Introduction to Computer Vision

Jean Ponce

Class notes by Antoine Groudiev

Last modified 22nd October 2024

Contents

1 Introduction to Computer Vision **2**

2 Camera Geometry **2**

3 Camera Calibration **2**
3.1 Affine models: weak perspective projection 2

4 Image processing using filters and convolutions **2**
4.1 Filters and convolution 2
4.1.1 Basic filters 2
4.1.2 Convolutions 2
4.1.3 Gaussian filters 3
4.2 Image derivatives 4
4.2.1 Finite differences 4
4.2.2 Smooth derivatives 5
4.2.3 Beyond smooth derivatives 5
4.3 Edge detection 6
4.4 The Canny edge detector 6
4.5 Denoising, sparsity and dictionary learning 6

5 Edge detection **6**

6 Radiometry and Color **6**

7 Color perception and Two-view geometry **6**

8 Epipolar Geometry and Binocular Stereopsis **6**

9 Markov random fields **6**

10 Recovering structure from motion **6**

11 Mean-shift algorithm for segmentation **6**

12 Multi-view object models **6**

13 Neural Networks for Visual recognition **6**

14 Learning methods **6**

Abstract

This document is Antoine Groudiev's class notes while following the class *Introduction to Computer Vision* (Introduction à la vision artificielle) at the Computer Science Department of ENS Ulm. It is freely inspired by the class notes written by Jean Ponce.

1 Introduction to Computer Vision

2 Camera Geometry

3 Camera Calibration

3.1 Affine models: weak perspective projection

4 Image processing using filters and convolutions

An image can be interpreted either as a continuous function $f(x, y)$ or as a discrete array $F_{u,v}$. While many applications, especially in image processing, use the discrete array, the intuition and operations are directly derived from the continuous function setup.

4.1 Filters and convolution

4.1.1 Basic filters

An image can be blurred using a filter, by replacing a point by the average of its neighbors. Blurring an image gives a smoother image, making it easier to compute derivatives.

4.1.2 Convolutions

Given two integrable functions $f, g : \mathbb{R} \to \mathbb{R}$, we can define their convolution as:

$$\begin{aligned} f * g : \mathbb{R} &\longrightarrow \mathbb{R} \\ x &\longmapsto \int_{-\infty}^{+\infty} f(x-t)g(t)dt \end{aligned}$$

Note that $f * g = g * f$ using a change of variable.

This is the definition of the convolution from a continuous perspective. When dealing with images, we want to apply the convolution to a discrete array. The definition becomes:

$$R_{i,j} = (F * G)_{i,j} = \sum_{u,v} F_{i-u,j-v} G_{u,v}$$

Convolution follow basic properties:

Commutativity $f * g = g * f$

Associativity $(f * g) * h = f * (g * h)$

Linearity $(af + bg) * h = af * h + bg * h$

Shift invariance $f_t * h = (f * h)_t$

where $f_t(x) = f(x - t)$. Note that is the only operator that is both linear and shift-invariant.

The convolution can be differentiated:

$$\frac{\partial}{\partial x}(f * g) = \frac{\partial f}{\partial x} * g \tag{4.1.1}$$

In practice, we are dealing with discrete and finite arrays; this causes border issues. When applying the convolution with a $K \times K$ kernel, the result is undefined for pixels closer than K pixels from the border of the image. There are multiple ways to solve this issue: *padding* the image with zeros, *cropping* the result, or *wraping around* the image.

4.1.3 Gaussian filters

Bluring images Gaussian filters are special filters that are used to blur images. Recall that in one dimension, the Gaussian function is defined as:

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

In computer vision, we will mostly use the 2-D Gaussian function:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

In the continuous setup, blurring a function is achieved by convoluting it with a Gaussian function. In the discrete setup, we can build a matrix kernel that approximates the Gaussian function. Note that the Gaussian function has infinite support, but in actual applications, we can truncate the kernel to a finite size.

Gaussian smoothing oftern provides better results than simple averaging. It is also quite effective to remove the noise in an image.

Properties of Gaussian filters Gaussian filters remove "high-frequency" components from the image; therefore, they are low-pass filters. The quantity of noise removed is proportional to the standard deviation σ of the Gaussian kernel. High values of σ will remove more noise but will also blur the image more.

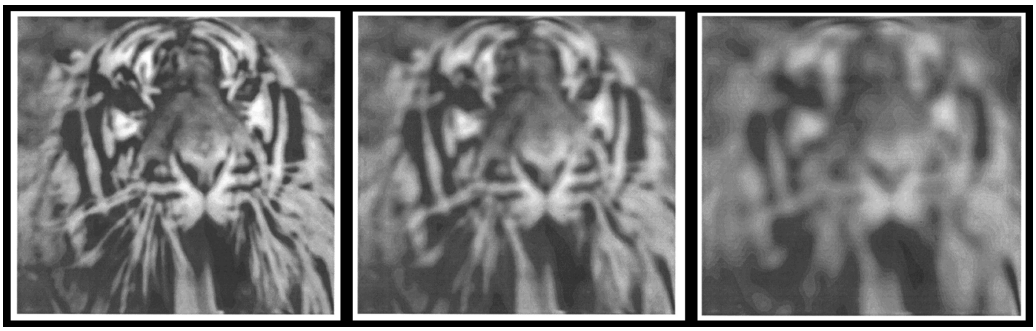

Figure 4.1: Effect of the standard deviation σ on the image.
The parameter σ is increased from left to right.

