[12] D. Lowe, Fitting Parametrized Three-Dimensional Models to Images, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-13, pp. 441-450, (1991).

[13] A.D. Marshall and R.R. Martin, *Computer Vision, Models and Inspection*, World Scientific, London (1992).

[14] J.E.W. Mayhew and J.P. Frisby, *3D Model Recognition from Stereoscopic Cues*, MIT Press, Cambridge (MA) (1991).

[15] H. Murase and S.K. Nayar, Visual Learning and Recognition of 3-D Objects from Appearance, *International Journal of Computer Vision*, Vol. 14, pp. 5-24 (1995).

[16] M.J L. Orr, On Estimating Rotations, DAI Working Paper 233, Dept. of Artificial Intelligence, University of Edinburgh, (1992).

A

Appendix

In this appendix, we give you some details on a number of facts needed to read and make profitable use of this book.

A.1 Experiments: Good Practice Hints

This section is a concise introduction to the *performance assessment* of computer vision programs, and is concerned mainly with the *design of experimental tests*. The purpose is only to provide a list of good-practice principles; be warned that the foundations of *sound* experimental testing are in statistics! Some references are provided at the end; notice the special issue of Machine Vision and Application journal on performance characterization of vision algorithms. In addition to the Web sites quoted in Chapter 1, check out the informative site on performance assessment techniques for computer vision systems and components, `http://pandora.imag.fr/ECVNet/benchmarking.html`, maintained by the European Computer Vision Network (ECV Net). Another interesting site describes HATE, a tool for generating scripts for testing computer vision programs (`http://peipa.essex.ac.uk/hate/`).

Why an Appendix on Experiments?

The reason for an appendix on experiments in a textbook on computer vision is twofold. First and foremost, computer vision is an experimental discipline: the ultimate proof of its theories is whether or not a system can be built which performs as predicted. It is therefore essential to know how to design experiments assessing whether an implementation does what expected, and how well. Second, the literature of computer vision is punctuated by algorithms supported by unconvincing or even scanty experimental evidence; it is important to recognize and avoid such bad practice.

Experimental Performance Assessment

Testing is not just about taking separate measurements, but *discovering how a system behaves as its parameters and experimental conditions vary*. This behavior can be investigated theoretically and experimentally; here we are interested in the latter method. Our objective is therefore *to assess experimentally and quantitatively how well a computer vision program performs its task*.

To achieve this objective, we must

1. identify the key variables and parameters of the algorithm;
2. predict the behavior of the output variables as functions of the parameters and input variables;
3. devise a systematic testing procedure, checking the results of the tests against expected values.

We discuss briefly each point in the next sections. In essence, we run the program undergoing testing in a large number of conditions, and record the value of target variables, their discrepancies from the expected true values, or *errors*, and the error statistics. If the true values are not known in advance, we can still measure the difference between each measure and its average, and its statistics.

Identifying the Key Variables

The key variables of an algorithm are:

- its *input variables*;
- its *output* or *target variables*;
- its *parameters*, for instance constants and thresholds.

We must also consider quantifiable experimental conditions which can affect the results of tests (e.g., illumination direction, uncertainties on input variables), that we shall call *experimental parameters*.

The nature of the target variables and their associated errors depends on the algorithm. For the purposes of this discussion, we identify two broad classes of algorithms, aimed respectively at *measurement* or *detection*.

Measurement Algorithms. If an algorithm outputs measurements like lengths or areas, such measurements are the target variables. In this case, typical errors used are

- the *mean absolute error*,

$$e_{ma} = \frac{1}{N}\sum_{i=0}^{N} |\hat{x}_i - x_T|,$$

where N is the number of tests, $\hat{x}_i$ is the value of the target variable, x, measured at the i-th test, and x_T the variable's true value;

- the *mean error*,

$$e_m = \frac{1}{N}\sum_{i=0}^{N} (\hat{x}_i - x_T);$$

- the *RMS (root mean square) error*,

$$e_{RMS} = \sqrt{\frac{1}{N}\sum_{i=0}^{N} (\hat{x}_i - x_T)^2}.$$

Detection Algorithms. If an algorithm is meant to detect instances of a particular entity, the errors above do not make sense; instead, we observe

- the numbers of *false positives*; that is, spurious responses not corresponding to real instances.
- the number of *false negatives*; that is, real instances missed by the algorithm.

