Appendix 3

A

Basics of Computer Vision

Here, we will describe some of the techniques and methods of computer vision that are used in this work. The description is intended for the reader who is not familiar with computer vision. It is far from being an exhaustive introduction into the field. For further reading please refer to [Forsyth and Ponce, 2003, Gonzales and Woods, 1992, Phillips, 1994]. Note also that many of the presented techniques are provided by the Open Source Computer Vision Library OpenCV [OpenCV, 2004].

A.1 Digital Filters

A common technique in computer vision to extract information from an image is to apply digital filters to the image. These filters are *masks* (also called *filter kernels*) which are applied to the image by a method called *convolution.* A digital filter extracts particular information from an image, for example the high frequencies of the image: the edges. If the high frequencies are preserved an edge image is obtained, if the high frequencies are removed the result is a smoothed image. In the following, we first explain the method of discrete convolution before we elaborate on several digital filters.

A.1.1 Discrete Convolution

Discrete convolution is the main operation in digital image processing. With this operation, digital filters are applied to the image to extract the desired information. Convolution is applicable for one dimensional and for two dimensional functions as well as for continuous and discrete functions. We concentrate here on the two dimensional discrete case. In the continuous case, the integrals instead of the sums are determined [Gonzales and Woods, 1992]. Convolution is also known as *linear filtering* since the process is linear: first, the output for the sum of two images is the same as the sum of the outputs

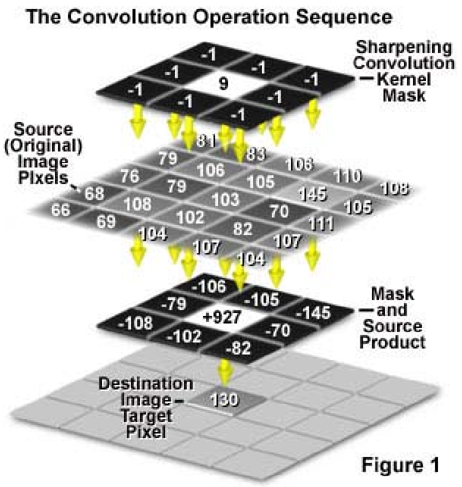

The Convolution Operation Sequence
Sharpening Convolution —Kernel Mask
Source (Original) Image Pixels
Mask —and Source Product
Destination Image - Target Pixel
130
Figure 1

Fig. A.1. Convolution. An original image is convolved with a filter mask for sharpening (edge detection). The mask is laid over the target pixel of the input image. Each component is multiplied with the corresponding pixel; the result is depicted here in the "mask and source product". Finally, the sum of these products gives the pixel value in the destination image (Fig. from [URL, 08])

obtained for the images separately and, second, the response to a scaled image is a scaled version of the response to the original image.

For convolving an image I with an $n \times n$ filter mask M (usually $n = 2k+1$ for $k > 0$), the following procedure is applied to each pixel: place the mask on the pixel and multiply each component of M with the corresponding pixel. Then sum the products and place the result in the center point of the image (see Fig. A.1) [Phillips, 1994]. Usually, the resulting pixel is also scaled to match the displayable intensity range. This procedure is repeated for every pixel of the input image by "moving" the mask over the image. That means, for the input image I and an 3×3 filter mask M, we obtain the following pixel value O_{ij} of the output image O:

$$O_{ij} = \begin{array}{llll} I_{(i-1)(j-1)} * M_{11} & + I_{(i-1)j} * M_{12} & + I_{(i-1)(j+1)} * M_{13} & + \\ I_{i(j-1)} * M_{21} & + \quad I_{ij} * M_{22} & + \quad I_{i(j+1)} * M_{23} & + \\ I_{(i+1)(j-1)} * M_{31} & + I_{(i+1)j} * M_{32} & + I_{(i+1)(j+1)} * M_{33} & + \end{array} \tag{A.1}$$

A common notation for convolving I with M is: $O = I * *M$ (two dimensional convolution); in the one dimensional case, it is $O = I * M$. The general form of two dimensional discrete convolution is:

$$O_{ij} = k_1 \sum_{x=-k}^{k} \sum_{y=-l}^{l} M(x,y) I(i+x, j+y) + k_0, \tag{A.2}$$

with k_0 and k_1 are constants for a scaling to the displayable intensity range [Hermes and Winter, 2003].

