Digital Image Processing

by Sarp Ertürk
University of Kocaeli

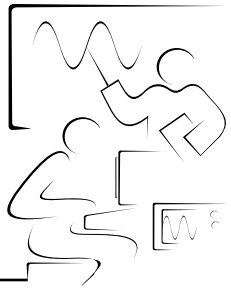
February 2003 Edition
Part Number 323604A-01

For More Information

Further information about this course can be obtained from the author:
sertur@kou.edu.tr

National Instruments Corporate Headquarters
11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices
Australia 61 2 9672 8846, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599,
Canada (Calgary) 403 274 9391, Canada (Montreal) 514 288 5722, Canada (Ottawa) 613 233 5949,
Canada (Québec) 514 694 8521, Canada (Toronto) 905 785 0085, Canada (Vancouver) 514 685 7530, China 86 21 6555 7838,
Czech Republic 42 02 2423 5774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24,
Germany 49 0 89 741 31 30, Greece 30 2 10 42 96 427, Hong Kong 2645 3186, India 91 80 4190000,
Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400, Malaysia 603 9059 6711,
Mexico 001 800 010 0793, Netherlands 31 0 348 433 466, New Zealand 64 09 914 0488, Norway 47 0 32 27 73 00,
Poland 48 0 22 3390 150, Portugal 351 210 311 210, Russia 7 095 238 7139, Singapore 65 6 226 5886, Slovenia 386 3 425 4200,
South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51,
Taiwan 886 2 2528 7227, United Kingdom 44 0 1635 523545



Contents

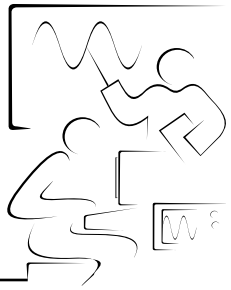
Lab 4
Image Enhancement—Spatial Operations

Lab 5
Color Images

Lab 6
The Frequency Domain

Lab 7
Morphological Image Processing

Introduction

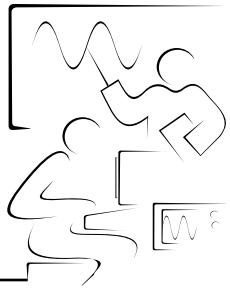


Digital image processing is rapidly becoming popular with many uses in scientific and engineering applications. Digital image processing is therefore included as graduate course in many electronics and computer engineering programs. The ease of LabVIEW programming and the many image processing functions incorporated into IMAQ Vision enables the implementation of simple and efficient digital image processing algorithms. This manual is designed to be useful as an aid for classroom demonstrations as well as a laboratory guide for interactive studies.

The image processing labs are structured to follow most digital image processing textbooks and include the fundamentals of the basic topics of digital image processing. The first two labs give an introduction to digital image processing and basic processing techniques. Labs 3 and 4 comprise point and spatial operations, which constitute popular image processing techniques commonly grouped under the title of image enhancement. Lab 5 is devoted to color images and introduces the commonly used RGB and HSI color formats. Lab 6 covers the frequency domain, introducing spatial frequency and demonstrating frequency domain image processing. Lab 7 deals with a more specific, but potentially useful, image processing approach called morphological image processing.

LabVIEW with IMAQ Vision installed is sufficient to run the included VIs, no special hardware or setup is required.

Lab 1
Digital Images



An Introduction to Digital Images

Images are a way of recording and presenting information in a visual form. Thus, images can be thought of as pictures, however in the broadest sense an image can correspond to any kind of two-dimensional data. Digital image processing refers to images being manipulated by digital means, most commonly computers. The naturally occurring form of images is not suitable for processing with computers, as computers cannot operate directly on pictorial data but require numerical data. Therefore images need to be converted into numerical data, referred to as digital images, to enable computer manipulation.

A digital image corresponds to an array of real or complex numbers represented by a finite number of bits, showing visual information in a discrete form. Although in some cases digital images might directly be synthesized in discrete form, most commonly they are translated from a physical image. The translation from a physical image into an appropriate digital form is accomplished by an **analogue to digital converter** (ADC) that carries out **sampling** and **quantization**.

Sampling

Sampling is the process of measuring the value of the physical image at discrete intervals in space. Most commonly a rectangular sampling grid is employed. Each image sample corresponds to a small region of the physical image, and is called a *picture element*, or *pixel*. The sampling process is shown in Figure 1-1. A digital image thus corresponds to a two-dimensional array of pixels.

A common approach is to index digital image pixels by x and y coordinates with the upper left corner taken as origin, and x and y being integer values. This convention is shown in Figure 1-2.

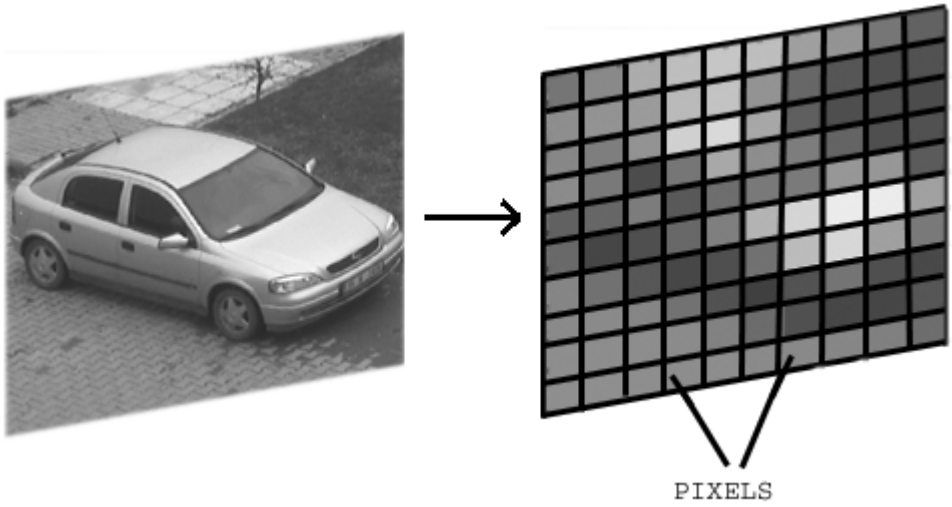


Figure 1-1. A physical image and the sampling process

Figure 1-2. Digital image coordinate system convention

The horizontal and vertical sampling rates, thus the number of pixels in the horizontal and vertical direction, give the **pixel dimensions** of the image. A digital image constructed by 640 samples in the horizontal direction and 480 samples in the vertical direction is referred to as a 640×480 image. An image pixel dimension of 640×480 is the common broadcast video standard. Digital still cameras directly produce digital images where usually the image dimension can be set by the user, such as 1024×768, 1280×1024, etc.

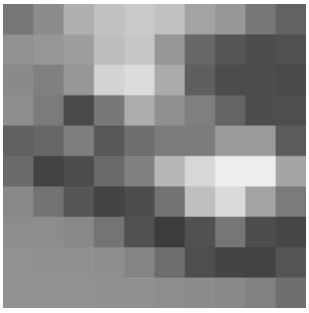
The physical size of a pixel in an image is defined by the **spatial resolution**. The spatial resolution of an image is commonly expressed synonymously in terms of dots-per-inch (dpi) or pixels-per-inch (ppi). For a fixed physical image region, dense sampling will result in a high resolution image with a large number of pixels each contributing a small part of the scene, while

coarse sampling will result in a low resolution image with a small number of pixels each contributing a large part of the scene.

The relation between the physical dimensions of the image, the pixel dimensions of the image and the spatial resolution is straight forward: the pixel dimensions of an image can be obtained by multiplying respectively the physical width and height of the image by the resolution. For example a 10” × 12” document scanned at 300 dpi will result in a digital image of dimensions 3000 × 3600 pixels.

$$\text{pixel dimensions} = \text{physical dimensions} \times \text{resolution}$$

Figure 1-3 shows the relation between image resolution, physical dimensions of the image and pixels dimensions of the image, and their effects on visual appearance. Commonly pixel dimensions are used analogously with physical dimensions, assuming a constant image resolution, in which case the physical dimensions of the image are changed according to changes in pixel dimensions.



Physical dim: 1.6” × 1.6”
Pixel dim.: 10 × 10 pixels
Resolution: 6 ppi

Physical dim: 1.6” × 1.6”
Pixel dim.: 50 × 50 pixels
Resolution: 30 ppi

Physical dim: 1.6” × 1.6”
Pixel dim.: 200 × 200 pixels
Resolution: 120 ppi

Physical dim: 0.7” × 0.7”
Pixel dim.: 50 × 50 pixels
Resolution: 70 ppi

Figure 1-3. Effects of the resolution, physical dimensions and pixel dimensions of an image on visual appearance

Quantization

Quantization is the process of replacing the continuous values of the sampled image with a discrete set of quantization levels. The number of quantization levels employed governs the accuracy by which the image values are displayed. Conventionally L quantization levels are defined by integers ranging from 0 to L-1, usually 0 corresponding to black and L-1 corresponding to white, with intermediate levels representing various shades of grey. All grey levels ranging from black to white are collectively referred to as **grayscale**. Grayscale images do not convey any colour information. For convenience and efficiency the number of quantization levels is usually an integral power of two: $L = 2^b$, so that that a total of b bits may used to represent L different quantization levels. The number of bits used to define a pixel value is called the **bit depth**. The greater the bit depth the greater the number of quantization levels and thus the tones that can be represent in the digital image. An image represented by only 1 pixel with 0 representing black and 1 representing white is called a bi-tonal or binary image.

Typically a bit depth of 8 bits is employed in digital imaging so that 256 possible grey levels ranging from 0 (black) to 255 (white) are used as pixel values. This is shown in Figure 1-4.

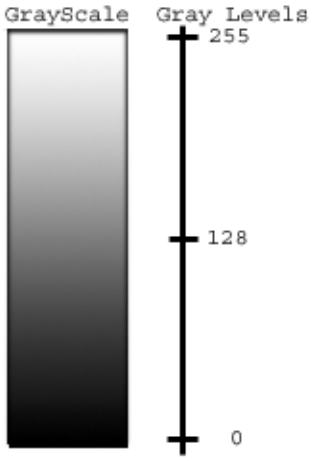


Figure 1-4. 256 different gray levels constructed for a bit depth of 8

Figure 1-5 shows an image at bit depths of 1 bit/pixel, 4 bits/pixel and 8 bits/pixel. It is clear that 1 bit/pixel is insufficient to display image detail but at 4 bits/pixel the visual appearance is reasonable.

1 bit/pixel 4 bits/pixel 8 bits/pixel

Figure 1-5. Effect of various bit depths on visual appearance

Image Re-sampling and Raw Images

It is possible to change the pixel dimensions of a digital image by **re-sampling**. A simple approach is to drop pixels at regular intervals to reduce the pixel dimensions, or duplicate pixels to increase pixel dimensions. While dropping is an efficient method for reducing pixel dimensions of an image, in the case of increasing pixel dimensions simple duplication results in visually observable blocking effects degrading visual quality of the constructed image. Instead of replicating image pixels it is preferable to interpolate new pixels in-between the original pixels in an appropriate way. There are various interpolation techniques, such as bi-linear interpolation, cubic spline interpolation and quadratic interpolation, which ensure that interpolated pixel values are reasonably computed from the original pixel pattern.

Figure 1-6 shows the effect of duplicating (also called zero-order interpolation) image pixels to increase pixel dimensions of an image as well as bi-linear interpolation of image pixels. It is clearly seen that interpolation avoids blocking artifacts.

Figure 1-6. Effect changing pixel dimensions by duplication (zero-order interpolation) and bi-linear interpolation. Top image: 70 × 70 pixels. Bottom-left image: 210 × 210 zero-order interpolated. Bottom-right image: 210 × 210 bi-linear interpolated.

Images that do not contain any formatting, but just the quantized pixel values are called **raw images**. Raw images may contain a header that specifies the pixel dimensions (width and height) of the image, or may contain only the pixel values with no header at all. In the second case it is required that the user specifies the width and height of the raw image so that it can be constructed properly. As no formatting is available for raw images, the raw image constitutes an array of pixel values of length width × height.

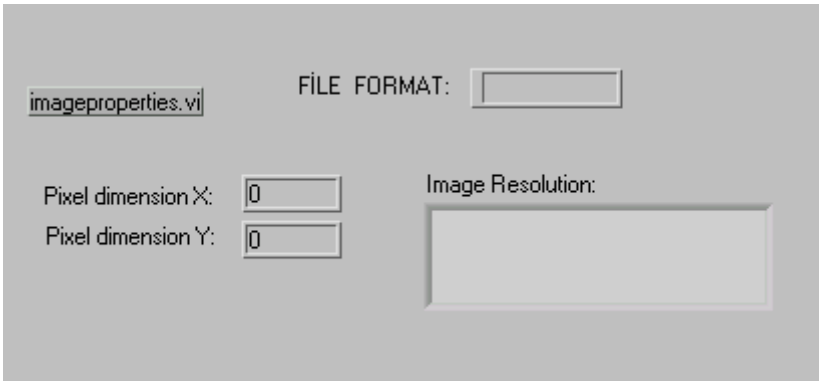
Images are by de-facto stored and processed in row-order, so that pixel values are encountered line by line. Figure 1-7 shows the effect of supplying incorrect pixel dimensions for a raw image. If the width is supplied correctly, the image is properly formatted as the end of each line of pixels is correctly known, yet if the height is incorrect the image is simply cropped at the specified height. If the width of the image is incorrect then the image cannot be formatted appropriately as the end of each line of pixels is taken incorrectly.

Figure 1-7. Effect of pixel dimensions on a raw image.
Top-left: correctly opened as 200×200.
Top-right: incorrectly opened as 100×200.
Bottom: incorrectly opened as 200×100.

Note LabVIEW IMAQ uses resolution analogous to pixel dimensions. Thus the horizontal pixel dimensions are referred to as ‘x resolution’ and the vertical dimensions are referred to as ‘y resolution’ in LabVIEW.

LabVIEW Demo 1.1: Image Properties

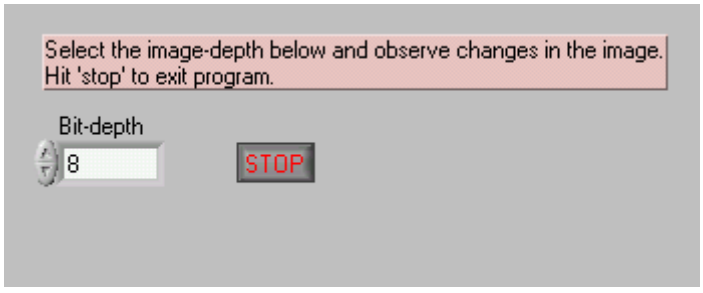
Launch the LabVIEW program entitled `imageproperties.vi` from the chapter 1 library. Run the program.



The file dialog will appear, asking for an image file to be selected. LabVIEW by default accepts standard image files in BMP, TIFF, JPEG, PNG, and AIPD formats. Select an image file in one of these formats (due to program restrictions the file will be processed as 8-bit grayscale even if a colour image is selected). The image will appear in the display window. The program panel will show the file format, pixel dimensions and image resolution (if specified). Note that the image resolution is not specified for all images, however that pixel dimensions must definitely be specified. Standard image formats include an image header previous to the visual information. The image header gives side information about the image, such as the image format, image dimensions, and possibly the image resolution.

LabVIEW Demo 1.2: Bit-Depth

Launch the LabVIEW program entitled `bitdepth.vi` from the chapter 1 library. Run the program.

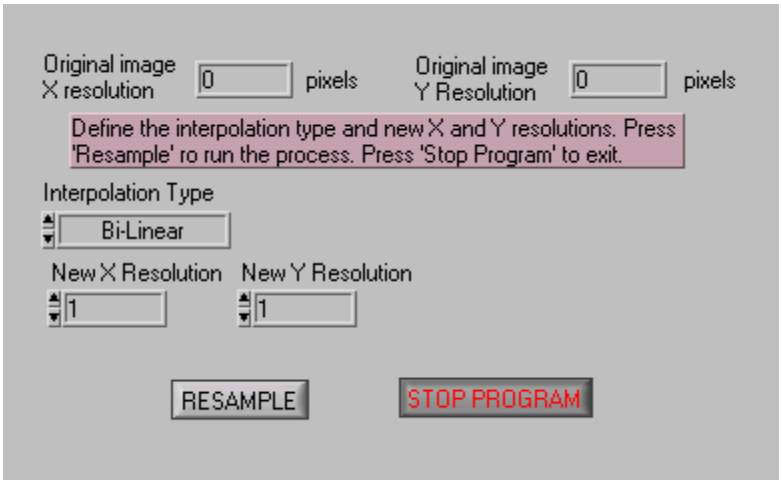


The file dialog will appear, asking for an image file to be selected. Select an image file in one of the formats supported by LabVIEW. The image will appear in the display window. Now change the bit-depth of the image and

examine changes in visual appearance. Observe that there is no obvious change between a bit-depth of 8 and a bit-depth of 7, while differences become visually significant when the bit-depth is further reduced.

LabVIEW Demo 1.3: Re-sampling

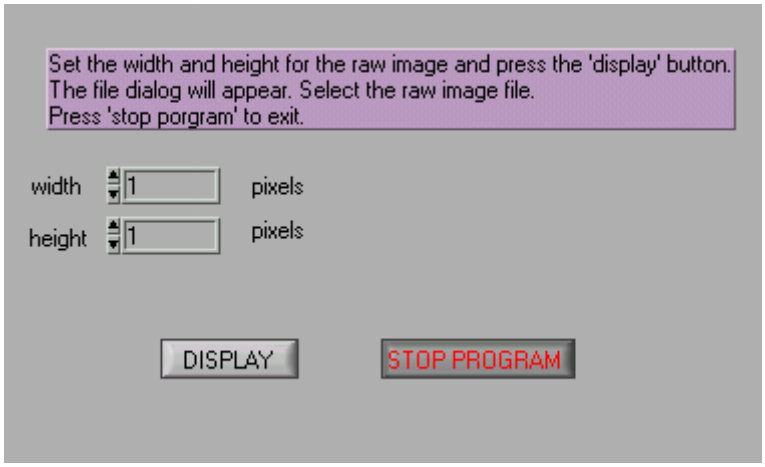
Launch the LabVIEW program entitled `resample.vi` from the chapter 1 library. Run the program.



The file dialog will appear, asking for an image file to be selected. Select an image file in one of the formats supported by LabVIEW. The image will appear in the display window and the original pixel dimensions of the image will be displayed. Change the pixel dimensions of the image and examine the visual appearance for various interpolation schemes. Observe that blocking artifacts occur for zero-order interpolation if the pixel dimensions are significantly increased, while for other interpolation schemes new pixels are smoothly interpolated so that the image may seem blurry, however blocking is avoided.

LabVIEW Demo 1.4: Raw Images

Launch the LabVIEW program entitled `displayrawimage.vi` from the chapter 1 library. Run the program.

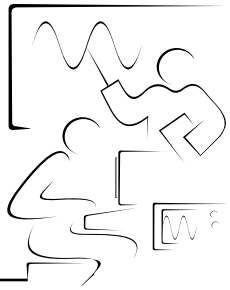


Load an image in raw format. Observe the effect of supplying incorrect pixel dimensions to the program.

Notes

Notes

Lab 2
Basic Image Processing



An Introduction to Image Processing

Image processing refers to the procedure of manipulating images. Commonly image processing is carried out in digital domain by means of computers. Digital image processing covers a wide range of different techniques to change the properties or appearance of an image. On the easiest level, image processing can be accomplished by changing the physical location of the pixels of an image.

It is possible to take the symmetry of an image by reversing the pixels according to a symmetry location. Figure 2-1 shows the result of a vertical symmetry and horizontal symmetry. In the case of a vertical symmetry, the process can be thought of image pixels being upturned with respect to a vertical line. If image1[x][y] represents the pixel value of the original image at row x and column y, and the image dimensions are given as $width \times height$, vertical symmetry can be formulated as image2[x][y]= image1[$width$-x][y] ($0 \le x < width$ and $0 \le y < height$). In the case of horizontal symmetry, the resultant image is given by image2[x][y]= image1[x][$height$-y]. It is possible to take increase the number of possible symmetries by combining vertical and horizontal symmetries and furthermore interchanging row and columns of pixels.

Original Image | Vertical Symmetry | Horizontal Symmetry

Figure 2-1. Image Symmetry

It is possible to change the location of pixels of an image by simple translation. If all pixels are shifted towards the right, left, up or down without changing the interconnection the entire image will be translated. Figure 2-2 shows the result of a horizontal and vertical shift of 20 pixels. The horizontal shift can be formulated as image2[x][y]= image1[$x+\Delta x$][y] and the vertical shift can be formulated as image2[x][y]= image1[x][$y+\Delta y$], where Δx and Δy are the horizontal and vertical translation amounts in pixels respectively. Due to the translation some part of the original image will move out of view, while some part of the resultant image is unknown as a result of corresponding pixels being not available in the original image, unknown regions are left blank (corresponding to pixel values of zero represented as black regions). It is possible to employ vertical and horizontal shifts concurrently.

Original Image | Horizontal Shift | Vertical Shift

Figure 2-2. Image Shift

Another transformation that can be applied to an image is rotation. In this case, the locations of pixels in an image are rotated around a certain origin by a determined rotation angle. Usually the center of the image is selected as origin, and the image is rotated respectively. Figure 2-3 shows the result of a rotation of 60 degrees in counter clockwise direction. As in the case of translation some parts of the original image may be lost, while some blank regions will appear in the resultant image. Note that rotation might require interpolation of pixel values, due to transformation characteristics.