☞ Notice that the quantities above are estimates of the *a posteriori* probability of false positives and false negatives.

Predicting the Behavior of the Output Variables

There are two basic cases, depending on the nature of the input data:

1. the test data are *synthetic*
2. the test data are *real*.

Synthetic data are simulated data sets. They allow us to run our program in perfectly controlled situations; as we design the data, we know the value of *all* input variables and parameters (sometimes called *ground truth*), as well as the exact value of the corresponding output variables.

Real data are obtained from real images. When testing with real data, you should endeavor to control the input variables, parameters and experimental parameters, so that results can be compared with the predictions of synthetic tests.

☞ Synthetic data are useful as long as they mirror realistic situations as closely as possible. It is good practice to begin testing with such synthetic data, then use the results to guide the choice of parameters and input variables to be used in tests with real data. It is *definitely bad practice* to claim that "the program works well" because "the results look good" with a few, uncharacterized real images.

Designing Systematic Tests

The next step is to design systematic experiments to investigate the behavior of the program as input variables, parameters and experimental parameters vary. Again, the procedure depends on the nature of data; we begin with synthetic data.

Synthetic Data.

Reference Procedure with Synthetic Data

1. Choose a realistic range of variation for each input variable, algorithm parameter, and experimental parameters. Discretize each range using sampling steps small enough to guarantee a significant sampling of the behavior of the output variables, but large enough to make the total number of runs in step 4 (below) feasible in reasonable times, given the computational resources available.
2. Select a model of random, additive noise to be added to the ideal data. The model (statistical distribution and its parameters) should reflect the real noise measured in the application; in the absence of such information, use Gaussian noise plus outliers. Choose a realistic range for the amount (standard deviation) of the noise, and discretize such range according to the criteria of point 1.
3. For each possible combination of values of input parameters, experimental parameters and amounts of noise:
   (a) generate the data set corresponding to the current values, and corrupt it with a number of different realizations of noise of the current amount;
   (b) run the program on all the noisy data sets obtained;
   (c) store the average values of the target variables together with the current values of noise amount, input variables, algorithm parameters, and experimental parameters.
4. Compute global statistics (over all tests) for the errors of the target variables.

It is convenient to show the results as graphs plotting the error statistics against variations of input variables, program and experimental parameters, and noise levels.

☞ As we cannot test the program in *all* possible situations, a practical question arises: What is the minimum number of tests guaranteeing that the results are representative for the general behavior of the program, within a given level of confidence? An exhaustive answer comes from statistics, and the considerations required are beyond the scope of this appendix. Refer to the Further Readings (for instance Lapin's book) for the whole story.

Real Data. Ultimately, the algorithm must work with real data from the target application. Hence, you should test your program with real, controlled images. What and how much you can control in real data depends on the application; for the purposes of performance assessment, you should know in advance reliable estimates of input and output variables, and experimental parameters. This includes the statistical properties of the image noise (Chapter 3) and the uncertainty on any quantity, which is input to the program.

In these assumptions, the reference algorithm for synthetic data can be adapted and applied. You cannot vary values as you please any more, but you can certainly use the results of the synthetic tests to identify key values for each controllable range, and run real tests for the selected combinations of input variables and parameters.

References

R.M. Haralick, C. Lee, K. Ottenberg, and M. Nolle, Analysis of Solutions of the Three Point Perspective Pose Estimation Problem, *Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition*, pp. 592-598 (1991).

L.W. Lapin, *Probability and Statistics for Modern Engineering*, 2nd edition, PWS-Kent, Boston (MA) (1990).

Machine Vision and Applications, Special Issue on Performance Characteristics of Vision Algorithms, Vol. 9, no. 5/6 (1997).

