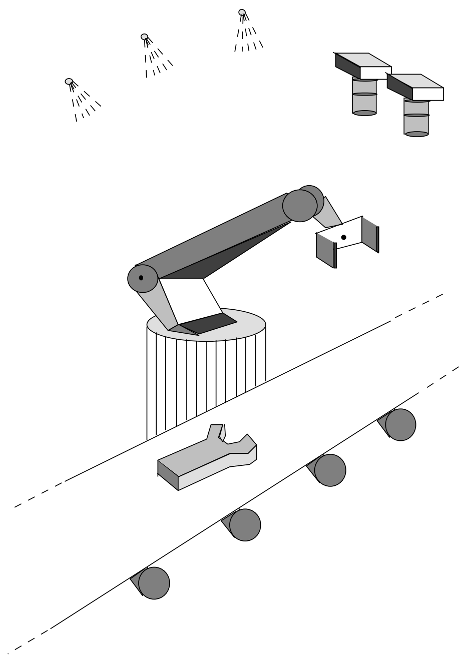
University of Groningen **Faculty of Mathematics and Natural Sciences**

Department of Mathematics and Computing Science

Lab Manual Computer Vision

M.H.F. Wilkinson
G.K. Ouzounis

August 2004



price € 3,–

RuG

Contents

1 Introduction

The practical assignment of the course on Computer Vision consists of a number of exercises from the following sessions.

1.1 Procedure

To qualify for the practical assignment there are the following requirements:

1. Reports of all sessions (except of the first session in which you will work with the image processing package ImageJ) with a concise explanation of each problem. All questions posed have to be answered.

2. All requested java-plugins (extension `.java`) with:

   - the name *as used in this document*;
   - a header of the form
     ```
     /* Author(s): .... */
     /* Date: .... */
     /* NAME: ....
     DESCRIPTION:....
     etc.*/
     ```
     containing the name(s) of the author(s), the date. (N.B. In a java file every line of comment has to begin with "`/*`" and end with "`*/`".)

3. Reports and java-plugins have to be handed in before the appropriate date to the lab instructor.

N.B. Whenever you are asked to *record* a *java-plugin* in the assignments below, you are also permitted to solve the problem by *writing* the plugin, and vice versa. You have to specify in the plugin file in which display windows you expect input/output, and assignment of parameter values must be explicitly done in the plugin file.

2 Working with ImageJ

In the first practical session you will learn some elementary aspects of the image processing package ImageJ.

To do so, you need to make use of the ImageJ manual, see Appendices D and E of this manual. Make sure during the first week that you acquire practical skills for working with ImageJ. You will need them in the rest of the course. If you have problems ask for help.

Exercise 1.

Before you use ImageJ, it is necessary to copy a few files to your local directory.

1. Create a directory `ImageJ`. In this directory, make a subdirectory with the name `plugins`.
2. Go to the `plugins` directory and make symbolic links by typing the following commands:

   ```
   ln -s /wing6/home/vakken/ComputerVision/ImageJ/plugins/ComputerVision .
   ln -s /wing6/home/vakken/ComputerVision/ImageJ/plugins/Demos .
   ```

3. Copy the file `/wing6/home/vakken/ComputerVision/ImageJ/imagej` to your private ImageJ directory. Open this file in an editor and look for the string
   ‘`-Dplugins.directory=/wing6/home/vakken/ComputerVision/ImageJ`’
   which defines the path to the plugins directory.
   Replace ‘`/wing6/home/vakken/ComputerVision/ImageJ`’ by your local `ImageJ` directory which you created in step 1. Save your modified file and leave the editor.
4. Start ImageJ by typing `imagej` at the command prompt.

Exercise 2.

Read the introductory material “ImageJ” of the IMAGEJ manual (Appendix E).

Exercise 3.

Follow the examples in “Examples of Image Analysis Using ImageJ” (Appendix E).

Exercise 4.

All lab sessions can be done as sequences of ImageJ menu commands, which you can *record* to create plugins. If you choose to record your plugins, then you should go through the “Recording ImageJ plugins” tutorial (Appendix D). If you solve the problems by *writing* plugins, then you should read the “Writing ImageJ plugins” tutorial (Appendix E). If you are not sure which method to choose, then read the “Recording ImageJ plugins” tutorial.

Important!!

All ImageJ operations are performed on the image which is currently selected (`Window/...`). To prevent the problem that you cannot select a different image from the menu, you have to name the images in such a way that they have a unique prefix. Apparantly, ImageJ cannot distinguish between images with names like `A`, `AA`, `AAA`, etc. It is best to use names like `1_A`, `2_AA`, `3_AAA`, etc.

3 Linear image filtering

3.1 Linear image restoration

By image restoration we denote the process of removing or reducing image degradations which occur during image formation. A simple model assumes linear degradation of the ideal image $f(x, y)$ described by a convolution kernel $k(x, y)$ and additive noise $n(x, y)$, resulting in the degraded image $w(x, y)$:

$$w(x, y) = (k * f)(x, y) + n(x, y). \tag{3.1}$$

Restoration now has to undo the degradation due to convolution and noise. Using a linear model for restoration (also called *deconvolution*), we compute an estimate $\tilde{f}(x, y)$ of the ideal image by using a convolution kernel $r(x, y)$:

$$\tilde{f}(x, y) = (r * w)(x, y)$$

The complete process of degradation and restoration is graphically represented in Fig. 1. A simplified

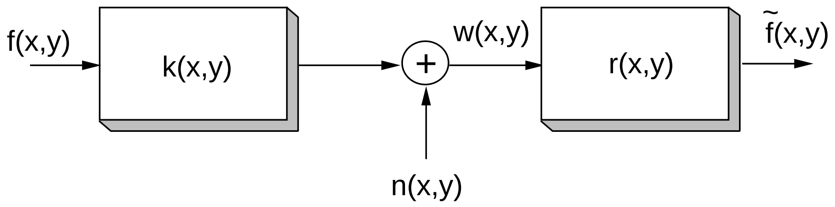


Figure 1: *Linear image restoration model.*

restoration problem results if we consider images of finite size, say $N \times N$. In that case we can work with matrix-vector calculus. The restoration problem (3.1) then has the form

$$\mathbf{w} = \mathbf{K}\,\mathbf{f} + \mathbf{n} \tag{3.2}$$

where $\mathbf{w}, \mathbf{f}, \mathbf{n}$ are column vectors of length N^2, and $\mathbf{K}$ is an $N^2 \times N^2$ matrix representing the convolution. For shift-invariant systems, the matrix $\mathbf{K}$ has a periodic structure which can be exploited to achieve efficient computation using the Fast Fourier Transform, see Appendix A.

Constrained least squares restoration A number of well-known restoration filters is captured by considering the following *constrained least squares restoration* model. First, the estimated solution vector $\tilde{\mathbf{f}}$ should satisfy (3.2); this gives the constraint that the norms of $\mathbf{w} - \mathbf{K}\tilde{\mathbf{f}}$ and $\mathbf{n}$ are equal:

$$\left\|\mathbf{w} - \mathbf{K}\tilde{\mathbf{f}}\right\|^2 = \|\mathbf{n}\|^2 \tag{3.3}$$

Second, one may subject the solution $\tilde{\mathbf{f}}$ to additional constraints; this can be realized by introducing a matrix $\mathbf{Q}$ such that the term

$$\left\|\mathbf{Q}\,\tilde{\mathbf{f}}\right\|^2 \tag{3.4}$$

is minimized. To take both contributions into account, we want to minimize

$$\mathcal{E}(\tilde{\mathbf{f}}) = \left\|\mathbf{Q}\,\tilde{\mathbf{f}}\right\|^2 + \gamma^{-1}\left(\left\|\mathbf{w} - \mathbf{K}\tilde{\mathbf{f}}\right\|^2 - \|\mathbf{n}\|^2\right) \tag{3.5}$$

where γ is a constant, called the *regularization parameter*, which determines the relative weight of the terms (3.3) and (3.4). The solution of (3.5) is:

$$\tilde{\mathbf{f}} = \left(\mathbf{K}^t\mathbf{K} + \gamma\,\mathbf{Q}^t\mathbf{Q}\right)^{-1}\mathbf{K}^t\mathbf{w} \tag{3.6}$$

where the superscript ‘t’ denotes transposition of vectors or matrices. Special cases are the following:

Pseudoinverse filter

Take $\mathbf{Q} = \mathbf{I}$, the identity matrix. Thus we minimize the norm of $\tilde{\mathbf{f}}$ subject to the constraint (3.3). The solution (3.6) then reduces to:

$$\tilde{\mathbf{f}} = \left(\mathbf{K}^t\mathbf{K} + \gamma\,\mathbf{I}\right)^{-1}\mathbf{K}^t\mathbf{w} \tag{3.7}$$

An alternative formulation of this result makes use of the discrete Fourier transforms $K[u,v]$, $W[u,v]$, $F[u,v]$ of $k[x,y]$, $w[x,y]$, $f[x,y]$ in the discrete analog of (3.1). The estimate $\tilde{F}[u,v]$ of $F[u,v]$ according to the pseudoinverse filter has the form:

$$\tilde{F}[u,v] = \left(|K'[u,v]|^2 + \gamma\right)^{-1} K'^*[u,v]\,W[u,v]. \tag{3.8}$$

Here

$$K'[u,v] = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} k-[x,y]\,e^{-i2\pi(ux/M+vy/N)} = \frac{K[u,v]}{c_N\,c_M}, \tag{3.9}$$

that is, $K'[u,v]$ equals the discrete Fourier transform $K[u,v]$ of $k[x,y]$, *apart from the normalization factor* used in the definition of the DFT, see Appendix A. By $K'^*[u,v]$ we denote the complex conjugate of $K'[u,v]$.

Inverse filter

For $\gamma = 0$ the formula for the pseudoinverse filter reduces to the 'inverse' filter:

$$\tilde{\mathbf{f}} = (\mathbf{K}^t\mathbf{K})^{-1}\mathbf{K}^t\mathbf{w} = \mathbf{K}^{-1}\mathbf{w} \tag{3.10}$$

or, in terms of Fourier transforms,

$$\tilde{F}[u,v] = \frac{W[u,v]}{K'[u,v]} = F[u,v] + \frac{N[u,v]}{K'[u,v]}, \tag{3.11}$$

where $K'[u,v]$ is as in (3.9) and $N[u,v]$ is the Fourier transform of the noise term $n[x,y]$ in (3.1).

3.2 Edge detection via Gaussian derivatives

The use of (discrete versions of) spatial derivatives to detect edges has the disadvantage of being very noise sensitive. For this reason methods have been developed to stabilize this operation by an averaging procedure. One of these methods is the `LOG` (Laplacian of Gaussian) edge detector. Here one first performs a convolution with a Gaussian kernel, followed by a second order derivative via a convolution with a Laplace kernel. Because the convolution is an associative operation, one can carry out a single convolution with the Laplacian of the Gaussian kernel, given by:

$$\begin{aligned}\mathtt{LOG}(x,y) &= \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\frac{1}{2\pi\sigma^2}e^{-\frac{1}{2}\left(\frac{x^2+y^2}{\sigma^2}\right)} \\ &= \frac{-1}{2\pi\sigma^4}\left(2 - \frac{x^2+y^2}{\sigma^2}\right)e^{-\frac{1}{2}\left(\frac{x^2+y^2}{\sigma^2}\right)}\end{aligned} \tag{3.12}$$

Gaussian derivatives filtering is also used in constructing a multiscale representation (*linear scale-space*) of images.

3.3 EXERCISES

3.3.1 Linear image restoration

Exercise 1.

Convolution theorem A discrete (circular) convolution has the form:

$$g[x,y] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} k[x-m, y-n]\, f[m,n], \tag{3.13}$$

for $x = 0, 1, \ldots, M-1, y = 0, 1, \ldots, N-1$. The index calculation in $k[x-m, y-n]$ must be done modulo M, N, respectively. Let F, G and K be the 2D forward discrete Fourier transforms of f, g and k, see Appendix A.

Prove that

$$G[u,v] = \frac{K[u,v]}{c_N c_M} F[u,v],$$

where c_N is the factor used in the forward discrete Fourier transform.

3.3.2 Edge detection via Gaussian derivatives

Exercise 2.

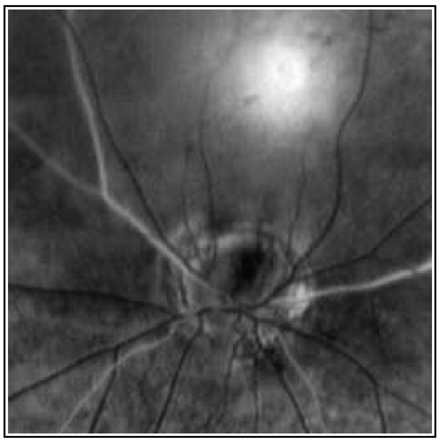
Write an ImageJ plugin (filename `Log_.java`) which computes the `LOG` edge detector of an input image and puts the result in an output image, for a given value of σ. You should use the ImageJ plugins:

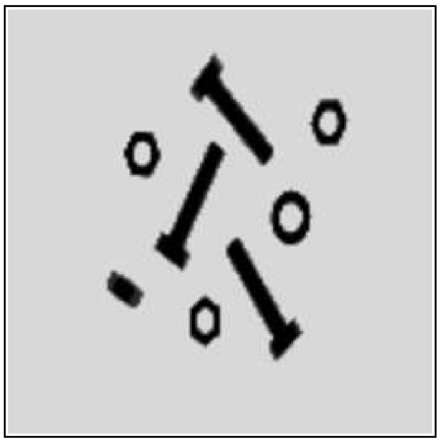
```
DirectFFT (Direct FFT transform )
InverseFFT (Inverse FFT transform )
Laplacian (the Laplace operator)
Gaussian Filter (Gaussian kernel)
```

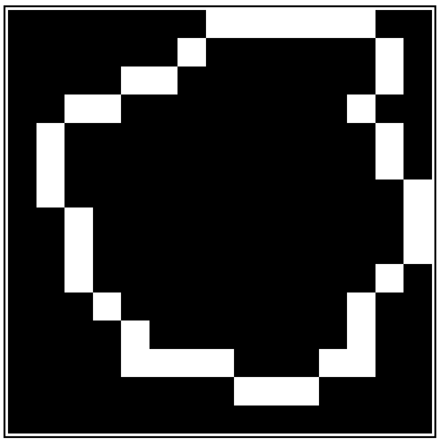
Note: Only use input images of size 256×256.

Exercise 3.

Apply the routine to the image `acros`, see Fig. 2d. Try a number of values of σ between 1.0 and 10.0. Which value gives the best result? Include in your answer a description of what *you* mean by “best”.


oog4


prak1


kromme

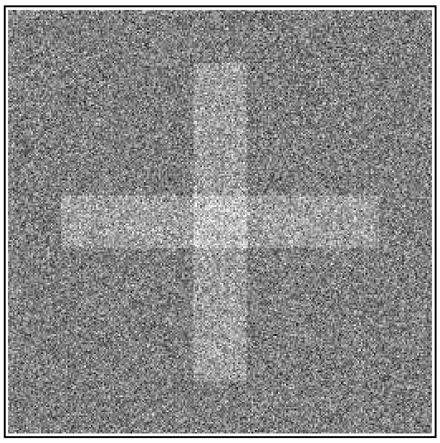

acros

Figure 2: *The images mentioned in the text.*

4 Morphological image processing I

4.1 Connected components

A binary image consists of 1-pixels (pixels with value 1, foreground) and 0-pixels (pixels with value 0, background). Connected components, are maximal connected regions of 1-pixels in the image. When only the pixels to the north, south, east and west of a pixel are considered as the neighbors of this pixel, this pixel with its 4 neighbors are said to form a *4-connected* region. When the pixels to the north, south, east, west, northeast, northwest, southeast and southwest of a pixel are considered as the neighbors of this pixel, this pixel with its 8 neighbors are said to form an *8-connected* region.