The combination of 2 Gaussian filters is a Gaussian filter:

$$G_{\sigma_1} * G_{\sigma_2} = G_{\sqrt{\sigma_1^2+\sigma_2^2}}$$

Each filter is separable, meaning that we can apply the filter in the x direction and then in the y direction:

$$G_\sigma * f = g_{\sigma\rightarrow} * g_{\sigma\uparrow} * f$$

This as a critical implication: filtering with a $n \times n$ Gaussian kernel can be implemented as two convolutions of size n, reducing the complexity from $O(n^2)$ to $O(n)$.

Oriented Gaussian Filters By default, G_σ smoothes the image by the same amount in all directions. This has the drawback of blurring edges in all directions, which might make edge detection harder later on in the image processing pipeline. If we have some information about preferred directions, we might want to smooth with some value σ_1 in the direction defined by the unit vector $\begin{bmatrix} a & b \end{bmatrix}$ and by σ_2 in the direction defined by $\begin{bmatrix} c & d \end{bmatrix}$. This can be achieved using:

$$G(x, y) = \frac{1}{C} \exp\left[-\frac{(ax + by)^2}{2\sigma_1^2} - \frac{(cx + dy)^2}{2\sigma_2^2} \right]$$

We can write this in a more compact form using the standard multivariate Gaussian notation:

$$G(x, y) = \frac{1}{C} \exp\left[-\frac{X^\top \Sigma^- 1 X}{2} \right] \quad \text{where} \quad X = \begin{bmatrix} x \\ y \end{bmatrix}$$

The two (orthogonal) directions of filtering are given by the eigenvectors of Σ, the amout of smoothing is given by the square root of the corresponding eigenvalues of Σ.

4.2 Image derivatives

4.2.1 Finite differences

We will see in the next subsection a variety of techniques to solve the *edge detection problem*. A building block of such methods are *image derivatives*: intuitively, we want to be able to measure how much the contrast of the image change locally. Peaks in contrast variation can be somehow interpreted as being close to edges, since this would be the point where the contours of the object contrast with the background.

Therefore, we want to compute at each pixel (x, y) the derivates. In the discrete case, we could take the difference between the left and right pixels:

$$\frac{\partial I}{\partial x} \simeq I[i + 1, j] - I[i - 1, j]$$

This is equivalent as convoluting the image by:

$$\partial_x = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

The problem of this method is that it increases noise. Consider a noise model in which the actual image I can be decomposed as the sum of the true, noiseless, image $\hat{I}$, and a noise n, following for instance a normal distribution. When then have $I = \hat{I} + n$, and we obtain:

$$\underbrace{I[i + 1, j] - I[i - 1, j]}_{\text{Actual image values}} = \underbrace{\hat{I}[i + 1, j] - \hat{I}[i - 1, j]}_{\text{True difference}} + \underbrace{n_+ + n_-}_{\text{Noises}}$$

Where $n_+ - n_-$ follows a normal distribution of larger variance, providing therefore more noise on the derivate image.

4.2.2 Smooth derivatives

A solution is to first smooth the image by a Gaussian G_σ, and *then* take derivatives:

$$\frac{\partial f}{\partial x} \simeq \frac{\partial G_\sigma * f}{\partial x}$$

Applying the differentiation property of the convolution (4.1.1):

$$\frac{\partial f}{\partial x} \simeq \frac{\partial G_\sigma}{\partial x} * f$$

Therefore, taking the derivative in x of the image can be done by applying a convolution with the derivative of a Gaussian:

$$\frac{\partial G_\sigma}{\partial x} = \frac{1}{C} \cdot x \exp\left[-\frac{x^2+y^2}{2\sigma^2}\right]$$

Another crucial property is that the Gaussian derivative is also separable, reducing drastically the computational cost.

Smoothing before the derivative improves the results by reducing the noise, but still blurs away the edge information. In practice, there is always a tradeoff to find between smoothing and good edge localization.

4.2.3 Beyond smooth derivatives

Other methods are sometimes used in practice to overcome the limitations detailled above. *Directional derivatives* are the equivalent of directional smoothing; we output the following quantity

$$\cos\theta \frac{\partial G_\sigma}{\partial x} + \sin\theta \frac{\partial G_\sigma}{\partial y}$$

This allows to avoid the smoothing of the edges while keeping the differentiation in directions that matter.

Second-order methods can also prove effective. This is a non-separable method, approximated by a difference of Gaussians. The output of the convolution is the Laplacian of the image; zero-crossing correspond to edges:

$$\nabla^2 G_\sigma(x,y) = \frac{\partial^2 G_\sigma(x,y)}{\partial x^2} + \frac{\partial^2 G_\sigma(x,y)}{\partial y^2}$$

4.3 Edge detection

4.4 The Canny edge detector

4.5 Denoising, sparsity and dictionary learning

5 Edge detection

6 Radiometry and Color

7 Color perception and Two-view geometry

8 Epipolar Geometry and Binocular Stereopsis

9 Markov random fields

10 Recovering structure from motion

11 Mean-shift algorithm for segmentation

12 Multi-view object models

13 Neural Networks for Visual recognition

14 Learning methods