Introduction to Computer Vision

Jean Ponce

Class notes by Antoine Groudiev

Last modified 27th October 2024

Contents

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

Camera calibration is the process of estimating the parameters of a camera model that relate the 3D world coordinates of a point to its 2D image coordinates. The camera parameters are represented in a *camera matrix* of shape 3×4, which essentially projects a 3D point (in homogeneous coordinates) to a 2D point.

The camera matrix is composed of two matrices: the *intrinsic matrix* and the *extrinsic matrix*. The intrinsic matrix contains the parameters that are specific to the camera, such as the focal length, the principal point, and the skew coefficient. The extrinsic matrix contains the parameters that describe the position and orientation of the camera in the world coordinate system.

3.1 Least squares calibration

Consider a set of n points $p_i \in \mathbb{R}^3$ and $P_i \in \mathbb{R}^4$, respectively the homogeneous coordinates of the 2D and 3D points that we will use for calibration:

$$p_i = \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \quad \text{and} \quad P_i = \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}$$

Each point p_i corresponds to the 2D projection of the 3D point P_i in the camera frame. We want to estimate the camera matrix $M \in \mathscr{M}_{3,4}(\mathbb{R})$, such that:

$$\forall i \in [\![1, n]\!], \quad p_i = \frac{1}{z_i} M P_i \tag{3.1.1}$$

Let's denote m_1, m_2, and m_3 the rows of M, such that:

$$M = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix}$$

We can rewrite (3.1.1) as:

$$\forall i \in [\![1, n]\!], \quad u_i = \frac{\frac{1}{z_i} m_1 P_i}{\frac{1}{z_i} m_3 P_i} \quad \text{and} \quad v_i = \frac{\frac{1}{z_i} m_2 P_i}{\frac{1}{z_i} m_3 P_i} \tag{3.1.2}$$

By clearing the denominators, we see that (3.1.2) is equivalent to:

$$\forall i \in [\![1, n]\!], \quad A_i X = 0_2 \tag{3.1.3}$$

where

$$A := \begin{bmatrix} P_i^\top & O_4^\top & -u_i P_i^\top \\ O_4^\top & P_i^\top & -v_i P_i^\top \end{bmatrix} \in \mathscr{M}_{2,12}(\mathbb{R}) \quad \text{and} \quad X := \begin{bmatrix} m_1^\top \\ m_2^\top \\ m_3^\top \end{bmatrix} \in \mathbb{R}^{12}$$

In practice, it is likely that (3.1.3) will not be exactly satisfied for all $i \in [\![1, n]\!]$, due to noise in the measurements. Therefore, we will not solve this exact problem, but instead solve the following minimisation problem:

$$\hat{X} = \underset{\|X\|^2=1}{\arg\min} \|\sum_i A_i X\|_2^2 = \underset{\|X\|^2=1}{\arg\min} \|AX\|_2^2 \tag{3.1.4}$$

where:

$$A := \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} = \begin{bmatrix} P_1^\top & O_4^\top & -u_1 P_1^\top \\ O_4^\top & P_1^\top & -v_1 P_1^\top \\ \vdots & \vdots & \vdots \\ P_n^\top & O_4^\top & -u_n P_n^\top \\ O_4^\top & P_n^\top & -v_n P_n^\top \end{bmatrix} \in \mathscr{M}_{2n,12}(\mathbb{R})$$

This can be solved using the *Singular Value Decomposition* (SVD) of A. If $A = U\Sigma V^\top$ is the SVD of A, then the solution $\hat{X}$ is given by the last column of V. Given $\hat{X}$, we can then extract the estimation of the camera matrix $\hat{M}$, by reshaping $\hat{X} \in \mathbb{R}^{12}$ into a 3×4 matrix.

Remark. *Estimating M does not give any explicit information on z_i; when projecting into the 2D plane, the u and v coordinates can be recovered by normalizing the projection by the third estimated value, which rescales by ensuring that the third coordinate is equal to 1.*

3.2 Parameter decomposition

Once the camera matrix M has been estimated, we can decompose it into its intrinsic and extrinsic parameters. One can show that any projection matrix M can be decomposed into the following form:

$$M = \mathcal{K}[R|t]$$

where:

$$\mathcal{K} = \begin{bmatrix} \alpha & -\alpha \cot\theta & u_0 \\ 0 & \beta/\sin\theta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{and} \qquad R = \begin{bmatrix} r_1^T \\ r_2^T \\ r_3^T \end{bmatrix}$$

Here, we denote by $r_1, r_2, r_2 \in \mathbb{R}^3$ the rows of the rotation matrix, and θ the skew rotation angle.