Usually, small filter masks are chosen (3×3 or 5×5) since it is rather time consuming to convolve an image with larger masks (the complexity is $O(nm)$ where n is the number of pixels in the image and m is the number of entries of the filter mask). The effect is that only small patches of the images are considered and for example a strong smoothing of an image is not possible. One possibility to regard larger filter kernels is to transform the image as well as the filter kernel into the frequency domain (with fast Fourier transform) and multiply the image with the kernel since convolution in the spatial domain is equivalent to multiplication in the frequency domain. However, this approach is also costly, thus as an alternative, another method is usually applied: the use of image pyramids (cf. sect. A.1.5).

Worth to note in the context of convolution is the *border problem*: the above procedure of moving the filter mask over the image is problematic for the image borders since the mask overlaps the image and equation A.2 can not be applied. There are different solutions of this problem:

- The border pixels of the image are cut of. This results in a slightly smaller output image. Usually, this is no problem, but if the mask is large or if the image is iteratively convolved, this procedure leads easily to very small images.
- The border pixels of the input image are copied into the output image.
- The image is periodically continued, that means the left border is considered to be adjacent to the right border, and the upper adjacent to the lower border.
- The overlapping entries of the mask are ignored and the weighted sum is restricted to the existing pixels. This is the most exact solution but requires a special treatment of the border pixels.

In our computations, we use the last solution.

A.1.2 Smoothing

Smoothing an image is used for blurring and for noise reduction. It is also referred to as *low-pass filtering* since the high frequencies are removed. The smoothing is done by convolving the image with a mask in which all values are positive. The simplest case is to use a mask in which all values are 1:

$$M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{A.3}$$

To prevent the resulting values to exceed the intensity range, the resulting value is divided by 9 (i.e., $k_1 = 1/9$ in equation A.2). Convolving an image with this mask has the effect of assigning each pixel the average value from its local neighborhood.

Usually, smoothing is done with a more sophisticated filter mask: the *Gaussian filter*. This mask adopts its values from a Gaussian, resulting in a smoothing that considers the center region more strongly than the surround. An example of a 3×3 Gaussian filter mask is:

$$M = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \tag{A.4}$$

with a factor $k_1 = 1/16$.

A.1.3 Edge and Bar Detection

Detecting edges in images is also done by convolution. It is also referred to as *high-pass filtering* since the high frequencies, the edges, are preserved. Edge detecting filter masks amplify the slope of the edge. Either a certain direction is preferred, for example horizontal edges, or all discontinuities in intensity are detected that means edges of arbitrary directions. The first kind of filters has zeros in the edge direction, positive values on the one side and negative values on the other. Thereby it enhances image regions with low values on one and high values on the other side. Edges of a particular direction are also called *bars*, whereas the direction-sensitive edge detection is also known as *bar detection*. The number of masks used for edge detection is almost unlimited. Some well known examples of direction-sensitive masks are the *Prewitt Operator*:

$$P_x = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \quad \text{and} \quad P_y = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} \tag{A.5}$$

with a factor $k_1 = 1/6$, and the *Sobel Operator*:

$$S_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad \text{and} \quad S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{A.6}$$

with a factor $k_1 = 1/8$. P_x and S_x find vertical and P_y and S_y horizontal edges. Equally, the numbers may be arranged to achieve the detection of diagonal edges. An example of a filter masks that detects edges of arbitrary directions is the *Laplace Operator*:

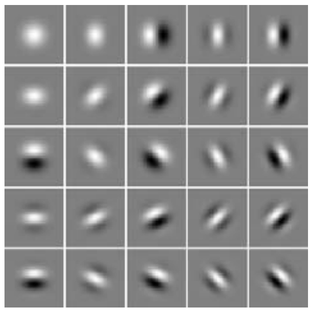


Fig. A.2. A set of Gabor filters for different spatial frequencies; Mid-grey values represent zero, dark values represent negative numbers and bright values represent positive numbers (Fig. from [URL, 17])

$$M = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \tag{A.7}$$

It responds strongly to regions, in which the value of the center region differs from its surrounding. Therefore, it responds strongly to edges but even more to single outlier pixels, making the operator highly sensitive to noise.