Original Image

Rotated Image

Figure 2-3. Image Rotation

Arithmetic Image Processing

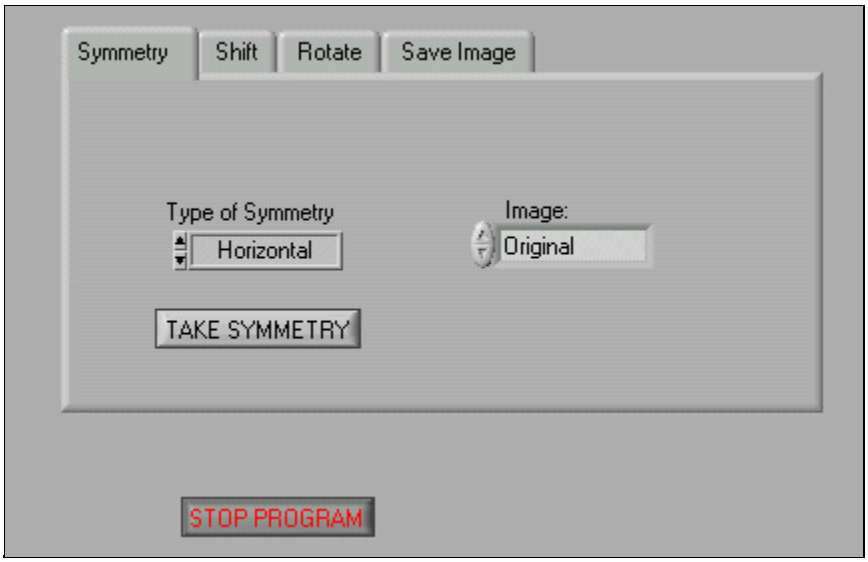
While basic image processing changes only the location of image pixels, i.e. takes the pixels of an image and moves them to another location, another way of manipulating an image is to carry out arithmetic operations on image pixels. It is possible to add/subtract an integer to/from pixel values or multiply or divide image pixels by a constant value. Figure 2-4 shows the effect of adding 50 to the values of all pixels of an image. Clearly the resultant image appears comparably brighter as pixel values are moved towards the white level. In the same figure the image resulting from an addition of 150 is displayed, due to the relatively large amount the image appears very bright, furthermore pixel values exceeding the representation range (255 for this 8-bit image) are truncated at the highest value resulting in dominant white regions and loss of image detail.

Original Image | 50 added to pixel values | 150 added to pixel values

Figure 2-4. Scalar Arithmetic

LabVIEW Demo 2.1: Basic Image Processing

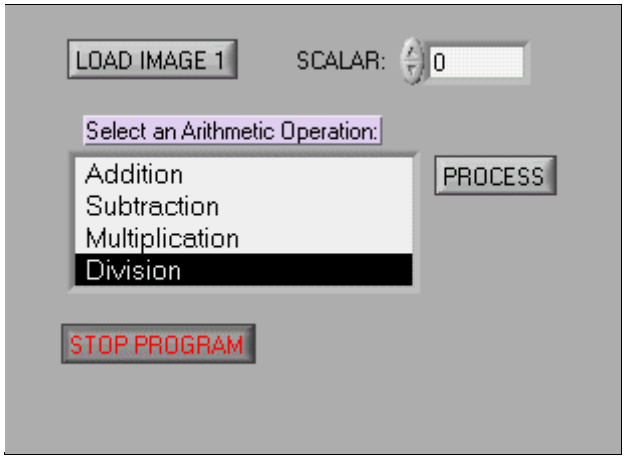
Launch the LabVIEW program entitled `basicprocess.vi` from the chapter 2 library.



A tab menu with “symmetry”, “shift”, “rotate” and “save image” options will appear. Run the program, and it will ask for an image file. In the symmetry tab you can choose between “horizontal”, “vertical”, “1st diagonal”, “2nd diagonal” and “central” symmetry. When you hit the “take symmetry” button the image with the corresponding symmetry will appear. You can choose to take the symmetry with respect to the original image or the already processed image so that you can conduct a sequence of different processes. In the shift tab you can define the horizontal and vertical shift amount in pixels and after hitting the “shift” button the processed image will appear. Again the source image can be chosen as either the original image or the already processed one. In the rotate tab the angle is entered in degrees and after hitting the “rotate” button the rotated image appears. It is possible to save the processed image from within the save image tab.

LabVIEW Demo 2.2: Scalar Arithmetic Image Processing

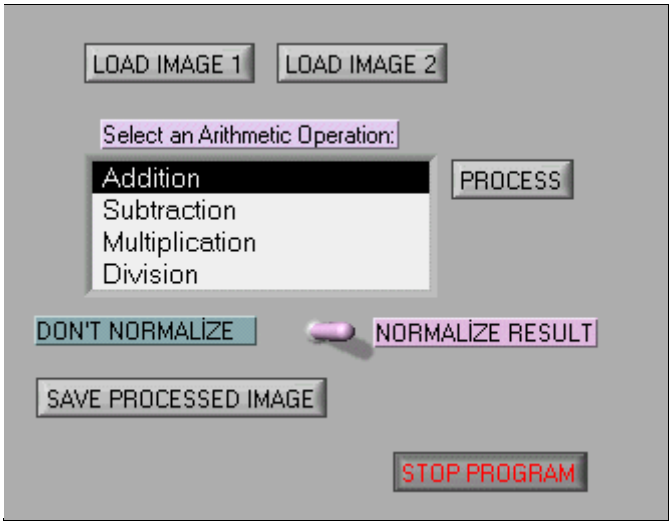
Launch the LabVIEW program entitled `scalararithmetic.vi` from the chapter 2 library. Run the program.



An image can be loaded using the “load image 1” button. Set the scalar value and after selecting the arithmetic operation as “addition”, “subtraction”, “multiplication” or “division” hit the “process” button and the processed image will appear. Observe that pixel values might surpass the representation range. The processed image can be saved using the “save processed image” button.

LabVIEW Demo 2.3: Arithmetic Image Processing

Launch the LabVIEW program entitled `arithmeticprocess.vi` from the chapter 2 library. Run the program.



Two images can be loaded using the “load image 1” and “load image 2” buttons respectively. After selecting the arithmetic operation as “addition”, “subtraction”, “multiplication” or “division” hit the “process” button and the processed image will appear. Observe what happens if you normalize the result and if you don’t. The processed image can be saved using the “save processed image” button.

Notes

Lab 3
Image Enhancement—Point Operations

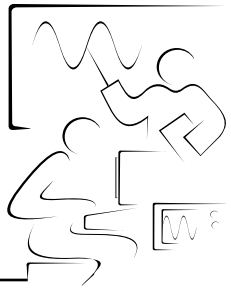


Image enhancement is the process of emphasizing certain image features, such as edges boundaries or contrast to improve visual appearance or content. Image enhancement is an important topic in image processing because of its usefulness in almost all image processing applications.

"Point operations" constitute a simple but efficient class of image enhancement techniques. These techniques are commonly referred to as point operations because the value of each pixel is recalculated independently of all other pixels according to a certain transformation. Point operations are also called contrast enhancement, contrast stretching or gray-scale transformation techniques. Point operations affect the appearance of an image when displayed and are often integrated into image digitizing and display applications.

Point operations commonly involve the adjustment of brightness and contrast in an image. These techniques are commonly applied to compensate for image acquisition drawbacks, such as insufficient lighting or contrast. A typical application is backlight compensation, which compensates for underexposed images in cases where an object of interest is backlit.

Linear Mapping (Linear Point Operations)

The overall brightness of a gray-scale image can be adjusted by adding a constant bias, *b*, to pixel values:

$$g(x,y) = f(x,y) + b \tag{3-1}$$

The overall brightness of the image is increased if $b > 0$, and decreased if $b < 0$. An example to increasing and decreasing the brightness of an image is shown in Figure 3-1.

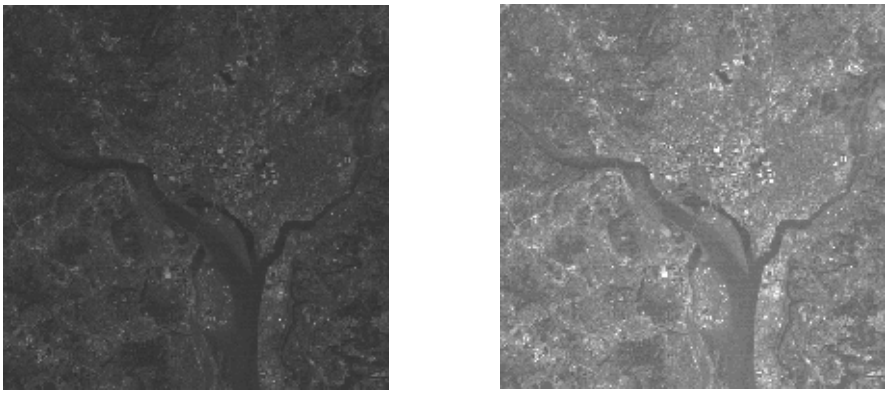
Original image | Brightness increased by 50 | Brightness decreased by 50

Figure 3-1. Image Brightness

The contrast of a gray-scale image can be adjusted by multiplying all pixel values by a constant gain, *a*:

$$g(x,y) = af(x,y) \tag{3-2}$$

The overall contrast of the image is increased if $a > 1$, and decreased if $a < 1$. An example to increasing and decreasing the contrast of an image is shown in Figure 3-2.



Original satellite image | Result of multiplying pixel value by 2

Figure 3-2. Image Contrast

Brightness and contrast equations can be combined to give a single expression for brightness and contrast adjustment:

$$g(x,y) = af(x,y) + b \tag{3-3}$$

Sometimes rather than specifying a certain gain or bias, it is preferred to define a mapping between pixel values, in which case a particular range of grey levels, $[f_1, f_2]$, is mapped onto a new range, $[g_1, g_2]$. This mapping can be accomplished by the transformation:

$$g(x,y) = g_1 + \left(\frac{g_2 - g_1}{f_2 - f_1}\right)[f(x,y) - f_1] \tag{3-4}$$

The graphical representation of brightness and contrast modification through bias and gain adjustment is shown in Figure 3-3 (a) by plotting the output grey level versus input grey level. The graphical representation confirms that bias and gain adjustment is equivalent to a linear mapping of pixel values. As the dynamic pixel range of the output image is changed and does not include low pixel values due to the bias, in some cases it is preferred to only stretch the contrast as shown in Figure 3-3 (b). In contrast stretching the slope is chosen greater than unity in the region of stretch, while the slope might be less than unity (contrast reduction) in regions that are of no concern. Contrast stretching is particularly used in images providing low contrast due to poor or nonuniform lighting conditions or due to the nonlinearity or small dynamic range of the imaging sensor.

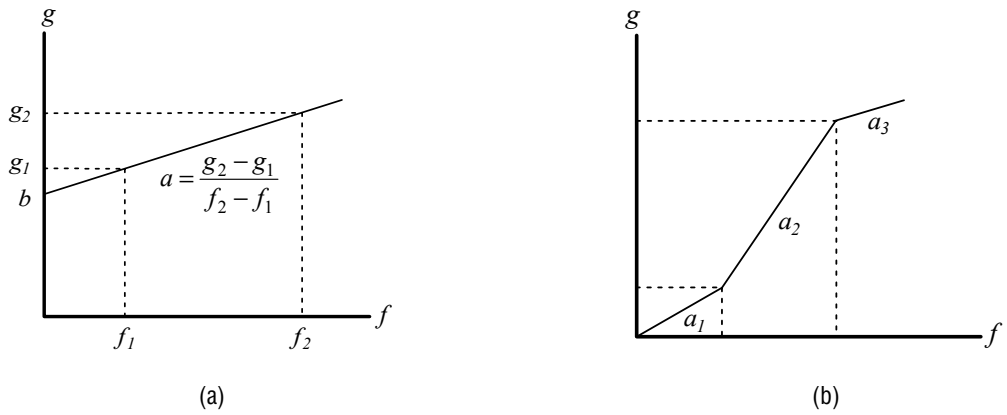


Figure 3-3. Graphical Representation of Linear Mapping
(a) Gain-bias representation (b) Contrast stretching

Clipping

Clipping is a special case of contrast stretching where a certain range of grey levels $[f_{\min}, f_{\max}]$, is mapped to the entire dynamic pixel range ($[0,255]$ for 8-bit images). The pixels below $f_{\min}$ are clipped to the lowest pixel value, and pixels above $f_{\max}$ are clipped to the highest pixel value. The graphical representation of clipping is shown in Figure 3-4. Clipping is particularly useful if the interest is in a particular pixel range and pixel values outside

this range are not important. As the output makes use of the full dynamic pixel range an optimal linear contrast stretching for the particular range is obtained.

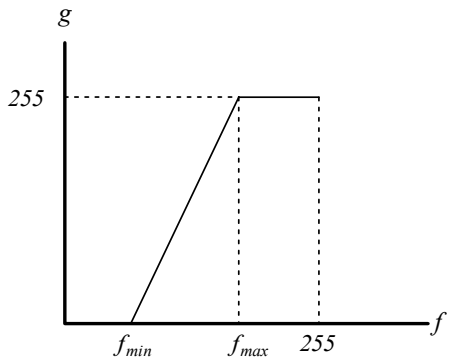
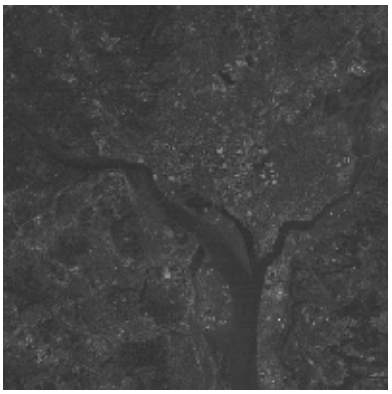


Figure 3-4. Graphical Representation of Clipping

Figure 3-5 displays an example, in which the pixels in the range 50–100 in the original image are linearly stretched to the full dynamic pixel range by clipping pixels below and above the range of interest. The visual appearance is clearly superior compared to Figure 3-2 that shows contrast enhancement by gain adjustment only. Clipping is particularly useful for noise reduction when the input signal is known to lie within a certain part of the dynamic range.



Original satellite image

Linear contrast stretching of pixels in the range of 50–100 by clipping

Figure 3-5. Contrast enhancement by clipping

Thresholding

Thresholding is a special case of clipping where $f_{min} = f_{max}$ and the resultant image becomes binary. In the case of 8-bit images, grey levels below and including the threshold (f_t) are mapped onto 0, while grey levels greater than the threshold value are mapped onto 255. The graphical representation of thresholding is shown in Figure 3-6. Thresholding produces a binary image and can for instance be used to locate objects in an image with respect to a plane background.

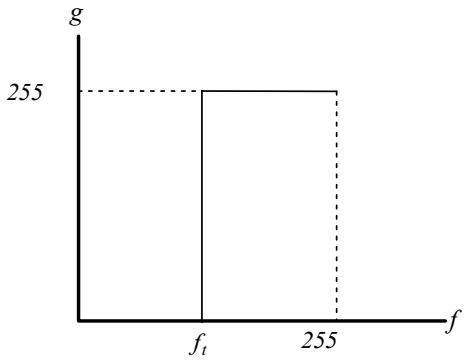
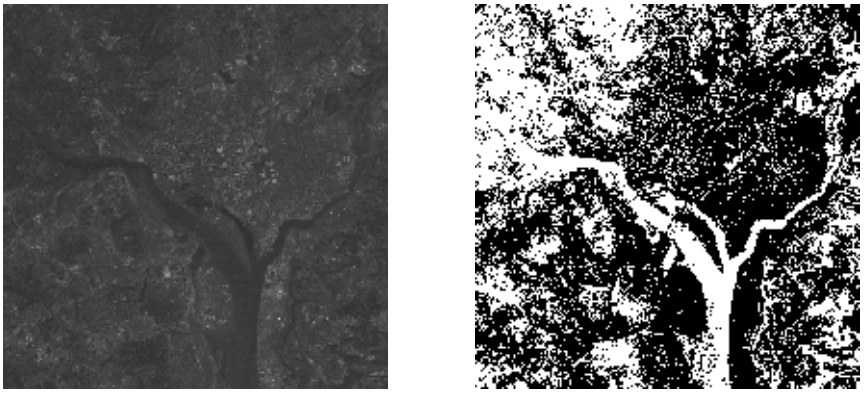


Figure 3-6. Graphical Representation of Thresholding

An example to image thresholding is given in Figure 3-7.



Original satellite image

Threshold set to 60

Figure 3-7. Image Thresholding

Negation

Negation or inversion of an image is accomplished by applying a negative gain factor to the image. A negative image is typically obtained by subtracting each pixel from the maximum pixel value. Thus for an 8-bit image the negative image can be achieved by reverse scaling of the grey levels, according to the transformation

$$g(x,y) = 255 - f(x,y) \tag{3-5}$$

The graphical representation of negation is shown in Figure 3-8.

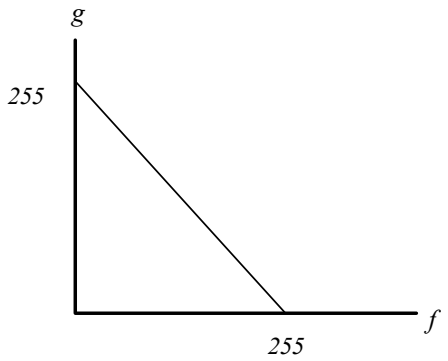


Figure 3-8. Graphical Representation of Negation

Negative images are particularly useful in the display of medical images and producing negative prints of images. Figure 3-9 shows an example to negation.

Original image Negative image

Figure 3-9. Image Negation

Non-linear Mapping

It is also possible to use non-linear functions for the mapping of input grey levels onto output levels. Non-linear mapping has the advantage that the gain applied to input grey levels can change for input levels. If a non-linear mapping is utilized it is important that care is taken so that any input level is mapped onto one and one output level, thus the mapping function needs to be one-to-one or many-to-one. An inappropriate mapping function, which maps some input levels to more than one output level, is shown in Figure 3-10.

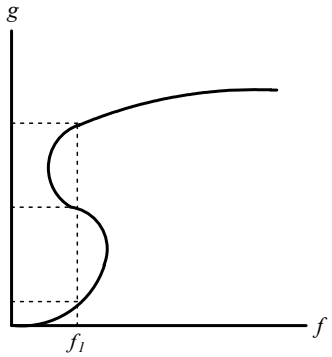


Figure 3-10. An inappropriate mapping function: to which level does f_1 map?

A commonly used non-linear mapping function is logarithmic mapping. The graphical representation of logarithmic mapping is shown in Figure 3-11. Logarithmic mapping increases contrast for low grey levels, while contrast is reduced for high grey levels. Logarithmic mapping is useful if it is desired to enhance detail in the darker regions of the image, at the expense of detail in the brighter regions. If only a particular range of grey levels is of interest it is possible to combine clipping with non-linear mapping.

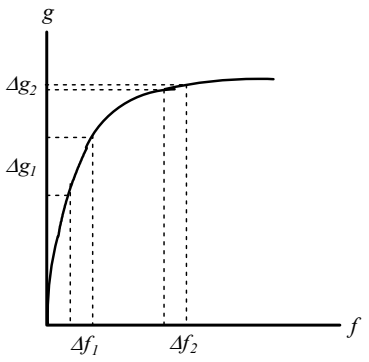
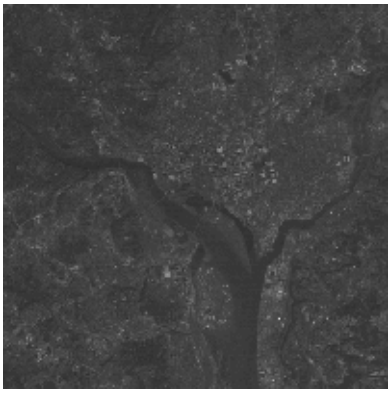


Figure 3-11. Graphical Representation of Logarithmic Mapping

Figure 3-12 shows an example for contrast enhancement by logarithmic mapping of a particular grey level range. Clearly darker parts of the original image are enhanced while contrast is reduced in the brighter parts of the original image.



Original satellite image

Logarithmic contrast stretching of pixels in the range of 50–100

Figure 3-12. Contrast enhancement by logarithmic mapping

An exponential mapping of grey levels will produce the opposite result of logarithmic mapping: contrast in brighter regions is increased at the expense of contrast in the darker regions. The graphical representation of exponential mapping is shown in Figure 3-13.

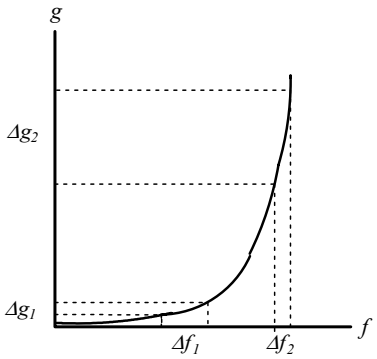
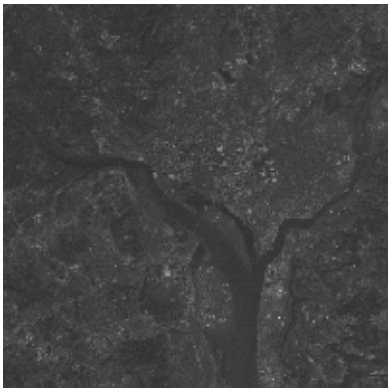


Figure 3-13. Graphical Representation of Exponential Mapping

Figure 3-14 shows an example for contrast enhancement by exponential mapping of a particular grey level range. Clearly brighter parts of the original image are enhanced while contrast is reduced in the darker parts of the original image.



Original satellite image

Exponential contrast stretching of pixels in the range of 50–100

Figure 3-14. Contrast enhancement by exponential mapping

It is possible to employ different non-linear mapping functions, such as square, square-root, or power functions, depending on the desired contrast enhancement attributes. As a large number of image enhancement techniques are available, the procedure is rather empirical and may require interactive procedures to obtain satisfactory results.

Intensity Level Slicing

Intensity level slicing can be used to segment certain grey level regions from the rest of the image. This technique is particularly useful if the region of interest in an image is within a particular grey level range. It is possible to utilize several implementations of intensity level slicing:

Segment a particular range keeping original pixel values and set the rest to zero:

$$g(x,y)=\begin{cases} f(x,y) & , f_{\min} \le f(x,y) \le f_{\max} \\ 0 & , \quad \text{otherwise} \end{cases} \tag{3-6}$$

Segment a particular range replacing pixel values by the highest value (255 for 8-bit images) and set the rest to zero:

$$g(x,y)=\begin{cases} 255 & , f_{\min} \le f(x,y) \le f_{\max} \\ 0 & , \quad \text{otherwise} \end{cases} \tag{3-7}$$

Segment a particular range replacing pixel values by the highest value (255 for 8-bit images) and keep the rest at the original pixel values (keep background):

$$g(x,y)=\begin{cases}255 & ,f_{\min}\le f(x,y)\le f_{\max}\\ f(x,y) & ,\quad \text{otherwise}\end{cases} \tag{3-8}$$

Intensity level slicing is sometimes managed as a thresholding operation with multiple boundaries. The lower threshold and upper threshold values are used to define the corresponding grey level range, and the option to keep pixel values within this range or replace them with a particular value determines the operation.

Efficient Implementation of Mapping

If it is desired to accomplish a point operation that involves some arithmetic operations, it is usually inefficient to apply all pixels of the entire image to the arithmetic due to computational load. For instance, in the case of a linear mapping in the form of $g(x,y)=af(x,y)+b$, for an image of dimensions $N\times N$, an algorithm that directly processes each pixel of the image requires N^2 multiplications and N^2 additions. The computational load is comparatively high, particularly for large images.