A.2 Numerical Differentiation

This section reminds you of some basic formulae for numerical derivatives, and how to apply them to digital images. Given our definition of digital image and the needs of the algorithms in this book, we limit ourselves to *first and second derivatives* in the case of *equally spaced data*. To dispel any potential impression that the problem is solved by a few, straightforward formulae, we list in the last section a few of the many dangers lurking in the perilous land of numerical differentiation.

Deriving Formulae for Numerical Derivatives

Consider a real-valued function, $f(x)$, of which we know only N samples taken at equidistant points, $x_1, \ldots, x_N$, such that $x_i = x_{i-1} + h$ for some $h > 0$. We want to compute finite-difference estimates of f' and f'', the first and second derivatives of f, as well as the error on these estimates[1]. To solve the problem we write Taylor's polynomial approximation of $f(x)$ at $x + h$ and $x - h$:

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + O(h^3) \tag{A.1}$$

$$f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) + O(h^3) \tag{A.2}$$

Subtracting (A.2) from (A.1) and solving for $f'(x_0)$ we get the desired formula for the first derivative:

$$f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h} + O(h^2).$$

The quantity $O(h^2)$ means that the *truncation error*, caused by stopping the polynomial approximation to second order, tends to zero as h^2 as h tends to zero. Similarly, summing up (A.1) and (A.2) and solving for $f''(x_0)$ we get the desired formula for the second derivative:

[1] Notice that we *assume* that f' and f'' exist, but this may not be true at some points; the samples do not really tell us!

$$f''(x_0) = \frac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{h^2} + O(h).$$

We can derive finite-difference formulae in which the truncation error vanishes more rapidly if we begin with higher-order polynomial approximations (which involve more samples).[2] Here is a summary of the useful formulae and their truncation errors, in the notation introduced at the beginning of the section.

First Derivatives: Central-difference Approximations

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h} + O(h^2)$$

$$f_i' = \frac{-f_{i+2} + 8f_{i+1} - 8f_{i-1} + f_{i-2}}{12h} + O(h^4)$$

Second Derivatives: Central-difference Approximations

$$f_i'' = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + O(h)$$

$$f_i'' = \frac{-f_{i+2} + 16f_{i+1} - 30f_i + 16f_{i-1} - f_{i-2}}{12h^2} + O(h^3)$$

☞ With digital images, one nearly invariably sets $h = 1$.

The equations above are called *central-difference approximations*, as they estimate the derivatives at x_i using samples from a symmetric interval centered on x_i. One can use asymmetric intervals instead, leading to the so-called *forward* and *backward approximations*; for instance, for the first derivatives,

$$f'(x_0) = \frac{f_{i+1} - f_i}{h} + O(h)$$

$$f'(x_0) = \frac{f_i - f_{i-1}}{h} + O(h)$$

are the forward and backward approximation respectively. Notice that these formulae carry larger truncation errors; therefore, *central differences should be your choice whenever derivatives must be estimated with simple formulae* (see the next subsection for hints to more sophisticated methods).

[2] Again, this implies the *assumption* that higher-order derivatives of f exist.

Computing Image Derivatives

One of the nice features of the formulae above is that *they allow us to compute image derivatives by convolution* (see algorithm LINEAR_FILTER in Chapter 3). For instance, suppose you want to estimate the image gradient, $\nabla I = [\partial I/\partial x, \partial I/\partial y]^\top$, at all pixels (i, j). You just convolve the rows and columns with the mask

$$[1 \quad 0 \quad -1]$$

and this implements the first formula in the box at all image points. Such masks are sometimes called *stencils*.

☞ Notice that the result can be represented as two images, one for each gradient component, assuming we allow noninteger pixel values (and do not consider peripheral pixels).

Building the masks implementing the other formulae in the box is left as an exercise. You may also want to try and show that the stencil for the Laplacian of $f = f(x, y)$,

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2},$$

is

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

A Word of Caution

What we said so far allows you to take numerical derivatives with simple formulae, but is just the tip of the iceberg; a word of caution must be spent on *errors*. The total error of a numerical approximation is due to two sources, *roundoff errors* and *truncation errors*.