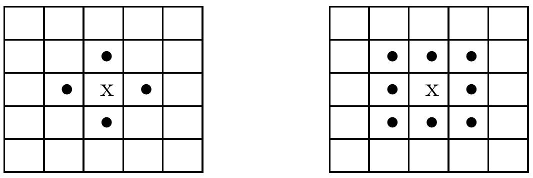


Figure 3: *Pixels • that are 4-connected (left) or 8-connected (right) to the center pixel* x.

Definition 1 *1-pixels p and q are said to belong to the* connected component C *if there is a sequence of 1-pixels $(p_0, p_1, \ldots, p_n)$ within C with $p_0 = p$, $p_n = q$ and p_i a* neighbor *of p_{i-1} for $i = 1, 2, \ldots, n$.*

Note that the result of applying this definition depends on the type of neighbor relation used (4-connectivity or 8-connectivity). The *border* of a connected component C of 1-pixels is the set of pixels of C which are a neighbor of 0-pixels.

4.2 Minkowski operations

Let A be a set in the plane: the '*structuring element*'. We write $x + y$ for the vector sum of x and y, and $-x$ for the reflected vector of x w.r.t. the origin. If we translate (i.e. shift) A along the vector h, we write the result as A_h, so

$$A_h = \{a + h : a \in A\},$$

see Fig. 4. As before, 1-pixels are represented by dots. The position of the origin in these sets is indicated by the symbol ⮣

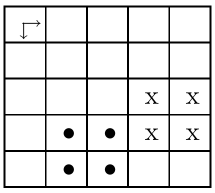


Figure 4: *A set X (black dots) and its shift X_h (crosses) with $h = (2, -1)$.*

Also, if we reflect all points of a set A, we get the *reflected set* $\check{A}$, i.e.

$$\check{A} = \{-a : a \in A\}.$$

Now we can define the Minkowski addition and subtraction:

$$addition \quad : \quad X \oplus A = \{x + a : x \in X, a \in A\} = \bigcup_{a \in A} X_a = \bigcup_{x \in X} A_x \tag{4.1}$$

$$subtraction \quad : \quad X \ominus A = \bigcap_{a \in A} X_{-a} \tag{4.2}$$

The dilation and erosion operations, denoted by δ and ε, respectively, are defined as follows. Let the set A be fixed.

$$\begin{aligned} &Dilation && \delta_A(X) = X \oplus A && (4.3) \\ &Erosion && \varepsilon_A(X) = X \ominus A. && (4.4) \end{aligned}$$

There is a simple geometrical interpretation:

$$\begin{aligned} Dilation &: \quad X \oplus A = \{x \in E : (\check{A})_x \cap X \neq \emptyset\} && (4.5) \\ Erosion &: \quad X \ominus A = \{x \in E : A_x \subseteq X\}. && (4.6) \end{aligned}$$

In words, the dilation of X by A is the collection of points h such that the translation of $\check{A}$ over the vector h 'hits' the set X. Similarly, the erosion of X by A is the collection of points h such that after shifting A over the vector h it 'fits' into X. We illustrate this for the discrete case in Fig. 5 and Fig. 6. There exists

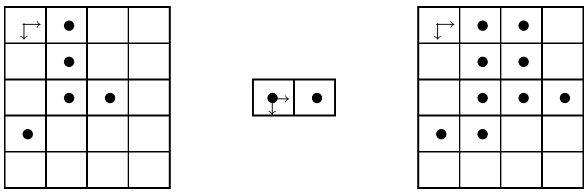


Figure 5: *Left: binary image X. Middle: structuring element A. Right: dilation of X by A.*

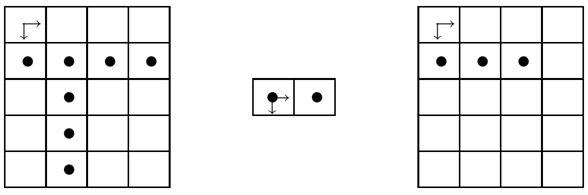


Figure 6: *Left: binary image X. Middle: structuring element A. Right: erosion of X by A.*

a *duality relation* with respect to set-complementation (remember that X^c denotes the complement of the set X):

$$X \oplus A = (X^c \ominus \check{A})^c,$$

i.e. dilating an image by A gives the same result as eroding the background by $\check{A}$.

4.2.1 Minkowski algebra

We mention some standard algebraic properties of Minkowski addition and subtraction. Some of the proofs are included, using the elementary set laws summarized in Appendix B. Here E is the Euclidean space, X, Y, Z, A, B arbitrary subsets of E, X^c the complement of X.

1. *Commutativity*

$$\begin{aligned} X \oplus A &= A \oplus X \\ X^c \ominus \check{A} &= A^c \ominus \check{X} \qquad (4.7) \end{aligned}$$

   We prove the first of these.

   Proof:

$$
\begin{aligned}
& X \oplus A \\
= \{ & \text{ definition of the Minkowski addition } \} \\
& \{a + b : a \in X \subseteq E, b \in A \subseteq E\} \\
= \{ & \text{ vector addition a+b is commutative } \} \\
& \{b + a : a \in X \subseteq E, b \in A \subseteq E\} \\
= \{ & \text{ definition } \} \\
& A \oplus X
\end{aligned}
$$

∎

2. *Iteration*

$$
\begin{aligned}
(X \oplus A) \oplus B = X \oplus (A \oplus B) \\
(X \ominus A) \ominus B = X \ominus (A \oplus B)
\end{aligned}
\tag{4.8}
$$

(The first property is usually called associativity.) We prove the second of these.

Proof:

$$
\begin{aligned}
& (X \ominus B) \ominus C \\
= \{ & \text{ definition erosion } \} \\
& \bigcap_{c \in C} \left(\bigcap_{b \in B} X_{-b} \right)_{-c} \\
= \{ & \text{ definition intersection } \} \\
& \bigcap_{c \in C} \bigcap_{b \in B} X_{-b-c} \\
= \{ & \text{ arithmetic } \} \\
& \bigcap_{c \in C} \bigcap_{b \in B} X_{-(b+c)} \\
= \{ & \text{ definition intersection and } \oplus \ \} \\
& \bigcap_{b+c \in B \oplus C} X_{-(b+c)} \\
= \{ & \text{ definition } \ominus \ \} \\
& X \ominus (B \oplus C)
\end{aligned}
$$

∎

3. *Translation invariance*

$$
\begin{aligned}
(X \oplus A)_h = X_h \oplus A. \\
(X \ominus A)_h = X_h \ominus A.
\end{aligned}
\tag{4.9}
$$

We prove the first of these.

Proof:

$$
\begin{aligned}
& (X \oplus A)_h \\
= \{ & \text{ definition } \oplus \ \} \\
& (X \oplus A) \oplus \{h\} \\
= \{ & \text{ associativity } \} \\
& X \oplus (A \oplus \{h\}) \\
= \{ & \text{ commutativity } \} \\
& X \oplus (\{h\} \oplus A) \\
= \{ & \text{ associativity } \} \\
& (X \oplus \{h\}) \oplus A \\
= \{ & \text{ definition } \oplus \ \} \\
& X_h \oplus A
\end{aligned}
$$

∎

4. *Distributivity*

$$\begin{aligned}(X \cup Y) \oplus A &= (X \oplus A) \cup (Y \oplus A)\\ (X \cap Y) \ominus A &= (X \ominus A) \cap (Y \ominus A)\\ X \oplus (A \cup B) &= (X \oplus A) \cup (X \oplus B)\\ X \ominus (A \cup B) &= (X \ominus A) \cap (X \ominus B)\end{aligned} \tag{4.10}$$

We prove the second of these.

Proof:

$$\begin{aligned}& (X \ominus A) \cap (Y \ominus A)\\ = \{ & \text{ definition } \ominus \ \}\\ & \left(\bigcap_{a \in A} X_{-a}\right) \cap \left(\bigcap_{a \in A} Y_{-a}\right)\\ = \{ & \text{ set theory } \}\\ & \bigcap_{a \in A} (X_{-a} \cap Y_{-a})\\ = \{ & \text{ translation invariance } \}\\ & \bigcap_{a \in A} (X \cap Y)_{-a}\\ = \{ & \text{ definition } \ominus \ \}\\ & (X \cap Y) \ominus A\end{aligned}$$

∎

A consequence of the distributivity property is that dilation and erosion are *increasing* transformations, i.e. transformations such that for all X, Y, $X \subseteq Y$ implies that $\psi(X) \subseteq \psi(Y)$.

4.3 Genus

One of the ways to characterize binary images is by the number of connected components and holes. A *hole* is a connected component of 0-pixels which is not connected to the border frame of the image.

Definition 2 *The* genus $g(I)$ *of a binary image* I *is the number of connected components of* I *minus the number of holes of* I.

To compute the genus erosions can be used. If we take 4-connected 1-pixels and 8-connected 0-pixels, the following formula holds ($\# I$ denotes 'number of 1-pixels in I'):

$$g_4(I) = \# I - \# I \ominus V - \# I \ominus H + \# I \ominus B \tag{4.11}$$

In the case of 8-connected 1-pixels and 4-connected 0-pixels, we have:

$$\begin{aligned} g_8(I) \quad = \quad & \# I - \# I \ominus V - \# I \ominus H - \# I \ominus D_1 - \# I \ominus D_2\\ & + \# I \ominus C_1 + \# I \ominus C_2 + \# I \ominus C_3 + \# I \ominus C_4 - \# I \ominus B \end{aligned} \tag{4.12}$$

The structuring elements V, H, B etc. occurring in these formulas are given in Fig. 7.

4.4 Region counting

Levialdi has developed an algorithm to count regions (i.e. connected components of 1-pixels) which makes use of erosions. First consider the case of 8-connected regions. Let I be the original image. Define the iteration

$$I_0 \quad = \quad I$$

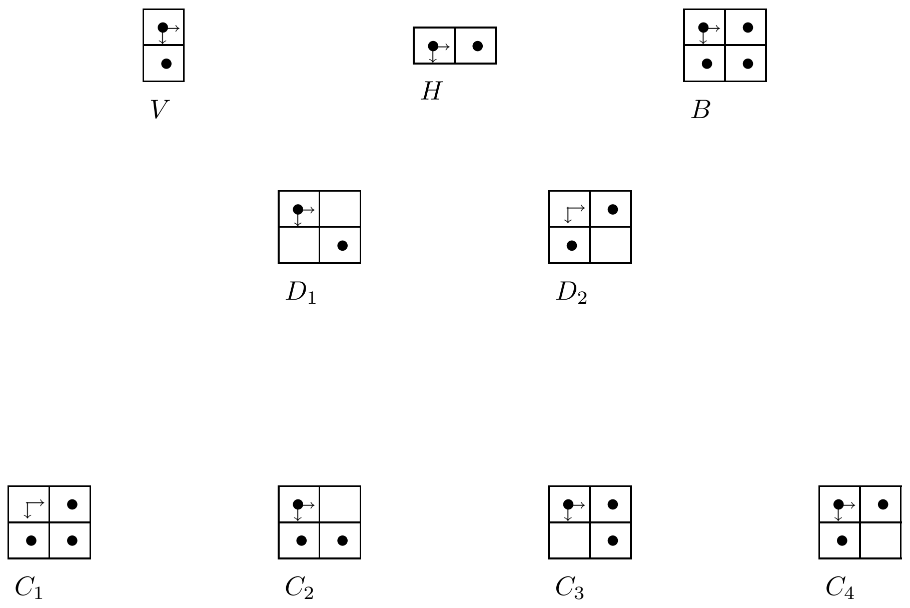


Figure 7: *Structuring elements to be used for computing the genus via erosions.*

$$I_{n+1} = \bigcup_{i=1}^{4} I_n \ominus K_i \tag{4.13}$$

with the structuring elements given in Fig. 8. This erodes I towards the top right without altering the connectivity. Now count the isolated pixels by

$$\begin{aligned} c_0 &= \# I_0 \circledast (L_1, L_2) & (4.14) \\ c_{n+1} &= c_n + \# I_{n+1} \circledast (L_1, L_2) & (4.15) \end{aligned}$$

and stop the iteration when $I_N = \emptyset$. Then c_N is the desired number of 8-connected regions.

In the case of 4-connected regions, the algorithm is the same but now the summation in (4.13) runs only from 1 to 3. The structuring elements for this case are shown in Fig. 9.

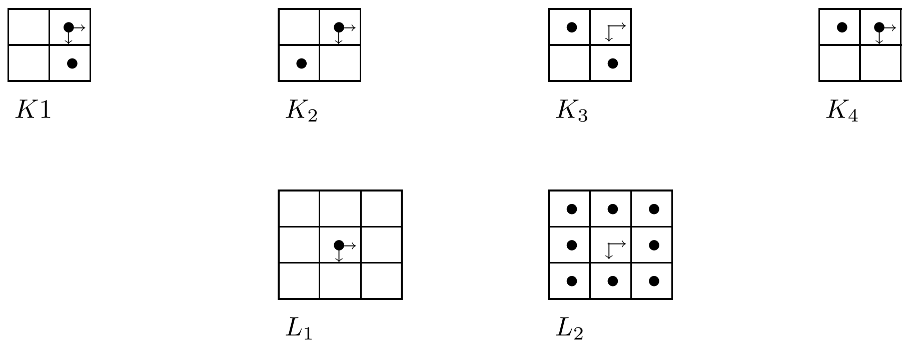


Figure 8: *Structuring elements to be used in Levialdi's algorithm for computing the number of 8-connected regions.*

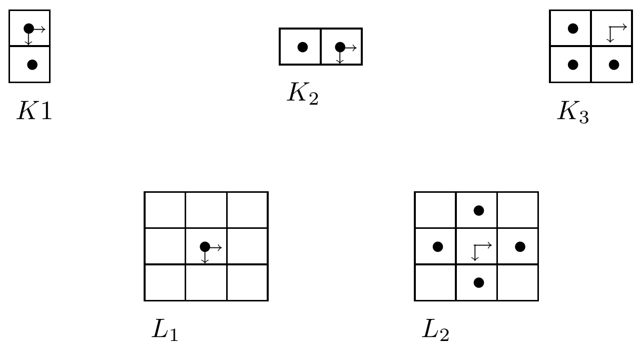


Figure 9: *Structuring elements to be used in Levialdi's algorithm for computing the number of 4-connected regions.*

4.5 Openings and Closings

Other important increasing transformations are the opening and closing by a structuring element A, which are obtained by an erosion followed by a dilation, and conversely:

$$Opening \quad : \qquad X \circ A := (X \ominus A) \oplus A = \bigcup \{A_h : h \in E, A_h \subseteq X\} \tag{4.16}$$

$$Closing \quad : \qquad X \bullet A := (X \oplus A) \ominus A = \bigcap \{(\overset{\vee}{A}{}^{c})_h : h \in E, (\overset{\vee}{A}{}^{c})_h \supseteq X\} \tag{4.17}$$

In words, the opening is the union of all the translates A_h of the structuring element A which are included in the set X. An opening smoothes contours, cuts narrow bridges, removes small islands and sharp corners; a closing fills narrow channels and small holes. For the discrete case, an example is given in Fig. 10. Again,

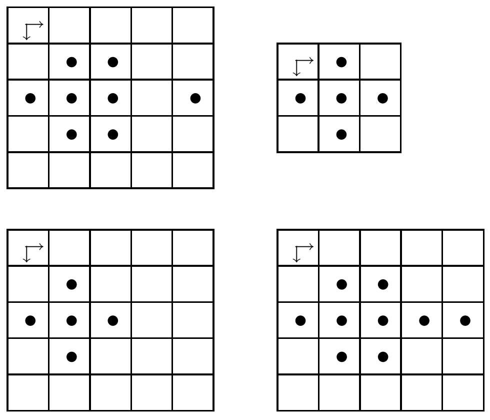


Figure 10: *Upper left: binary image X. Upper right: structuring element A. Lower left: opening of X by A. Lower right: closing of X by A.*

if we fix the set A, we can define the opening γ_B and closing ϕ_A by

$$\gamma_A(X) := X \circ A, \qquad \phi_A(X) := X \bullet A. \tag{4.18}$$

Note that $\gamma_B := \delta_B \varepsilon_B$ and $\phi_B := \varepsilon_B \delta_B$.