In the case of 8-bit images, as there are only 256 possible input grey levels the number of possible mappings is equal to the number of grey levels and hence 256. It is possible to perform the mapping calculation for every possible input grey level, in this case 256 times, and store the corresponding values in a look-up-table (LUT). For instance, in the case of linear mapping for all possible input levels the values in the table are computed as $table[i]=ai+b$. Then the mapping of the image is simply accomplished by replacing each pixel of the input image with the value at the corresponding look-up-table location: $g(x,y)=table[f(x,y)]$. The computational load is reduced significantly particularly for large image dimensions.

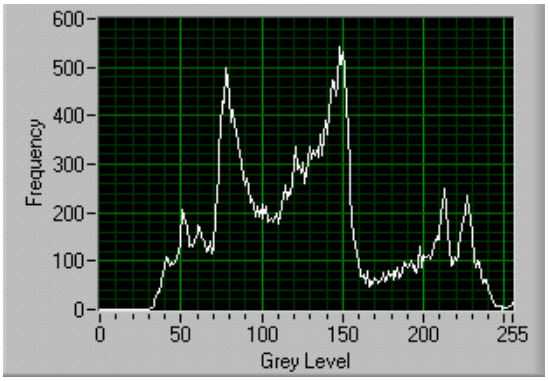
Instead of utilizing an arithmetic expression to accomplish the mapping procedure, it is also possible to define the corresponding mapping function in terms of entries of a look-up-table. The LUT entries can be defined explicitly upfront, and the image enhancement can be accomplished by performing the corresponding mapping.

Image Histograms

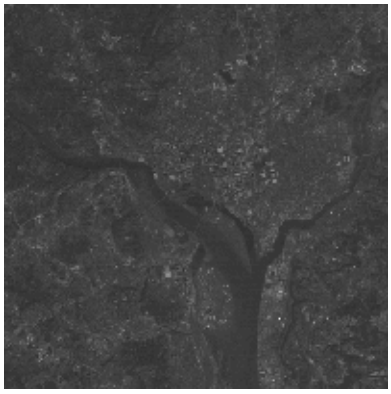
The histogram of an image represents the frequency of occurrence of grey levels in the image. The histogram of an 8-bit image will correspond to a table with 256 entries, or ‘bins’, indexed from 0 to 255, that will record the numbed of occurrences of each level in the image. An image histogram can simply be computed by creating a table of size equivalent to the number of grey levels of the image, with table entries initialized to zero, each entry is incremented for a pixel in the image of that grey level value.

The histogram of an image provides useful information about the importance and frequency of different grey levels in the image, and can particularly useful in selecting a brightness or contrast modification procedure. Figure 3-15 shows the histograms of the car and satellite images. While pixels in the car image comprise almost the entire dynamic pixel range and vary from 30 to 240. It is clearly seen that pixels in the satellite image are mainly located in the range of 50–100, thus an enhancement process will be most successful if only this range is taken into account.

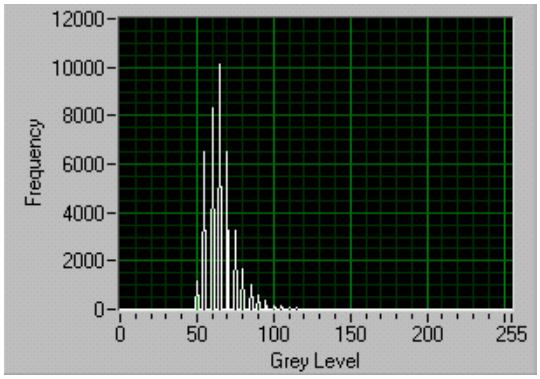
Original car image



Corresponding histogram



Original satellite image



Corresponding histogram

Figure 3-15. Image Histograms

Point operations usually affect the histogram of an image in a predictable fashion. For instance in the case of brightness modification, the addition of a constant bias to pixel values will shift the histogram along the grey level axis by a corresponding distance without changing the histogram shape except may be at the boundaries as some pixels might be clipped due to the limited dynamic range of pixel values. In the case of contrast modification by gain adjustment the histogram will be spread evenly if the multiplication factor is greater than one increasing the spacing between occupied bins as now non-occupied bins will be introduced, or compressed if the multiplication factor is less than one which might result in the merge of bins. A non-linear mapping of grey levels will usually stretch some parts of the histogram, whist compressing other parts.

Histogram Equalization

As the histogram of an image gives the frequency distribution of all grey levels, it should be possible to define a non-linear mapping of grey-levels that produces an optimal improvement in contrast making use of the histogram information. By allocating more grey levels in the parts of the histogram with most of the pixels and fewer grey levels for parts of the histogram with a smaller amount of pixels, an optimal contrast will be achieved as contrast is varied according to the number of pixels within the image itself.

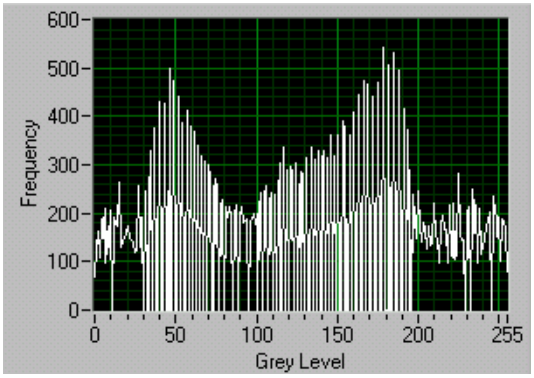
In order to increase contrast for most frequently occurring grey levels a mapping function with a gain greater than unity is required for this regions, while a mapping function with gain less than unity is required for less populated parts of the grey level range to reduce the contrast for this regions. The process for histogram equalization of an 8-bit image can be summarized into the following steps:

1. Define a scaling factor d = 255/total number of pixels
2. Calculate the histogram of the image
3. The first grey level mapping is computed to be $c[0] = d \times histogram[0]$
4. For all remaining grey levels compute $c[i] = c[i-1] + d \times histogram[i]$
5. For all pixels in the image perform the mapping: $g(x,y) = c[f(x,y)]$

This procedure ensures that the distance between two mapping levels is proportional to the number of occurrences of the levels. Less occurring grey levels will be mapped closer to each other, while more frequent levels will be mapped further apart. Note that the mapping is one-to-one, i.e. all pixels at a certain grey level are mapped to the same level at the output. As bins are not split, histogram equalization will not produce a totally flat (uniform) histogram at the output, although this is the motivation of the procedure.

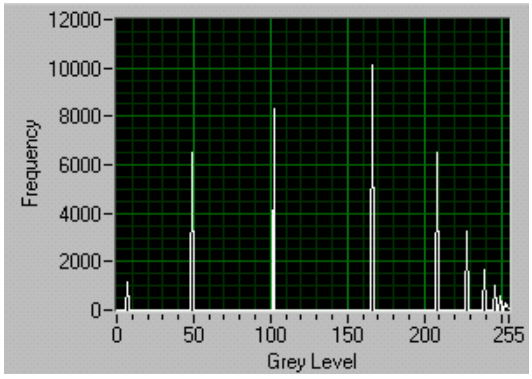
Figure 3-16 shows the result for histogram equalization for the car and satellite images and the corresponding histograms.

Histogram equalized car image



Corresponding histogram

Histogram equalized satellite image

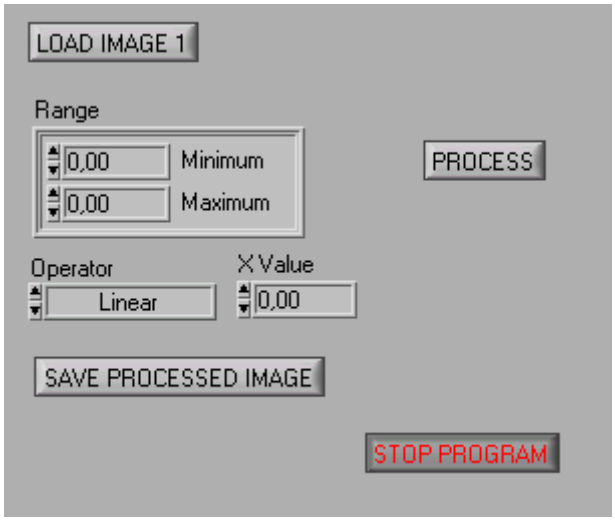


Corresponding histogram

Figure 3-16. Histogram Equalization

LabVIEW Demo 3.1: Linear and Non-linear Mapping

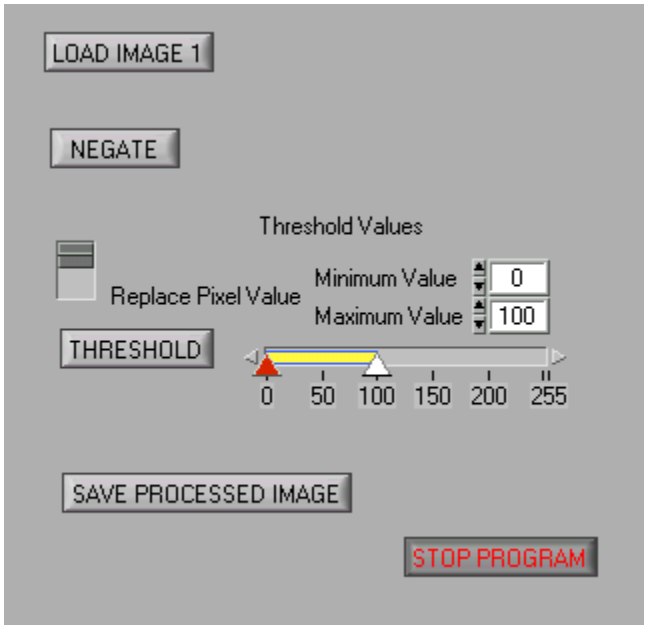
Launch the LabVIEW program entitled `mapping.vi` from the chapter 3 library. Run the program.



Press the “load image 1” button to select, load and display an image. Set the input grey level range that is to be mapped using the minimum and maximum values in range (set minimum to 0 and maximum to 255 to map the entire range). Select the mapping operator as linear, logarithmic, exponential, square, square-root, power *X*, or power 1/*X* to define the mapping function. Note that it is required to set the *X* value if power *X* or power 1/*X* mapping is desired. Hit the process button and observe the resultant images. Try various mapping functions for separate images and different ranges and in each case observe the results.

LabVIEW Demo 3.2: Negation and Thresholding

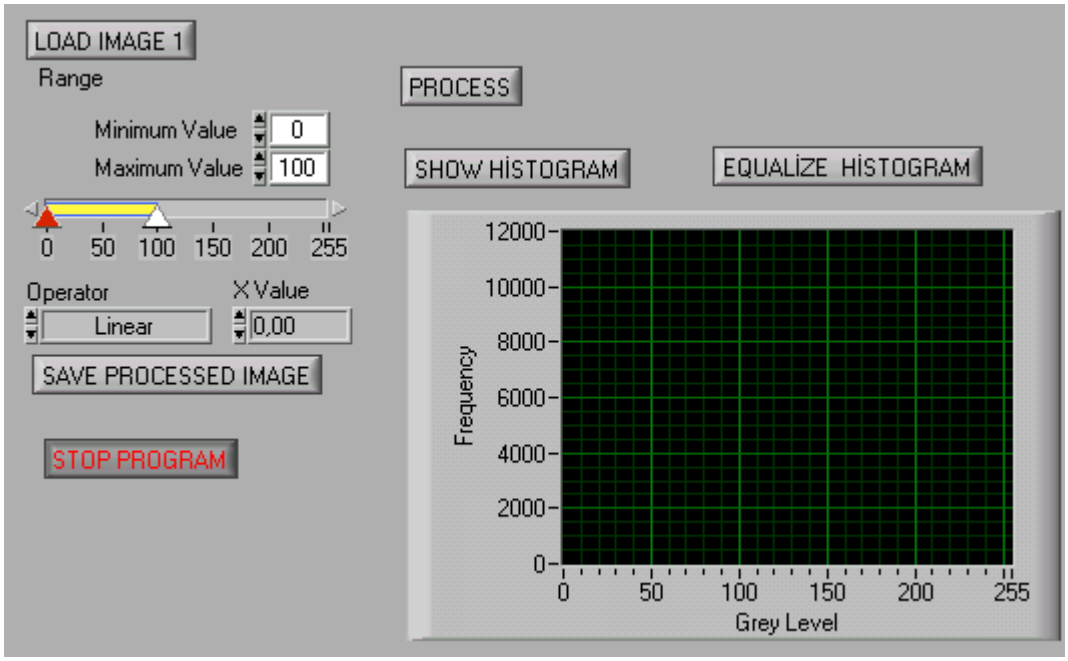
Launch the LabVIEW program entitled `negate-threshold.vi` from the chapter 3 library. Run the program.



Press the “load image 1” button to select, load and display an image. Hit the “negate” button to display the negative image. For thresholding an image select the threshold range by setting the minimum and maximum values or using corresponding sliders. It is possible to keep original pixel values within the range or replace pixel values by 255 (white) using the slide switch. Try various thresholds for separate images and observe results for keeping and replacing pixel values.

LabVIEW Demo 3.3: Image Histograms

Launch the LabVIEW program entitled `histogram.vi` from the chapter 3 library. Run the program.

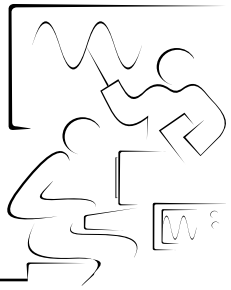


Press the “load image 1” button to select, load and display an image. Hit the “show histogram” button to display the histogram of the image. Hit the “equalize histogram” button to observe the histogram equalized image, hitting the “show histogram” button will now display the histogram of the histogram equalized result. Try various mapping functions for separate images and observe their effect on the image histogram.

Notes

Notes

Lab 4
Image Enhancement—Spatial Operations



"Spatial operations" constitute an efficient class of image enhancement techniques, in which the new value of a pixel is calculated not only from the original value of the pixel but also from pixels in the local neighborhood. As a single pixel considered independently of all other pixels can only give information about the intensity (and possibly color) of a single point in the image, but does not give information about the spatial variation of these properties in the image, point operations provide only restricted capabilities. In order to investigate or modify spatial variations in an image, it is required to perform operations over a certain image area. Thus a pixels new value has to be computed from the pixels old value and the values of pixels in the neighborhood. For that reason, spatial operations are also called neighborhood operations.

Convolution

Convolution is one of the basic linear neighborhood operations that can be used for image processing. Convolution basically computes the new value of a pixel as a weighted sum of pixel values in a certain neighborhood surrounding the pixel. The neighborhood commonly includes the pixel under consideration, and although not necessarily it is customarily symmetric about that pixel. A neighborhood that is centered on a pixel will have odd dimensions, and symmetric neighborhoods can be represented as 3×3, 5×5, 7×7, etc.

The convolution sum is computed as a weighted sum of pixels in the predefined neighborhood. The convolution is commonly computed by weighting neighborhood grey levels by coefficients that are defined in the form of a matrix called **convolution kernel**. In effect, the dimension of the kernel defines the neighborhood dimensions in which the convolution is calculated. Due to the high computational load of convolution it is usually preferred to utilize a convolution kernel that is relatively small compared to image dimensions.

For a kernel *k,* of width *w* and height *h* (*w* and *h* being both odd, and *m*=(*w*-1)/2 and *n*=(*h*-1)/2 being the half-width and half-height respectively) and the input image denoted by *f*, the convolution process is defined as

$$g(x,y)=\sum_{j=-n}^{n}\sum_{i=-m}^{m}k(i,j)f(x-i,y-j)=k*f \tag{4-1}$$

Note that for each kernel coefficient is in turn multiplied by a pixel value from the neighborhood that is symmetric with respect to the centre of the kernel (or the pixel for which the convolution is being computed). For instance the top-left corner of the kernel is multiplied by the pixel value at the bottom-right corner of the neighborhood.

Figure 4-1 shows an example 3 × 3 kernel and a corresponding image neighborhood. Assume that the convolution sum for the centre pixel of the image neighborhood is to be computed:

$$\begin{aligned}g(x,y)=\;&h(-1,-1)f(x+1,y+1)+h(0,-1)f(x,y+1)+h(1,-1)f(x-1,y+1)\\&+h(-1,0)f(x+1,y)+h(0,0)f(x,y)+h(1,0)f(x-1,y)\\&+h(-1,1)f(x+1,y-1)+h(0,1)f(x,y-1)+h(1,1)f(x-1,y-1)\end{aligned} \tag{4-2}$$

For the kernel and pixel values given in Figure 4-1 the result of the convolution sum for the centre pixel is $g(x,y) = (-1 \times 52) + (-1 \times 54) + (-1 \times 58) + (-2 \times 55) + (10 \times 60) + (-2 \times 69) + (-1 \times 63) + (-1 \times 68) + (-1 \times 72) = -15$.

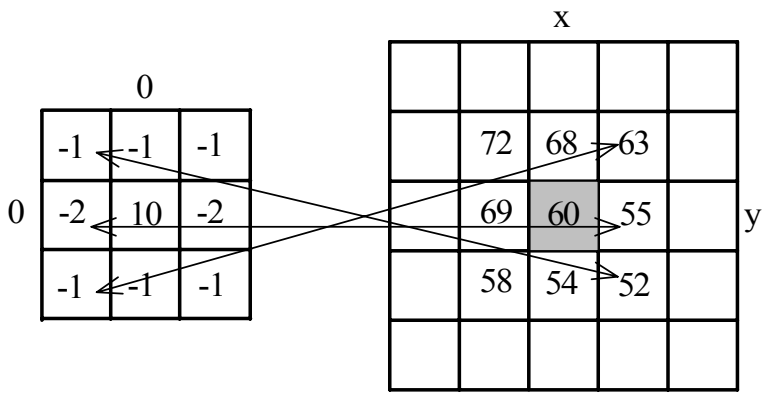


Figure 4-1. A 3 × 3 convolution kernel and the corresponding image neighborhood

Due to the convolution sum computation the output image values might exceed the dynamic representation range of the input pixels, or might be negative if the kernel includes negative coefficients. Therefore it is important to take care of the output image representation it might be required to use a signed data type, normalize the result, add a constant bias to the output or utilize clipping of out of range values.

In the convolution of an image special consideration has to be given to the borders of the image, as due to definition of the convolution kernel the convolution sum will go beyond the image borders for pixels nearby the borders. Different approaches can be employed to overcome this problem:

- Don't process borders: ignore pixels on the borders of the image for which the convolution can not be computed. If the dimensions of the output image are the same as the dimensions of the input image, pixels for which the convolution can not be computed will normally be initialized to zero value, which will result in a black border in the output image. This might irritate the viewer is the output image is displayed or cause possible problems or performance degradations for further processing.
- Copy input image pixels: copy input image pixels to the output image for locations for which the convolution can not be computed. As a result the output image will include a number of unprocessed pixels. Although this solution might provide a suitable approach for some cases depending on the convolution kernel, it is not possible to ensure an appropriate result for all cases.
- Truncate image: the output image can be truncated removing the pixels for which the convolution can not be computed. The result is an output that is smaller than the input image. This approach might be utilized if only the display of the result is in order, however the size reduction might be a problem if further processing is desired.
- Truncate kernel: borders of the image can be treated as a special case and truncated kernels can be utilized for the borders. Thus instead of utilizing the symmetric kernel which will exceed image dimensions, particular asymmetric kernels possibly obtained from truncation of the original kernel can be utilized at the image borders. This solution provides a similar processing to pixels on the border of the input image, however increases the complexity of convolution considerably.
- Use reflected indexing: in cases where the convolution pixel coordinate exceeds the image dimensions, it is reflected back into the image as shown in Figure 4-2. This process is equivalent to mirroring the image at its borders to increase image dimensions for successful convolution computation, or equivalently reflection of the kernel whenever the corresponding pixel is out of the image range.
- Use circular indexing: by assuming that the image repeats itself endlessly in all directions it is possible to use circular indexing as shown in Figure 4-2 to overcome border problems. This assumption is based on frequency domain properties of images.

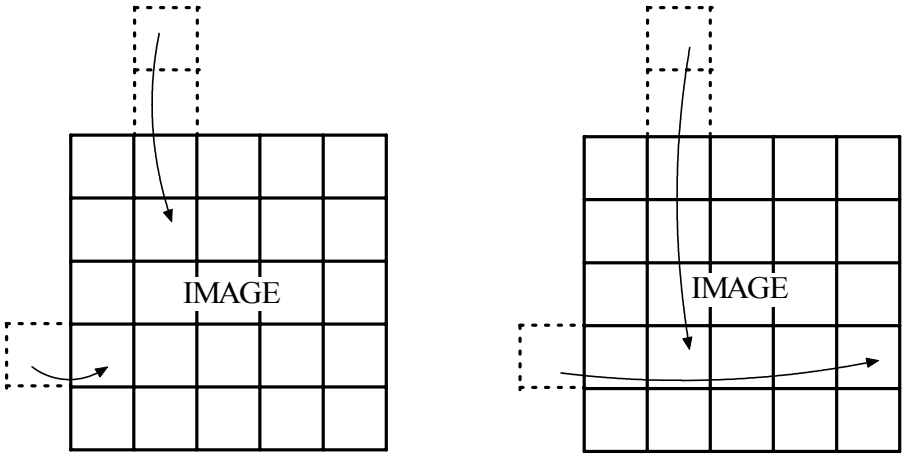


Figure 4-2. Indexing of pixel coordinates falling out of the image range during convolution

As will be shown later on, convolution is commonly used for the **linear filtering** of an image. Figure 4-3 shows an example to the output image if a high-pass kernel is employed in the convolution: sudden changes in grey-levels such as variations that occur at the edges of objects are preserved, while smooth changes are removed from the image.