This section has considered only the latter. The former is due to the fact that most numbers are not represented exactly by binary codes in the computer; this can lead to significant errors in the result of even simple computations. Moreover, it turns out that there is an optimal value of h minimising the total error of a given derivative of f at x_i; to make things worse, this value varies, in general, with x_i. Since we do not control the spatial sampling of a digital image, we must expect that the errors associated with numerical image derivatives vary across the image.

☞ A useful take-home message is: *the fractional accuracy of the simple finite-difference approximations in the box is always worse than the fractional accuracy with which the function can be computed* (which, in turn, is generally worse than the machine accuracy).

Better accuracies can be obtained with more sophisticated methods. For instance, one can fit a spline to the data, and compute the derivatives from the spline's coefficient. A good, concise introduction to these methods (including C code) is given by the Numerical Recipes.

References

C.F. Gerald and P.O. Wheatley, *Applied Numerical Analysis*, Fourth edition, Addison-Wesley, Reading (MA) (1970).

W.H. Press, S.A. Teulosky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C*, 2nd edition, Cambridge University Press, Cambridge (UK) (1992).

A.3 The Sampling Theorem

This section states the celebrated *Sampling Theorem* and discusses a few related concepts. This should give you the minimum knowledge necessary to go through Chapters 2 and 3. But if this is the first time you hear about the Sampling Theorem, you are strongly encouraged to read more about it.

The Theorem

Let $F(\omega)$ be the Fourier transform of a function $f(t)$, with $t \in (-\infty, \infty)$. We assume that f is *band-limited*; that is, $F(\omega) = 0$ for $|\omega| > \omega_c > 0$. Then the following theorem holds true.

Sampling Theorem

The function f can be *exactly* reconstructed for every $t \in (-\infty, +\infty)$ from a sequence of equidistant samples, $f_n = f(n\pi/\omega_c)$, according to the formula

$$f(t) = \sum_{n=-\infty}^{+\infty} f_n \frac{\sin(\omega_c t - n\pi)}{\omega_c t - n\pi} = \sum_{n=-\infty}^{+\infty} f_n sinc(\omega_c t - n\pi). \qquad \text{(A.3)}$$

☞ The Sampling Theorem is nontrivial in at least three respects. First, it tells you that any bandlimited function can be reconstructed *exactly* at locations where it has not been observed. Second, the reconstruction relies upon discrete observations. Third, the information content of a function is not "local": as shown by (A.3), the reconstruction of any value $f(t)$ receives finite contributions from *all* samples.

A few important remarks now follow.

Aliasing and the Nyquist Frequency

The frequency $\nu_c = \omega_c/\pi$, inverse of the sampling interval $T_c = \pi/\omega_c$, is named *Nyquist frequency* and is typical of the signal. It is *the minimal sampling frequency necessary to reconstruct the signal*. This means that, for $\omega < \omega_c$, the series

$$f_\omega(t) = \sum_{n=-\infty}^{+\infty} f_n \frac{\sin(\omega t - n\pi)}{\omega t - n\pi}$$

does *not* converge to $f(t)$. The difference between f and f_ω, that is, the *reconstruction error*, is due to the fact that the sampling distance is too coarse to capture the higher frequencies of the signal. This phenomenon is called *aliasing* because f_ω, the reconstruction of the original signal f, is corrupted by higher frequencies which behave as if they *were* lower frequencies.

If the Function is Not Band-limited

How strong is the assumption that f is band-limited? In practice, not very: After all, because of the integrability conditions, the Fourier transform of any function (if it exists) must be very close to zero outside some finite frequency range. The problem is, one does not necessarily know in advance the value of ω_c, and it may well be the case that π/ω_c is too small with respect to the finest sampling distance achievable in practice. The consequence of this fact can be appreciated by means of the following example.