If image orientations of a certain direction have to be obtained, a good method is to use *Gabor filters*. A Gabor filter responds strongly to image regions which have a particular spatial frequency and orientation. Their behavior is similar to the response of orientation sensitive cells in the human cortex. Mathematically, the Gabor filter kernels are the product of a symmetric Gaussian with an oriented sinusoid. They come in pairs, one recovers symmetric components in a particular direction, the other recovers antisymmetric components. The form of the symmetric kernel is

$$G_{sym}(x,y) = cos(k_x x + k_y y)\ exp - \Big(\frac{x^2+y^2}{2\sigma^2}\Big), \tag{A.8}$$

and of the antisymmetric kernel

$$G_{anitsym}(x,y) = sin(k_0 x + k_1 y)\ exp - \Big(\frac{x^2+y^2}{2\sigma^2}\Big), \tag{A.9}$$

where (k_x, k_y) give the spatial frequency to which the filter responds most strongly, and σ is the scale of the filter [Forsyth and Ponce, 2003]. Fig. A.2 shows some examples of Gabor filter kernels for different spatial frequencies.

A.1.4 Morphological Filtering

Morphological filter operations deal with the shape of image regions. These operations work usually on binary images with one or several regions of a

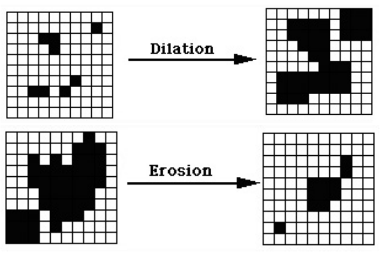


Fig. A.3. The morphological filter operations dilation and erosion. Dilation enlarges image regions whereas erosion shrinks them

particular intensity value and background pixels with the value zero. In the following, we assume that the regions have the value 1. We focus here on the two operations *dilation* and *erosion*, since only dilation is used in this work and erosion is its counterpart. Note that there are other morphological filters, for example *opening* and *closing* or *thinning* and *skeletonization*. These are explained for example in [Gonzales and Woods, 1992] or [Phillips, 1994].

Dilation and erosion are both *neighbor operations*, that means the value of a pixel is changed according to the value of its neighbors. The operations are used to smooth the shape of regions, join broken or discontinuous shapes, or to separate touching regions.

Dilation

Dilation makes a region larger by adding pixels around its edges. This can be done by defining a threshold t and setting a zero pixel to 1 if the number of differing neighbors exceeds t [Phillips, 1994]. If $t = 0$, all pixels in the 3×3 neighborhood of a region pixel are set to 1. This case is shown in Fig. A.3, top.

Erosion

Erosion is the counterpart of dilation and makes a region smaller. The technique is equivalent to dilation: if the number of zero neighbors of a region pixel exceeds a threshold t the region pixel is set to zero too [Phillips, 1994]. If $t = 0$, all border pixels of a region are eliminated. This case is shown in Fig. A.3, bottom.

A.1.5 Image Pyramids

A common technique in computer vision to extract information on different scales is the use of image pyramids [Tanimoto and Pavlidis, 1975,Burt, 1980]. The idea is that applying large filter masks to an image is very costly, so a

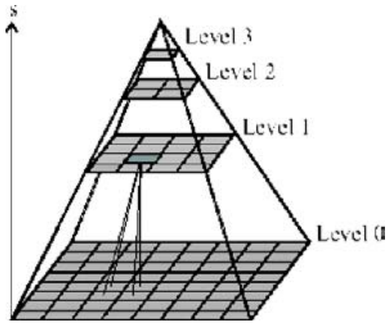


Fig. A.4. An image pyramid with four levels

better approach is to shrink the image and apply small filters to each shrunken image version. An image pyramid is simply a collection of representations of an image of different sizes (see Fig. A.4). Usually, each layer of the pyramid is half the width and height of the previous layer.

The easiest way to obtain an image pyramid is to take every kth pixel (typically every 2nd pixel) from layer n to obtain layer $n + 1$. This method is called *resampling*, *subsampling* or *downsampling*. Since *Nyquist's theorem* states that the sampling frequency must be at least twice the highest frequency present for a signal to be reconstructed from a sampled version, this approach might lead to problems, namely to an effect called *aliasing* [Forsyth and Ponce, 2003]. Aliasing means that a signal which is sampled too slowly will be misrepresented by the samples since high spatial frequencies will appear as low spatial frequencies. An example is a chessboard: imagine that each field of the board is represented by one pixel. If you take now every 2nd pixel for subsampling, the resulting image is only white or black (depending on the pixel you start with). Aliasing can be avoided by filtering the image so that spatial frequencies above the new sampling frequency are removed. This is done by smoothing. Therefore, a common procedure is to first smooth the image with a Gaussian filter and then subsample it. The resulting image pyramid is called *Gaussian Pyramid*. An example of a Gaussian pyramid was depicted in Fig. 4.2.