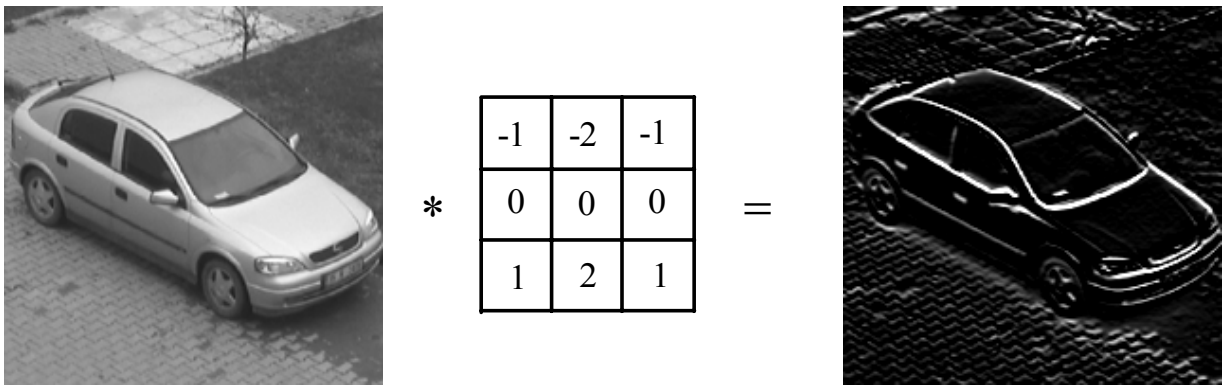


Figure 4-3. Convolution with a high-pass kernel

Convolution is computationally a very expensive procedure. Evaluation of a direct two-dimensional convolution for an $m \times m$ kernel requires m^2 multiplications and m^2 additions per image pixel. Therefore software or hardware aids are commonly utilized to speed up the computation process.

The computational load of convolution is reduced considerably if the utilized kernel is separable, that is the $m \times m$ kernel can be represented as a vector product of two orthogonal one-dimensional kernels each of width m. Then one of the one-dimensional kernels is applied along the columns of the image to produce a temporary result, and the other kernel is applied along the rows of the temporary image to produce the final result. Thus the two-dimensional convolution process is reduced to two one-dimensional convolution computations. The computational load is reduced comparably, and the saving is about proportional with kernel size. A kernel is separable if it can be written in the form of a vector product of two orthogonal one-dimensional kernels: $k = k_x \times k_y$.The process of using a separable kernel is presented in Figure 4-4. The two-dimensional kernel utilized in the example of Figure 4-3 is separated into two orthogonal one-dimensional kernels.

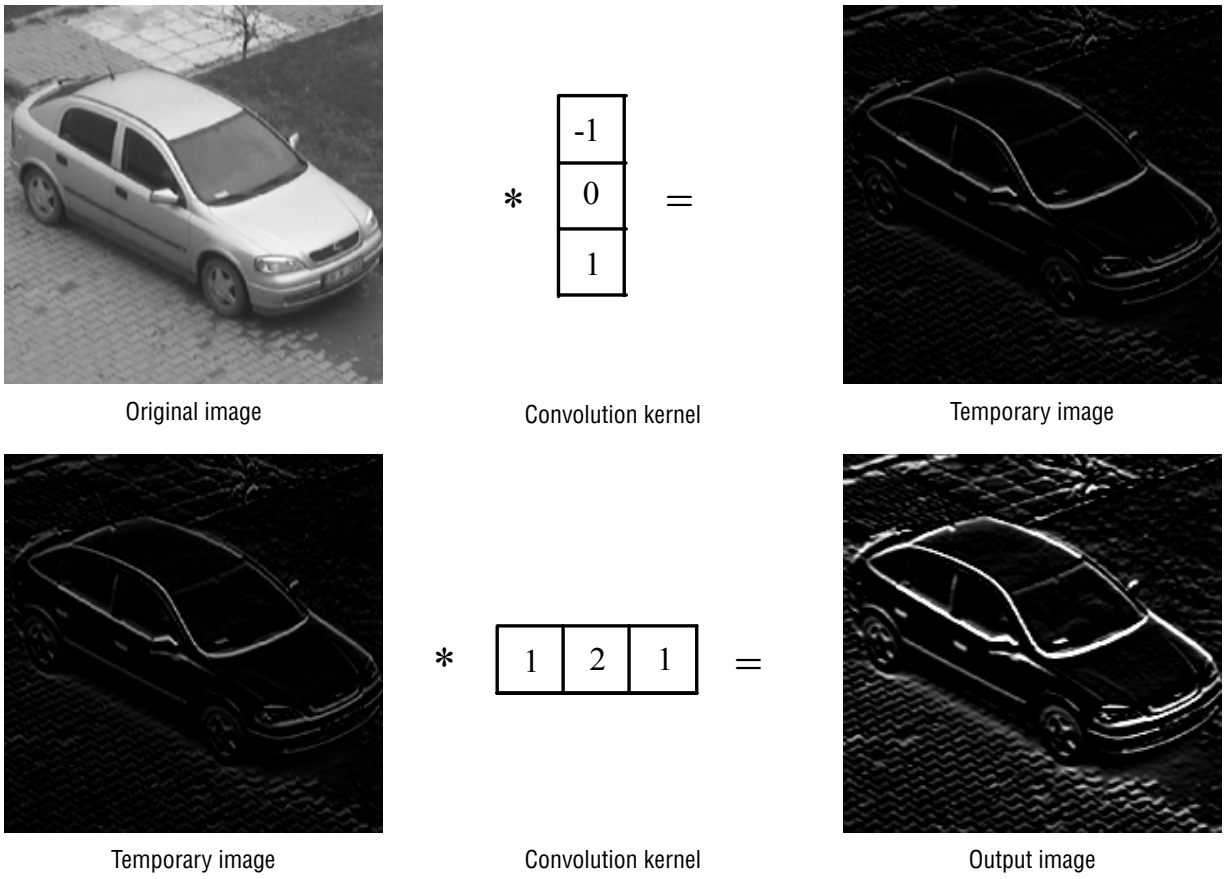


Figure 4-4. Two-dimensional convolution by 2 one-dimensional convolutions

Correlation

For a kernel *k,* of width *w* and height *h* (*w* and *h* being both odd, and *m*=(*w*-1)/2 and *n*=(*h*-1)/2 being the half-width and half-height respectively) and the input image denoted by *f*, the correlation process is defined as

$$g(x,y)=\sum_{j=-n}^{n}\sum_{i=-m}^{m}k(i,j)f(x+i,y+j) \tag{4-3}$$

In correlation, putting the kernel centre at the pixel location for which the correlation measure is being computed, each kernel coefficient is paired with the image pixel that lies directly beneath it. Kernel coefficients and image pixel values within the neighborhood defined by the kernel dimensions are multiplied ad the sum gives the corresponding correlation measure.

Correlation is commonly used to determine similarities between images or parts of images. As the correlation measure of Equation 4-3 will produce larger values for brighter image parts, which can harden the similarity identification process, the correlation measure is commonly normalized. A simple approach for normalization is to divide the total correlation measure by the sum of grey levels in the image neighborhood for which the correlation is computed:

$$g'(x,y)=\frac{\sum_{j=-n}^{n}\sum_{i=-m}^{m}k(i,j)f(x+i,y+j)}{\sum_{j=-n}^{n}\sum_{i=-m}^{m}f(x+i,y+j)} \tag{4-4}$$

Correlation is commonly used to measure similarities between images. It is possible to use correlation to determine the global displacement between two images of the same scene, in which case the input image will comprise the first image and the correlation kernel will comprise the second image. It is also possible to find certain parts inside an image given a template for the searched pattern, in which case the image is correlated with the template which makes up the correlation kernel. As the aim of correlation is to find similarities between an image and a template image, commonly the term template (or template image) is used for the correlation process, whereas the term kernel is saved for the convolution process.

Figure 4-5 shows correlation results for two images of which one is the original image and the other is the same image displaced horizontally by 20 pixels. The displayed image is used as correlation image, and the original image as template to determine the global displacement of the image by

utilization of the correlation measure. Note that the template has the same dimensions as the image and therefore the computation time is considerably increased, compared to the utilization of a relatively small template.

It is possible to display the correlation measure in the form of an image as well as a 3-D graph. If correlation results are displayed as an image, bright parts will correspond to good matches while dark regions will indicate poor matches. Therefore the best match is at the location at which the highest pixel value is obtained within the correlation image. If correlation results are considered in the form of 3-D data, the best match will correspond to the peak location of the correlation surface. For the example given in Figure 4-5 for instance best match is obtained for the location (120, 100) which defines the best match location of the template to the image with respect to the top-left corner of the image. In this case it is desired to find the global translation of the images, therefore it is required to consider the displacement with respect to the centre of the image: due to image dimensions of 200×200, the centre of the image corresponds to the position (100, 100). Therefore the displacement between images is correctly obtained to be 20 pixels in the horizontal direction from the correlation measure.

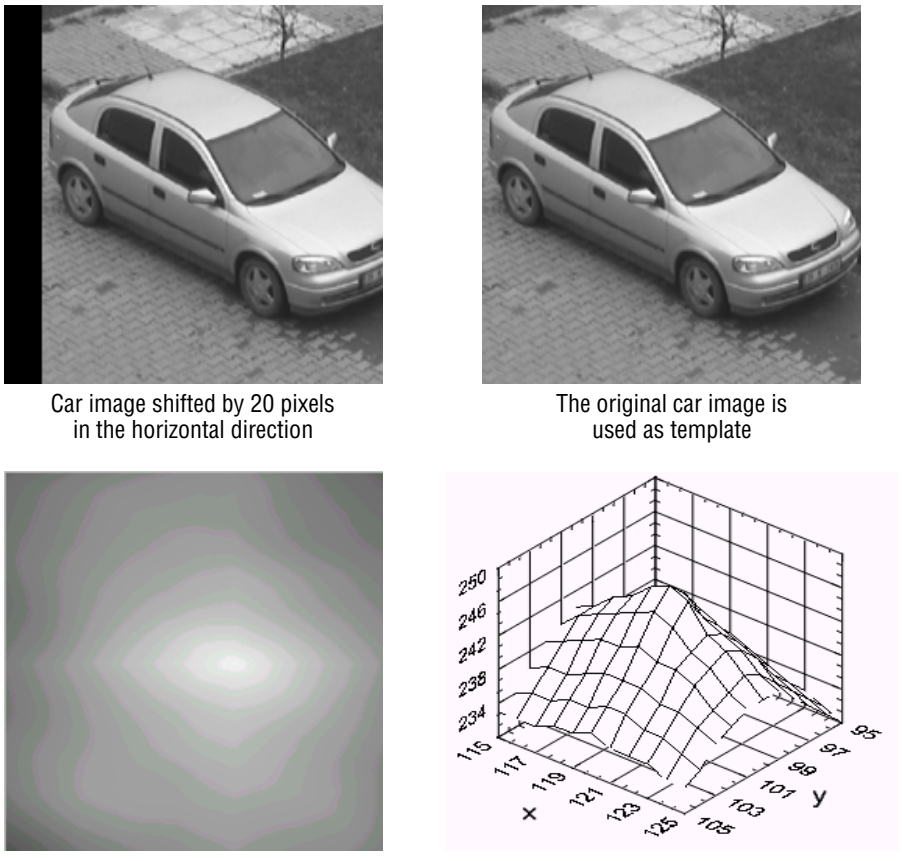


Figure 4-5. Correlation of a displaced image with the original version

Figure 4-6 shows the resulting correlation surface that is obtained from the correlation of a noisy image and a net template. Despite the large amount of noise included in the image the correlation measure is highest when the template is directly above the Q letter in the noise image. As it is aimed to find a certain template pattern within the image the coordinates of the peak in the correlation surface will define the best suited location for the centre of the template with respect to the top-left corner of the image. For the example given in Figure 4-6 for instance, the best match is obtained for the location (149, 99) which corresponds to about the centre of the Q letter in the image.

Noisy image Template Image Correlation Surface

Figure 4-6. Correlation of a noisy image with a template

Linear Filtering

An image can be described in terms of spatial frequencies: while smooth image features correspond to low spatial frequencies, sudden changes in the pixel values of an image, such as it is the case for object boundaries and edges, will result in high spatial frequency content. Hence it is possible to refer to spatial operations that have an effect on the spatial properties of images as **filtering**. Consequently a low pass filter will smoothen or blur the image, while a high pass filter will preserve sudden grey level variations but suppress regular image parts. Convolution can be used to carry out the linear filtering process.

Low Pass Filtering

A convolution using a kernel with all positive coefficients will result in a low pass filtering process. For all positive coefficients a weighting of pixel values within the neighborhood is carried out, resulting in a smoothened image. To avoid an overall increase in image brightness kernel coefficients can be normalized so that they sum to unity.

For example the following 3×3 and 5×5 uniform kernels shown in Figure 4-7 are normalized so that kernel coefficients sum to unity. These kernels result in unit weighting of all pixels within the neighborhood and the result is normalized through division by the total number of pixels within the neighborhood. The convolution process is therefore equivalent to a spatial averaging. Each pixel is replaced by the average or mean of its corresponding neighborhood. These kernels are therefore referred as **mean filters**.

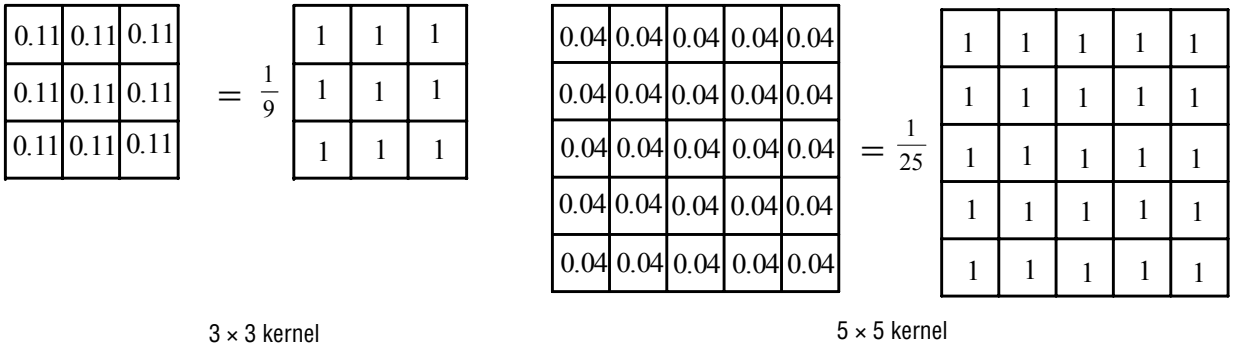


Figure 4-7. Mean filter kernels

Figure 4-8 displays images the resulting from 3×3, 5×5, and 7×7 mean filtering. It is clearly seen that image blur is intensified for increased kernel size. The effect of a large smoothing kernel can also be achieved by repeatedly applying the image to small smoothing kernels.

Figure 4-8. Effect of mean filtering for various kernel sizes

While low-pass filtering can be carried out using mean filters, i.e. uniform kernels that have all equal coefficients, it is also possible to achieve low-pass filtering with non-uniform kernels that have all positive but not necessarily equal coefficients. Some commonly used smoothing kernels are given in Figure 4-9. Note that normalization through division of the convolution sum by the sum of kernel coefficients is carried out inherently.

3 × 3 kernels:

0	1	0
1	0	1
0	1	0

0	1	0
1	1	1
0	1	0

0	2	0
2	1	2
0	2	0

0	4	0
4	1	4
0	4	0

1	1	1
1	0	1
1	1	1

1	1	1
1	1	1
1	1	1

2	2	2
2	1	2
2	2	2

4	4	4
4	1	4
4	4	4

5 × 5 kernels:

1	1	1	1	1
1	1	1	1	1
1	1	0	1	1
1	1	1	1	1
1	1	1	1	1

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

7 × 7 kernels:

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

Figure 4-9. Some common smoothing kernels

A commonly used class of low pass filters is the **Gaussian filter**, for which kernel coefficients are obtained from samples of a two-dimensional Gaussian function, which is defined as

$$k(x, y) = e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{4-5}$$

The two-dimensional Gaussian function is plotted in Figure 4-10. By definition Gaussian function values are greater about the centre location and function values decrease further away from the centre location. Thus, if kernel coefficients are constructed from samples of the Gaussian function, more weight will be given to centre pixels and the role of the pixels will

diminish with increasing distance to the centre. This is a meaningful feature, as due to spatial correlation it is expected that pixels close to the centre pixel will demonstrate a high similarity, while the correlation is normally expected to diminish with distance. The spread of the Gaussian function is governed by σ, a larger value will result in a wider peak and therefore a greater blur.

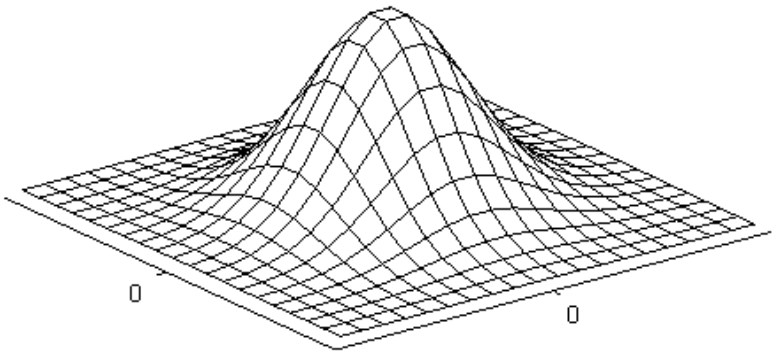


Figure 4-10. Two-dimensional Gaussian function

Commonly pre-defined Gaussian kernels are used instead of evaluating the Gaussian function and performing a sampling process. Some common pre-defined Gaussian kernels are displayed in Figure 4-11.

3 × 3 kernels:

0	1	0
1	2	1
0	1	0

0	1	0
1	4	1
0	1	0

1	1	1
1	2	1
1	1	1

1	1	1
1	4	1
1	1	1

1	2	1
2	4	2
1	2	1

1	4	1
4	16	4
1	4	1

5 × 5 kernels:

1	2	4	2	1
2	4	8	4	2
4	8	16	8	4
2	4	8	4	2
1	2	4	2	1

7 × 7 kernels:

1	1	2	2	2	1	1
1	2	2	4	2	2	1
2	2	4	8	4	2	2
2	4	8	16	8	4	2
2	2	4	8	4	2	2
1	2	2	4	2	2	1
1	1	2	2	2	1	1

Figure 4-11. Some common Gaussian kernels

Figure 4-12 displays results for low-pass filtering of a noisy image. Comparing Gaussian filters to Mean filters it is clear that Gaussian filters are more successful in retaining spatial detail to some extend. The images resulting from mean filtering are more affected by blur; however the noise reduction capabilities of Mean and Gaussian filters are comparable.

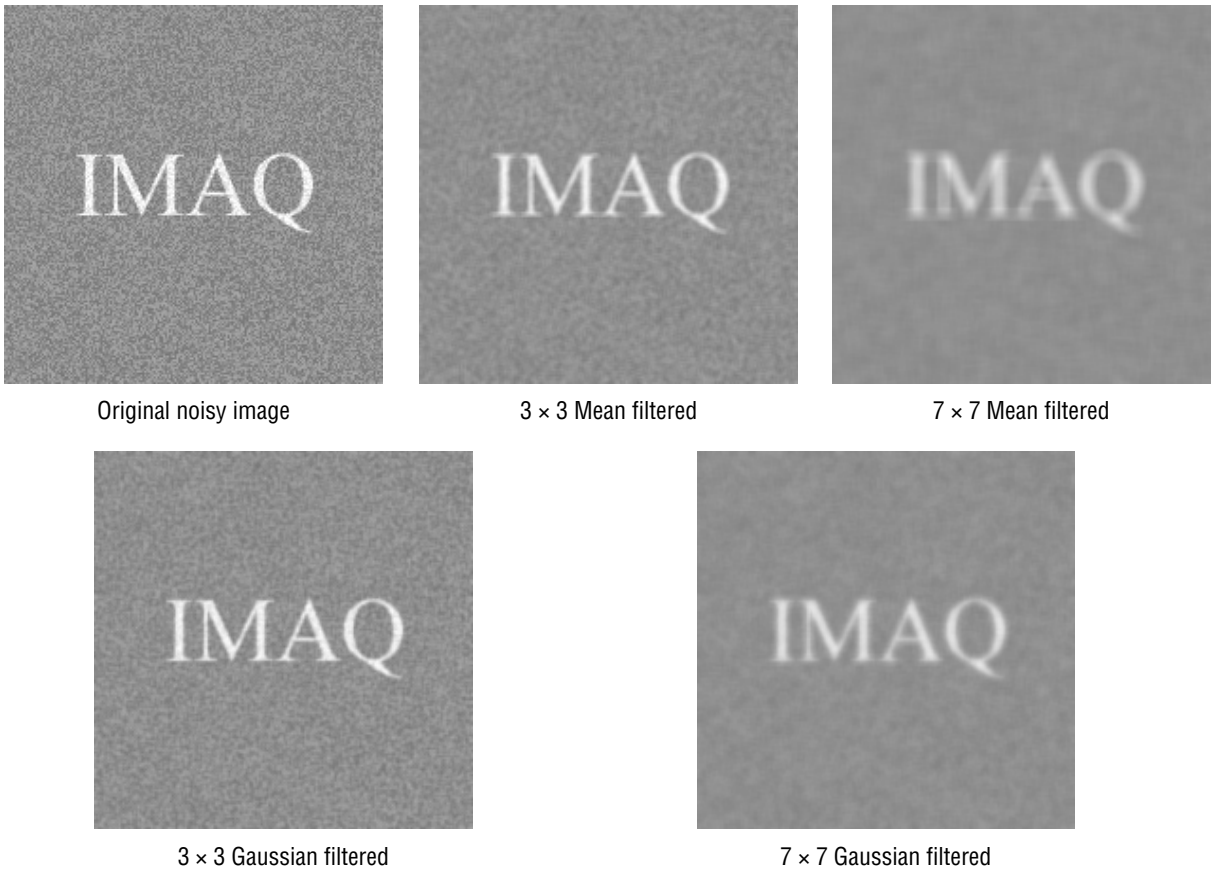


Figure 4-12. Noise reduction by low pass filtering

High Pass Filtering

High pass filters are accomplished using kernels that contain negative as well as positive coefficients. Therefore the convolution result might be negative and either an output image representation that supports negative numbers is in order or a constant bias can be added to the output. Usually a bias of half of the maximum pixel value is added so that actually a filter response of 0 maps onto the middle of the dynamic range, in which case negative filter outputs will be displayed darker and positive filter outputs will be brighter. High pass filters have two main uses: **edge detection** and **high frequency emphasis**.

Edge detection is the process of locating parts of an image that display sudden variations in the grey level or color of pixels. Edges are for instance generated by object boundaries; hence the detection of edges is helpful in locating and possibly recognizing objects. Locating meaningful edges is a tricky situation as noise and uninteresting features might also result in edges within the image. Commonly convolution kernels with a total coefficient sum of zero are used for the edge detection process. Due to the zero coefficient sum, the filter output will be zero (or comparatively small) if there is no (or little) variation in the grey level or color of pixels. On the

contrary, if there are sudden changes in the grey levels the filter output will have large amplitude (either positive or negative) revealing an edge in the output image.