Assume you are given a sequence of equidistant samples $\ldots, g(-T), g(0), g(T), g(2T), \ldots$ of a function $g(t)$. You don't know whether or not g is band-limited. If you let $\omega = \pi/T$, the series

$$\sum_{n=-\infty}^{\infty} g_n \frac{\sin(\omega t - n\pi)}{\omega t - n\pi}$$

converges to a band-limited function $\tilde{g}$ with $\omega_c = \omega$ (the proof of this fact is left to you as an exercise). If g is not band-limited, or perhaps its band is larger than 2ω, g and $\tilde{g}$ differ, and the difference increases with the amplitude of G, the Fourier transform of g, outside the band $[-\omega, \omega]$.

Function Reconstruction

Perhaps a weakness of the Sampling Theorem is that the reconstruction given by (A.3) converges rather slowly. The function $\sin x/x$ goes to zero as $1/x$ for $x \to \infty$. This means that samples far away from the location where the reconstruction takes place might still give important contributions.

It is instructive to evaluate the derivative of a band-limited function $f(t)$ making use of (A.3). To compute $f'(0)$, the derivative of $f(t)$ at $t = 0$, we take the derivative of both sides of (A.3), set $t = 0$, and obtain

$$f'(0) = \frac{\omega_c}{\pi} \sum_{n \neq 0} (-1)^{n+1} \frac{f_n}{n},$$

where n ranges from $-\infty$ to $+\infty$ (with the exception of $n = 0$, where the derivative of $sinc(x)$ vanishes). Thus for $\omega_c = \pi$ (the typical computer vision setting in which the pixel width is one) we have

$$f_i' = f_{i+1} - f_{i-1} + \frac{f_{i+2} - f_{i-2}}{2} + \ldots \frac{f_{i+n} - f_{i-n}}{n} + \ldots.$$

This formula should be compared with the stencils proposed in section A.2.

References

A. Papoulis, *The Fourier Integral and Its Applications*, McGraw-Hill, New York (1962).

A.4 Projective Geometry

In this section, which is not meant to be a rigorous introduction to projective geometry, we give you the minimum information necessary to go through the projective material of the book. We first define projective transformations and standard bases, then discuss briefly the most important projective invariant, the cross-ratio.

Definitions

The projective geometry immediately relevant for computer vision deals with points, lines and their relations in 2-D and 3-D. In this section, we discuss the main concepts in the *planar case*; the extension to the 3-D case is straightforward.

The Projective Plane

We begin by defining the *projective plane*.

Definition: Projective Plane

The *projective plane*, P^2, is the set of equivalence classes of triplets of real numbers (not all zero), where two triplets, $\mathbf{p} = [x, y, z]^\top$ and $\mathbf{p}' = [x', y', z']^\top$, are equivalent if and only if

$$[x, y, z]^\top = \lambda [x', y', z']^\top,$$

where λ is a real number.

A point $\mathbf{p} \in P^2$ is thus identified by three numbers, called *homogeneous coordinates*, defined up to an undetermined factor. This redundant representation allows us to develop a more general geometry than Euclidean geometry. We retain only the elementary concepts of *point*, *line*, and *incidence*, but we do not talk of *angles* and *lengths*.

A Useful Model

A useful model of the projective plane can be obtained in 3-D space: each projective point $\mathbf{p}$ is put in correspondence with a 3-D line through the origin. The proof that this is a *faithful* model of the projective plane is left to you as an exercise.

In this setting, all 3-D lines (or, equivalently, all points of P^2) stand on an equal footing. Instead, if we cut the bundle of 3-D lines in our model with a plane, π, not going through the origin (say the plane of equation $z = 1$), we can distinguish between *proper* and *improper* points:

- each point of the projective plane with $z \neq 0$ is a *proper* point, identified by the coordinates $[x/z, y/z, 1]^\top$;

- each point with $z = 0$ is an *improper* point, identified by the coordinates $[x, y, 0]^\top$.