Just as in the case of dilation and erosion, opening and closing are related by *duality*:

$$(X^c \circ A)^c = X \bullet \overset{\vee}{A}.$$

So, the closing of X by $\overset{\vee}{A}$ equals the opening of the complement of X by A. As image transformations, opening and closing are:

1. *(Anti)extensive*
   - $X \circ A \subseteq X$: opening is *anti-extensive*, i.e. by opening an image the result can only become smaller.
   - $X \bullet A \supseteq X$: closing is *extensive*.
2. *Idempotent*: $(X \circ A) \circ A = X \circ A$, $(X \bullet A) \bullet A = X \bullet A$. Repeating an opening or closing does not change the result, so an opening or closing has to be carried out only once.
3. *Increasing*: $X \circ A \subseteq X' \circ A$ when $X \subseteq X'$, and similarly for the closing.

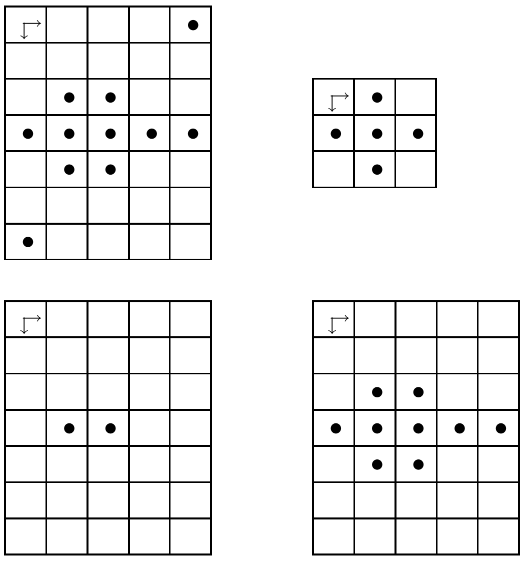


Figure 11: *Upper left: binary image X. Upper right: structuring element A. Lower left: erosion of X by A. Lower right: conditional dilation $Y \oplus A; X$ where $Y = X \ominus A$.*

4.6 Conditional dilations and erosions

A practically useful transform is the *conditional* dilation of a set with respect to a *mask* X.

Definition 3 *Let A be a structuring element containing the origin. The* conditional dilation *of Y by the structuring element A with mask X, denoted by $Y \oplus A; X$, is defined by*

$$Y \oplus A; X = \bigcup_{m=0}^{\infty} J_m, \tag{4.19}$$

where

$$J_0 = Y \tag{4.20}$$
$$J_n = (J_{n-1} \oplus A) \cap X \tag{4.21}$$

In the discrete case we have

$$Y \oplus A; X = J_m \tag{4.22}$$

with m the smallest integer such that $J_m = J_{m-1}$. This transformation can be used to eliminate small noise pixels from an image X as follows. First erode X by A, where A is large enough to eliminate the noise pixels. Then do a conditional dilation by A of the result with mask X to restore small bridges (see Fig. 11).

4.7 EXERCISES

4.7.1 Region counting & Genus

The plugins `Count` and `Genus` from the `ComputerVision` menu under the `Plugins` menu compute the number of 4-connected or 8-connected regions and the 4-connected or 8-connected genus, respectively.

Exercise 1.

Use these plugins with the images `test1` to `test3` for both types of connectivity. Is the result correct?

4.7.2 Classification

Exercise 2.

Record a plugin `Classify_.java` which reads the image `prak1` (see Fig. 2b) with nuts and bolts, and produces an output image where the nuts have the grey value 64, the bolts the grey value 128 and the background pixels the value 0.

4.7.3 Region filling

Suppose we want to fill a region within a curve $\mathcal{C}$ of 1-pixels (give all pixels within the curve the value 1). Let $p = (x, y)$ be a given point within this curve.

Exercise 3.

Give an iterative procedure for this purpose, using only dilations, complementation and intersections. Implement this algorithm as an ImageJ plugin (use `Region_.java` as the filename).

Hint: In the `Plugins/ComputerVision` menu of ImageJ you find a function `SetPixel` which can be used to set a single pixel in a binary image. Also, after you generated the java file, you should probably add a loop to iterate your function calls.

Use the plugin with the binary image `kromme` (see Fig. 2c). How many iterations are needed by the algorithm?

5 Morphological image processing II

In the second session on morphology we will study some of the more advanced morphological transforms.

5.1 Morphology for grey value images

A grey value image is a function $f : E \to G$, where E is $\mathbb{R}^n$ or $\mathbb{Z}^n$ and G — the set of grey values — is (a subset of) $\mathbb{R}$ or $\mathbb{Z}$. Let $F, K \subseteq E$, $f : F \to G$ and $k : K \to G$ two grey value images. Then the dilation of f by k, denoted as $f \,\overline{\oplus}\, k$, is defined as

$$
\begin{aligned}
(f \,\overline{\oplus}\, k)(x) &= \mathsf{Max}_{y \in K, x-y \in F}\,[f(x-y) + k(y)], &(5.1)\\
(f \,\overline{\ominus}\, k)(x) &= \mathsf{Min}_{y \in K, x+y \in F}\,[f(x+y) - k(y)]. &(5.2)
\end{aligned}
$$

That is, $f \,\overline{\oplus}\, k$ equals the maximum of $f(x-y) + k(y)$ when y runs over the set K, and similarly for $f \,\overline{\ominus}\, k$. For this reason these operations are called **minimum** and **maximum** filters.

5.2 Grey value opening and closing

By taking products of the grey value dilation and erosion we can construct grey value openings and closings just as in the binary case.

Definition 4 *Let $F, K \subseteq E$, $f : F \to G$ and $k : K \to G$ two grey value functions. Then the* grey value opening *of f by k is the function $f \circ k$ defined by*

$$f \circ k = (f \,\overline{\ominus}\, k) \,\overline{\oplus}\, k. \qquad (5.3)$$

Similarly, the grey value closing *of f by k is the function $f \bullet k$ defined by*

$$f \bullet k = (f \,\overline{\oplus}\, k) \,\overline{\ominus}\, k. \qquad (5.4)$$

There is a *duality relation* between grey value opening and closing. Introduce the *reflection* of the function k by

$$\overset{\vee}{k}(x) = k(-x).$$

Then duality is expressed by

$$-(f \circ k)(x) = ((-f) \bullet \overset{\vee}{k})(x).$$

If $k : K \to E$ is *constant* (such functions k are called *flat* structuring functions), one often writes $f \oplus K$ instead of $f \,\overline{\oplus}\, K$, and similarly for erosion, opening and closing. Figure 12 shows the result of the grey value dilation, erosion, opening and closing on a real grey value image. If we apply the grey value opening with a *flat* structuring element to an image A and subtract the result from A, we get Meyer's *top-hat transform*. This transform extracts the peaks from a grey value image A.

5.3 Distance transform

Let $A \subseteq E$ be the 2-dimensional Euclidean space $\mathbb{R}^2$, or the discrete grid $\mathbb{Z}^2$. (The generalization to N dimensions is straightforward.) Between points $x = (x_1, x_2)$ and $y = (y_1, y_2)$ of $\mathbb{R}^2$ there is defined the usual Euclidean distance:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}. \qquad (5.5)$$

On the discrete grid we use a *digital* distance function,which depends on the type of connectivity. If 4-connectivity is used, each of the 4 neighbors is by definition at distance 1 of the center pixel; this is called the *city-block distance*. In the case of 8-connectivity, the center pixel is at distance of 1 of its 8 neighbors; this is called the *chessboard distance*. Other choices are possible, which give a closer approximation of the *Euclidean distance transform*. Paths on the grid are formed by steps in the horizontal and vertical directions (each of length 1). The length of a path is the sum of the lengths of the individual steps. The total distance

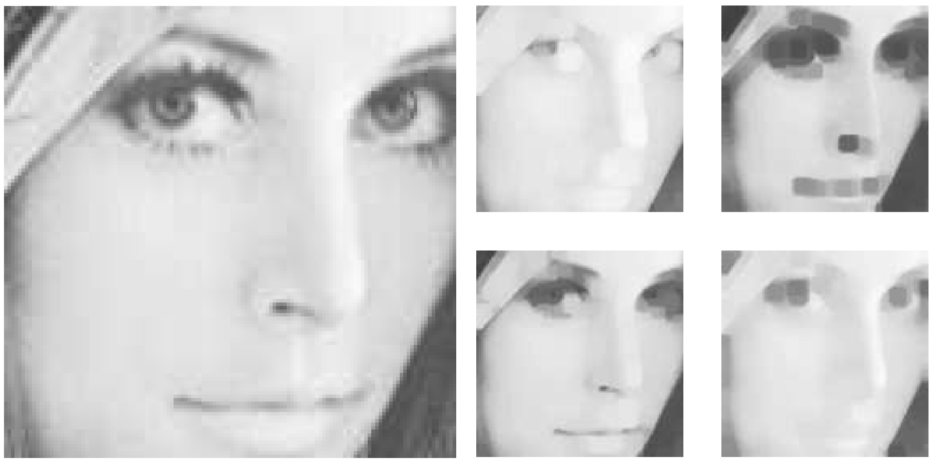


Figure 12: *Left: a grey value image. Right: upper left: grey value dilation; upper right: grey value erosion; lower left: grey value opening; lower right: grey value closing.*

between two points x and y on the grid is then the minimum of the lengths of all allowed paths from x to y on the grid.

For the two distance definitions just mentioned this leads to the following expressions for the distance $dist(x, y)$ between two points $x = (x_1, x_2)$ and $y = (y_1, y_2)$ on the grid:

$$\begin{aligned}
\text{City-block}: \quad & dist(x, y) = |x_1 - y_1| + |x_2 - y_2| \\
\text{Chessboard}: \quad & dist(x, y) = \mathsf{Max}(|x_1 - y_1|, |x_2 - y_2|) \\
\text{Euclidean}: \quad & dist(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}
\end{aligned}$$

We can also define the distance between a point x and a set A in E as the minimum of the distances between x and points of A:

$$d(x, A) = \mathsf{Min}_{a \in A} d(x, a). \tag{5.6}$$

Definition 5 *Let $A \subseteq E$. The* distance transform *of A is the grey value function $D : E \to \mathbb{R}$ which associates to point x the distance of x to the complement of A:*

$$D(x) = d(x, A^c) \tag{5.7}$$

Notice that $D(x) = 0$ for points $x \in A^c$.

5.4 Generalized distance transform

Let $K \subseteq E$ be a structuring element containing the origin: $\mathbf{0} \in K$. Define the n-fold dilation of K with itself by

$$\underset{n}{\oplus} K := \overbrace{K \oplus K \oplus \ldots \oplus K}^{n\ times} \tag{5.8}$$

with

$$\underset{0}{\oplus} K = \{0\}. \tag{5.9}$$

The n-fold erosion of A by K is defined as

$$A \underset{n}{\ominus} K := \overbrace{(\ldots((A \ominus K) \ominus K) \ominus \cdots \ominus K)}^{n\ times} = A \ominus (\underset{n}{\oplus} K). \tag{5.10}$$

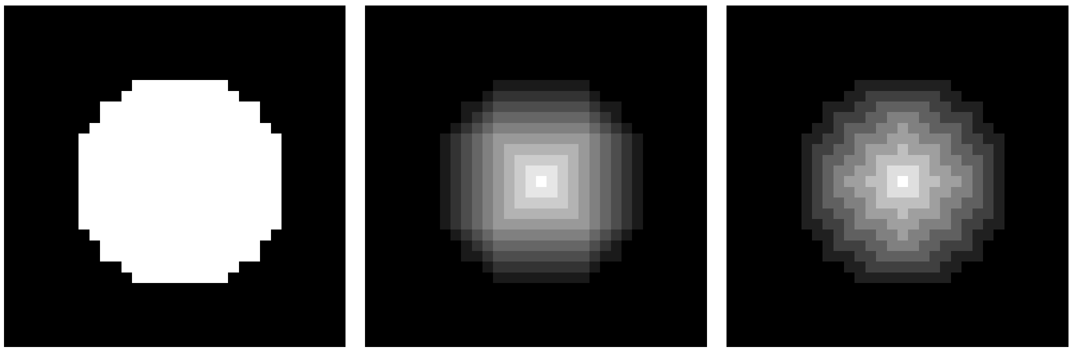


Figure 13: *Left: binary image. Middle: distance transform with cross as structuring element. Right: distance transform with square as structuring element.*

Definition 6 *The* generalized distance transform $d[A,K]$ *of* A *w.r.t.* K *is the function defined by:*

$$d[A,K](x) = \begin{cases} \mathsf{Max}\{n : x \in A \underset{n-1}{\ominus} K\}, & x \in A \\ 0 & x \notin A \end{cases} \tag{5.11}$$

If K is the cross structuring element the generalized distance transform is just the city-block distance transform discussed earlier.

5.5 Morphological skeleton

For purposes of shape description and efficient coding it is desirable to extract features from binary images which represent only the 'backbone' or skeleton of the set. Roughly spoken these are connected lines within the binary image which are in the middle of the set and 'as thin as possible'.

Let $X, B \subseteq E$. Define the n-fold dilation of X by B by

$$X \underset{n}{\oplus} B := \overbrace{(\ldots((X \oplus B) \oplus B) \oplus \cdots \oplus B)}^{n\ times} = X \oplus (\underset{n}{\oplus} B), \tag{5.12}$$

and, similarly, define the n-fold erosion of X by B by

$$X \underset{n}{\ominus} B := \overbrace{(\ldots((X \ominus B) \ominus B) \ominus \cdots \ominus B)}^{n\ times} = X \ominus (\underset{n}{\oplus} B). \tag{5.13}$$

The *morphological skeleton* $SK(X)$ of X with structuring element B is defined as

$$SK(X) = \bigcup_{n=0}^{N} S_n(X) \tag{5.14}$$

with

$$S_n(X) = X \underset{n}{\ominus} B - (X \underset{n}{\ominus} B) \circ B \tag{5.15}$$

$$X \underset{0}{\ominus} B = X \tag{5.16}$$

where N denotes the largest integer such that $S_N(X) \neq \emptyset$.

The set X can be exactly reconstructed from its skeleton sets $S_n(X)$ by:

$$X = \bigcup_{n=0}^{N} S_n(X) \underset{n}{\oplus} B \tag{5.17}$$

A compact way to encode all skeleton sets $S_n(X)$ is to introduce the *skeleton function* $skf(X)$ as follows:

$$[skf(X)](i,j) = \begin{cases} n+1, & (i,j) \in S_n(X) \\ \\ 0, & (i,j) \notin SK(X) \end{cases} \tag{5.18}$$

This is a single-valued function because the skeleton sets $S_n(X)$ are disjoint (see below). In Fig. 14 we show a binary image and its skeleton function (the lines are somewhat thickened and contrast stretched for display purposes).