High frequency emphasis is the process of highlighting sharp features within an image. Also called high boost filters, high frequency emphasis filters are made up of kernels that contain negative as well as positive coefficients but have a coefficient sum that is larger than zero. These filters are commonly used to sharpen an image for improved visual appearance.

Figure 4-13 shows example edge detection and high frequency emphasis kernels. The edge detection kernel has a coefficient sum of zero, and because only the centre coefficient is positive with all coefficients around the centre being negative the filter will show the same response whatever the direction in which the grey level changes (edges) occur, i.e. the kernel provides an omni-directional filter. The high frequency emphasis kernel has a centre coefficient that is larger than 8 so that the overall sum of the coefficients is positive. It can be thought the greater the centre coefficient is the greater will be the sharpening effect; however this is not the case. For very large centre coefficients the high pass filtering effect will be very small as the centre coefficient will dominate the convolution sum. Therefore the utilization of a centre coefficient of 9 or 10 is best suited for high frequency emphasis purpose.

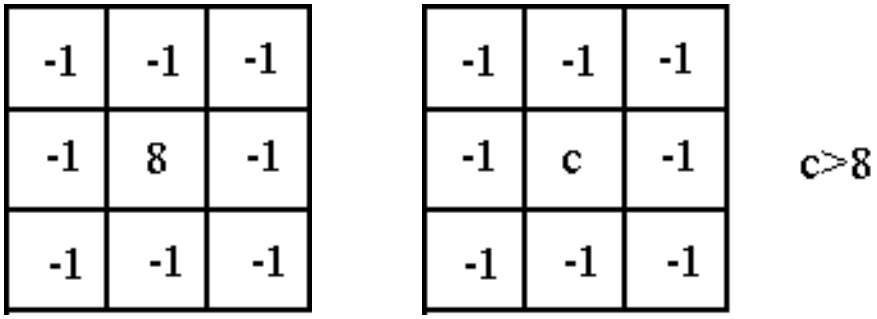


Figure 4-13. Example high pass kernels

Figure 4-14 displays the result of applying the kernels given in Figure 4-13 to an image. In the case of the edge detection kernel, all edges are successfully located irrespective of the edge direction. Dark regions in the output image represent parts of the image that display smooth grey level variations and do not contain any edge information, while sharp edges corresponding to sudden changes in grey levels have resulted in a brighter output, while moderate grey level changes are represented by a less bright output. Furthermore some scattered bright pixels are encountered in the output image due to noise and uninteresting grey level changes not directly caused by object boundaries.

In the case of the high frequency emphasis kernel, it is obvious that edges are strengthened so that the resultant image provides a sharper view more pleasant to the viewer. While the original image seemed to be fine until this point, when compared to the sharpened cases it is clear that the original image is slightly blurred and sharpening the image through edge emphasis filters clearly improves the visual appearance.

Original car image

Filtered with edge detection kernel of Figure 4-13

Filtered with edge emphasis kernel of Figure 4-13 (c=9)

Filtered with edge emphasis kernel of Figure 4-13 (c=10)

Figure 4-14. Effect of high pass filtering

A typical operator used for high pass filtering is the **gradient**. The gradient is a measure of change in a function and a discrete approximation to the gradient can be used to determine changes in grey levels of an image. The gradient is the two-dimensional equivalent of the first derivative and is defined as the vector

$$\mathbf{G}[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{4-6}$$

The gradient has two important features: the gradient vector points in the direction of the maximum rate of increase of $f(x,y)$, and the magnitude of the gradient gives the maximum rate of increase of $f(x,y)$ per unit distance in the direction of the gradient vector.

For application to digital images the gradient can be approximate by differences. The simplest gradient approximation can be formulated as

$$\begin{aligned} G_x &\cong f(x,y) - f(x-1,y) \\ G_y &\cong f(x,y) - f(x,y-1) \end{aligned} \tag{4-7}$$

It is possible to implement this approximation using simple kernels shown in Figure 4-15.

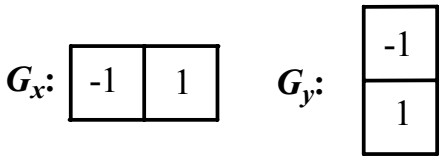


Figure 4-15. Simple gradient kernels

When computing an approximation to the gradient, horizontal and vertical partial derivatives should be computed at exactly the same position in space. Therefore it is required to compute 2×2 differences instead of the simplified 2×1 and 1×2 kernels. The corresponding kernels for 2×2 gradient computation are given in Figure 4-16. In this way it is ensured that the gradients in the horizontal and vertical directions are computed at the same point, which lies in between all four pixels. Commonly it is preferred to use 3×3 kernels so that the gradient is calculated for the centre pixel of the 3×3 neighborhood.

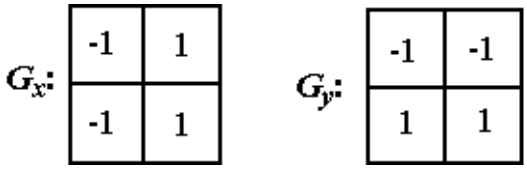


Figure 4-16. 2 × 2 gradient computation kernels

A typical class of gradient filters comprises a filter group called Prewitt filters for which kernels are shown in Figure 4-17. The notations West (W), South (S), East (E), and North (N) indicate the edges of bright regions outlined by the filter.

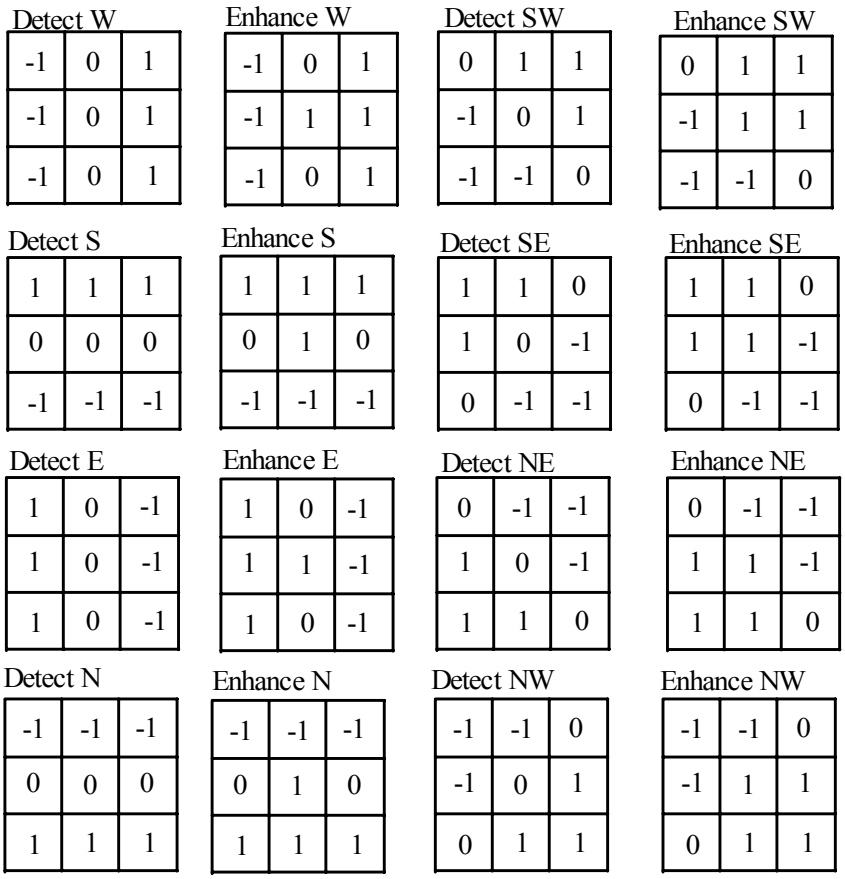


Figure 4-17. Prewitt filter kernels

Using specific kernels it is possible to control the direction of edges that are to be detected/enhanced in the image. It is possible to combine the effect of various kernels by applying the original image to corresponding kernels separately and the averaging the resultant outputs to obtain a combined effect.

A second well known class of gradient filters comprises a filter group called Sobel filters for which kernels are shown in Figure 4-18. Sobel filters emphasize pixels that are closer to the centre of the kernel.

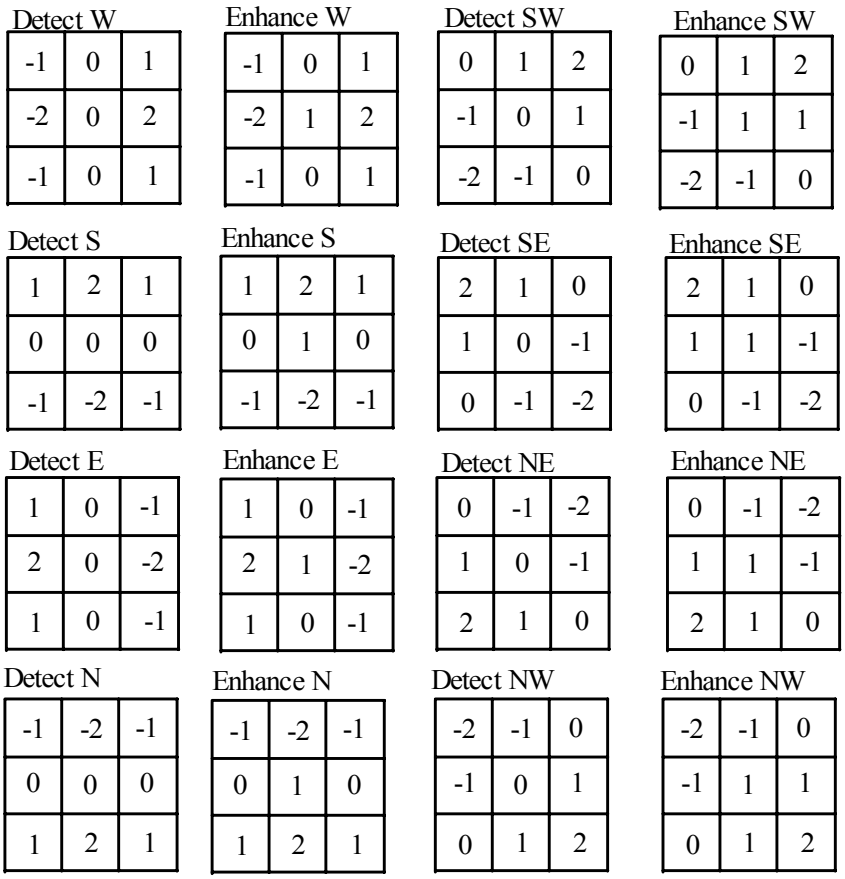


Figure 4-18. Sobel filter kernels

It is possible to use larger kernels, however in this case the effect of uninteresting edges and noise increases and the output image is dominated by edge information as shown in Figure 4-19. Therefore it is usually preferred to utilize small filter kernels commonly of size 3×3.

The gradient operator is based on the computation of the first derivative and results in the detection of too many edge points as all sport of grey level variations are directly assigned as edge information. The performance can be improved by considering only points that result in local maxima in gradient values to correspond to edges. This procedure requires the detection of peaks of the first derivate, or equivalently zero crossings of the second derivative. In order to detect zero crossings of the second derivative of the image intensity it is possible to employ the **Laplacian** operator, which can be considered as the two-dimensional equivalent of the second derivative.

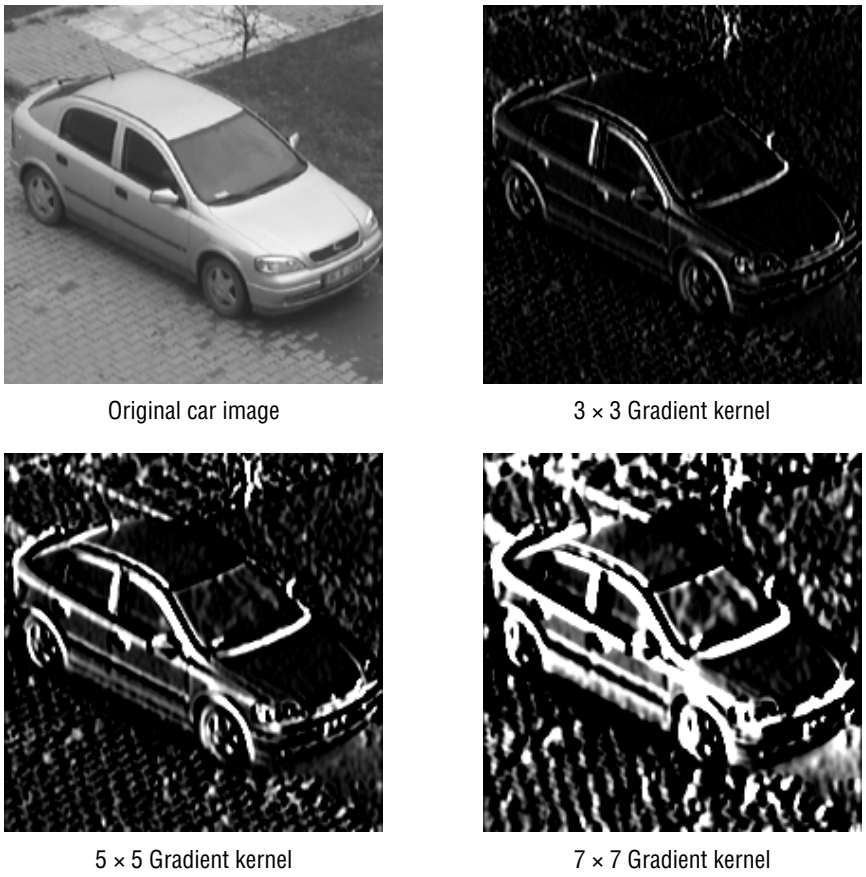


Figure 4-19. Detecting edges in the West direction using Gradient filters

The Laplacian of an image combines the second order derivatives in the horizontal and vertical directions in the form of

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{4-8}$$

The second derivative can be approximated for instance using the difference equations

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

By combining these difference equations into a single operator, it is possible to obtain the kernel shown in the top left corner of Figure 4-20. Some commonly used Laplacian kernels are displayed in Figure 4-20. Note that it is also possible to approximate the Laplacian by using higher weights for pixels close to the centre of the neighborhood, which might be convenient for some applications. Note that high-pass filtering results presented in Figure 4-14 use Laplacian filters.

Since any operator involving second order derivatives affected by noise more than an operator involving first order derivatives, the Laplacian is very sensitive to noise. The Laplacian is therefore seldom used on it own, but most commonly combined with Gaussian filtering. The Laplacian of Gaussian (LoG) filter thus combines Gaussian filtering for noise reduction with a Laplacian to detect or enhance edges. The effect of smoothing previous to edge detection is shown in Figure 4-21.

3 × 3 kernels:

Edge Detect

0	-1	0
-1	4	-1
0	-1	0

Edge Enhance

0	-1	0
-1	5	-1
0	-1	0

Edge Enhance

0	-1	0
-1	6	-1
0	-1	0

Edge Detect

-1	-1	-1
-1	8	-1
-1	-1	-1

Edge Enhance

-1	-1	-1
-1	9	-1
-1	-1	-1

Edge Enhance

-1	-1	-1
-1	10	-1
-1	-1	-1

Edge Detect

-1	-2	-1
-2	12	-2
-1	-2	-1

Edge Enhance

-1	-2	-1
-2	13	-2
-1	-2	-1

5 × 5 kernels:

Edge Detect

-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	24	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

Edge Enhance

-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	25	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

7 × 7 kernels:

Edge Detect

-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	48	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1

Edge Enhance

-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	49	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1

Figure 4-20. Some common Laplacian kernels

5 × 5 Laplacian filter 5 × 5 LoG filter

Figure 4-21. Effect of low pass filtering before edge detection

Edge Detection

Edge detection is an important part of image processing, and therefore worth investigating supplementary to filtering. Edge detection algorithms typically contain three steps:

1. *Noise Reduction*—Since edge detection algorithms are susceptible to noise, noise reduction, commonly accomplished by low pass filtering, will improve the edge detection performance.
2. *Edge Enhancement*—Edge enhancement emphasizes pixel locations that show a significant change in local intensity values. Edge enhancement is commonly carried out by using high pass filters that respond strongly at edges and weakly elsewhere.
3. *Edge Detection*—Edge detection aims to discriminate meaningful edges with from other points in the output image that have a nonzero value possibly due to noise. Therefore some method that determines which points are edges is in order. Frequently, thresholding provides a reasonably accurate criterion for detection.

The edge detection process merely indicates that an edge is present near a pixel in the image, but does not necessarily result in an accurate estimate of the edge location or orientation. Edge detection can result in two sorts of errors: false edges and missing edges. Note that many edge detection techniques have been developed and edge detection is a huge field of study. Therefore it is sensible to merely investigate basic and commonly used edge detection algorithms.

Simple edge detection can be accomplished using the Roberts operator that provides a simple approximation to the gradient magnitude, for which kernels are shown in Figure 4-22, or the already investigated Prewitt, Sobel, or Laplacian operators.

G_x:

1	0
0	-1

G_y:

0	-1
1	0

Figure 4-22. Roberts kernels

The **Canny edge detector** is a somewhat more sophisticated and frequently used edge detection technique which is optimal for step edges corrupted by white noise. Canny edge detection can be summarized by the following algorithm:

1. *Smoothing*—The image is smoothened with a Gaussian filter.
2. *Edge Enhancement*—Compute the gradient vector using finite difference approximations for the partial derivatives.
3. *Non-maxima Suppression*—Thin wide ridges around local maxima in gradient magnitude down to edges that are only one pixel wide. This is accomplished by suppressing all values along the line of the gradient that are not peak values of a ridge.
4. *Hysteresis thresholding*—The typical approach to reduce the number of false detections is to apply a threshold and set all values below the threshold to zero. In the case of using a fixed single threshold a threshold too low might result in many false edges to be registered while a threshold too high might suppress meaningful edge points. Hysteresis thresholding offers a more effective scheme using two thresholds. The higher threshold is used to mark the best edge pixel candidates, which are grown into contours by searching for candidate neighbor edge points in the low threshold edge map.

Figure 4-23 shows an example to Canny edge detection. The effect of non-maxima suppression is clearly visible: all edges are only one pixel wide. It is possible to adjust the response of the edge detector by setting Gaussian filter parameters as well as high and low threshold values.

Original car image

Canny edge detection

Figure 4-23. Effect of low pass filtering before edge detection

Rank Filtering

As an alternative to linear filtering through convolution it is also possible to use non-linear techniques to filter an image. A widely used class of non-linear filters is collectively known as order statistics (also called **nth order)** filters or **rank** filters. Rank filters compile a list of the grey levels in the neighborhood of a given pixel, sort these values in ascending order and then select a value from a particular position of the sorted list. Again, image borders must be given special consideration.

The most commonly used rank filter is the **median filter**, which selects the middle ranked value from the sorted neighborhood values. Figure 4-24 displays the operation of a 3×3 Median filter. Median filters are very effective in removing impulse noise while retaining image details. Because neighborhood pixel values are sorted and the middle ranked value is outputted, values that are significantly different from typical values within the neighborhood will directly be expelled. Furthermore pixels corrupted by noise will not affect surrounding pixels at the output as no more a weighted sum is computed as it is the case in linear low pass filtering.

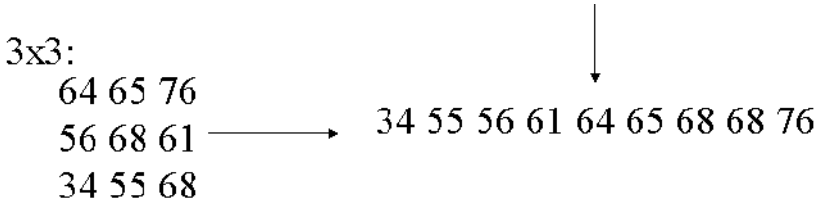


Figure 4-24. Operation of a 3 × 3 Median filter for a sample image neighborhood

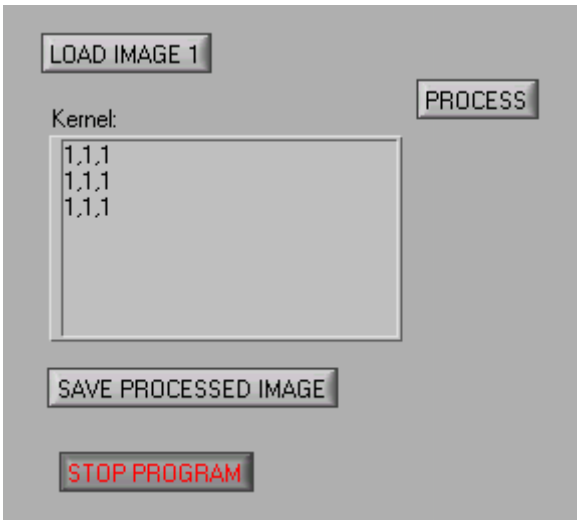
Figure 4-25 shows the effect of median filtering of an image corrupted by impulsive noise and compares the result to the mean filter. It is clearly seen that the median filter provides a better outcome.

Image corrupted by noise | 3 × 3 Median filtered | 3 × 3 Mean filtered

Figure 4-25. Comparing Mean and Median filtering

LabVIEW Demo 4.1: Convolution

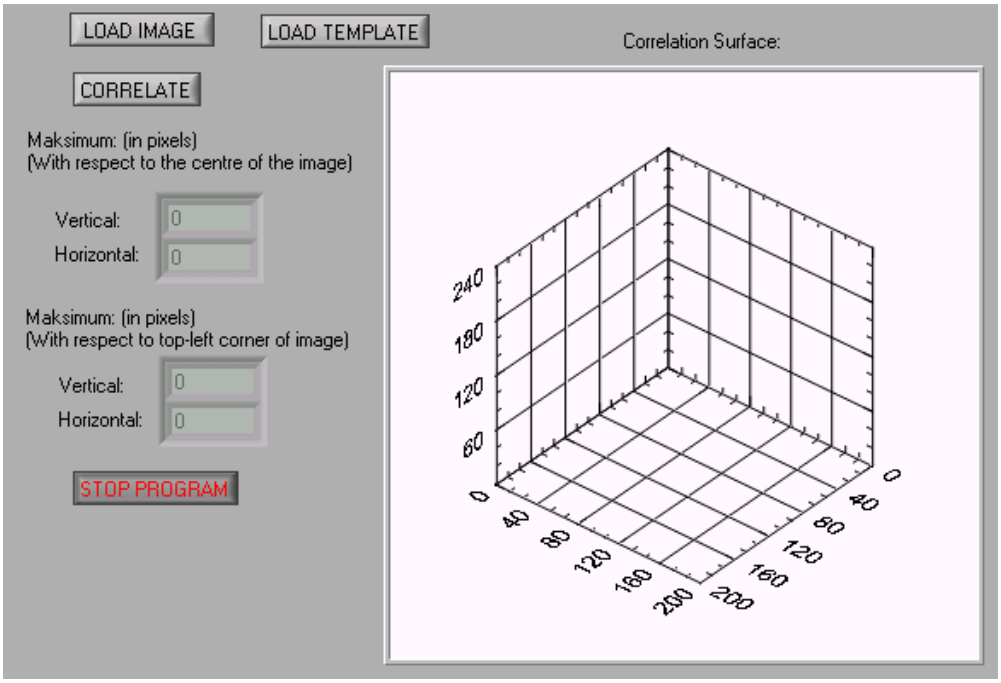
Launch the LabVIEW program entitled `convolution.vi` from the chapter 4 library. Run the program.



