Table of Contents

Digital Images

An image is a matrix $I(x,y)$ with a predefined number of rows and columns. Images that have only 8 bits per pixel are monochromatic b/w images with 256 tones.

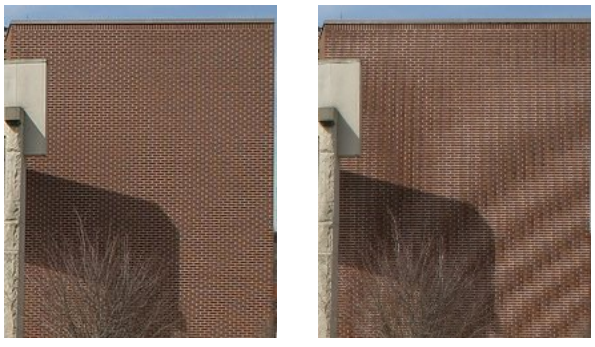
Spatial Sampling

If the distance d represents the distance between CCD elements, then the highest frequency that the camera can capture is:

$$v_c = \frac{1}{2d}$$

If a function contains no frequencies higher than f, it is completely determined by the samples resulting from a sampling rate higher than $2f$. When this is not the case, the high frequencies will alias with lower ones (see picture).

The highest spatial frequency of an image (ignoring the CCD camera) is given by $v_c' = \frac{a}{\lambda f}$, where a is the diameter of the aperture, λ is the light wavelength, and f is the focal length. Spatial frequencies larger than v_c may well generate aliasing, as is demonstrated by the following pictures of a brick wall taken with different resolutions. The left picture is said to be aliased. In general, the situation is described by $v_c < v_c'$, where the difference is usually at least an order of magnitude. As a result, we expect aliasing in imagery in general.



Acquisition Noise

Two images taken by the same camera and of the same scene under the same conditions are never exactly identical. We have come accustomed to regard these noisy variations as probabilistic noise, which we may model with random

variables. Estimating the amount of noise in a sequence of n images of the same scene is usually accomplished with computing pixel-wise averages through the frames of the sequence:

$$\bar{I}(x,y)=\frac{1}{n}\sum_{k=0}^{n-1} I_k(x,y)$$

followed by determining the standard deviation at each pixel:

$$\sigma(x,y)=\left(\frac{1}{n}\sum_{k=0}^{n-1}\left(\bar{I}(x,y)-I_k(x,y)\right)^2\right)^{\frac{1}{2}}$$

This way, we can obtain an estimate of the acquisition noise that a particular visual sensor generates. However, in the general case, pixel values are correlated (CCD temperature, light-wave diffusion, etc). To obtain estimates of this correlation, an auto covariance measure may be used.

Camera Parameters

Two groups of parameters describe cameras. They are intrinsic and extrinsic parameters. Intrinsic parameters are those which link pixel coordinates of an image point with the corresponding coordinates in the reference frame. Extrinsic parameters define the location and orientation of the camera reference frame with respect to a world reference frame. Camera calibration involves finding both the extrinsic and intrinsic parameters of cameras.

A frequent task when using computer vision equipment is to determine the location and orientation of the camera frame with respect to some known reference, with image-related information only. The typical choice for describing the transformation between the camera and the world frames of reference is to model a 3D translation and a 3D rotation as $\vec{T}^T=(T_x,T_y,T_z)$ and

$$R=\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

a rotation matrix (note that this matrix is orthogonal: $R^T R=R R^T=I$). The relation between a world point P_w and its transform into the reference of the camera P_c is given by:

$$P_c=R(P-\vec{T})$$

Intrinsic parameters characterize the optical, geometric, and digital properties of a visual sensor. They are the focal length, optical distortions, and the transform

between camera frame of reference and pixel coordinates onto the imaging plane.
In order to find the transformation from camera to pixel coordinates we need to establish the relation between pixel coordinates $(x_{im}, y_{im})^T$ and the coordinates $(x, y)^T$ of the same point in the reference frame of the camera:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -(x_{im} - o_x) s_x \\ -(y_{im} - o_y) s_y \end{pmatrix}$$

where $(o_x, o_y)^T$ are the pixel coordinates of the optical center, and $(s_x, s_y)^T$ are the effective size of the pixels (CCD elements).

Usually, the optics introduces distortions that are most visible toward the periphery of the image. These are radial distortions, and they can be modeled with the following relation:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_d (1 + k_1 r^2 + k_2 r^4) \\ y_d (1 + k_1 r^2 + k_2 r^4) \end{pmatrix}$$

where $(x_d, y_d)^T$ are the coordinates of the distorted points, and $r^2 = x_d^2 + y_d^2$.