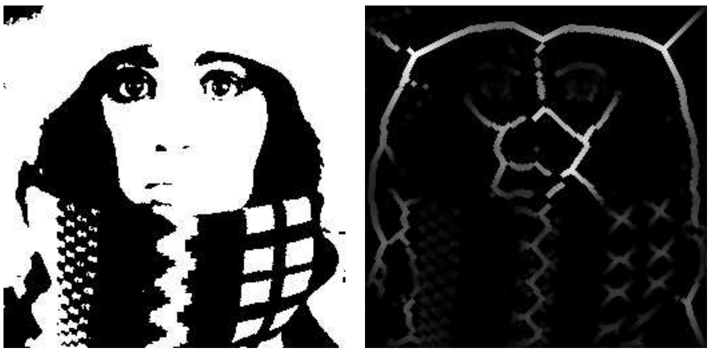


Figure 14: *A binary image (left) and its skeleton function (right).*

If the first k skeleton sets $S_n(X)$ are omitted (partial reconstruction), one gets the opening of X with structuring element $\underset{k}{\oplus} B$:

$$X \circ (\underset{k}{\oplus} B) = \bigcup_{n=k}^{N} S_n(X) \underset{n}{\oplus} B \tag{5.19}$$

This can be proved by induction on k.

Assume that $\mathbf{0} \in B$. Then the skeleton sets $S_n(X)$ are disjoint:

Proof:

$\{\mathbf{0}\} \subseteq B$

: { $\forall Y : Y \subseteq Y \oplus B$, so $\forall Z : Z \ominus B \subseteq (Z \ominus B) \oplus B = Z \circ B$ }

$X \underset{n+1}{\ominus} B = (X \underset{n}{\ominus} B) \ominus B \subseteq (X \underset{n}{\ominus} B) \circ B$

: { definition S_{n+1} }

$S_{n+1} \subseteq X \underset{n+1}{\ominus} B \subseteq (X \underset{n}{\ominus} B) \circ B$

: { $S_n \cap (X \underset{n}{\ominus} B) \circ B = \emptyset$ }

$S_{n+1} \cap S_n = \emptyset$

∎

5.6 EXERCISES

5.6.1 Contrast enhancement

Global contrast enhancement of a grey value image is given by the formula

$$f' = \frac{f - f_{min}}{f_{max} - f_{min}} f_M \tag{5.20}$$

with f the original grey value image, f_{min} and f_{max} the minimal and maximal value of f, and f_M the maximal grey value (say 255).
Local contrast enhancement is described by the formula

$$f' = \frac{f - f \ominus S}{f \oplus S - f \ominus S} f_M \tag{5.21}$$

where $f \oplus S$ and $f \ominus S$ are a grey value dilation and erosion with a flat circular (in the sense of the city-block distance) structuring element.

Exercise 1.

Write a plugin `Global_.java` which performs **global** contrast enhancement on the input image and outputs the result in another window.

Write a plugin `Local_.java` which performs **local** contrast enhancement on the input image and outputs the result in another window.

Use both plugins with the image `oog4` (see Fig. 2a). Use different sizes of the structuring element. Describe the results and explain the observed differences between the two methods.

Hint: Use the plugins `Erode` and `Dilate` under the `Process/Binary` menu to perform the erosion and dilation transforms. Both transforms use a default 8-connected structuring element of size 3x3. Transforms with larger structuring elements can be obtained by *iterating* 3x3 transforms. Use the command `Process/Binary/Set Iterations` for this purpose.

5.6.2 Morphological edge detection

Let B be a 3×3 square structuring element. The edge of a binary image X can be found by computing the difference

$$X - (X \ominus B) \tag{5.22}$$

In the case of a grey value image one can take the difference

$$f - (f \ominus B) \tag{5.23}$$

Exercise 2.

Write an ImageJ plugin which implements these formulas (use `Edge_.java` as the filename). Make sure that the plugin works both on binary and grey value images.

Use the plugin with a grey value image, and compare the results visually with those obtained with the `Roberts` and `Sobel` edge detectors. Describe your observations.

Hint: You can implement filtering operations with small kernels using the `Convolve` command under the `Process/Filters` menu. Specify the kernel coefficients in the `Convolver` window.

5.6.3 Morphological skeleton

Let $X, B \subseteq E^D$, and let $SK(X)$ be the morphological skeleton of X with structuring element B. Assume that $\mathbf{0} \in B$.

Exercise 3.

a. Prove that $SK(X) = \emptyset$ if $B = \{\mathbf{0}\}$.

b. Prove that $SK(X) = X$ if $X \ominus B = \emptyset$.

The ImageJ plugin `SkeletonDecomposition` performs the skeleton decomposition of a binary image, and outputs a grey value image containing the skeleton function $skf(X)$, for a given structuring element.

The ImageJ plugin `SkeletonReconstruction` performs the skeleton reconstruction of a grey value image containing the skeleton function $skf(X)$, and outputs a binary image containing the original image X, for a given structuring element (the same as was used in the decomposition!).

Exercise 4.

Test the skeleton decomposition and reconstruction algorithms on the binary image which you get by thresholding the grey value image `prak1` and inverting it. (For visual inspection, you may want to apply some contrast enhancement on the image containing the skeleton function.)

How can you check whether the reconstruction equals the original? Describe your observations.

5.6.4 Generalized distance transform

The ImageJ plugin `DistanceTransform` computes the generalized distance transform of a binary input image for a given structuring element and writes the result to a grey value image.

Exercise 5. Write a plugin file `GenDist_.java` which:

- reads a binary image I (choose one yourself) and displays it;
- use the plugin `DistanceTransform` with this binary image I for the structuring elements: K =`cross3x3` (a center pixel and its 4-connected neighbors) and K=`box3x3` (a center pixel and its 8-connected neighbors).
- displays the output images in two new windows.

In order to check whether the generalized distance transform has computed the correct results you can erode the input image m times with structuring element K and compare this with the appropriate 'cross-sections' of the output image O produced by the generalized distance transform. The cross-section at level t of a grey value image O is nothing but the binary image O_t produced by thresholding:

$$\begin{aligned} O_t(r,c) &:= 1 \quad if \quad O(r,c) \geq t \\ O_t(r,c) &:= 0 \quad if \quad O(r,c) < t \end{aligned}$$

So the comparison is between binary images.

Exercise 6.

1. Formulate precisely which erosion of I has to be compared with which cross-section of O.
2. For the 4-connected case: write a plugin `Compare_.java` which computes the 'difference' images at each level $t = 1, \ldots, NL$ where NL is the number of grey levels of the output image O. A difference image should have a 1 (grey level 255) where the corresponding pixels — in the images which are compared — differ, and a 0 otherwise. If the generalized distance transform is correct then all these difference images are identically zero. Is the result correct?
   Hint: You can use the ImageJ command `Analyze/Measure` to find the maximum grey value in an image; you may need this value to iterate some functions calls in your plugin. The same command can be used to find out if the difference image contains non-zero values.

6 2D → 3D inference

In this session the topic is 3D Scene Analysis. We start by explaining how images of 3D scenes are made by perspective projection. Then we look at a very simple 2D→ 3D inference problem.

6.1 Central or perspective projection

In central projection (**cp**) we have a distinguished point called the *center of projection* C and a plane, called the *image plane*, where the image will be formed. Each 3D point $P = (x, y, z)$ is projected upon the image plane by connecting P and C by a straight line. The intersection point of this line with the image plane is called the *projection* of P.

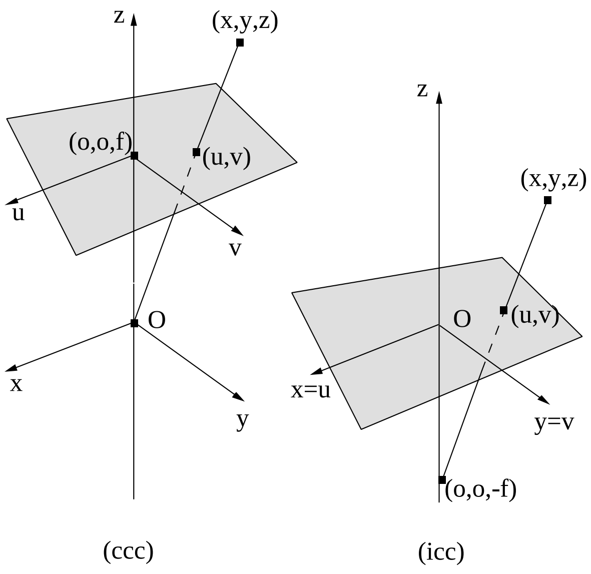


Figure 15: *Central projection using the pinhole-camera model. Left: camera-centered (***ccc***). Right: image centered (***icc***).*

This type of projection occurs in two variants:

- *camera-centered coordinate system* (**ccc**): Here we put the camera in the origin of the coordinate system with the image plane parallel to the x-y plane at a distance f (this is the *camera constant*, usually identified with the *focal length* as explained above), see Fig. 15. So the projection center $C = (0, 0, 0)$. The z-axis is called the *optical axis*. If we use the variables u and v as coordinates in the image plane, then the projection can be described by the equation:

$$\boxed{u = fx/z, \quad v = fy/z.} \tag{6.1}$$

  It is sensible to use these equations only for points *above* the image plane, that is, for $z > f$.

- *image-centered coordinate system* (**icc**): Here the projection center $C = (0, 0, -f)$, so we put the camera at a distance f below the image plane, which is the x-y plane in this case, see Fig. 15. This projection can be described by the equation:

$$\boxed{u = fx/(f + z), \quad v = fy/(f + z).} \tag{6.2}$$

  Again, it is sensible to use these equations only for $z > 0$.

6.1.1 Parallel projection

As an approximation to central projection *parallel projection* (**pp**) is obtained: take the limit $f \to \infty$ in the **icc** case to find

$$u = x \quad v = y.$$

These equations say that the projection takes place along straight lines perpendicular to the projection plane. This approximation is valid if the objects are very small compared to f or if they are very far removed from the camera.

The basic question of 3D vision now is to what extent 3D properties can be reconstructed from the projective images by inverting the projection process, that is, by *backprojection*.

6.1.2 Lines to lines

Under perspective projection a 3D line is mapped onto a 2D line. This can be proved as follows. Assume a **ccc** projection, see (6.1). Let $\vec{p}, \vec{w} \in \mathbb{R}^3$ and let L be the 3D line with representation:

$$L = \left\{ \vec{p} + \lambda \vec{w} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} + \lambda \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} : \lambda \in \mathbb{R} \right\}. \tag{6.3}$$

The vector $\vec{w}$ is called the 'direction vector' and will be assumed to be normalized: $\|\vec{w}\| = 1$. Then the coordinates (u, v) of the projection of this line have the form:

$$\begin{aligned} u &= f(p_1 + \lambda w_1)/(p_3 + \lambda w_3) \\ v &= f(p_2 + \lambda w_2)/(p_3 + \lambda w_3) \end{aligned} \tag{6.4}$$

Now a little algebra shows that

$$\begin{pmatrix} u \\ v \end{pmatrix} = f \begin{pmatrix} p_1/p_3 \\ p_2/p_3 \end{pmatrix} + \mu \begin{pmatrix} w_1 - (p_1/p_3)w_3 \\ w_2 - (p_2/p_3)w_3 \end{pmatrix}, \qquad \mu = \frac{f\lambda}{p_3 + \lambda w_3},$$

which is the parameter representation of a 2D line (when λ runs over $\mathbb{R}$ so does μ).

6.1.3 Relation between line parameters

There is a relation between the parameters of a 3D line and those of its perspective projection. Represent the 3D line L as

$$L = \left\{ \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} + \lambda \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} : \lambda \in \mathbb{R} \right\}$$

and the perspective projection of this line (again a line) as

$$L' = \left\{ \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + \eta \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} : \eta \in \mathbb{R} \right\}$$

Taking a **ccc** system, we find by (6.1) that λ, η must satisfy

$$c_1 + \eta d_1 = f(p_1 + \lambda w_1)/(p_3 + \lambda w_3) \tag{6.5}$$

$$c_2 + \eta d_2 = f(p_2 + \lambda w_2)/(p_3 + \lambda w_3) \tag{6.6}$$

Eliminating η we find

$$(c_1 d_2 - c_2 d_1)p_3 - (d_2 p_1 - d_1 p_2)f + \lambda[(c_1 d_2 - c_2 d_1)w_3 - (d_2 w_1 - d_1 w_2)f] = 0.$$

Now this equation must be true for any λ. Therefore the following two equations must hold:

$$\boxed{\begin{aligned} d_2 f p_1 - d_1 f p_2 + (c_2 d_1 - c_1 d_2)p_3 &= 0 \\ d_2 f w_1 - d_1 f w_2 + (c_2 d_1 - c_1 d_2)w_3 &= 0 \end{aligned} \qquad (6.7)}$$

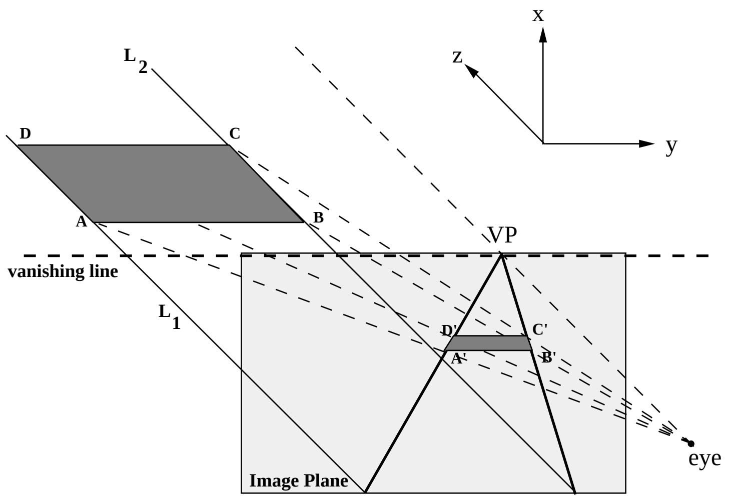


Figure 16: *Vanishing point VP of parallel 3D lines L_1 and L_2 in a plane parallel to the y-z plane. The eye is at the projection center. Dark regions: square ABCD with projection A'B'C'D'.*

6.1.4 Vanishing point

Perspective projection of parallel 3D lines having nonzero slope along the z-axis meet (in general) in a *vanishing point* on the image projection plane, see Fig. 16. This point can be found as the intersection with the image plane of a line through the projection center parallel to the given 3D lines. This can be seen as follows. Consider again the 3D line L with representation (6.3) and projection (6.4). For lines with nonzero slope along the z-axis, $w_3 \neq 0$. Points on the line which are infinitely far removed from the projection center will have coordinates u_∞, v_∞ given by

$$\boxed{\begin{aligned} u_\infty &= \lim_{\lambda\to\infty} f(p_1+\lambda w_1)/(p_3+\lambda w_3) = f(w_1/w_3) \\ v_\infty &= \lim_{\lambda\to\infty} f(p_2+\lambda w_2)/(p_3+\lambda w_3) = f(w_2/w_3) \end{aligned} \qquad (6.8)}$$

Now the point (u_∞, v_∞) is independent of p_1, p_2, p_3, so all lines with the same direction vector $\vec{w} = (w_1, w_2, w_3)$ — which are therefore *parallel* — with nonzero slope along the z-axis have perspective projections which meet in a single point, called the *vanishing point*.

Lines L with $w_3 = 0$ have perspective projections which are parallel to L. For, in this case

$$\begin{pmatrix} u \\ v \end{pmatrix} = f\begin{pmatrix} p_1/p_3 \\ p_2/p_3 \end{pmatrix} + \mu \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}, \qquad \mu = \frac{f\lambda}{p_3},$$

which is the parameter representation of a 2D line with direction vector proportional to $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$. It is therefore parallel to L which has direction vector $\begin{pmatrix} w_1 \\ w_2 \\ 0 \end{pmatrix}$.