Press the “load image 1” button to select, load and display an image. Hit the “process” button to display the result of the convolution with the specified kernel. Try kernels with positive only and both positive and negative coefficients for separate images and observe their effect.

LabVIEW Demo 4.2: Correlation

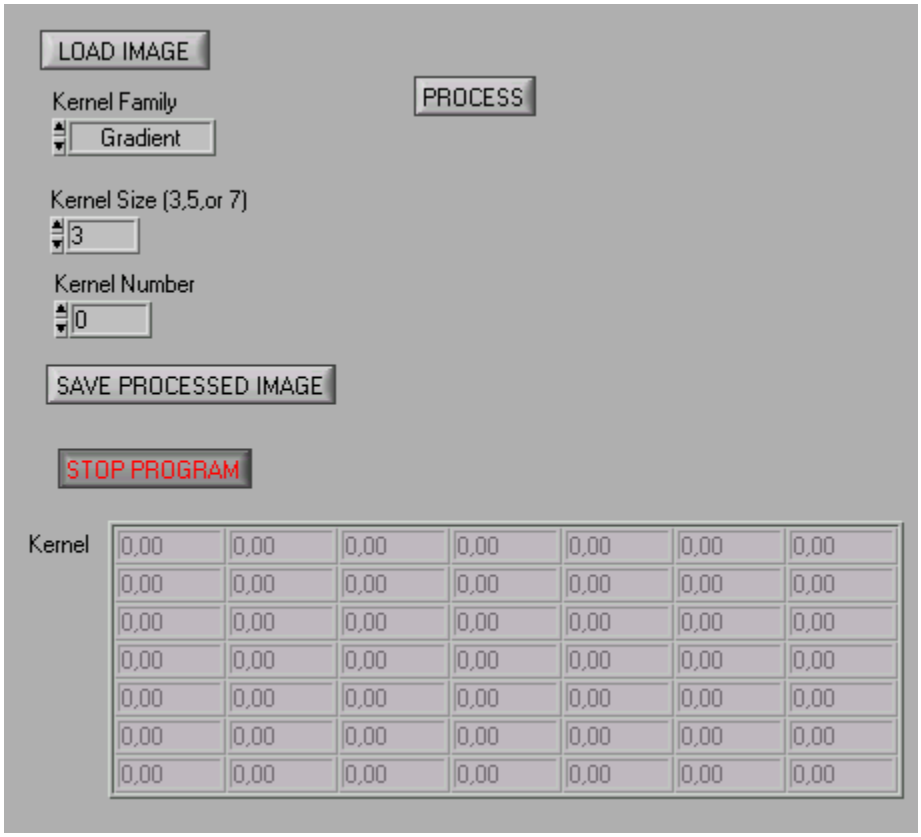
Launch the LabVIEW program entitled `correlation.vi` from the chapter 4 library. Run the program.



Press the “load image” button to select, load and display an input image and the “load template” button to select, load and display an image template. Hit the “correlate” button to display the correlation result in the form of a correlation image as well as 3-D correlation surface. The point at which the maximum correlation is obtained is displayed. Try the program with various images and templates, trying to locate a part of the image or global image displacement. Observe the computation times for various image and particularly template sizes. Observed what happens if the input and template is entirely different.

LabVIEW Demo 4.3: Linear Filtering

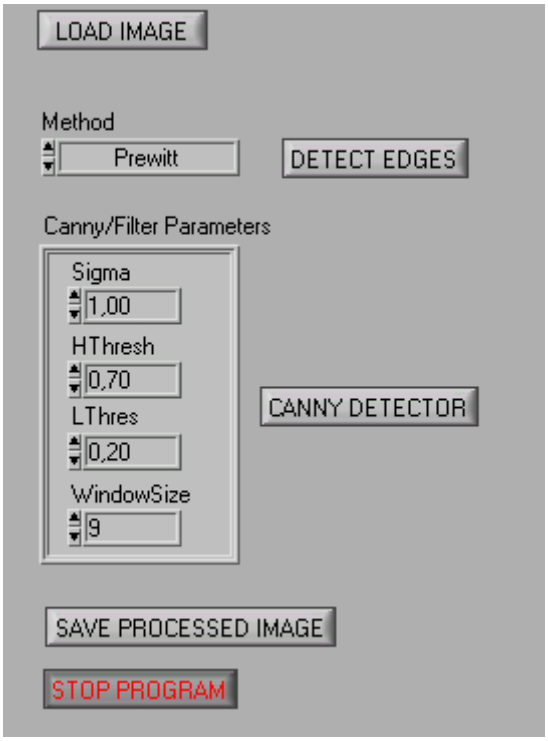
Launch the LabVIEW program entitled `linearfiltering.vi` from the chapter 4 library. Run the program.



Press the “load image” button to select, load and display an image. Select the desired kernel family (Smoothing, Gaussian, Gradient, Laplacian), kernel size and kernel number. Hit the “process” button to display the result of linear filtering with the specified kernel. The kernel coefficients will be displayed. Try various kernels with different sizes and observe their effects. Notice that using the kernel number you might select different kernels for the same family of the same size performing various processes.

LabVIEW Demo 4.4: Edge Detection

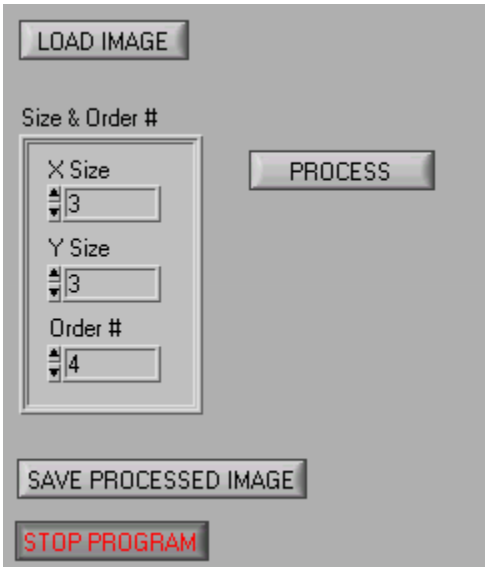
Launch the LabVIEW program entitled `edgedetection.vi` from the chapter 4 library. Run the program.



Press the “load image” button to select, load and display an image. Hit either the “detect edges” button to display the result of edge detection using the selected method (Prewitt, Sobel, Roberts, etc.) or hit the “Canny detector” button to facilitate Canny edge detection. Observe the results for various edge detection algorithms. Change the parameters of the Canny detector and examine the effects on the resultant edge map.

LabVIEW Demo 4.5: Rank Filtering

Launch the LabVIEW program entitled `rankfilter.vi` from the chapter 4 library. Run the program.

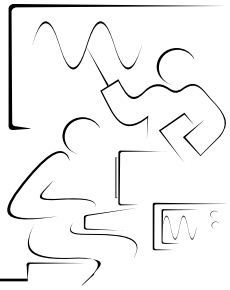


Press the “load image” button to select, load and display an image. Hit the “process” button to display the result of rank filtering. X size and Y size determines the horizontal and vertical size of the filtering neighborhood, and the order # determines the number of the pixel which is outputted after sorting. For instance for a 3×3 Median filter there are a total of 9 pixels in the neighborhood, thus the output pixel should have an order of 4, while for a 5×5 Median filter the rank should be 12. Try the rank filter with different parameters on various images and observe the results particularly for images corrupted by noise.

Notes

Notes

Lab 5
Color Images



While grey-level images only contain information of light intensity, color imaging is not only more pleasant for the viewer but it is also enables the incorporation of extra visual information. The perceived color depends on three factors:

- The spectral reflectance characteristic of surfaces which specifies the way color is reflected from the surfaces will affect the perceived color.
- Light consists of a spectrum of wavelengths and perceived color will depend on the color content of light shining on the surfaces which is specified by the spectral content of the present illumination.
- Sensors used in the imaging system will have a certain spectral response which will have an effect on perceived color.

Because of the nature of the human visual system, imaging equipment is mainly based on tricolor systems. Color photography and color television are typical examples of tricolor systems, in which the visible spectrum is divided intro three bands, red, green, and blue (RGB model), approximating the behavior of the human eye. A tricolor digital image is commonly managed as an ordinary two-dimensional image having three grey levels (e.g. red, green, blue) at each pixel, or an image composed of three color planes respectively.

Color Models

The **RGB model** is based on the observation that by mixing red (R), green (G), and blue (B) in various proportions it is possible to obtain a wide range of colors. Therefore it is possible to construct a color image using the red, green and blue components of the detected light for each pixel. The color of each pixel is then determined by the weight of each primary color. In color television, for instance, three imaging sensors are utilized, one behind red optical filters, one behind green optical filters and one behind blue optical filters. For display purposes, red, green and blue images are superimposed to obtain a color image.

Commonly R, G, and B values are normalized to vary between zero and one. In a color imaging conforming to the RGB model, the value of each pixel of the image can be thought as of a vector of three components, namely the red, green and blue values. Thus the color space can be defined such that R, G, and B are regarded as the orthogonal axes defining a three-dimensional color space represented in terms of a color cube in the first quadrant as shown in Figure 5-1. The origin of the color space represents no brightness of any of the primary colors and therefore corresponds to the color black.

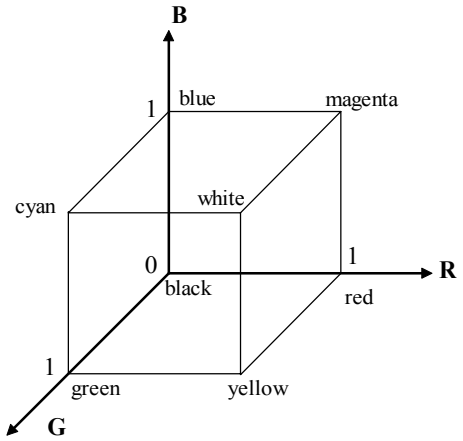


Figure 5-1. RGB color space

Each pixel of the color image will have three values, one each for the red, green and blue component. Therefore a color image can be represented to be composed of three color planes, namely the red plane, green plane and blue plane. The red plane will include the red component values of all pixels, the green plane will include the green component values of all pixels and the blue plane will include the blue component values of all pixels. Commonly each of the three color components is quantized to 8 bits, so that the resultant color image is described by 24 bits. Note that about 16 million different colors are therefore available in a 24-bit color image. Figure 5-2 presents an example to the color planes of an image.

"Kids" image

Red color components of the image

Green color components of the image

Blue color components of the image

Figure 5-2. RGB color planes of an image

An alternative color model uses cyan (C), magenta (M), and yellow (Y) as primary colors. Although in theory almost any color should be produced by mixing cyan, magenta, and yellow; in practice this process does not produce a satisfactory black color. Therefore a forth component representing the black color is added. The black component is labeled as K, resulting in the **CMYK model** for color imaging. This color model is for instance used in color printers to print digital images.

The RGB and CMYK models are of limited use in image processing as each color components in these models contains color as well intensity information. However, due to properties of the human visual system in perceptual terms color and intensity are distinct from each other. Therefore color models that decompose color information and intensity are preferable for image processing applications.

The **HSI model** is more suited to image processing as color information contained in hue (H) and saturation (S) is decoupled from intensity (I) information. Hue specifies the dominant perceived color, i.e. pure color, and is not limited to just red, green, cyan, or magenta, etc. The hue can be any color such as orange, pink, etc. Saturation measures the amount of white

contained in the pure color showing how washed out the color appears to be. Hue and saturation specify the color information. The intensity is used for the brightness of the color specified by hue and saturation. For example, an orange in sunlight and the same orange in shade will have different brightness values. The HSI model is more suited to image processing as it matches human perception characteristics. A rather complex geometric transformation can be employed to map a color from RGM space to HIS space or vice versa. Figure 5-3 presents an example to the HSI planes of an image.

"Kids" image

Hue components of the image

Saturation components of the image

Intensity components of the image

Figure 5-3. HSI color planes of an image

In the HSI model the first value encodes the colors dominant hue (H), progressing cyclically from red at 0.0, through green at 0.33, blue at 0.67 and back to red at 1.0. Intermediate values encode composite colors such as yellow, turquoise and purple. The second value encodes the purity or tone of the color (its saturation S), ranging from no color (a grey tone) at 0.0, to a vivid tone of the color (no grey) at 1.0. The final value encodes the intensity from black at 0.0 to full brightness at 1.0. Note that in the HSI model the intensity value of a pixel corresponds to just the average of the red, green and blue grey level values.

An alternative color model is the **HSV model**. The HSV model that is similar to the HSI model in that H represents hue and S represents saturation, although the relation between hue and saturation and the RGB colors is slightly different established compared to HSI. Although different transformations are used to compute hue and saturation values from RGB, nonetheless resulting values for HIS and HSV are about the same. V stands for value and encodes the maximum of red, green, and blue values for a pixel.

Another similar color model is the **HSL model**, which represents a color image in the form of hue (H), saturation (S) and luminance (L) values. The same transformation of HSV is used for hue resulting in the same hue plane; however a slightly different transformation is employed for saturation resulting in a slightly different saturation plane. L stands for luminance and encodes the average of the maximum and minimum of red, green, and blue values for a pixel.

Color Enhancement

If a color image is processed in the RGB domain, care has to be taken not to disturb the color balance. Figure 5-4 displays an example of the effect of changing RGB balance.

"Kids" image

Red channel brightness increased

Red and green channel brightness increased

R, G, and B channel brightness increased

Figure 5-4. RGB color enhancement

If only one of the RGB channels is processed, it is likely that the color balance will be disturbed resulting the dominance or lack of the processed channel color. As shown in the example of Figure 5-4, for instance, increasing the red channel brightness only results in red color dominance in the output image. If for instance, two of the channels are processed, their cumulative effect will dominate, e.g. increase in red and green channel brightness results in dominant yellow. Thus unless the image is "out of balance" to start with and the aim is to balance the color, care should be taken while processing an image in the RGB domain.

Image processing in the HSI domain is more straightforward. As the human vision system is more sensitive to intensity than color, it is possible to use this feature for image processing purposes. Essentially all image processing techniques available for grey-level images can be applied to the intensity of

an image successfully, and the color information embedded in hue and saturation can be kept unchanged to accomplish the desired process. However it is possible to modify the saturation or hue of an image for particular purposes.

Saturation enhancement can be utilized to change the brightness of colors. A decrease in saturation will result in duller colors, while a saturation increase will cause colors to appear shinier. In some cases it might be desirable to maximize the saturation in order to determine just what colors (hues) are actually present in an image. Figure 5-5. shows the result of increasing the saturation.

Original image Result of increased saturation

Figure 5-5. Saturation enhancement

In some cases it might be desired to change the hue of an image. Since hue directly determines the color content, this will result in change in color. If the change is small, the result might appear just as a “cooling” or “warming” the image color. A large change in hue might result in a totally different color.

Note that geometric operations, such as translation, must naturally be carried out exactly the same way in all three components in HSI as well as RGB domain.

Color Histograms

The histogram of a color image can be considered to consist of three separate one-dimensional histograms, one for each tricolor component. However, the true histogram of a color image is three-dimensional. It is possible to consider the true histogram for an RGB image for instance as the RGB color cube shown in Figure 5-1 divided into bins, giving a $256 \times 256 \times 256$ array. Therefore every possible color value in 24-bit format will be represented by one bin of the histogram making up over 16 million bins. Three separate histograms recording R, G, and B values actually correspond to projections of the 3D histogram onto the primary color axes.

There are mainly two reasons why it is preferred to use three separate one-dimensional histograms. The first reason is the amount of storage space required. For a three-dimensional histogram with over 16 million bins, if a 32-bit integer representation is used for bin counts the total storage space for a single color histogram requires 64 megabytes. However storage space requirements can be reduced to some extend making use of the sparse population of such histogram. The second disadvantage of a three-dimensional histogram is the difficulty in visualizing 3-D data. As comprise it is possible to compute two-dimensional histograms for R and G, R and B, and G and B to visualize the correlation between color components.

Figure 5-6 displays the RGB and HSI histograms for the “kids” image. While not much information is gained from the RGB histograms, the hue histogram unveils that the scene is dominated by the green color (corresponding to one third of the dynamic range of hue) and the saturation diagram shows that colors in the image have quite low saturation.

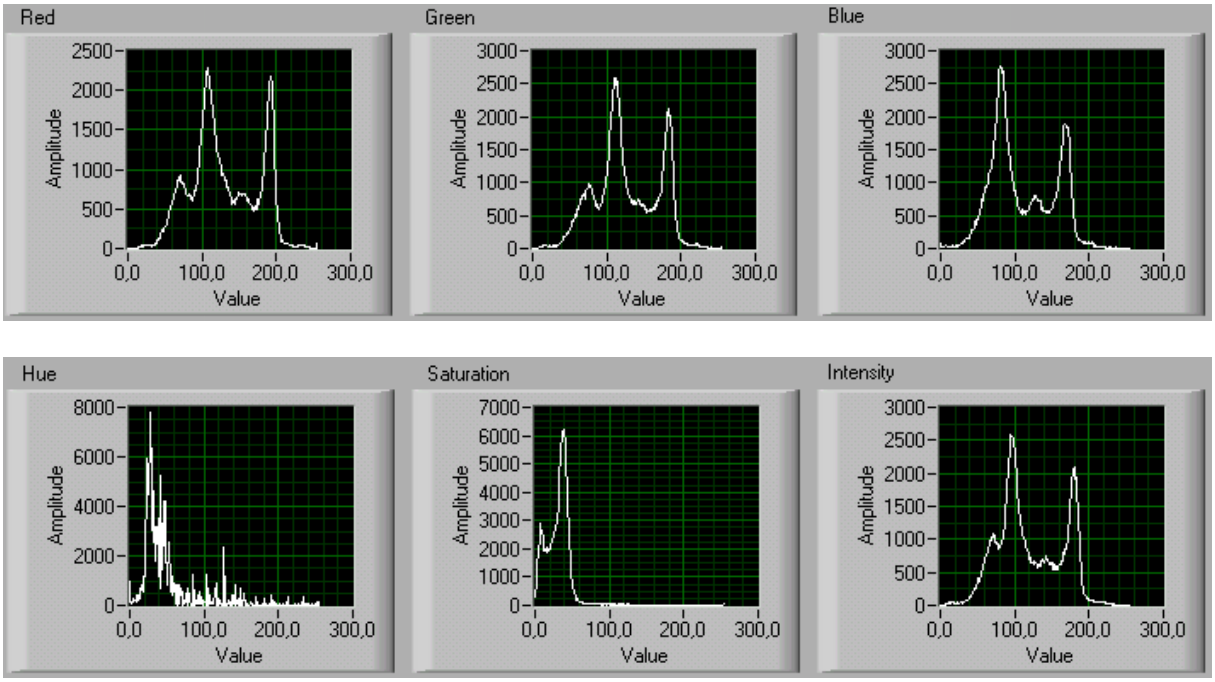


Figure 5-6. RGB and HSI histograms for the “kids” image

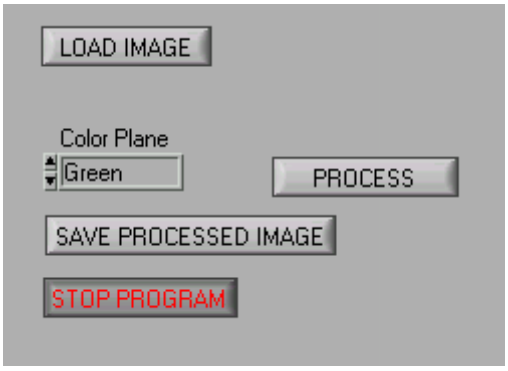
Expanding the idea of histogram equalization to color images, one option seems to equalize all three color planes separately. However if R, G, and B planes are equalized independently, the color balance of the image might be upset. An alternative approach is to equalize the intensity values only, with no changes to saturation and hue. This approach ensures that image contrast is enhanced without affecting the color balance of the image.

Color Thresholding

In the case of thresholding again processing in the RGB domain is quite difficult as color planes are not decoupled and it is required to develop a tricky thresholding approach for the R, G, and B planes that will result in the desired thresholding. On the contrary, color thresholding is quite simple in the HSI domain. It is possible to threshold pixels, according to their pure color (hue), saturation or intensity. For example, if it is desired to identify regions of green color, the range of green in hue is well known and it possible to separate green colored regions by simple thresholding in the hue plane.

LabVIEW Demo 5.1: Color Planes

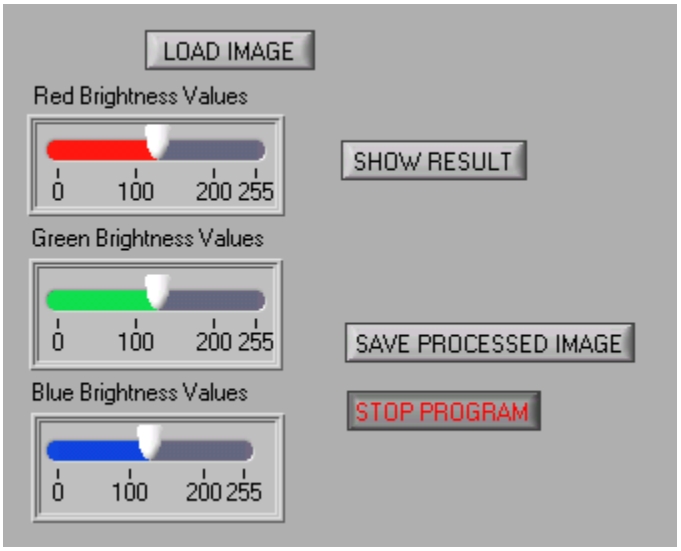
Launch the LabVIEW program entitled `extractcolorplane.vi` from the chapter 5 library. Run the program.