Another important pyramid is the *Laplacian Pyramid*. It consists of bandpass filtered versions of the input image: each stage of the pyramid is constructed by subtracting two corresponding adjacent levels of the Gaussian pyramid. Thereby, the smoothed "lowpass" image is subtracted from the original "highpass" image resulting in a bandpass image which contains most of the image's important textural features such as edges. The Laplacian pyramid is named as such because the process is approximately equivalent to convolving the image with the Laplacian of the Gaussian smoothing filter.

From the Laplacian pyramid, an *oriented Laplacian pyramid* is obtained by applying Gabor filters of different directions to each level of the pyra-

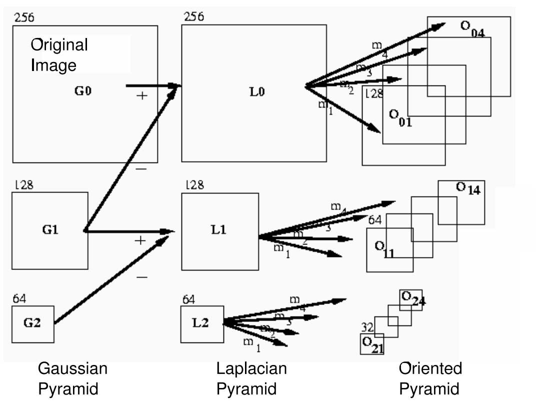


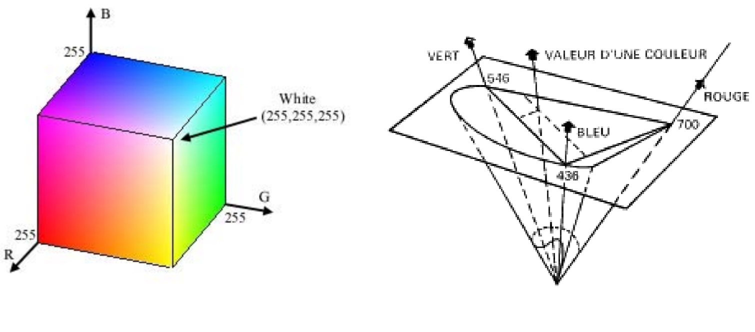
Fig. A.5. To obtain an oriented pyramid O, first a Gaussian pyramid G is computed from the input image, second a Laplacian pyramid L is obtained from the Gaussian pyramid by subtracting two adjacent levels and, finally, Gabor filters of 4 directions are applied to each level of the Laplacian pyramid. O_{ij} denotes the ith level of the pyramid with orientations of direction j (Fig. reprinted with permission from [Greenspan et al., 1994]. ©1994 IEEE)

mid [Greenspan et al., 1994]. In other words, an oriented pyramid consists of several pyramids of edge images, one for each direction. This method was applied to obtain the orientation scale maps on page 61 by considering 4 orientations: $0°, 45°, 90°, 135°$. In Fig. A.5 we show how an oriented pyramid is obtained.

A.2 Color Spaces

Colors are organized in so called color spaces. These spaces are three dimensional since three parameters are sufficient to define almost all colors we perceive. One of the most common color spaces is the RGB color space which is represented as a cube (the RGB cube, see Fig. A.6, left). A color is produced by adding different quantities of the three components red, green, and blue. Another important color space is HSV (H = hue, S = saturation, V = value (luminance)). It can be thought of as a RGB cube tipped up onto one corner. This color space is more intuitive and enables to regard colors independently of their intensity (V) by just ignoring one parameter.

In 1931, an international committee, the CIE (Commission Internationale de l'Eclairage), defined the color space CIE XYZ in which all possible colors that could be made by mixing red, green and blue light sources can be represented using only positive values of X, Y and Z. These colors are arranged in a three-dimensional coordinate system in which Y stands for luminance whereas X and Z give coloring information. Usually, colors are not specified



(a) RGB color cube (b) CIE XYZ color space

Fig. A.6. **(a)** the RGB cube. Colors are expressed by a red (R), a green (G), and a blue (B) component (Fig. from [URL, 09]). **(b)** the CIE XYZ color space (Fig. from [URL, 10])

in XYZ coordinates but in *chromaticity coordinates* which are independent of the luminance (hue and saturation taken together are called chromaticity). The shape of the represented colors in a two-dimensional space forms a horseshoe with white in the middle and the colors, with increasing saturation, are arranged circularly around this point. The CIE XYZ color space is depicted in Fig. A.6, right. The XYZ coordinates are obtained from RGB by the following equation:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{A.10}$$

where X_r, X_g and X_b are the weights applied to the monitor's RGB colors to find X, and so on [Foley et al., 1990]. In this work, we used the following values which are from the converting function of the OpenCV library [OpenCV, 2004]:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.412411 & 0.357585 & 0.180454 \\ 0.212649 & 0.715169 & 0.072182 \\ 0.019332 & 0.119195 & 0.950390 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{A.11}$$