6.1.5 Vanishing line

We now show that the vanishing points of all 3D lines which lie in parallel planes all lie on a line in the image plane called the *vanishing line*. This line can be found as the intersection of the image plane with the plane through the projection center parallel to the given 3D planes. This can be seen as follows. Consider a plane V with normal (A, B, C). Points in this plane satisfy the equation

$$Ax + By + Cz + D = 0$$

for some D. Let L be a 3D line with dv $\vec{w} = (w_1, w_2, w_3)$ ($w_3 \neq 0$) lying in a plane parallel to V. Then the dv $\vec{w}$ of L must be perpendicular to (A, B, C), in other words:

$$Aw_1 + Bw_2 + Cw_3 = 0.$$

Using (6.8) this can be written as

$$A\frac{u_\infty w_3}{f} + B\frac{v_\infty w_3}{f} + Cw_3 = 0.$$

Multiplying by f and dividing by w_3 (allowed since $w_3 \neq 0$ by assumption) yields

$$Au_\infty + Bv_\infty + Cf = 0.$$

For fixed A, B, C, f this is the equation of a line in the image plane, called the *vanishing line*.

In conclusion, the vanishing points of lines lying in parallel planes with normal vector (A, B, C) lie on the line L_∞ defined by:

$$\boxed{L_\infty = \left\{ \begin{pmatrix} u \\ v \end{pmatrix} : Au + Bv + Cf = 0. \right\}} \tag{6.9}$$

Also, every point on the line L_∞ is the vanishing point of some line in such a plane. This is illustrated in Fig. 17.

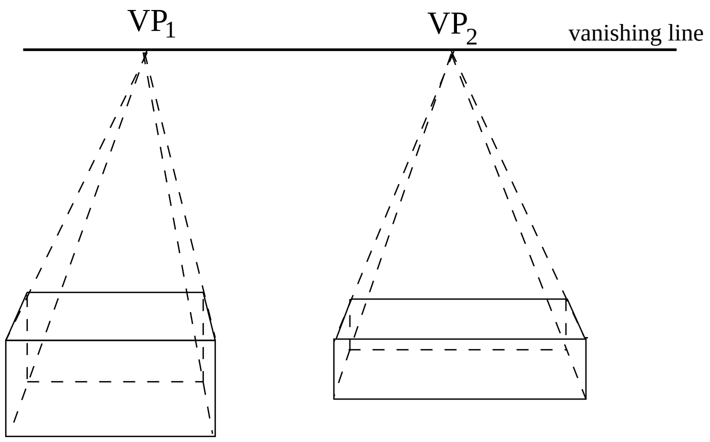


Figure 17: *Lines in parallel planes have vanishing points on the vanishing line.*

6.2 2D→ 3D inference: Direction of 3D parallel lines

Let us show how to determine the orientation of a set of two or more parallel 3D lines from a perspective image of these lines. In particular, the position of the vanishing point will allow us to recover the *direction vector* of the parallel lines. We start with the case of two parallel lines. To reduce noise influences we might want to take measurements of the projections of more than two parallel lines. This case is considered too.

6.2.1 Two parallel lines

Consider two parallel 3D lines with dv $\vec{w}$ whose projections meet in the vanishing point with coordinates u_∞, v_∞ given by (6.8). From this equation and the normalization condition $w_1^2 + w_2^2 + w_3^2 = 1$ we can solve for w_1, w_2, w_3:

$$\boxed{\begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \frac{1}{\sqrt{u_\infty^2 + v_\infty^2 + f^2}} \begin{pmatrix} u_\infty \\ v_\infty \\ f \end{pmatrix}.}$$

6.2.2 N parallel lines

Assume that the perspective projection of N parallel lines with unknown direction vector $\vec{w} = (w_1, w_2, w_3)$ is observed. Represent the projection of the nth line by

$$L_n = \left\{ \begin{pmatrix} c_n \\ d_n \end{pmatrix} + \eta \begin{pmatrix} g_n \\ h_n \end{pmatrix} : \eta \in \mathbb{R} \right\}, \quad n = 1, \ldots, N.$$

By the 2D-3D relation (6.7) we have

$$h_n f w_1 - g_n f w_2 + (d_n g_n - c_n h_n) w_3 = 0, \quad n = 1, \ldots, N.$$

In matrix form this can be written as

$$\mathbf{A}\vec{w} = \vec{0},$$

or, explicitly,

$$\boxed{\mathbf{A}\,\vec{w} = \begin{pmatrix} h_1 f & -g_1 f & d_1 g_1 - c_1 h_1 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ h_N f & -g_N f & d_N g_N - c_N h_N \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \end{pmatrix}} \tag{6.10}$$

The additional constraint is that $\vec{w}$ is a unit vector:

$$\|\vec{w}\| = 1. \tag{6.11}$$

For $N > 2$ this is an overdetermined system of equations (more equations than unknowns). To solve it we use *least squares* techniques, see Appendix C (in particular Section C.6). Since in general the equation $\mathbf{A}\vec{w} = \vec{0}$ has no nontrivial solution satisfying (6.11) we take as solution the vector $\vec{w}$ that minimizes $\|\mathbf{A}\vec{w}\|^2$ and satisfies (6.11). This vector can be computed by performing a *singular value decomposition* of the matrix $\mathbf{A}$ and choosing the right singular vector of $\mathbf{A}$ having smallest singular value.

Note that this method requires that the camera constant f is known.

If we define the vector $\vec{x}$ by

$$x_1 = w_1 f, \quad x_2 = w_2 f, \quad x_3 = w_3 \tag{6.12}$$

we can also solve the equation $\mathbf{A}'\vec{x} = 0$ with $\mathbf{A}'$ *independent of f*:

$$\mathbf{A}'\,\vec{x} = \begin{pmatrix} h_1 & -g_1 & d_1 g_1 - c_1 h_1 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ h_N & -g_N & d_N g_N - c_N h_N \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \end{pmatrix}. \tag{6.13}$$

Afterwards we then have to compute $\vec{w}$ by

$$w_1 = x_1/f, \quad w_2 = x_2/f, \quad w_3 = x_3 \tag{6.14}$$

followed by a normalization of the vector $\vec{w}$ thus obtained.

6.3 EXERCISES

The problem is to determine the orientation of a set of two or more parallel 3D lines from a perspective image of these lines. Now assume that both the direction vector $\vec{w}$ of the parallel lines, and the camera constant f is unknown, but that the available information is extended by giving the perspective projection of a rectangle (or a number of rectangles with parallel sides), see Fig. 18.

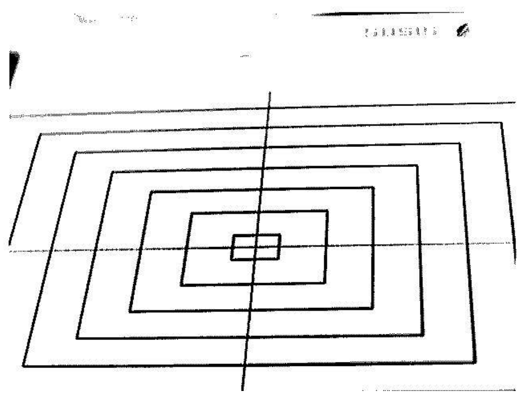


Figure 18: *Camera image of a set of rectangles.*

Exercise 1.

Show how the camera constant f can be obtained from the perspective projection of a rectangle, and from this, the direction vectors of the sides of the rectangles (and therefore the normal of the planar patch containing the rectangles).

Exercise 2.

You will need to process a picture taken by camera for you, containing a number of concentric rectangles with parallel sides as shown in Fig. 18. From the convergence of the parallel lines in the image (both for the horizontal and vertical sides of the rectangles), the vanishing points can be easily determined. The picture is in TIFF file format, with filename `rectangle.tif`.

Start ImageJ and read in the 'rectangle' image. Now you have to obtain the endpoints of two sets of parallel lines in this image, one set each for the horizontal and vertical sides of the rectangles. To this end an ImageJ plugin `ScanPoints` is provided under the `ComputerVision` submenu. This plugin will prompt you for the number of sets of parallel lines you want to scan (choose 2 in this case), and the number of lines per set you want to scan (a value of 2–4 is sufficient). Scan the sides of the rectangles by clicking the left mouse button over their endpoints. You should see the coordinates of points you clicked in the `Results` window. Save them in a file called `rectangles.dat`. Then you may quit ImageJ.

Exercise 3.

Copy the subdirectory `inference` of the lab directory to your home directory by typing:

```
cp -r /wing6/home/vakken/ComputerVision/inference $(HOME)
```

Look at the file `par_lines.c`. This routine determines the orientation of N parallel 3D lines in a perspective image. It reads in a data file `par_lines.dat` (made with the ImageJ plugin `ScanPoints`) containing the coordinates of the endpoints of the images of the parallel lines (take a look at the contents), and then computes the least squares solution of the matrix system (6.10). Study the code and try to understand what it does. You can make slight modifications and recompile by typing

```
make par_lines
```

Warning! To compute mathematical functions like a square root (sqrt), the line

```
#include <math.h>
```

must be present in your C source file.

Exercise 4.

Modify the routine `par_lines.c` to a routine `rectangles.c` which reads in the file `rectangles.dat` and writes the following information on `stdout`:

1. the camera constant f
2. the direction vectors of the sides of the rectangles
3. the normal of the planar patch containing the rectangles

Assuming you use the same functions and data structures as in `par_lines.c` (such as `matrix( )` etc.) you can compile by typing

```
make rectangles
```

7 Two-view planar motion algorithm

In this session we will try to extract 3D-information from two views of a moving planar patch.

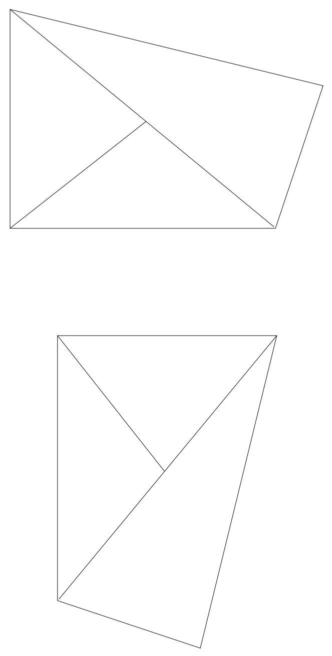


Figure 19: *Camera images of a quadrangle before and after a motion.*

The assignment is to find the motion parameters (rotation axis $\vec{n}$, rotation angle ϕ and translation vector $\vec{t}$ using the 'Two-view planar motion algorithm' from the Lecture Notes.

7.1 Camera pictures

Input are two images in TIFF file format of a quadrangle (`quadrangle` and `quadr_rot`), see Fig. 19, taken by camera for you:

1. a picture `quadrangle.tif` of the sheet of paper containing a quadrangle.
2. a picture `quadr_rot.tif` of the same quadrangle, but rotated over an angle less than $90°$.

The same position and settings of the camera have been used for these pictures as for the picture `rectangle.tif` of Section 6.

7.2 Exercises

Copy the subdirectory `optflow` of the lab directory to your home directory by typing:

```
cp -r /wing6/home/vakken/ComputerVision/optflow $(HOME)
```

Exercise 1. First we will obtain the coordinates of vertices of the quadrangles images. The correspondence between the points before and after the motion has to be made by yourself.

Start ImageJ and read in the `quadrangle` and `quadr_rot` images. Create an ImageJ stack by selecting '`Convert Images to Stack`' from the `Image/Stacks` submenu. To obtain the required vertices, the ImageJ plugin `ScanCorners` under the `Plugins\ComputerVision` menu can be used.

This plugin will prompt you for the number of point sets you want to scan (choose 2 in this case) and the number of points per set; first take 4 points per set.

Scan the vertex points of the first slice, i.e. image `quadrangle` (first data set) and then the vertex points of the second slice, image `quadr_rot`, (second data set). Save the vertex-coordinates file.

Make sure you scan the points in corresponding order!
Repeat the procedure for 5 points per set. So now you should have two output files.
Then quit ImageJ.

$\boxed{\textbf{Exercise 2.}}$ Now that the required image points are detected we can derive 3D-information using the Two-view planar motion algorithm.

This has been implemented in the C-routine `two_view.c`. Edit the file `two_view.c` to set the camera constant f to the value you have determined in the session on 2D-3D inference. Then compile the `two_view` procedure by typing

```
make two_view
```

1. Compute the motion parameters (rotation axis $\vec{n}$, rotation angle ϕ and translation vector $\vec{t}$), both for the case of 4 and 5 points per data set. Report your results.
2. How does the computed normal vector of the plane compare to the normal you have found in the session on 2D-3D inference?
3. Compare and discuss the results of the motion recovery algorithm for the cases with 4 and 5 vertices, respectively.

A Appendix: Linear shift-invariant systems

A.1 Definition of LSI systems

Two basic assumptions are often made about the systems which transforms signals to signals (or, for that matter, images to images). The first one is additivity: operating on the sum of two input signals gives the same result as operating on the two input signals separately and adding the results. A second basic assumption concerns time shifts: it makes no difference whether the system operates on a time-shifted signal, or on the original signal followed by a time shift. (In the case of 2D images there are two shift parameters, one for the horizontal and one for the vertical direction.) This leads to the following precise definition. A linear shift-invariant (LSI) system $\mathcal{O}$ maps an input signal $f(t)$ to an output signal $g(t)$, denoted as $f(t) \xrightarrow{\mathcal{O}} g(t)$, such that the following two conditions hold:

- *Linearity*: if $f_1(t) \xrightarrow{\mathcal{O}} g_1(t)$ and $f_2(t) \xrightarrow{\mathcal{O}} g_2(t)$, then, for arbitrary constants a, b,

$$a\, f_1(t) + b\, f_2(t) \xrightarrow{\mathcal{O}} a\, g_1(t) + b\, g_2(t)$$

- *Shift invariance*: if $f(t) \xrightarrow{\mathcal{O}} g(t)$, then for each time shift T,

$$f(t-T) \xrightarrow{\mathcal{O}} g(t-T)$$

A.2 Convolution and impulse response function

The output of a LSI system is given by the *convolution* of the input with a kernel $k(t)$ which is characteristic for that system:

$$g(t) = (k * f)(t) = \int_{-\infty}^{\infty} k(t-s)\, f(s)\, \mathrm{d}s \tag{A.1}$$

If the input is an impulse, i.e., a delta function at time 0, then the output is precisely $k(t)$. Therefore the filter kernel $k(t)$ is usually called the *impulse response function*.

A.3 The 1D Fourier transform

LSI systems can be conveniently described by using Fourier transforms. The Fourier transform of a function $f(t)$, denoted by $\widehat{f}(u)$, is defined by

$$\widehat{f}(u) = \int_{-\infty}^{\infty} e^{-i2\pi ut} f(t)\, \mathrm{d}t \tag{A.2}$$

For discrete and finite data the Fourier transform (A.2) is replaced by the forward discrete Fourier transform (DFT), which transforms a signal represented by the vector $(f[0], f[1], \ldots, f[N-1])$ into a transformed vector $(F[0], F[1], \ldots, F[N-1])$

$$F[u] = c_N \sum_{x=0}^{N-1} f[x]\, e^{-i2\pi ux/N}, \qquad u = 0, 1, \ldots, N-1$$

The original signal can be recovered from its discrete Fourier transform by the inverse discrete Fourier transform (IDFT):

$$f[x] = \frac{1}{c_N \cdot N} \sum_{u=0}^{N-1} F[u]\, e^{i2\pi ux/N}, \qquad x = 0, 1, \ldots, N-1$$

For the constant c_N different conventions are in use. Common choices are:

$$c_N = \begin{cases} 1 \\ \frac{1}{N} \\ \frac{1}{\sqrt{N}} \end{cases} \tag{A.3}$$

The last choice has the advantage that the same expression for the forward and inverse transform can be used (only the sign in the exponential changes). The DFT can be implemented efficiently by the Fast Fourier Transform (FFT) algorithm.