For a certain scale factor $\rho \in \mathbb{R}$, it is shown that:

$$\rho M = \begin{bmatrix} \alpha r_1 - \alpha \cot\theta r_2 & \alpha t_x - \alpha \cot\theta t_y + u_0 t_z \\ \frac{\beta}{\sin\theta} r_2 + v_0 r_3 & \frac{\beta}{\sin\theta} t_y + v_0 t_z \\ r_3 & t_z \end{bmatrix}$$

The normalisation by ρ comes from the fact that the Frobenius norm of the rotation matrix is equal to 1. Furthermore, we know from the rotation matrices properties that they are orthogonal ($R^\top R = I_3$), and that the determinant of a rotation matrix is equal to 1. Hence, we have:

$$\forall i \in [\![1, 3]\!], \quad \|r_i\|^2 = 1$$

If we write $M = [\mathcal{A}b]$ with $b \in \mathbb{R}^3$ the last column of M, and $a_1, a_2, a_3 \in \mathbb{R}^3$ the columns of $\mathcal{A}$, we have by identification that:

$$\rho a_3 = r_3$$

Hence, we can deduce the scale factor ρ:

$$\rho = \frac{\varepsilon}{\|a_3\|}$$

where $\varepsilon = \pm 1$. The choice of ε determines the orientation of the camera. In practical situations, the sign of t_z is often known, since it corresponds to knowing whether the origin of the world coordinate system is in front of or behind the camera.

Using the orthogonality of the rotation matrix, we have that:

$$r_i^\top \cdot r_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Hence:

$$r_3^\top (\alpha r_1 - \alpha \cot\theta r_2 + u_0 r_3) = (\rho a_3) \cdot (\rho a_1)$$

Which gives us:

$$u_0 = \rho^2 a_3 \cdot a_1$$

Similarly, we can find that:

$$v_0 = \rho^2 a_3 \cdot a_2$$

It can also be shown that we can recover the skew rotation angle θ by computing:

$$\cos\theta = \frac{(a_1 \times a_3) \cdot (a_2 \times a_3)}{||a_1 \times a_3||||a_2 \times a_3||}$$

The skew rotation angle can later be used to retrieve the values of α and β:

$$\alpha = \rho^2 ||a_1 \times a_3|| \sin\theta$$

$$\beta = \rho^2 ||a_2 \times a_3|| \sin\theta$$

We then have all the elements to compute the lines of R:

$$\begin{cases} r_1 = \frac{a_2 \times a_3}{||a_2 \times a_3||} \\ r_3 = \rho a_3 \\ r_2 = r_3 \times r_1 \end{cases}$$

Finally, we can recover $t = \rho \mathcal{K}^{-1} b$. This method provides an implementation-ready way to decompose the camera matrix. In practice, the only unknown is the sign of the scale factor, ε.

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

We will see in the next chapter 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.

4.2.1 Finite differences

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:

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

5 Edge detection

5.1 Gradient-based edge detection

The edge detection problem aims at identifying the *edges* inside an image; this requires a proper definition of "edge", which often depends on the method used to compute it. Intuitively, an edge is a discontinuity of intensity in some direction; like explained previously, it could be detected by looking for place where the derivatives of the iamge have large values.

Gradient-based edge detectors run into three major issues:

1. The gradient magnitudes at different scalres are different: which one should we choose?
2. tThe gradient magnitude is large along thick trails: how do we identify the significant points?
3. How do we link the relevant points up into curves?

Another way to detect an extremal first derivative is to look for a null second derivative. In practice, applying a Laplacian method always require smoothing with a Gaussian kernel first. The method goes as follows: smooth the image, apply the Laplacian, and mark the zero points where there is a sufficiently large derivative, and enough contrast.

5.2 The Canny edge detector

5.3 Denoising, sparsity and dictionary learning

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