Grouping the sum of these elements together and using the perspective projection equations

$$x = f \frac{X}{Z} \qquad y = f \frac{Y}{Z}$$

we obtain an expression to transform 3D points in the world coordinates into their pixel coordinates onto the imaging surface of the camera. Introducing the transformation from world to camera coordinates and the transformation from camera coordinates to pixel coordinates into the perspective equations yields:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -(x_{im} - o_x) s_x \\ -(y_{im} - o_y) s_y \end{pmatrix} = \begin{pmatrix} f \dfrac{R_1^T (\vec{P}_w - \vec{T})}{R_3^T (\vec{P}_w - \vec{T})} \\ f \dfrac{R_2^T (\vec{P}_w - \vec{T})}{R_3^T (\vec{P}_w - \vec{T})} \end{pmatrix}$$

where R_i is the i^{th} row of the rotation matrix R . If we omit the radial distortion parameters, we can rewrite the complete transformation as a product of two matrices:

$$M_l = \begin{pmatrix} \frac{-f}{s_x} & 0 & o_x \\ 0 & \frac{-f}{s_y} & o_y \\ 0 & 0 & 1 \end{pmatrix}$$

and

$$M_e = \begin{pmatrix} r_{11} & r_{12} & r_{13} - R_1^T \vec{T} \\ r_{21} & r_{22} & r_{23} - R_2^T \vec{T} \\ r_{31} & r_{32} & r_{33} - R_3^T \vec{T} \end{pmatrix}$$

The matrix M_e performs the transform between the world coordinates and the camera reference frame while the matrix M_l performs the transform between the camera frame and the image frame. The complete transformation is written as $\boldsymbol{x} = M_l M_e P_w$.

Image Noise

Image noise is defined as spurious and somewhat random fluctuations of pixel values. A common assumption is that noise is additive and random:

$$\hat{I}(x, y) = I(x, y) + n(x, y)$$

Different types of noise are countered by different techniques. However it is important to note that what is considered noise for one computer vision task might constitute useful information for another.

The amount of noise in imagery is often quantified by a signal-to-noise ratio, such as:

$$SNR = \frac{\sigma_s}{\sigma_n}$$

Alternatively, the signal-to-noise ratio may be expressed in decibels:

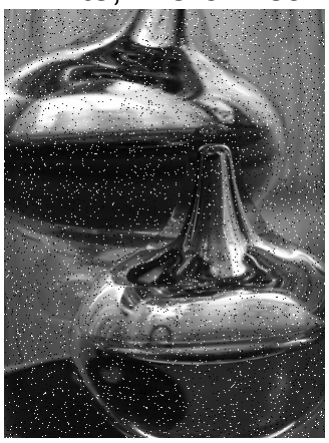
$$10 \log_{10}\left(\frac{\sigma_s}{\sigma_n}\right)$$

In general, an additive model for noise is adequate for most computer vision systems.

Gaussian Noise

The Gaussian model for noise is the most widely used in the absence of noise

information. In this case, we assume that $n(x,y)$ is a white, zero-mean Gaussian stochastic process. That is to say: for each pixel $I(x,y)$ there is a random variable $n(x,y)$ whose behavior is Gaussian.



Impulsive Noise

Impulsive noise, as opposed to Gaussian noise, alters pixels randomly, in such a way as to make their values very different from what they should be. A form of impulsive noise is salt-and-pepper noise:

$$\hat{I}(h,k)=\begin{cases} I(h,k) & \text{if} \quad x<l \\ i_{min}+y(i_{max}-i_{min}) & \text{if} \quad x\geq l \end{cases}$$

where x and y are random variables in the interval $[0,1]$, while l is a parameter controlling the noise density over the image, and i_{max} and i_{min} control the severity of the noise.

Noise Filtering

Given a noisy image, noise filtering attempts to attenuate the noise as much as possible without significantly altering the signal itself. A common technique to noise reduction is linear filtering. Suppose an image $I(x,y)$, and a kernel containing a linear filter $A(h,k)$. The filtered version of the image is obtained by the following discrete convolution:

$$I_A(x,y)=I*A=\sum_{h=\frac{-m}{2}}^{\frac{m}{2}}\sum_{k=\frac{-m}{2}}^{\frac{m}{2}}A(h,k)I(x-h,y-k)$$

where the kernel $A(h,k)$ is of (odd) size $m\times m$. A linear filter replaces the pixel values $I(x,y)$ with a weighted sum of the pixel values from a neighborhood of size $m\times m$ centered at (x,y). The weights are the entries in the kernel $A(h,k)$.