Introducing the Fourier transforms $\widehat{f}, \widehat{g}$ and $\widehat{k}$ of f, g and k, we can write the convolution formula (A.1) in the form:

$$\widehat{g}(u) = \widehat{k}(u)\,\widehat{f}(u). \tag{A.4}$$

So in the Fourier domain, convolution becomes multiplication, which is the basis of computer implementation using the FFT: first compute the Fourier transform of k and f, then perform the multiplication of the Fourier transforms $\widehat{k}$ and $\widehat{f}$, and finally compute the inverse DFT of $\widehat{g}$ to obtain g. The function $\widehat{k}(u)$, which is the Fourier transform of the impulse response function, is called the *transfer function* of the linear system.
Warning: in the discrete case, (A.4) may contain an extra factor, depending on the choice of c_N.

A.4 2D systems

In the 2D case, the convolution takes the form

$$g(x,y) = (k * f)(x,y) = \iint k(x-x', y-y')\, f(x',y')\, \mathrm{d}x'\, \mathrm{d}y' \tag{A.5}$$

Now $k(x,y)$ is often referred to as the *point spread function*. Introducing the 2D Fourier transforms $\widehat{f}, \widehat{g}$ and $\widehat{k}$, we can again write the convolution formula (A.5) in the form of a product:

$$\widehat{g}(u,v) = \widehat{k}(u,v)\,\widehat{f}(u,v).$$

In the case of digital images, the convolution integral (A.5) becomes a summation:

$$g[x,y] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} k[x-m, y-n]\, f[m,n], \tag{A.6}$$

for $x = 0,1,\ldots,M-1, y = 0,1,\ldots,N-1$. The index calculation in $k[x-m, y-n]$ must be done modulo M, N, respectively. Examples of discrete convolution occur in spatial filtering.

A.5 The 2D Fourier transform

For the case of images we use a two-dimensional DFT:

$$F[u,v] = c_N\, c_M \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f[x,y]\, e^{-i2\pi(ux/M+vy/N)},$$

with $u = 0,1,\ldots,M-1; v = 0,1,\ldots,N-1$, and the constants c_N, c_M are defined by (A.3). The inverse 2D DFT is:

$$f[x,y] = \frac{1}{c_N \cdot N \cdot c_M \cdot M} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} F[u,v]\, e^{i2\pi(ux/M+vy/N)},$$

with $x = 0,1,\ldots,M-1; y = 0,1,\ldots,N-1$.

B Appendix: Set laws

Notation:

union	$\cup$
intersection	$\cap$
set inclusion	$\subseteq$
element of	$\in$
universal set	E
empty set	$\emptyset$
set difference	$X \backslash Y = \{x \in X : x \notin Y\}$
complement	$X^c = E \backslash X$

1. Relations involving the universal set E and the empty set .

$$\begin{aligned} E \cup X &= E \quad E \cap X = X \quad X \cup \emptyset = X \quad X \cap \emptyset = \emptyset \\ X \cup X &= X \quad X \cap X = X \end{aligned}$$

2. Commutativity

$$X \cap Y = Y \cap X \quad X \cup Y = Y \cup X$$

3. Distributivity

$$\begin{aligned} X \cap (Y \cup Z) &= (X \cap Y) \cup (X \cap Z) \\ X \cup (Y \cap Z) &= (X \cup Y) \cap (X \cup Z) \end{aligned}$$

4. Associativity

$$\begin{aligned} X \cup (Y \cup Z) &= (X \cup Y) \cup Z \\ X \cap (Y \cap Z) &= (X \cap Y) \cap Z \end{aligned}$$

5. De Morgan's Laws

$$\begin{aligned} (X \cup Y)^c &= X^c \cap Y^c \\ (X \cap Y)^c &= X^c \cup Y^c \end{aligned}$$

6. Minimax Theorem

$$\bigcap_i \left(\bigcup_j X_{ij} \right) \supseteq \bigcup_j \left(\bigcap_i X_{ij} \right)$$

C Appendix: Least Squares and the Singular Value Decomposition

A large body of techniques exists to handle sets of linear equations that are either singular or numerically close to being singular. One of the most robust is the Singular Value Decomposition (SVD) which diagnoses what the problem is and sometimes also solves it. In particular SVD is the method of choice to solve *linear least squares* problems.

We first introduce the problem of simultaneous linear equations, then discuss the SVD and finally look at its application to least squares solution of linear equations. For a fuller discussion see [1–3].

C.1 Simultaneous linear equations

A set of simultaneous linear equations can be written as the matrix equation:

$$\boxed{\mathbf{A}\vec{x} = \vec{b}} \tag{C.1}$$

where $\mathbf{A}$ is a square $M \times N$ matrix, $\vec{x}$ is an N-dimensional vector and $\vec{b}$ is an M-dimensional vector.

If we denote the jth column vector of the matrix $\mathbf{A}$ by $\vec{a}_j$, then (C.1) can be rewritten as:

$$\sum_{j=1}^{N} \vec{a}_j x_j = \vec{b}. \tag{C.2}$$

This equation says that linear combinations of the column vectors $\vec{a}_j$ with multipliers $\{x_j\}$ should equal the vector $\vec{b}$.

If you have fewer equations than unknowns ($M < N$) you do not expect a unique solution. If there are more equations than unknowns ($M > N$) there will in general be no solution at all. Then we want to find the so-called ‘minimum norm least squares solution’.

The most familiar case is when $M = N$, so let us first discuss this as a reminder and then come back to the general case.

C.1.1 Simultaneous linear equations with $M = N$

The following distinction is essential in studying the solution of (C.1) when $\mathbf{A}$ is a square matrix.

- The matrix $\mathbf{A}$ is *nonsingular*. This is the case when the determinant of $\mathbf{A}$ is not zero: $\det(\mathbf{A}) \neq 0$. In this case $\mathbf{A}$ is invertible, and the solution of (C.1) is given by
$$\vec{b} = \mathbf{A}^{-1}\vec{x},$$
where $\mathbf{A}^{-1}$ is the *inverse* of $\mathbf{A}$.
- The matrix $\mathbf{A}$ is *singular*. This is the case when $\det(\mathbf{A}) = 0$. In this case $\mathbf{A}$ is not invertible, and the solution of (C.1) either does not exist or else there are infinitely many solutions.

Example 7 Let

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{C.3}$$

When $\det(\mathbf{A}) = ad - bc \neq 0$, this matrix is nonsingular with inverse

$$\mathbf{A}^{-1} = \frac{1}{ad - bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix},$$

as can be easily verified by checking that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. Techniques to compute the inverse for nonsingular matrices of arbitrary dimension are discussed below, see Remark 19.

For example, take $a = 2, b = c = d = 1$. Then for any vector $\vec{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$ the solution of (C.1) is given by

$$\vec{x} = \mathbf{A}^{-1}\vec{b} = \begin{pmatrix} b_1 - b_2 \\ -b_1 + 2b_2 \end{pmatrix}.$$

$\diamondsuit$

C.2 Nullspace and range

For questions of existence and uniqueness of solutions of (C.1), the concepts of *nullspace* and *range* are important. Eq. C.1 defines $\mathbf{A}$ as a mapping from $\mathbb{R}^N$ to $\mathbb{R}^M$: N-dimensional vectors $\vec{x}$ are mapped to M-dimensional vectors $\mathbf{A}\vec{x}$.

It may happen that a vector $\vec{x}$ is mapped to the zero vector $\vec{0}$ in $\mathbb{R}^M$. The set of vectors $\vec{x}$ such that $\mathbf{A}\vec{x} = \vec{0}$ is called the *nullspace* of $\mathbf{A}$ and denoted by $N(\mathbf{A})$. The dimension of the nullspace (number of linearly independent vectors that can be found in it) is called the *nullity* of $\mathbf{A}$.

There is also some subspace of $\mathbb{R}^N$ that can be 'reached' by $\mathbf{A}$, in the sense that there exists some $\vec{x}$ which is mapped to this subspace by $\mathbf{A}$. This subspace is called the *range* of $\mathbf{A}$ and denoted by $R(\mathbf{A})$. From (C.2) we see that the range of $\mathbf{A}$ is the set of linear combinations of the column vectors $\vec{a}_j$ (also called the *span* of $\{\vec{a}_j\}$).

Define the *rank* of $\mathbf{A}$ as the number of independent columns (which equals the number of independent rows). Then it is clear that the dimension of the range equals the rank of $\mathbf{A}$. If $\mathbf{A}$ is of *full column rank* (all columns $\vec{a}_j$ of $\mathbf{A}$ are independent) then the nullspace reduces to the zero vector $\vec{0}$. (Of course this can only happen if $M \geq N$).

A square ($N \times N$) matrix $\mathbf{A}$ of full column rank is nonsingular and its range is all of $\mathbb{R}^N$. It can be shown that for such a matrix

$$\boxed{\text{rank plus nullity } = N.} \tag{C.4}$$

Example 8 Let

$$\mathbf{A} = \begin{pmatrix} 2 & 1 \\ 2 & 1 \end{pmatrix}$$

This matrix has rank 1 (the columns are dependent). Since $\mathbf{A}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ has the form $(2x_1 + x_2)\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, the range is spanned by the vector $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and its dimension is 1. The equation $\mathbf{A}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \vec{0}$ is true when $2x_1 + x_2 = 0$, so the nullspace consists of vectors $\begin{pmatrix} x_1 \\ -2x_1 \end{pmatrix} = x_1\begin{pmatrix} 1 \\ -2 \end{pmatrix}$. The nullspace is spanned by the vector $\begin{pmatrix} 1 \\ -2 \end{pmatrix}$, and its dimension (the nullity) is also 1. Notice that rank plus nullity $= 2$, in agreement with (C.4). $\diamondsuit$

C.3 The SVD

The SVD explicitly constructs orthonormal bases for the nullspace and range of a matrix. It is based on the following theorem of linear algebra.

Theorem 9 *Any $M \times N$ matrix $\mathbf{A}$ can be written as a product:*

$$\begin{pmatrix} \\ \quad \mathbf{A} \quad \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \quad \mathbf{U} \quad \\ \\ \end{pmatrix} \cdot \begin{pmatrix} \\ \quad \mathbf{D} \quad \\ \\ \end{pmatrix} \cdot \begin{pmatrix} \quad \mathbf{V}^T \quad \\ \\ \end{pmatrix} \tag{C.5}$$

where $\mathbf{U}$ is an $M \times M$ orthogonal matrix, $\mathbf{V}$ is an $N \times N$ orthogonal matrix, and $\mathbf{D}$ is an $M \times N$ diagonal matrix of the form

$$\left(\begin{array}{cccc|c} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \cdots & & \\ & & & \sigma_r & \\ --- & --- & --- & --- & --- \\ & & & & \end{array}\right). \tag{C.6}$$

The entries σ_j *of* $\mathbf{D}$*, called the* singular values*, are all* positive*. All other elements of* $\mathbf{D}$ *are zero. Here* r *is the* rank *of* $\mathbf{A}$.

The orthonormality of the columns of $\mathbf{U}$ and $\mathbf{V}$ is expressed by the equations

$$\sum_{i=1}^{M} U_{ik}\, U_{in} = \delta_{kn}, \quad 1 \le k \le M,\ 1 \le n \le M, \tag{C.7}$$

$$\sum_{j=1}^{N} V_{jk}\, V_{jn} = \delta_{kn}, \quad 1 \le k \le N,\ 1 \le n \le N, \tag{C.8}$$

which can be compactly written in matrix form as

$$\mathbf{U}^T \mathbf{U} = \mathbf{I}_M, \tag{C.9}$$

$$\mathbf{V}^T \mathbf{V} = \mathbf{I}_N, \tag{C.10}$$

where, for any integer L, $\mathbf{I}_L$ is the L-dimensional unit matrix.

The first r columns $\{\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_r\}$ of $\mathbf{U}$ form an orthonormal basis for the range $R(\mathbf{A})$; the remaining columns $\{\vec{u}_{r+1}, \ldots, \vec{u}_M\}$ form an orthonormal basis of[1] $R(\mathbf{A})^\perp = N(\mathbf{A}^T)$. Similarly, the first r columns $\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_r\}$ of $\mathbf{V}$ form an orthonormal basis for $R(\mathbf{A}^T)$; the remaining columns $\{\vec{v}_{r+1}, \ldots, \vec{u}_N\}$ form an orthonormal basis of the nullspace $N(\mathbf{A}) = R(\mathbf{A}^T)^\perp$.

Remark 10 By keeping only the first r rows and columns of $\mathbf{D}$ (all other ones are identically zero) it follows from (C.5) that $\mathbf{A}$ can also be represented as

$$\mathbf{A} = \mathbf{U}_r \cdot \mathbf{W} \cdot \mathbf{V}_r^T, \tag{C.11}$$

where $\mathbf{U}_r$ and $\mathbf{V}_r$ are the matrices obtained from $\mathbf{U}$ and $\mathbf{V}$ by keeping the first r columns only, and[2] $\mathbf{W} = \operatorname{diag}(\sigma_j)$ is the $r \times r$ matrix with the singular values on the diagonal. $\diamondsuit$

C.3.1 Left and right singular vectors

Using the representation $\mathbf{A} = \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{V}^T$ and the orthonormality conditions we can derive that

$$\begin{aligned} \mathbf{A}\,\mathbf{V} &= \mathbf{U}\,\mathbf{D} \\ \mathbf{A}^T\,\mathbf{U} &= \mathbf{V}\,\mathbf{D} \end{aligned} \tag{C.12}$$

Define the vectors $\vec{u}_j, \vec{v}_j$ as the jth column of the matrices $\mathbf{U}$ and $\mathbf{V}$, respectively:

$$(\vec{u}_j)_i = \mathbf{U}_{ij}, \quad (\vec{v}_j)_i = \mathbf{V}_{ij}.$$

Then (C.12) can be rewritten as

$$\boxed{\begin{aligned} \mathbf{A}\,\vec{v}_j &= \sigma_j\,\vec{u}_j, & j &= 1, \ldots, r & \text{(C.13)} \\ \mathbf{A}\,\vec{v}_j &= 0, & j &= r+1, \ldots, N & \text{(C.14)} \\ \mathbf{A}^T\,\vec{u}_j &= \sigma_j\,\vec{v}_j, & j &= 1, \ldots, r & \text{(C.15)} \\ \mathbf{A}^T\,\vec{u}_j &= 0, & j &= r+1, \ldots, M & \text{(C.16)} \end{aligned}}$$

These equations tell us that for $j = 1, \ldots, r$ the action of $\mathbf{A}$ on $\vec{v}_j$ produces $\vec{u}_j$ multiplied by the singular value σ_j, and similarly, that the action of $\mathbf{A}^T$ on $\vec{u}_j$ produces $\vec{v}_j$ multiplied by the singular value σ_j. Notice that any column of $\mathbf{V}$ with $j > r$ yields a solution to the *homogeneous equation* $\mathbf{A}\vec{x} = \vec{0}$.

[1] $R(\mathbf{A})^\perp$ denotes the complement of the range of $\mathbf{A}$.

[2] $\operatorname{diag}(a_j)$ is the diagonal matrix with entries $a_1, a_2, \ldots$ on the diagonal.

Eq. C.15 is equivalent to

$$\vec{u}_j^T\,\mathbf{A} = \vec{v}_j^T\,\sigma_j.$$

Therefore we call the $\{\vec{v}_j\}$ the *right* singular vectors and the $\{\vec{u}_j\}$ the *left* singular vectors of $\mathbf{A}$.

