda_1 + \lambda_2 \tag{30}$$

To prevent the explicit calculation of eigenvalues, which is computationally intensive, one can determine the corner response $\mathbf{r}$ for all image points (x, y) as follows:

$$r(x,y) = \frac{L_{xx}(x,y) \cdot L_{yy}(x,y) - (L_{xy}(x,y))^2}{L_{xx}(x,y) + L_{yy}(x,y)} \tag{31}$$

An example of the corner reponse $\mathbf{r}$ can be seen in Figures 5b and 5c.



(a)

Figure 4 – Eigenvalues from matrix **m**. Corners are characterized by $\lambda_1 \sim \lambda_2$ where both are of significant value. Edges are indicated by $\lambda_1 \gg \lambda_2$ or $\lambda_2 \gg \lambda_1$ for large enough values.

6. Optionally, you can implement a measure that stipulates that a potential corner $r(x, y)$ is only accepted if it is significantly larger than all other 'corner-ness' scores in a certain region of interest.
7. Find points for which $r(x, y)$ is the most strong, and write out their coordinates.

Exercise

1. Implement the detector and run it on a number of images.
2. Plot the found corners with a marker over the input images.

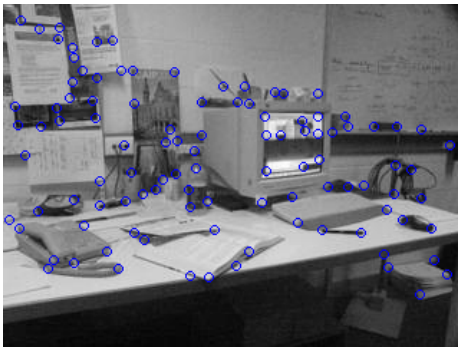
Exercise

The corner response **r** is rotation invariant, which naturally, is a direct consequence of the fact that it is based on eigenvalues. The feature points are, however, not invariant to scale. Large corners can also be classified as two big edges pending the scale the detector is working in. This scale can be set by function parameter σ of the Gaussian filter that was used in step x to blur the derivatives.

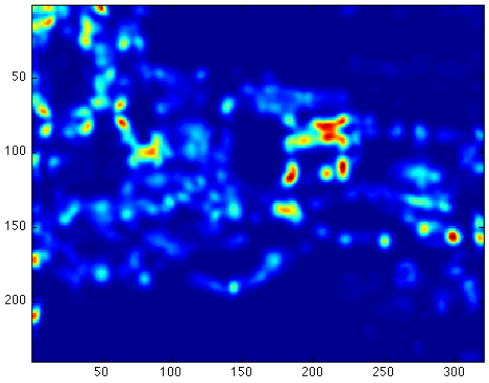
1. Experiment with parameter σ
2. Run the detector on the 'square' images from `Chamilo` with different values of σ. Show the results.

Hints

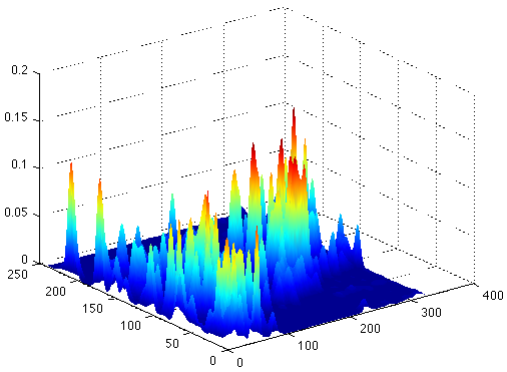
- You may generate the Gaussian filter of step 3 as follows:
  `g = fspecial('gaussian',max(1,fix(6*sigma)), sigma);`
- Do not define a separate matrix **m** for all pixels for $\mathbf{L}_{xx}$, $\mathbf{L}_{xy}$ and $\mathbf{L}_{yy}$. Access all separate **L**-components when needed.
- To find the local maxima in the corner response matrix or image **r**, you can use Matlab `ordfilt2`.



(a) Detected corners



(b) Corner response **r**



(c) Corner response **r**

Figure 5 – Detecting corners and Edges

Notes

Features like the one you just implemented are used throughout computer vision for applications such as semantic object recognition, image stabilization and "stitching" of multiple images into a panorama.