In 1976, the CIE defined two new color spaces to achieve more accurate models: the CIE LUV color space (also $L^*u^*v^*$) and the CIE LAB color space (also $L^*a^*b^*$) [Hunt, 1991]. These color spaces are substantially *uniform*, that means, if the distance between two colors in coordinate space is below some threshold, a human observer is not able to distinguish the colors, and the perceived difference between two colors depends on the distance of the colors in the color space, regardless of where in the space the colors are (note that this is not true for other color spaces, for example the HSV or the XYZ color space!).

We concentrate here on the CIE LAB color space, which is well suited for our application since it represents colors similar to human perception. The three parameters in the model represent the luminance of the color (L, the smallest L yields black), its position between red and green (A, the smallest A yields green) and its position between yellow and blue (B, the smallest B yields blue). A rendering of the CIE LAB space is shown in Figure A.7. The coordinates of a color in LAB are obtained as a nonlinear mapping of the XYZ coordinates:

$$L = 116 \left(\frac{Y}{Y_n}\right)^{1/3} - 16 \tag{A.12}$$

$$a = 500 \left[\left(\frac{X}{X_n}\right)^{1/3} - \left(\frac{Y}{Y_n}\right)^{1/3}\right] \tag{A.13}$$

$$b = 200 \left[\left(\frac{Y}{Y_n}\right)^{1/3} - \left(\frac{Z}{Z_n}\right)^{1/3}\right], \tag{A.14}$$

where X_n, Y_n and Z_n are the XYZ coordinates of a reference white patch [Forsyth and Ponce, 2003, Hunt, 1991].

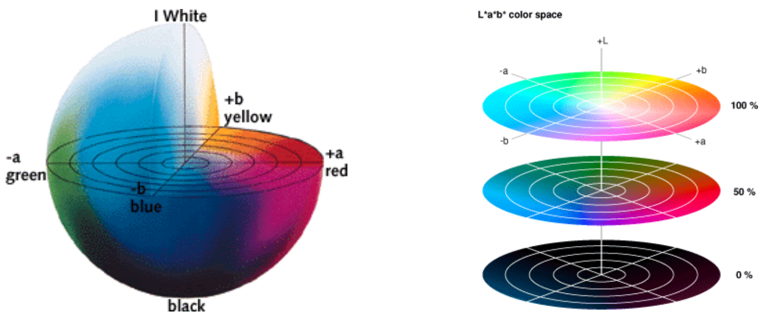


Fig. A.7. The CIE LAB color space. The space is spanned by the axes L (luminance), A (red-green), and B (blue-yellow). On the right, three disks for constant luminance. In chapter 4, we regard the middle disk for the computation of the color maps (Figures from [URL, 11] and [URL, 12])

A.3 Segmentation

Segmentation is the process of dividing an image into regions. It is the first step in the process of image analysis. In contrast to image processing, in image analysis the image is not altered but its content is analyzed. The basic idea of segmentation is to group similar pixels together into regions. Similarity is defined due to intensity, texture, or other properties. Autonomous segmentation is one of the most difficult tasks in image analysis since it is difficult to decide which pixels belong to the same region.

There are many techniques for segmentation (see for example [Gonzales and Woods, 1992] or [Phillips, 1994]). Here, we focus on the simple approach of *seeded region growing* which was used in chapter 4 to determine the most salient region in the saliency map. It starts with a set of "seed" points and from these it grows regions by appending to each seed those neighboring pixels that have similar properties such as intensity, color, or texture. There are two difficulties with this approach. The first is how to select the seeds. The output of the segmentation depends highly on this selection. Fortunately in the application of determining the most salient region, the choice of the (single) seed is obvious: it is simply the brightest pixel in the saliency map. The second difficulty is the selection of the similarity criteria. One solution is to consider all pixels that are neighbored to the seed and differ from it less than p percent according to one or several properties. When determining the most salient region in the saliency map, we accepted all pixels that differed in intensity at most 25% from the seed.