Finally we can form the matrices $\mathbf{A}^T\mathbf{A}$ or $\mathbf{A}\mathbf{A}^T$ and act with them on $\vec{v}_j$ or $\vec{u}_j$, respectively:

$$\begin{aligned}
\mathbf{A}^T\mathbf{A}\,\vec{v}_j &= \sigma_j\,\mathbf{A}^T\vec{u}_j = \sigma_j^2\,\vec{v}_j, \qquad j=1,\ldots,r &\text{(C.17)}\\
\mathbf{A}\mathbf{A}^T\,\vec{u}_j &= \sigma_j\,\mathbf{A}\vec{v}_j = \sigma_j^2\,\vec{u}_j, \qquad j=1,\ldots,r &\text{(C.18)}
\end{aligned}$$

This shows that $\vec{v}_j$ is an eigenvector of $\mathbf{A}^T\mathbf{A}$ and that $\vec{u}_j$ is an eigenvector of $\mathbf{A}\mathbf{A}^T$, both with eigenvalue σ_j^2, for $j=1,\ldots,r$.

Remark 11 If $\mathbf{A}$ is of rank r, the eigenvalues of $\mathbf{A}^T\mathbf{A}$ can be ordered as

$$\lambda_1 \ge \lambda_2 \ge \ldots \ge \lambda_r > \lambda_{r+1} = \lambda_{r+2} = \ldots = \lambda_N = 0.$$

Then we define w_j as the square root of the λ_j:

$$w_j = \sqrt{\lambda_j}, \quad j=1,\ldots,N.$$

The first r of the w_j's are the singular values σ_j[3]. ♢

Example 12 Consider the same matrix as in Example 8:

$$\mathbf{A} = \begin{pmatrix} 2 & 1 \\ 2 & 1 \end{pmatrix}$$

We know already that the rank $r=1$. To find the right singular vectors $\vec{v}_j$ construct the matrix $\mathbf{A}^T\mathbf{A}$:

$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 8 & 4 \\ 4 & 2 \end{pmatrix}$$

The eigenvalues of this matrix are $\lambda_1 = 10$ and $\lambda_2 = 0$ (so $w_1 = \sqrt{10}$ and $w_2 = 0$) with eigenvectors:

$$\vec{v}_1 = \frac{1}{\sqrt{5}}\begin{pmatrix} 2 \\ 1 \end{pmatrix}, \qquad \vec{v}_2 = \frac{1}{\sqrt{5}}\begin{pmatrix} 1 \\ -2 \end{pmatrix} \tag{C.19}$$

To find the left singular vectors $\vec{u}_j$ construct the matrix $\mathbf{A}\mathbf{A}^T$:

$$\mathbf{A}\mathbf{A}^T = \begin{pmatrix} 2 & 1 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 5 & 5 \\ 5 & 5 \end{pmatrix}$$

which has the same eigenvalues $\lambda_1 = 10$ and $\lambda_2 = 0$, with eigenvectors:

$$\vec{u}_1 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \vec{u}_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{C.20}$$

To check the SDV representation, we form the product

$$\mathbf{U}\cdot\mathbf{D}\cdot\mathbf{V}^T = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} \sqrt{10} & 0 \\ 0 & 0 \end{pmatrix}\frac{1}{\sqrt{5}}\begin{pmatrix} 2 & 1 \\ 1 & -2 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 2 & 1 \end{pmatrix}$$

which indeed equals $\mathbf{A}$. ♢

[3]Sometimes the w_j's themselves (including the zero ones) are called 'singular values'.

C.4 Consistent and inconsistent systems

The system of linear equations (C.1) falls in either of two classes depending on whether a solution does or does not exist.

C.4.1 Consistent systems

Eq. C.1 is said to be *consistent* when a solution exists. This can only happen when $\vec{b}$ lies in the range of $\mathbf{A}$.

For consistent equations another distinction arises depending on whether the solution is *unique* or not. If $b \in R(\mathbf{A})$ then the solution is unique if and only if $\mathbf{A}$ is of full column rank. Otherwise there is more than one solution: any vector in the nullspace can be added to a particular solution $\vec{x}^*$ in any linear combination: As a representative we can select the solution $\vec{x}$ with smallest length $\|\vec{x}\|$.

Definition 13 *A vector $\vec{x}^*$ is called a* minimum norm solution *of (C.1) if $\vec{x}^*$ is a solution to $\mathbf{A}\vec{x} = \vec{b}$ and $\|\vec{x}^*\| < \|\vec{w}\|$ for all other solutions $\vec{w}$.*

It can be shown that the minimum norm solution is *unique* and given by the solution of $\mathbf{A}\vec{x} = \vec{b}$ lying in the range of $\mathbf{A}^T$, or, what is the same, by the solution orthogonal to the nullspace $N(\mathbf{A})$, see Example 17.

Theorem 14 *Let $\mathbf{A}\vec{x} - \vec{b} = \vec{0}$ with $b \in R(\mathbf{A})$. Then $\vec{x}$ is of the form*

$$\vec{x} = \vec{x}^* + \lambda\vec{h} \text{ with } \vec{h} \in N(\mathbf{A}).$$

The minimum norm solution *is given by the solution of $\mathbf{A}\vec{x} = \vec{b}$ orthogonal to the nullspace $N(\mathbf{A})$.*

C.4.2 Inconsistent systems

Eq. C.1 is said to be *inconsistent* when no solution exists. This happens when $\vec{b}$ does not lie in the range of $\mathbf{A}$. Now we can look for the least squares solution, that is find

$$\vec{x} \text{ which minimizes } \rho := \left\|\mathbf{A}\vec{x} - \vec{b}\right\|.$$

The number ρ is called the *residual* of the solution.

Geometrically this means that we first project $\vec{b}$ onto the range of $\mathbf{A}$ and then solve the resulting consistent system $\mathbf{A}\vec{x} = P_{R(\mathbf{A})}\vec{b}$ as above (by $P_{R(\mathbf{A})}$ we denote the orthogonal projection on the range $R(\mathbf{A})$), see Example 17.

A vector $\vec{x}$ is a least squares solution of $\mathbf{A}\vec{x} = \vec{b}$ if and only if $\vec{x}$ is a solution of the following consistent system, called the *normal equation*:

$$\boxed{\mathbf{A}^T\mathbf{A}\,\vec{x} = \mathbf{A}^T\vec{b}.}$$

The matrix $\mathbf{A}^T\mathbf{A}$ is square of dimension $N \times N$. The least squares solution is unique only when $\mathbf{A}$ is of full column rank. In that case $\mathbf{A}^T\mathbf{A}$ is nonsingular and the unique least squares solution is given by

$$\vec{x}^* = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b}. \tag{C.21}$$

Example 15 Consider the linear system with two equations for one unknown x:

$$a_1 x = b_1 \tag{C.22}$$

$$a_2 x = b_2 \tag{C.23}$$

which can be written in matrix form as

$$\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} x = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

Then $\mathbf{A}^T\mathbf{A} = a_1^2 + a_2^2$, so that the least squares solution (C.21) is:

$$x = \frac{1}{a_1^2 + a_2^2}(a_1\ a_2)\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \frac{1}{a_1^2 + a_2^2}(a_1 b_1 + a_2 b_2).$$

$\diamondsuit$

If the least squares solution is not unique we may again select the one with minimum norm.

Definition 16 *A vector $\vec{x}^*$ is called a* least squares solution *of (C.1) if $\left\|\mathbf{A}\vec{x}^* - \vec{b}\right\| \leq \left\|\mathbf{A}\vec{v} - \vec{b}\right\|$ for all $\vec{v} \in \mathbb{R}^N$. A vector $\vec{x}^*$ is called a* minimum norm least squares solution *of (C.1) if $\vec{x}^*$ is a least squares solution to $\mathbf{A}\vec{x} = \vec{b}$ and $\|\vec{x}^*\| < \|\vec{w}\|$ for all other least squares solutions $\vec{w}$.*

Again it can be shown that the minimum norm least squares solution is *unique*.

If $\mathbf{A}\vec{x} = \vec{b}$ is a consistent system then the least squares solution is simply an actual solution.

Example 17 Again take the matrix of Example 8:

$$\mathbf{A} = \begin{pmatrix} 2 & 1 \\ 2 & 1 \end{pmatrix}$$

We have seen that the range is spanned by the vector $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and that the nullspace is spanned by the vector $\begin{pmatrix} 1 \\ -2 \end{pmatrix}$. Both the range and the nullspace can be represented as straight lines in the x_1-x_2 plane, see Fig. 20. Now take $\vec{b}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, which is in the range of $\mathbf{A}$. A particular solution is $\vec{x}^* = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. So the general solution is of the form

$$\vec{x} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \lambda \begin{pmatrix} 1 \\ -2 \end{pmatrix}. \tag{C.24}$$

This is the straight line L through $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ parallel to the line representing the nullspace. To find the minimum norm solution, we have to draw a perpendicular line from the origin to the line L: the value of λ corresponding to the intersection point can be found from the orthogonality condition

$$\begin{pmatrix} \lambda \\ 1 - 2\lambda \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -2 \end{pmatrix} = 0$$

yielding $\lambda = 2/5$. The minimum norm solution $\vec{x}^*$ is found by substituting this value of λ in (C.24):

$$\vec{x}^* = \begin{pmatrix} 2/5 \\ 1/5 \end{pmatrix}. \tag{C.25}$$

To check that this is the correct answer, compute the square norm of the general solution (C.24):

$$\left\| \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \lambda \begin{pmatrix} 1 \\ -2 \end{pmatrix} \right\|^2 = \lambda^2 + (1 - 2\lambda)^2 = 5(\lambda - 2/5)^2 - 1/5.$$

The minimum of the norm is indeed obtained for $\lambda = 2/5$.

Next we take a vector $\vec{b}_2$ outside the range of $\mathbf{A}$, for example $\vec{b}_2 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$. To compute the least squares solution, we first project $\vec{b}_2$ perpendicularly on the line representing the range, which yields the point $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, see Fig. 20. Then we have so solve the equation $\mathbf{A}\vec{x} = P_{R(\mathbf{A})}\vec{b}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. But this problem we already solved above, so that we conclude that the minimum norm least squares solution is again given by (C.25). $\diamondsuit$

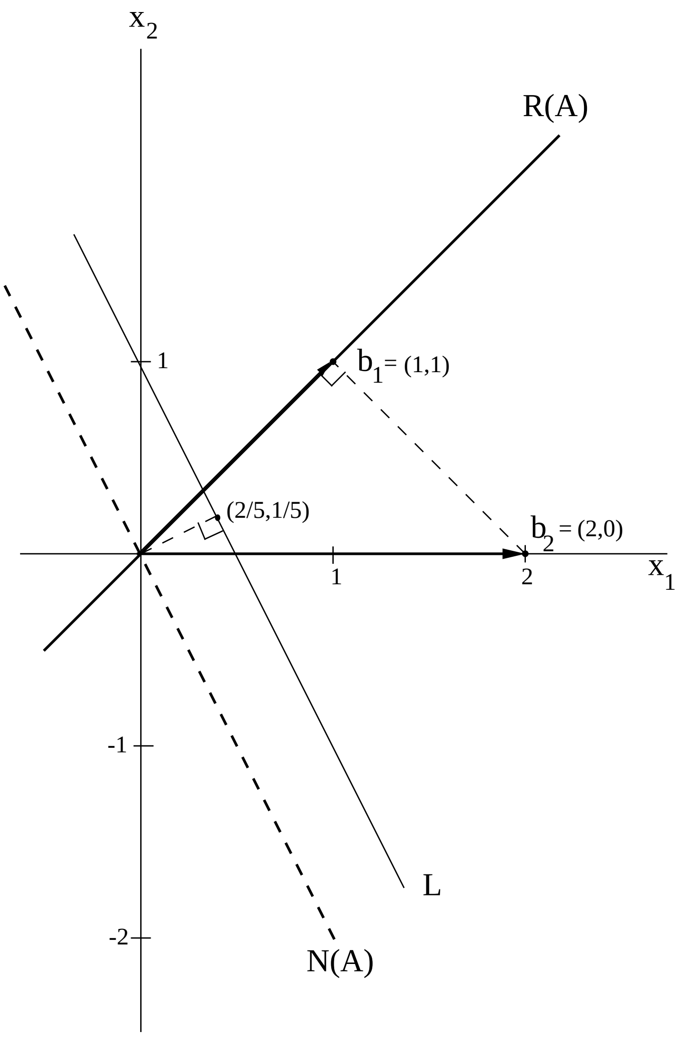


Figure 20: *Range $R(\mathbf{A})$ and nullspace $N(\mathbf{A})$ of the matrix in Example 17 in the x_1-x_2 plane. L: solution set of $\mathbf{A}\vec{x} = \vec{b}_1$. The minimum norm solution $(2/5, 1/5)$ is found by intersecting L with its perpendicular through the origin. The minimum norm least squares solution of $\mathbf{A}\vec{x} = \vec{b}_2$ is found by projecting $\vec{b}_2$ perpendicularly on the range and solving the resulting consistent equation.*

C.4.3 The Moore-Penrose inverse

Let $\mathbf{A}$ be a $M \times N$ real matrix. Then there is a unique matrix $\mathbf{A}^\dagger$, called the *Moore-Penrose inverse* of $\mathbf{A}$, satisfying the four equations:

$$\begin{aligned} \mathbf{A}\mathbf{A}^\dagger\mathbf{A} &= \mathbf{A} \\ \mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger &= \mathbf{A}^\dagger \\ (\mathbf{A}\mathbf{A}^\dagger)^T &= \mathbf{A}\mathbf{A}^\dagger \\ (\mathbf{A}^\dagger\mathbf{A})^T &= \mathbf{A}^\dagger\mathbf{A} \end{aligned}$$

The relevance of the Moore-Penrose inverse for least squares problems is given by the following theorem:

Theorem 18 *The following statements are equivalent:*

- $\vec{x}^*$ *is a least squares solution of* $\mathbf{A}\vec{x} = \vec{b}$.
- $\vec{x}^*$ *is a solution of* $\mathbf{A}\vec{x} = \mathbf{A}\mathbf{A}^\dagger\vec{b}$.
- $\vec{x}^*$ *is a solution of* $\mathbf{A}^T\mathbf{A}\vec{x} = \mathbf{A}^T\vec{b}$.
- $\vec{x}^*$ *is of the form* $\mathbf{A}^\dagger\vec{b} + \vec{h}$ *where* $\vec{h} \in N(\mathbf{A})$.

The unique minimum norm least squares solution is given by $\mathbf{A}^\dagger\vec{b}$.

C.5 Computing least squares solutions

Computation of the minimum norm least squares solution, that is of the Moore-Penrose inverse $\mathbf{A}^\dagger$, can be done using the SVD. The result is that

$$\mathbf{A}^\dagger = \mathbf{V} \cdot \mathbf{D}^\dagger \cdot \mathbf{U}^T \tag{C.26}$$

where

$$\mathbf{D}^\dagger = \left(\begin{array}{cccc|c} \frac{1}{\sigma_1} & & & & \\ & \frac{1}{\sigma_2} & & & \\ & & \cdots & & \\ & & & \frac{1}{\sigma_2} & \\ --- & --- & --- & --- & --- \\ & & & & \end{array} \right).$$

That is, you *simply replace* σ_j *by* $\frac{1}{\sigma_j}$ *and leave all zero entries of* $\mathbf{D}$ *unchanged*. From (C.11) we also can write

$$\mathbf{A}^\dagger = \mathbf{V}_r \cdot [\mathrm{diag}\,(\frac{1}{\sigma_j})] \cdot \mathbf{U}_r^T. \tag{C.27}$$

Then compute

$$\boxed{\vec{x} = \mathbf{A}^\dagger\vec{b}.} \tag{C.28}$$

This is the minimum norm least squares solution for inconsistent equations, which reduces to the minimum norm solution for consistent equations (which in turn equals the unique solution if $\mathbf{A}\vec{x} = \vec{b}$ is uniquely solvable).