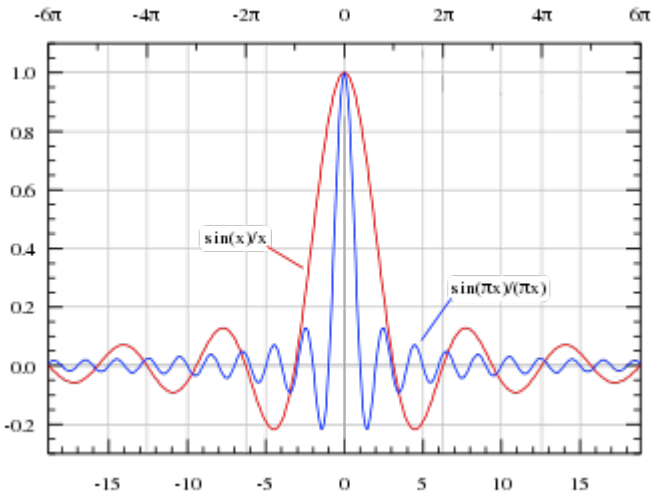
The convolution of two signals (for instance, an image and a kernel) is equivalent to the multiplication of their respective Fourier transforms: $I*A=F^{-1}[F[I]\times F[A]]$, where F and F^{-1} are the Fourier transform and its inverse, respectively.
There are various kernels that can be applied to images to perform different types of image filtering. For instance, an averaging 3 by 3 kernel can be used to attenuate noise. This kernel is:

$$A=\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

This kernel attenuates noise because, in general, noise is a high-frequency component and hence, averaging will attenuate sharp local variations. To understand this, consider the Fourier transform of a 1D box function (which is a 1D profile of the averaging kernel):

$$F[\text{box}(x)]=\frac{2\sin(\omega)}{\omega}$$

The result of this Fourier transform is the sinc function, as depicted on the right. Some problems are associated with attenuating the high frequency contents of images. For example, high frequencies are not always caused by noise. In the particular case of an averaging filter, the secondary lobes of its Fourier transform lets in some high frequency content, which may be undesirable.



The most commonly used approach to noise reduction is Gaussian filtering, which is implemented with a sampled Gaussian function as kernel entries. One of the main reasons this is a common approach is that the Fourier transform of a Gaussian is also a Gaussian, and hence there are no secondary lobes in the spectrum, as opposed to the averaging kernel. In addition, 2D Gaussian functions are separable, which allows performing 2D image convolutions very efficiently. The Gaussian filtered version of an image is obtained by the following discrete convolution:

$$I_G(x,y)=I*G=\sum_{h=\frac{-m}{2}}^{\frac{m}{2}}\sum_{k=\frac{-m}{2}}^{\frac{m}{2}}G(h,k)I(x-h,y-k)$$

and can be separated along the two dimensions as follows:

$$\sum_{h=\frac{-m}{2}}^{\frac{m}{2}}\exp\left(\frac{-h^2}{2\sigma^2}\right)\sum_{k=\frac{-m}{2}}^{\frac{m}{2}}\exp\left(\frac{-k^2}{2\sigma^2}\right)I(x-h,y-k)$$

The programming technique to perform this 2D convolution efficiently is to proceed with the following steps:

1. Convolve in 1D all rows of the original image, and store the result in a temporary image

2. Convolve in 1D all columns in temporary image, and store result as a Gaussian filtered image.

Note that you may start this process with columns and then rows, as this is interchangeable.
To create a 1D Gaussian kernel, we must sample a Gaussian function which has the appropriate variance for the filtering task at hand. The relationship between the variance of the continuous Gaussian and the kernel size is given by $m=5\sigma$, which subtends 98.76% of the area under the Gaussian curve. Note that m must be odd. Add 1 if necessary.

To speed up computations further, we may construct Gaussian kernels that contain integer values instead of floating points. Since the pixels are also represented by integer values, this amounts to performing discrete 1D convolutions in integer arithmetic. The following steps are taken to construct an integer Gaussian kernel:

1. Create a floating point Gaussian kernel

2. Find its smallest value

3. Find the coefficient which scales this value to 1

4. Apply this coefficient to all the values in the kernel

5. Store the resulting kernel entries as integers

6. Compute the sum of these integer kernel values

7. Use this sum as a divisor in front of the kernel

Alternatively, Gaussian filtering can be achieved with repeated average filtering. For instance, convolving a 3×3 averaging kernel n times approximates a Gaussian convolution with a Gaussian kernel built with:

$$\sigma=\sqrt{\frac{n}{3}}$$

and size $2n+3$.

Non-linear filtering can also be performed on images. This type of filtering cannot be applied to images through the means of convolutions. As an example of such filtering, consider a median filter which replaces each pixel value by a median value obtained from a local neighborhood. Clearly, such techniques cannot be implemented using convolutions.