Press the “load image” button to select, load and display a color image. Select a Color plane and hit the “process” button to display the corresponding plane. Extract different color planes for various images try to relate bright and dark parts of the color planes to color features in the images.

LabVIEW Demo 5.2: RGB Enhancement

Launch the LabVIEW program entitled `RGBbrightness.vi` from the chapter 5 library. Run the program.



Press the “load image” button to select, load and display a color image. At the centre position of the slides R, G, B planes are at their original brightness. Decreasing the brightness values using the slides reduces the values of the corresponding color plane, increasing the brightness values using the slides adds to the values of the corresponding color plane. Hit the “show result” button to display the output image. Observe how image get out of color balance and they can be brought into balance.

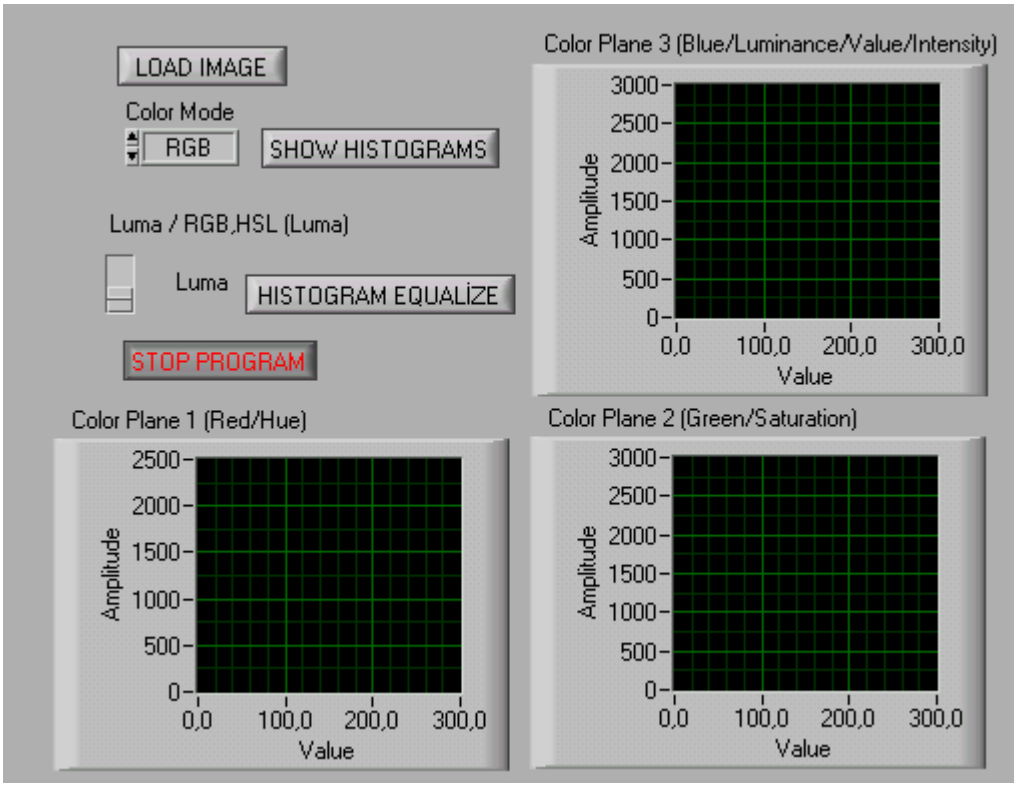
LabVIEW Demo 5.3: HSI Enhancement

Launch the LabVIEW program entitled `HSIbrightness.vi` from the chapter 5 library. Run the program.

Redo demo5.2 in the HSI domain. By changing the hue, saturation and intensity of a color image, observe the effect of each and try get a feeling of these identities.

LabVIEW Demo 5.4: Color Histograms

Launch the LabVIEW program entitled `colorhistogram.vi` from the chapter 5 library. Run the program.



Press the “load image” button to select, load and display a color image. Display the RGB and HSI histograms of various images. Observe the relation between the histogram and color image content for both cases. Perform histogram equalization on the RGB planes as well as luminance only and observe the results. Notice that histogram equalization in RGB domain upsets the color balance.

LabVIEW Demo 5.5: Color Thresholding

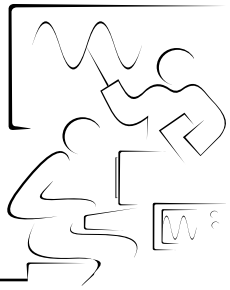
Load the LabVIEW program entitled `Color Threshold Example.vi` from the IMAQ vision example folders. (`C:\Program Files\National Instruments\LabVIEW 6\examples\Vision\2. Functions\Color\ColorThreshold Example.vi`)

Follow the instruction step by step, and observe that processing HSI mode is advantageous to processing in the RGB domain.

Notes

Notes

Lab 6
The Frequency Domain



Images typically display variations of brightness or color in space, and are therefore defined in the spatial domain. Several image processing techniques deal with images in the spatial domain and achieve manipulation of images through point operations or spatial operations. An alternative representation of image content is available in the frequency domain, in which the image content is represented based on the frequencies of brightness or color variation. It is possible to convert an image from the spatial domain to the frequency domain, and vice verse, without any loss of information. Furthermore it is possible to process an image in the frequency domain by manipulating its frequency spectrum.

Note that when talking of the frequency domain of images, it is the **spatial frequency** that is considered, not the time-frequency relationship of time varying signals. As images can be considered as functions of brightness or color in space, i.e. horizontal and vertical spatial directions, the spatial frequency is characterized by the pace of change of brightness or color values for a certain spatial distance. Assuming periodic changes of image values, by definition, spatial frequency refers to the inverse of the periodicity with which image values change. Image features that show a great change over a short spatial distance, such as edges, will correspond to the high spatial frequency content, while image features that are smooth and show a small change correspond to low spatial frequency content.

The Fourier Transform

The Fourier Transform is of fundamental importance in digital image processing because it can be used to convert an image from the spatial domain to the frequency (or Fourier) domain. The Fourier transform typically decomposes an image into sine and cosine components of different frequency. The Fourier transform produces a Fourier domain image, in which each point represents a particular frequency contained within the image.

Because digital images constitute of sampled data, the Discrete Fourier Transform (**DFT**) is used and not the Continuous Fourier Transform. The DFT corresponds to the sampled Fourier Transform and therefore contains only a set of frequency samples large enough to fully describe the spatial domain image so that no information loss is introduced by the

transformation. The number of frequencies required to fully represent an image is equal to the number of pixels in the spatial domain. Thus the image in the spatial domain and Fourier domain are of the same size. Because images are two-dimensional function, the two-dimensional Fourier transform is utilized.

The two-dimensional DFT for a square-image of size $N \times N$ is defined as

$$F(u,v) = \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) e^{-j\frac{2\pi}{N}(ux+vy)} \tag{6-1}$$

The exponential term constituting the basis functions of the Fourier domain, the DFT can be interpreted as the sum of the multiplication of spatial image values with the corresponding base function. The basis functions are sine and cosine waves with increasing frequencies. Thus $F(0,0)$ corresponds to the DC component and represents the average brightness of the image. The highest frequency is represented by the last Fourier component $F(N-1, N-1)$. Note that the DFT produces a complex representation, thus a real image is represented in the Frequency domain in complex form, i.e. the Fourier Domain representation is a complex image.

While the DFT transforms an image from the spatial domain to the frequency domain, there also exists an inverse discrete Fourier transform (IDFT) that converts a set of Fourier coefficients into an image. The IDFT is defined as

$$f(x,y) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} F(u,v) e^{j\frac{2\pi}{N}(ux+vy)} \tag{6-2}$$

For frequency domain processing of images, the image is initially transformed into the frequency domain using the DFT. Then the frequency domain representation $F(u,v)$ of the image is manipulated. Finally the manipulated frequency values are transformed back into the spatial domain by the IDFT. Note that while manipulations carried out in the frequency domain may result in a loss of information, the forward or inverse Fourier transform in itself does not result in any information loss.

The computation of the DFT in form of its definition is very costly and rather time consuming. The complexity of the computation of the Fourier transform is reduced by employing the **Fast Fourier Transform** (**FFT**). The FFT achieves a considerable speed up making use of the separability of the Fourier transform to separate the two-dimensional Fourier transform into one-dimensional Fourier transforms and enables fast computation of one-dimensional Fourier transforms by recursively decomposing the Fourier transform into even and odd parts. However in this case the

computation size and hence the size of the image is restricted to be a power of two: $N = 2^n$. Thus, sometimes it might be necessary to crop the image or pad it out to the appropriate dimensions.

Frequency Domain Representation of Images

The Fourier transform of an image produces a complex image. The frequency domain representation of an image can be established using the real and imaginary parts of the complex image, however this representation is not particularly informative. Therefore it is commonly preferred to represent the complex image by the magnitude and phase components.

$F(u,v)$ representing the Fourier transform of an image, the magnitude denoted by $|F(u,v)|$ and phase by $\phi(u,v)$ is given by

$$\begin{aligned} |F(u,v)| &= \sqrt{\mathrm{Re}^2\{F(u,v)\} + \mathrm{Im}^2\{F(u,v)\}} \\ \phi(u,v) &= \tan^{-1}\frac{\mathrm{Im}\{F(u,v)\}}{\mathrm{Re}\{F(u,v)\}} \end{aligned} \tag{6-3}$$

While it is commonly preferred to display the spectra of images also in the form of images, several adjustments need to be made for a satisfactory result.

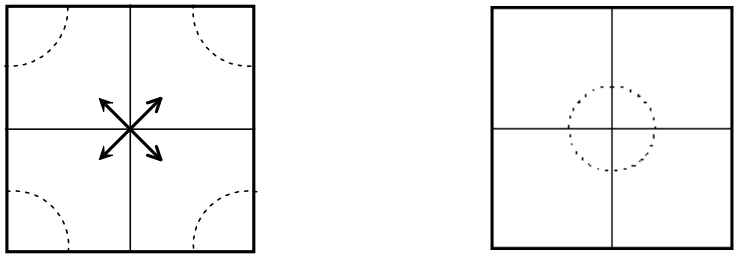
Usually the dynamic range of the Fourier coefficient magnitudes is too large to be displayed directly. The DC coefficient is the largest coefficient and although it is possible to linearly scale the magnitude such that the DC coefficient corresponds to the largest pixel value (e.g. 255 for 8-bit images), usually all other coefficients are comparably smaller resulting in all Fourier coefficients except the DC value to appear as black. Therefore commonly a logarithmic mapping is employed to enhance contrast for visualization. It is possible to implement logarithmic mapping of the magnitude spectrum by

$$|F(u,v)|_{\log} = C\log\left[|F(x,y)| + 1\right] \tag{6-4}$$

where the coefficient *C* is chosen so that the resulting values occupy the full dynamic range. Note that the before computation of the logarithm the magnitude spectra is incremented to ensure that originally zero valued coefficients are brought into the logarithmic range as the logarithm of zero is undefined.

The phase spectrum on the other hand includes values in the range $(-\pi,\pi]$, or $(-180°,180°]$. In this case it is preferable to use a linear mapping of phase values into the dynamic range of image used for display (e.g. 0–255 for 8-bit representations) as virtually a uniform distribution of phase values is encountered.

Another point that has to be taken care of is the location of frequency components after the Fourier transform. Commonly FFT routines result in low frequencies to be located at the corners of the resultant two-dimensional spectrum, while high frequencies are grouped at the centre. For visualization purposes, however, it is preferred to cluster low frequencies at the centre of the spectrum, while high frequencies can be located further outside. For this purpose it might be required to flip the quadrants of the spectrum to produce a central symmetric representation of the spatial frequencies. This process is illustrated in Figure 6-1.

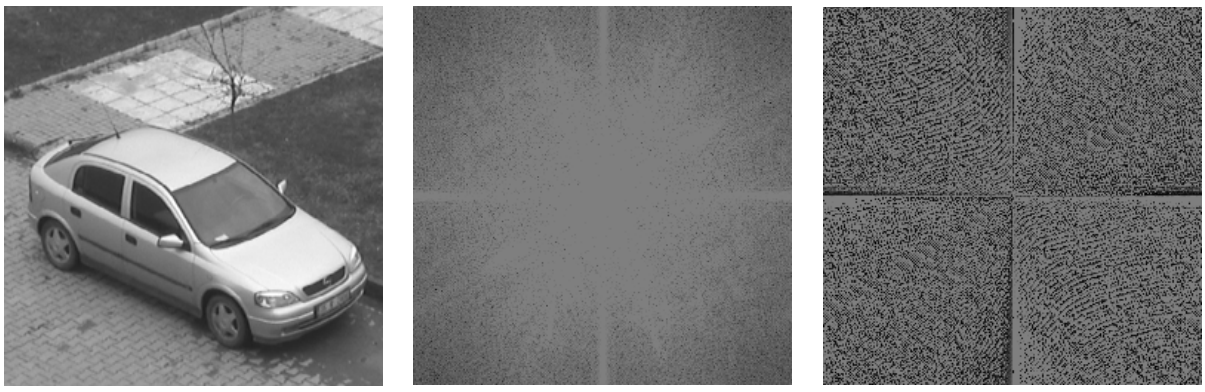


Low frequencies at corners

Low frequencies at centre

Figure 6-1. Flipping of quadrants to cluster low frequencies at the centre of the spectrum

Figure 6-2 demonstrates an example to the frequency domain representation of an image. The magnitude spectrum is transformed using logarithmic mapping to utilize the full dynamic range of the 8-bit image. Brighter parts in the spectrum indicate higher amplitudes of the corresponding frequency. The spectra is low frequency centered, thus it is seen that as expected low frequency components have higher amplitude. The phase spectrum is linearly mapped such that a phase value of 180° is represented by white and a phase of –180° is represented by black



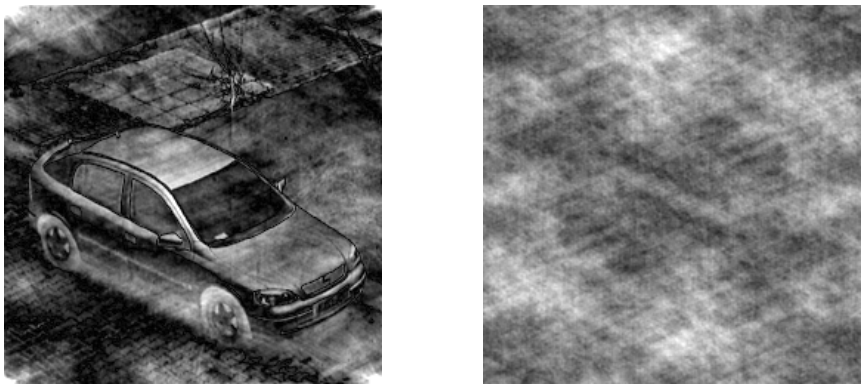
Original image

Magnitude of FFT

Phase of FFT

Figure 6-2. Frequency domain representation of an image

Using the magnitude and phase spectra it is possible to reconstruct the spatial image with no loss of information using the inverse Fourier transform. Although it seems from the magnitude and phase spectra of the image shown in Figure 6-2 as if most of the information is contained in the magnitude plane, while the phase plane look rather like noise, this is not the case. Figure 6-3 shows the reconstructed spatial images for the case of destroyed magnitude information by retained phase, and the case of destroyed phase information by retained magnitude. It is clearly seen that destroying the magnitude information results only in changes of brightness while visible information is still comprehensible, while destroying the phase information results in a totally meaningless result.



Magnitude information destroyed Phase information destroyed

Figure 6-3. Reconstruction of the image from spectra

The phase information encodes the location of image features and disrupting the phase information upsets the spatial coherence of the image. Therefore most image processing applications are only concerned with the manipulation of the magnitude spectrum while retaining phase information.

Frequency Domain Filtering

It is possible to perform filtering in the frequency domain by specifying the frequencies that should be kept and the frequencies that should be discarded. Frequency domain filtering not only has the advantage of simplifying the definition of the filter but also reduces the computational load.

While spatial domain filtering is accomplished by convolving the image with a filter kernel, the convolution theorem proves that convolution in the spatial domain implies multiplication of the corresponding Fourier transforms. If filtering is defined in the spatial domain as the convolution of the image with a filter kernel

$$g(x,y) = f(x,y) * k(x,y) \tag{6-5}$$

In the frequency domain filtering corresponds to the multiplication of the image spectrum by the Fourier transform of the filter kernel, $K(u,v)$, which is referred to as the Frequency response of the filter.

$$G(u,v) = F(u,v)K(u,v) \tag{6-6}$$

The resultant spectrum $G(u,v)$ depicts the filtered spectrum, and the filtering result can be obtained as a spatial image by the inverse Fourier transform of $G(u,v)$. As the frequency spectra is complex, theoretically the multiplication of the image spectrum with the frequency response of the filter can affect the magnitude as well as the phase of the original image. In practice, however, it is usually desired to keep the phase of the original image to avoid distortion, thus zero phase shift filters, which affect only the magnitude but not the phase, are used.

While frequency domain filtering can be accomplished by taking the Fourier transform of the image, the Fourier transform of the kernel, multiply the two Fourier transforms, and take the inverse Fourier transform of the result, the multiplication of Fourier transforms need to be carried out point-by-point. This point-by-point multiplication requires that the Fourier transforms of the image and the kernel, and hence the image and kernel themselves have the same dimensions. As convolution kernels are commonly much smaller than the images they are used to filter, it is required to zero pad out the kernel to the size of the image to accomplish this process.

Instead of defining a convolution kernel and taking its Fourier transform for frequency domain filtering it is possible to design the filter directly in the frequency domain. Taking initially low pass filters into consideration, the simplest type of low pass filter that can be designed in the frequency domain is the **ideal low pass filter**. The ideal low pass filter has a sharp cut-off at some specific frequency. In the frequency domain, the filter characteristics can be considered to keep the frequency components up to a certain distance from the centre of the spectrum and discard the rest. As the frequency response of the filter is to be multiplied with the image spectrum, the frequency response can be formulated as

$$K(u,v) = \begin{cases} 1, & r(u,v) \le r_0 \\ 0, & r(u,v) > r_0 \end{cases} \tag{6-7}$$

where r_0 is the filter radius and $r(u,v) = \sqrt{u^2 + v^2}$ is the distance from the centre of the spectrum. The resultant cylindrically shaped frequency response of an ideal low pass filter is illustrated in Figure 6-4.

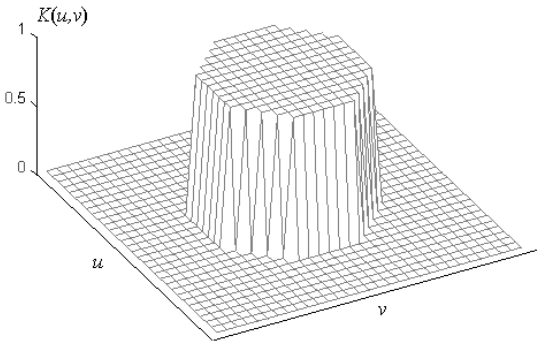
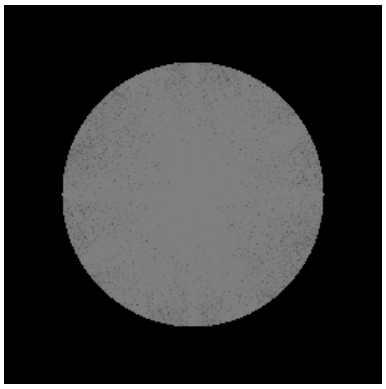


Figure 6-4. Frequency response of an ideal low pass filter

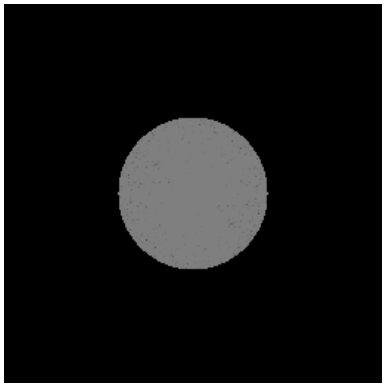
Figure 6-5 shows examples for low pass filtering in the frequency domain. Spatial images reconstructed by the inverse Fourier transform and the magnitude spectra resulting from low pass filtering are displayed. Commonly the pass band of the filter is specified by the filter radius in terms of the total spectrum radius. A low pass filter with radius of 70% for instance designates a filter radius that corresponds to seventy percent of the full spectrum radius.

LPF filter radius at 70%



Corresponding spectrum

LPF filter radius at 40%



Corresponding spectrum

Figure 6-5. Effect of low pass filtering in frequency domain

The ideal low pass filter, in addition to blurring the image which is expected, also introduces wave-like artifacts that are particularly visible if the pass band of the filter is comparably small. This phenomenon is known as ringing and results from the sharp cutoff in the frequency response of the filter. To avoid ringing, low pass filters with frequency responses that smoothly fall to zero are required. It is possible to accomplish such filters through linear, or raised-cosine shaped drop band, or through exclusively defined filter characteristics such as the Butterworth filter.

A high pass filter operates exactly the opposite way to the low pass case: frequencies up to the cut-off frequency are suppressed while larger frequencies are passed. The frequency response of an ideal high pass filter is illustrated in Figure 6-6.

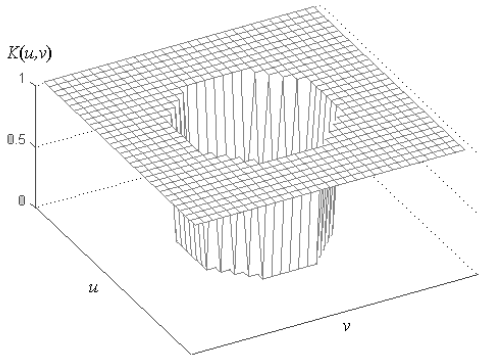
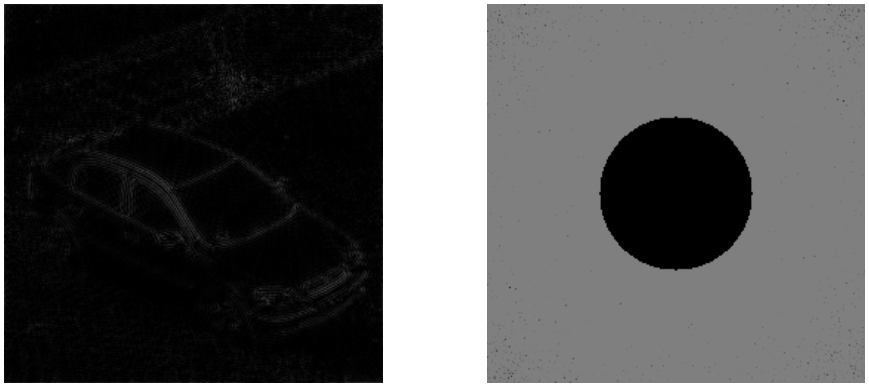


Figure 6-6. Frequency response of an ideal high pass filter

Figure 6-7 shows an example to ideal high pass filtering. The filter radius is expressed as percentage with respect to the full radius of the spectrum. As expected ideal high pass filtering maintains the frequencies that are discarded at the ideal low pass filter with the same filter radius. As a result the high pass filter displays the differences between the low pass filtered image and the original image.