The same reasoning can be applied to both the 1-D and 3-D case, for the definitions of P^1 and P^3 respectively). *Mutatis mutandis*, the picture is identical. You add one coordinate for the description of a n-dimensional point, subject to the condition that the $(n+1)$-tuple of numbers (not all zero) are unique up to an undetermined factor. Hence a point in the projective line is identified by two numbers, whereas a point in the projective space by four numbers. In both cases, the homogeneous coordinates are unique up to an undetermined factor.

The Projective Line

We now close this preliminary section by introducing the notion of *projective line*. This can be easily done through the model above, since *collinear points in P^2 correspond to coplanar lines in the 3-D model*.

Definition: Projective Line

A *projective line*, $\mathbf{u}$, is represented by a 3-D plane going through the origin, or

$$\mathbf{u}^\top \mathbf{p} = 0. \tag{A.4}$$

☞ In the projective plane, points and lines are *dual*. In (A.4), one can alternatively think of $\mathbf{p}$ as (*a*) a point lying on the line $\mathbf{u}$, or (*b*) a line going through the point $\mathbf{u}$.

Projective Transformations

A projective transformation is *a linear transformation between projective spaces*. In computer vision, there are at least two important classes of projective transformations:

- linear invertible transformations of P^n, $n = 1, 2, 3$, into themselves.
- transformations between P^3 and P^2, which model image formation.

In what follows, we are interested in the first class. In particular, we want to establish that *a projective transformation of P^n onto itself is completely determined by its action on $n+2$ points*. For the sake of simplicity, we prove this general result in the particular case of $n = 2$; the extension to the case of a generic $n > 0$ does not pose any problem and is left for an exercise.

Determining a Projective Transformation

A projective transformation of the projective plane onto itself is completely determined once the transformation is known on four points, of which no three are collinear.

As a projective transformation is a linear, invertible transformation, we can represent it in matrix form and write

$$T\mathbf{p}' = \mathbf{p}.$$

Since the coordinates of both $\mathbf{p}$ and $\mathbf{p}'$ are known up to an undetermined factor, the entries of T are also known up to an undetermined factor.

We want to show that T can be written in terms of the four points $\mathbf{p}'_i = [x'_i, y'_i, z'_i]^\top$, $i = 1, \ldots, 4$, image of $\mathbf{p}_1 = [1, 0, 0]^\top$, $\mathbf{p}_2 = [0, 1, 0]^\top$, $\mathbf{p}_3 = [0, 0, 1]^\top$, and $\mathbf{p}_4 = [1, 1, 1]^\top$, respectively. Thanks to the fact that T is a 3×3 invertible matrix, our statement is proven if it can be proven for T^{-1}.

We start by writing

$$T^{-1}\mathbf{p}_1 = \mathbf{p}'_1.$$

From this equation we find that the first column of T^{-1} can be written as $\lambda[x'_1, y'_1, z'_1]^\top$, with λ undetermined. By using the knowledge of $\mathbf{p}'_2$ and $\mathbf{p}'_3$, we then find that T^{-1} can be written as

$$T^{-1} = \begin{pmatrix} \lambda x'_1 & \mu x'_2 & \nu x'_3 \\ \lambda y'_1 & \mu y'_2 & \nu y'_3 \\ \lambda z'_1 & \mu z'_2 & \nu z'_3 \end{pmatrix}.$$

Since no three of the four points $\mathbf{p}'_i$ are collinear, we can now determine λ, μ, and ν, up to an unknown factor. To do this, we use the last available point, $\mathbf{p}_4$, to find

$$\begin{aligned} \lambda x'_1 + \mu x'_2 + \nu x'_3 &= \rho x'_4 \\ \lambda y'_1 + \mu y'_2 + \nu y'_3 &= \rho y'_4 \\ \lambda z'_1 + \mu z'_2 + \nu z'_3 &= \rho z'_4. \end{aligned}$$

The nine entries of T^{-1} are therefore known up to an undetermined factor.

In summary, we have shown that a projective transformation of P^2 onto itself is characterized by its action on four points, no three of which are collinear. The four points, $\mathbf{p}_1 \ldots \mathbf{p}_4$, are called the *standard basis of* P^2.