Remark 19 For a square nonsingular matrix $\mathbf{A}$, the matrix $\mathbf{A}^\dagger$ is simply the inverse $\mathbf{A}^{-1}$ as mentioned in Example 7. $\diamondsuit$

Remark 20 Sometimes the singular values σ_j may become very small. Therefore it is a good idea to 'zero' the small singular values before using Eq. C.26. This is an example of *regularization*. Another possibility is to replace $\frac{1}{\sigma_j}$ by $\frac{1}{\lambda+\sigma_j}$ where λ is small positive real number. This is called *Tichonov-Phillips regularization*. $\diamondsuit$

Example 21 Consider again Example 12. Let us compute $\mathbf{A}^\dagger$ according to the recipe above:

$$\mathbf{A}^\dagger = \mathbf{V}\mathbf{D}^\dagger\mathbf{U}^T = \frac{1}{\sqrt{5}}\begin{pmatrix} 2 & 1 \\ 1 & -2 \end{pmatrix}\begin{pmatrix} \frac{1}{\sqrt{10}} & 0 \\ 0 & 0 \end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{10}\begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix}. \tag{C.29}$$

The minimum norm least squares solution of $\mathbf{A}\vec{x} = \vec{b}$ is therefore of the form:

$$\vec{x}^* = \mathbf{A}^\dagger\vec{b} = \frac{1}{10}\begin{pmatrix} 2(b_1 + b_2) \\ b_1 + b_2 \end{pmatrix}.$$

Both when $\vec{b} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and when $\vec{b} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$, this yields $\vec{x}^* = \begin{pmatrix} 2/5 \\ 1/5 \end{pmatrix}$, in agreement with what we found in Example 17. ♢

Example 22 Consider the rectangular matrix

$$\mathbf{A} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \tag{C.30}$$

This is a matrix of full column rank $r = 2$, so the least squares solution will be unique. The matrix $\mathbf{A}^T\mathbf{A}$ equals $\begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix}$, with eigenvalues and eigenvectors $\lambda_1 = 4$, $\vec{v}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $\lambda_2 = 1$, $\vec{v}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. The corresponding $\vec{u}_j = \frac{1}{w_j}\mathbf{A}\vec{v}_j$ are: $\vec{u}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, $\vec{u}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$. The 3×3 matrix $\mathbf{A}\mathbf{A}^T$ has an additional eigenvalue $\lambda_3 = 0$, but according to Equations (C.11) and (C.27) we don't need λ_3 and the associated $\vec{u}_3$ to represent $\mathbf{A}$ or to compute the Moore-Penrose inverse. Indeed,

$$\mathbf{U}_2 \cdot \mathbf{W} \cdot \mathbf{V}_2^T = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} = \mathbf{A},$$

and

$$\mathbf{A}^\dagger = \mathbf{V}_2 \cdot \mathbf{W}^\dagger \cdot \mathbf{U}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

♢

C.6 Constrained least squares

Sometimes we need to solve the system

$$\boxed{\begin{aligned} \mathbf{A}\vec{x} &= \vec{0} && \text{(C.31)} \\ \text{subject to } \|\vec{x}\| &= 1. && \text{(C.32)} \end{aligned}}$$

Then we take as solution the vector $\vec{x}$ of norm 1 that minimizes $\|\mathbf{A}\vec{x}\|^2$. It can be shown that this vector equals the right singular vector $\vec{v}_j$ of $\mathbf{A}$ having smallest singular value. The corresponding minimal value is:

$$\|\mathbf{A}\vec{v}_j\|^2 = \|w_j\vec{u}_j\|^2 = w_j^2.$$

There may be several such solutions if there are several singular values of smallest value.

Example 23 Consider again Example 22. The matrix (C.30) is of full column rank so (C.31) will only have the trivial solution $\vec{x} = \vec{0}$. According to the rule above the unit vector minimizing $\|\mathbf{A}\vec{x}\|^2$ is $v_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ and the minimal value equals $w_2^2 = 1$.

This can be checked as follows. We have that

$$\|\mathbf{A}\vec{x}\|^2 = \left\| \begin{pmatrix} 2x_1 \\ x_2 \\ 0 \end{pmatrix} \right\|^2 = 4x_1^2 + x_2^2.$$

Since $\vec{x}$ has to be a unit vector, we also have that $\|\vec{x}\|^2 = x_1^2 + x_2^2 = 1$, so $\|\mathbf{A}\vec{x}\|^2 = 3x_1^2 + 1$. This is minimal when $x_1 = 0$, from which $x_2 = 1$, and the minimal value equals 1. ♢

Example 24 As a final example consider the underdetermined system

$$\begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \vec{0}. \tag{C.33}$$

If we require $\|\vec{x}\|^2 = 1$ we have 3 equations with 3 unknowns.

Again apply the rule given above, that is, compute the eigenvalues and eigenvectors of

$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix},$$

which are $\lambda_1 = 3$, $\lambda_2 = 1$, $\lambda_3 = 0$. So the smallest w_j equals zero, and the singular vector $v_3 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$. So the minimal value of $\|\mathbf{A}\vec{x}\|^2$ equals 0, and this value is attained by $\vec{x} = \vec{v}_3$, which in this case is an actual solution of (C.33). ♢

References

[1] BEN-ISRAEL, A., AND GREVILLE, T. N. E. *Generalized Inverses and Applications*. J. Wiley, N. Y., 1974.

[2] CAMPBELL, S. L., AND JR, C. D. M. *Generalized Inverses of Linear Transformations*. Pitman, London, 1979.

[3] PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., AND VETTERLING, W. T. *Numerical Recipes, the Art of Scientific Computing*. Cambr. Univ. Press, New York, 1986.

D Recording ImageJ plugins - Tutorial

D.1 Introduction

The first part of this tutorial is an introduction to recording and running plugins for ImageJ. However, it does not explain the concept of plugins in ImageJ, which is covered in the "Writing ImageJ Plugins" tutorial. The second part contains short descriptions of the supplementary plugins which should help you during your lab sessions; examples of using some of them are also given.

D.2 Recording plugins

All exercises can be done as sequences of ImageJ menu commands, which you can record. The menu command *Plugins/Record...* opens a window and records your actions, as long as the "Record" checkbox is checked. While recording, you can modify (edit) the recorded pseudocode which is displayed in the Recorder's window. After you finished recording, the "Create Plugin" button can be clicked to generate a Java class from the pseudocode. The plugin class can be opened in the built-in editor from where you can also compile and run it. You can also compile and run existing plugins using the *Plugins/Compile and Run...* command. However, the plugins generated by recording should be placed in the ImageJ/plugins directory.

Example

In this example we will use the ConditionalDilation plugin found in the *Plugins/ComputerVision* menu.

- Start the plugin recorder by selecting *Plugins/Record...*.
- Open the mask image "cond_dil_mask.jpg" found in ImageJ/cv_images by selecting *File/Open...*.
- Open the seed image "cond_dil.jpg" found in ImageJ/cv_images by selecting *File/Open...*.
- Convert the images to a ImageJ stack by selecting *Image/Stacks/Convert images to stack*.
- Do a conditional dilation of the seed image inside the mask image by selecting *Plugins/ComputerVision/ConditionalDilation*.
- Stop the recorder by unchecking the "Record" checkbox.
- Create the plugin class by pressing the "Create Plugin" button.
- Save the plugin by selecting *File/Save As...*; accept the default name "Macro_java".
- Close the Recorder and the Editor windows and copy the file "Macro_java" in the ImageJ/plugins directory.
- Compile and run the macro by selecting *Plugins/Compile and Run...* and browse to the "Macro_java" file.

You should end up with a new image called "result", which (in this case) is the same with the mask image.

Remarks

- The order in which you open both the mask and the seed images is important. The ConditionalDilation plugin expects a stack with at least two slices, the first slice being the mask image and the second, the seed image.
- Details about ImageJ stacks can be found in the "Writing ImageJ Plugins" tutorial (p. 20).
- The name of an ImageJ plugin must contain at least one underscore.

D.3 Supplementary plugins

In this section some examples and short descriptions of the plugins found in the *Plugins/ComputerVision* submenu are given. All images used in the examples are found in the ImageJ/cv_images directory.

Genus

The *Genus* plugin computes the genus of a binary image; you can select either 4-connected or 8-connected genus.

Count

The plugin *Count* computes the number of 4-connected or 8-connected regions of a binary image.

ConditionalDilation

This plugin computes the conditional dilation. It requires an ImageJ stack with at least two slices (binary images), the mask image and the seed image, respectively. Its output is an 8-bit binary image.

Dilation3x3

Performs a morphological dilation with a flat structuring element of size 3x3. The input should be an 8-bit binary or grey-scale image. The input image is replaced with the dilated image (in-place computation).

Erosion3x3

Performs a morphological erosion with a flat structuring element of size 3x3. The input should be an 8-bit binary or grey-scale image, and the computation is done in-place.

Example

This example plugin computes the morphological opening of an 8-bit image; let us assume that a new image called “result” should contain the opened image. Since both Dilation3x3 and Erosion3x3 plugins compute their results in-place, we will use the menu command *Image/Duplicate...* to copy the original image in the result image, and then perform the operations on this image.

- Start the plugin recorder by selecting *Plugins/Record...* and open the image “blobs.jpg”.
- Copy the image in a new image by selecting *Image/Duplicate...*. Rename the image to “result” by selecting *Image/Rename...*.
- Select *Window/result* to select the current image, the image on which the operations are applied.
- Select *Plugins/ComputerVision/Erosion3x3* and then Dilation3x3. For both of them choose 8-connectivity.
- Create the plugin, copy it in the ImageJ/plugins directory, compile and run it.

Try to modify the generated java file to compute an opening with a 5x5 structuring element. To do this, you can repeat each erosion and dilation operations two times. The run function in the generated java file should be similar to:

```
public void run(String arg) {
     IJ.run("Open...", "path=/ImageJ/cv_images/blobs.jpg");
     IJ.run("Duplicate...");
     IJ.run("Rename...", "title=result");
     for(int i=0;i<2;i++)
```

```
         IJ.run("Erosion3x3 ", "connectivity=8");
      for(int i=0;i<2;i++)
         IJ.run("Dilation3x3 ", "connectivity=8");
 }
```

Compile and run the plugin.

DistanceTransform

Computes the generalized distance transform of a binary input image for a given structuring element, and writes the result to an 8-bit grey-scale image “gen_dist”.

Example

An example which uses the *DistanceTransform* plugin is given below.

- Start the plugin recorder and open the image “prak1.tif’.
- Select *Process/Binary/Threshold*. This command uses an adaptive algorithm to threshold the input 8-bit grey-scale image.
- Invert the binary image by selecting *Plugins/Demos/Inverter*.
- Select the distance transform plugin and choose the image “cross3x3.tif” as structuring element.
- Enhance the output image “gen_dist” by selecting *Image/Adjust/ Brightness/Contrast...* and in the B&C window select Auto.
- Create the plugin, copy it in the ImageJ/plugins directory, compile and run it.

DirectFFT

This plugin computes the discrete direct FFT transform of an 8-bit grey-scale input image; the output image is a 32-bit, grey-scale image.

InverseFFT

This plugin computes the discrete inverse FFT transform of a 32-bit grey-scale input image; the output image is also a 32-bit, grey-scale image. You can convert it to an 8-bit grey-scale by selecting *Image/Type/8-bit*.

Gaussian Filter

This plugin calculates a 2-D Gaussian lowpass filter using a 2-D Gaussian density function. It outputs an 8-bit grey-scale image of 256×256 pixels. The cutoff parameter defines the cutoff frequency. The DC-level parameter defines the height of the center component.

Example

This example uses the FFT and the Gaussian filter plugins. The plugin reads a grey-scale image of 256×256 pixels and it convolves the image in the Fourier domain with a 2-D Gaussian kernel.

- Start the plugin recorder and open the image “acros.tif”.
- Compute the direct FFT transform using the DirectFFT plugin.
- Generate a 2-D Gaussian kernel with a cutoff frequency of 10.0.

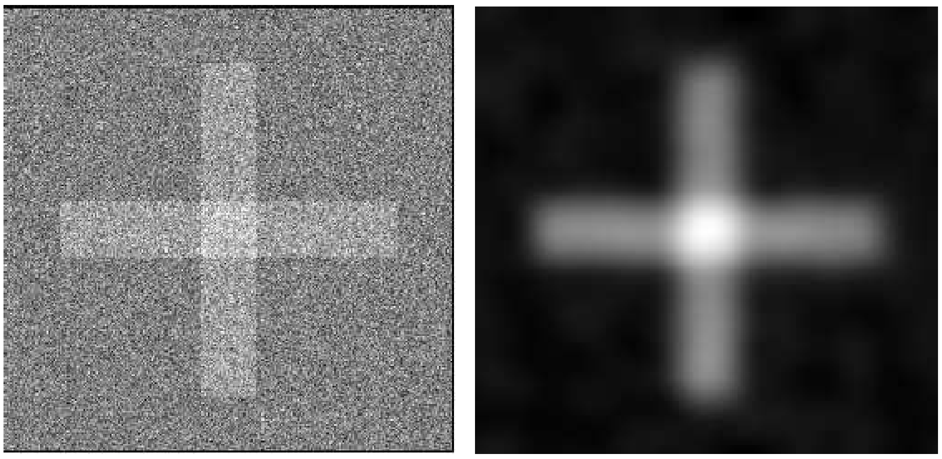


Figure 21: *Left: input image; Right: smoothed image*

- Compute the direct FFT transform of the kernel.
- Multiply the two FFT images by selecting *Process/Image Calculator....* Select the parameters: (i) Image1: “acros.tifFFT”, (ii) Operation: “Multiply” and (iii) Image2: “2D FilterFFT”. Uncheck the “create new window” checkbox; the resulting image will be stored in the selected Image1.
- Compute the inverse FFT transform of the resulting image.
- Create the plugin, copy it in the ImageJ/plugins directory, compile and run it.

The input and the output (filtered) images are shown in Fig. 21, and the recorded pseudocode should be similar to:

```
run("Open...",
 "path=/ImageJ/cv_images/acros.tif");
run("DirectFFT ");
run("Gaussian Filter", "cutoff=10 dc-level=255");
run("DirectFFT ");
run("Image Calculator...", "image1=acros.tifFFT
 operation=Multiply image2='2D FilterFFT'");
run("InverseFFT ");
```

Remarks

- The menu command *Process/Image Calculator...* can be used to perform blitting (pixelwise) operations such as: multiplication, addition, division, subtraction and logical operations.
- This is in contrast to the operations found in *Process/Math* which use only one value to perform the selected computations.

Laplacian

Performs convolution of a 32-bit grey-scale image with a discrete 3x3 approximation of the Laplacian operator; the result is returned in-place as a 32-bit grey-scale image.

ScanCorners and ScanPoints

The use of these plugins is explained during the lab sessions.

SetPixel

This plugin can be used to set one pixel in an 8-bit (binary) image. It waits the user to click a point in the selected image and it sets the corresponding pixel to 255.

SkeletonDecomposition and SkeletonReconstruction

The SkeletonDecomposition plugin performs the skeleton decomposition of an 8-bit binary image. It outputs an 8-bit grey-scale image containing the skeleton function for the chosen structuring element.

The SkeletonReconstruction plugin performs the skeleton reconstruction of an 8-bit grey-scale image containing the skeleton function. It outputs an 8-bit binary image for the chosen structuring element (the same as was used in the decomposition).

Threshold

This plugin thresholds the input 8-bit grey-scale image at the selected threshold value. It outputs the result in-place as an 8-bit binary image.

The source codes for all recorded example plugins can be found in the ImageJ/cv_doc/examples directory. Nevertheless, to get acquainted with ImageJ you should try to generate all examples by yourself.

E Manual ImageJ