HPF filter radius at 40% Corresponding spectrum

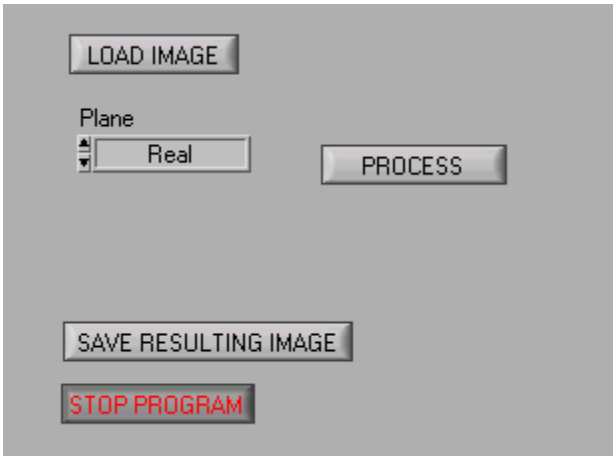
Figure 6-7. Effect of high pass filtering in frequency domain

It is possible to obtain band pass or band stop filtering effects by using a combination of low pass and high pass filters.

Note that besides filtering it is also possible to carry out more advanced image processing tasks in the Fourier domain, making commonly use of Fourier transform properties. A typical example is translation or rotation estimation by comparing the spectra of images.

LabVIEW Demo 6.1: Frequency Domain Representation

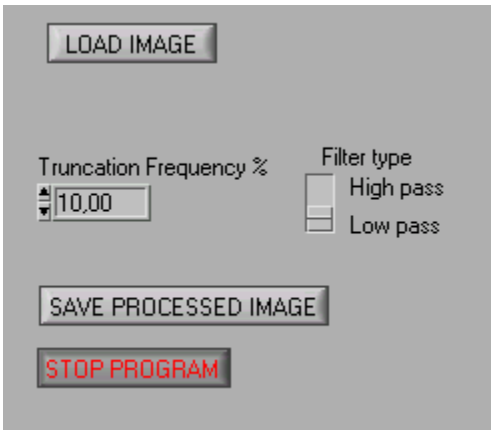
Launch the LabVIEW program entitled `frequencydomain.vi` from the chapter 6 library. Run the program.



Press the “load image” button to select, load and display an image. Select a spectrum plane (Real, Imaginary, magnitude, Phase) and hit the “process” button to display the corresponding plane. Note that planes are normalized to the 8-bit image range 0–255. Observe the spectra of various images with various characteristics.

LabVIEW Demo 6.2: Frequency Domain Filtering

Launch the LabVIEW program entitled `frequencyfilter.vi` from the chapter 6 library. Run the program.

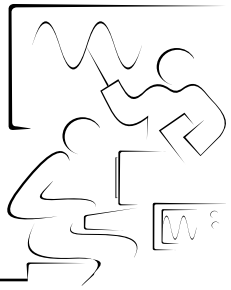


Press the “load image” button to select, load and display an image. Select the filter type as either low pass or high pass. Set the truncation frequency in terms of the radius percentage and observe the result. For the low pass filter start at 100% of the frequency to include all components and progressively reduce the frequency components to detect at what rate ringing becomes visible.

Notes

Notes

Lab 7
Morphological Image Processing



Morphological image processing constitutes a powerful range of non-linear processing techniques. Morphological image processing is based on the idea of probing an image with a small shape or template known as a **structuring element**. The structuring element is positioned at all possible locations in the image and a certain operation is performed according the relation between image content and the structuring element. It is possible to use structuring elements of different shapes to perform a specific task and also define various processing relationships between the structuring element and the image.

While morphological image processing is commonly used on binary images, the approach can as well be extended to gray-scale images. Therefore, morphological image processing is initially explored for binary images and then an extension to grey-scale images is provided.

Morphological Image Processing Essentials

The structuring element can be represented in the form of a matrix with elements either 0 or 1. The size of the structuring element is basically defined by the dimensions of the matrix, and the shape of the structuring element is determined by the pattern of zeros and ones. Figure 7-1 shows for example square, triangle, diamond and cross-shaped 5×5 structuring elements.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Figure 7-1. 5×5 structuring elements with different shapes

Structuring elements need to have an origin. Alike the convolution process, the morphological operation will produce a certain result, which will be taken as the outcome of the process for the image pixel corresponding to the existing location of the origin of the structuring element. Hence the structuring element is commonly defined to have odd dimensions, and the

centre of the matrix is usually utilized as the origin of the structuring element.

When placing a structuring element in a binary image, each of its members is associated with a pixel in the neighborhood under the structuring element. If for all members of the structuring element that are 1, the corresponding image pixel is also 1 the structure element is said to *fit* the image at that position. A structuring element is said to intersect or *hit* an image if not for all but at least one of its members with a value of 1, the corresponding image pixel is also 1. Image pixels for which the corresponding structure element is 0 are not taken into account.

Figure 7-2 shows an example to probing a binary image with a structuring element, where grey boxes denote values of 1 and white boxes values of zero. The structuring element has a size of 3×3 and a square shape. In the figure, the structuring element fits the image at A, hits the image at B, and neither fits nor hits the image at C.

Morphological operations can be defined in relation to a structure element fitting or hitting the image content. In this sense, a morphological operation can be considered as a binary convolution with logical operations.

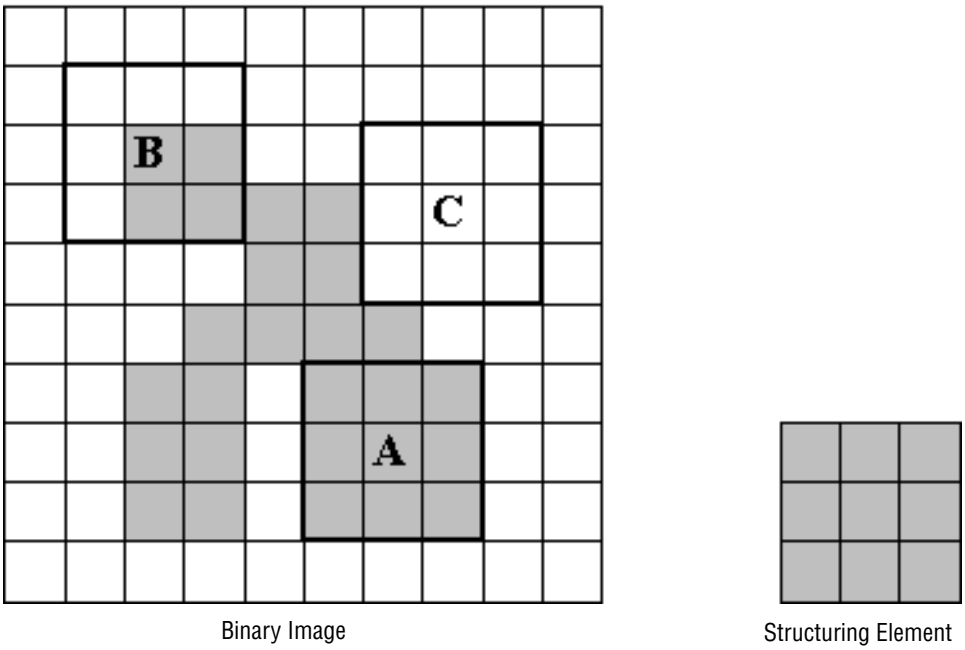


Figure 7-2. Probing of a binary image with a structuring element

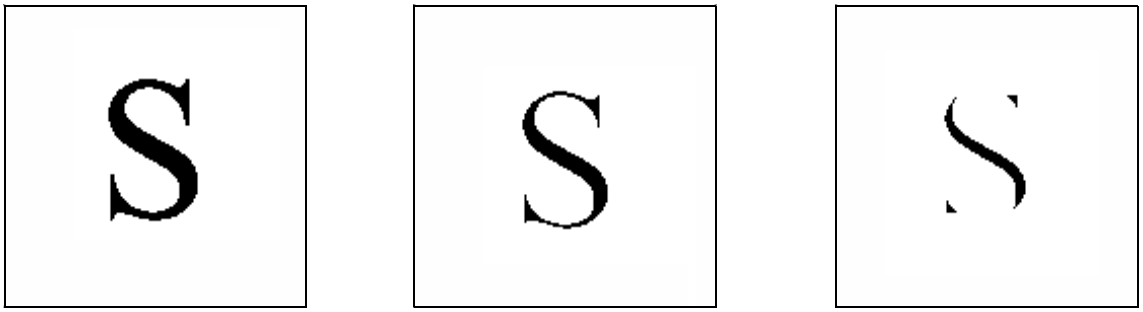
Basic Morphological Operations

The basic morphological operations are erosion and dilation. Most of the other morphological operations can be defined in terms of these two basic operations.

The **erosion** of an image f by a structuring element s can be defined by the rule

$$g(x,y)=\begin{cases}1 & \text{if } s \text{ fits } f\\ 0 & \text{else}\end{cases} \tag{7-1}$$

The erosion process can be denoted as $g = f \otimes s$. Thus probing the image with the structuring element by successively placing the origin of the structuring element to all possible pixel locations, the output is 1 if the structuring element is completely contained in the image (i.e. it fits), 0 otherwise. Figure 7-3 shows the effect of erosion.



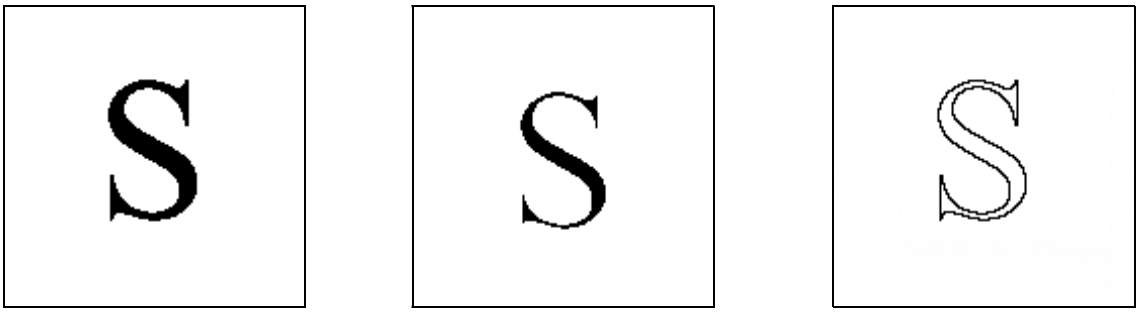
(a) Binary Image

(b) Erosion by 3×3 square structuring element

(c) Erosion by 5×5 square structuring element

Figure 7-3. Binary erosion

Erosion can be used to detect the boundaries in a binary image by subtracting the eroded image from the original image. As the eroded image will lack the boundary pixels of the original image, the result of the subtraction will give the boundaries. Thus it is possible to obtain the boundaries by $g = f - (f \otimes s)$. Figure 7-4 shows an example to this process.



(a) Binary Image

(b) Erosion by 3×3 square structuring element

(c) Difference between the original and eroded image

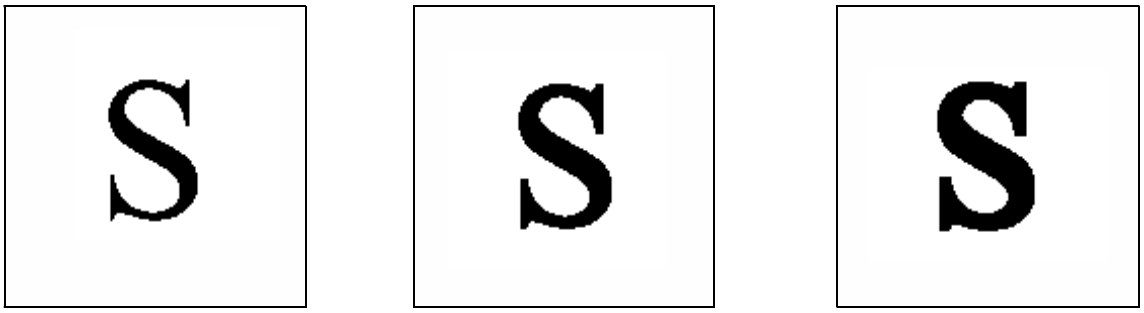
Figure 7-4. Boundary detection using erosion.

The **dilation** of an image *f* by a structuring element *s* can be defined by the rule

$$g(x,y)=\begin{cases}1 & \text{if } s \text{ hits } f \\ 0 & \text{else}\end{cases} \tag{7-2}$$

The dilation process can be denoted as $g = f \oplus s$. Thus probing the image with the structuring element by successively placing the origin of the structuring element to all possible pixel locations, the output is 1 if the structuring element and the image have a nonzero intersection (i.e. the structuring element hits), 0 otherwise. Figure 7-5 shows the effect of dilation. The difference between the original and dilated image will again produce the boundaries.

While the idea of erosion and dilation is fixed, the shape and size of the structuring element will clearly affect the outcome of the morphological process. It is therefore possible to improve the performance for specific applications by employing proper structuring element shapes and sizes. The use of erosion and dilation on their own is however limited, much more effective morphological operations can be achieved by combining these two basic operations.



(a) Binary Image

(b) Dilation by 3×3 square structuring element

(c) Dilation by 5×5 square structuring element

Figure 7-5. Binary dilation

Binary Morphology

In addition to the two basic erosion and dilation operations it is possible to obtain additional morphological operations through the combination of erosion and dilation.

The **opening** operation is defined as an erosion followed by a dilation. The opening of an image *f* by a structuring element *s* can be denoted by $f \circ s$ and expressed in the form of

$$g_{open} = f \circ s = (f \otimes s) \oplus s \tag{7-3}$$

The opening operation has the effect of eliminating small and thin objects, smoothing the boundaries of large objects without changing the general appearance and breaking objects connected only at thin points (i.e. opening objects, thus the name). It is possible to apply the opening process successively a several times to enhance its effect.

The **closing** operation is defined as a dilation followed by an erosion. The opening of an image *f* by a structuring element *s* can be denoted by $f \bullet s$ and expressed in the form of

$$g_{close} = f \bullet s = (f \oplus s) \otimes s \tag{7-4}$$

The closing operation has the effect of filling small and thin holes in an object, connecting nearby objects and generally smoothing the boundaries of large objects without changing the general appearance. It is possible to apply the closing process successively a several times to enhance its effect.

It is possible to combine the idea of erosion and dilation using the **hit–miss** operation. This operation probes the inside and outside of objects at the same time, using two separate structuring elements. A pixel belonging to an object is preserved by the hit-miss operation if the first structuring element translated to that pixel fits the object, and the second translation element misses the object (i.e. fits outside the object). Note that it is assumed that the two structuring elements do not intersect as otherwise it is not possible to meet the requirements at the same time. The hit-miss transform can be expressed in the form of

$$g_{hit-miss} = (f \otimes s_{hit}) / (f \oplus s_{miss}) \tag{7-5}$$

where "/" represents the set difference.

The hit-miss operation is particularly useful for the detection of specific shapes. While the hit structuring element will ensure that all object parts that are the same as the structuring element are preserved, the miss structuring element will ensure that all object parts that are different than the structuring element are removed. Usually the complement of the hit structuring element is by default used for the miss structuring element. Therefore if a single structuring element is utilized, the operation eliminates all pixels that do not have the same pattern as the structuring element.

The **thinning** operation accomplishes erosion without breaking objects. Thinning can be accomplished as a two-step approach, with the first step being an erosion that marks all candidates for removal without actually removing them and in the second pass candidates that do not destroy the connectivity are removed. Thinning can be expressed and realized as the difference between the original object and the hit-miss operation result

$$g_{thin} = f / g_{hit-miss} = f / ((f \otimes s_{hit}) / (f \oplus s_{miss})) \tag{7-6}$$

The **thickening** operation accomplishes dilation without merging nearby objects. Thickening can be accomplished as a two-step approach similar to thinning. Alternatively it is also possible to complement the binary input image and use thinning on the backgrounds, and complement the result. Thickening can be expressed and realized as the union of the original object with the hit-miss operation result

$$g_{thick} = f \cup g_{hit-miss} = f \cup ((f \otimes s_{hit}) / (f \oplus s_{miss})) \tag{7-7}$$

where "$\cup$" represents the set union.

The **gradient-in** operation extracts the interior contours of the object and is simply implemented as the difference between the original image and the eroded version. The **gradient-out** operation extracts the external contours of the object and is simply implemented as the difference between the dilated image and the original image. The **gradient** operation finds internal and external counters together and can simply be implemented by the union of the gradient-in and gradient-out operations.

Morphological operations such as opening and closing can be regarded as morphological filters that remove undesired features from an image. As morphological operations function according to the shape of the structuring element, they can be considered as filters of shape. Furthermore it is possible to filter out image features that are smaller in size than the structuring element.

Grayscale Morphology

While morphological image processing is mainly utilized for binary images, it can as well be extended to grayscale images. For grayscale morphology the operation need to be modified to take the extra dimension provided by pixel grey levels into account. The structuring element is not binary but can take any integer value.

Grayscale erosion is for instance defined as placing the structuring element beneath the grey level image and pushing it as far up as possible without any part of the structuring element exceeding the grey levels of the image. The maximum distance the structuring element can be pushed up is recorded as the new value of the pixel. This distance might be negative, which is the case if the structuring element has pixel values greater than the grey level at the corresponding image pixel. The structuring element might also take negative or zero values. Commonly the output distances are truncated or scaled for instance to a 0–255 range for 8-bit images. Thus grayscale erosion can be implemented to produce the minimum difference between pixel grey levels and the corresponding structure element values. For the special case of a flat structuring element with all elements set to zero value, this process has the effect of a rank filter that selects the minimum pixel value over the structuring element domain.

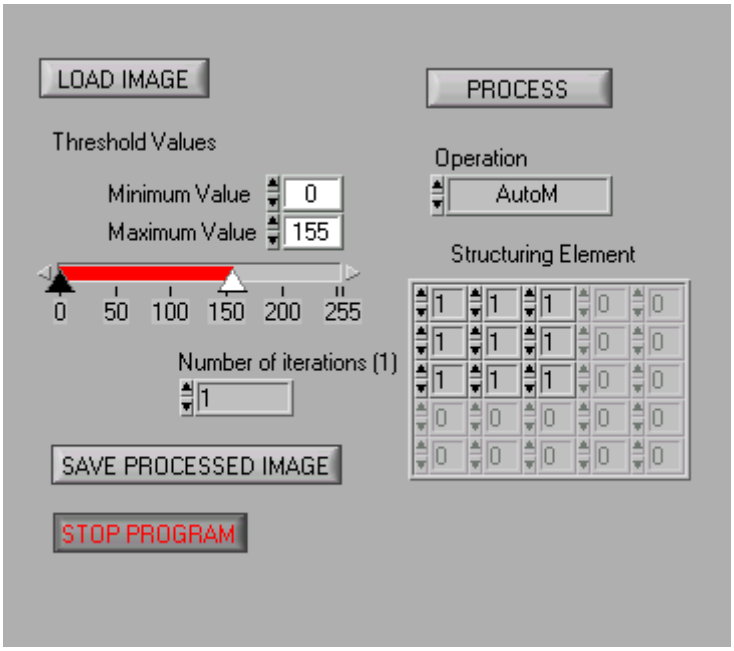
Grayscale dilation is defined in a dual manner to erosion. The structuring element is flipped upside down and the minimum distance required for the structure element to exceed pixel grey levels when pushed upwards is recorded as the output. For a flat structuring element this process is equivalent to a rank filter that selects the maximum pixel value over the structuring element domain.

Grayscale morphology is rarely used compared to binary morphology as the functionality is rather limited and the process is more complicated. Therefore morphological image processing is commonly considered as a binary image processing technique only, and main attention is given to binary operations.

It is possible to accomplish complex tasks by combining these simple morphological operations in specific ways, according to the desired assignment.

LabVIEW Demo 7.1: Binary Morphology

Launch the LabVIEW program entitled `binarymorphology.vi` from the chapter 7 library. Run the program.



Press the “load image” button to select, load and display an image. Thresholding will ensure that the processed image is binary. If a grayscale image is loaded set the thresholds as desired.

Select the operation (AutoM accomplished median filtering) and construct the desired structuring element. Hit the “process” button to display the result of morphological processing. Try all operations on various images with different structuring elements to observe the results.

LabVIEW Demo 7.2: Blob Analysis Example

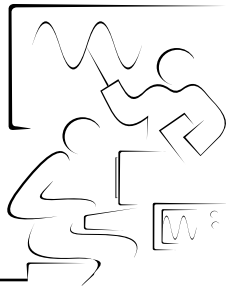
Load the LabVIEW program entitled `Blob Analysis Example.vi` from the IMAQ vision example folders. (`C:\Program Files\National Instruments\LabVIEW 6\examples\Vision\2. Functions\Binary Analysis\Blob Analysis Example.vi`)

Follow the instruction step by step, and observe the uses of morphological image processing.

Notes

Notes

Appendix A
References



[1] Nick Efford, *Digital Image Processing, a practical introduction using Java*, Addison Wesley, USA, 2000.

[2] Anl K. Jain, *Fundamentals of Digital Image Processing*, Prenctice Hall, USA, 1989.

[3] K.R. Castleman, *Digital Image Processing*, Prentice Hall, USA, 1996.

[4] Milan Sonka, Vaclac Hlavac, and Roger Boyle, *Image Processing, Analysis and Machine Vision*, Chapman & Hall, UK, 1993.

[5] HIPR2, Image Processing Learning Resources Web site, `http://www.dai.ed.ac.uk/HIPR2/`

[6] Ramesh Jain, Rangachar Kasturi, and Brian G. Schunck, *Machine Vision*, Mc Graw Hill, Singapore, 1995.

Notes