By means of similar arguments, you should be able to show that one can pick $[1, 0]^\top, [0, 1]^\top, [1, 1]^\top$ as the standard basis for the case of line-to-line projective transformations, and $[1, 0, 0, 0]^\top, [0, 1, 0, 0]^\top, [0, 0, 1, 0]^\top, [0, 0, 0, 1]^\top, [1, 1, 1, 1]^\top$ as the standard basis for the case of space-to-space projective transformations.

The Cross-ratio

We close this brief appendix on projective geometry by touching upon the vast subject of *invariants*. We consider the most important and simplest invariant, the *cross-ratio*.

Definition: Cross-ratio

Given four distinct points of P^1, described in homogeneous coordinates as $\mathbf{p}_i = [x_i, y_i]^\top$ $(i = 1, \ldots, 4)$, the *cross-ratio* c is defined as

$$c(1, 2, 3, 4) = \frac{d(1, 2)d(3, 4)}{d(1, 3)d(2, 4)}$$

with

$$d(i, j) = x_i y_j - x_j y_i$$

the *determinant* between $\mathbf{p}_i$ and $\mathbf{p}_j$.

Cross-ratio Invariance

The cross-ratio is invariant to projective transformations of P^1 onto itself.

☞ Notice that the order of the points matters in the definition of cross-ratio, and each point appears once in the numerator and once in the denominator. In addition, the cross-ratio can also take on the improper values $\pm\infty$.

You should be able to prove that, given four points, you can define six different cross-ratios. A slightly more difficult exercise is to show that, if λ is one of the six cross-ratios, the other five are $1/\lambda$, $1 - \lambda$, $1/(1 - \lambda)$, $(\lambda - 1)/\lambda$, and $\lambda/(\lambda - 1)$. This tells you also that if, for some labelling, the cross-ratio goes to infinity, it is always possible to find a different labeling which evaluates to a finite value.

It is instructive to go through the proof of the cross-ratio invariance step by step. If we form the determinant between two of the four points, say $\mathbf{p}_1$ and $\mathbf{p}_2$, and assume y_1 and y_2 are not zero, we can write

$$d(1, 2) = x_1 y_2 - x_2 y_1 = y_1 y_2 \left(\frac{x_1}{y_1} - \frac{x_2}{y_2} \right). \tag{A.5}$$

As $[x, y]^\top$ are homogeneous coordinates of a line point, $d(1, 2)$ can be interpreted as the Euclidean distance between $\mathbf{p}_1$ and $\mathbf{p}_2$ times an undetermined factor, $y_1 y_2$, depending on both $\mathbf{p}_1$ and $\mathbf{p}_2$. Now let T, a 2×2 invertible matrix, represent a generic projective transformation,

$$T\mathbf{p}_i = \mathbf{p}'_i.$$

If the determinant between $\mathbf{p}'_1$ and $\mathbf{p}'_2$ is denoted by $d'(1, 2)$, we have

$$d'(1, 2) = d(1, 2)|T|,$$

where $|T|$ denotes the determinant of T. We can see that the determinant of the transformation cancels out in any ratio of determinants like (A.5), but not the factors $y_i y_j$; such factors cancel out, however, in the definition of cross-ratio, for which invariance is achieved.

References

J.L. Mundy and A. Zisserman, Appendix—Projective Geometry for Machine Vision. In *Geometric Invariants in Computer Vision*, Mundy, J.L. and Zisserman, A., eds, MIT Press, Cambridge (MA) (1992).

C.E. Springer, *Geometry and Analysis of Projective Spaces*, Freeman (1964).

A.5 Differential Geometry

This section complements our discussion of range image segmentation (Chapter 4) by recalling a few concepts from the differential geometry of surfaces.

Surface Tangent, Normal, and Area

Consider a parametric surface, $\mathbf{S}(u, v) = [x(u, v), y(u, v), z(u, v)]^\top$, and a point P on $\mathbf{S}$. Assume that all first and second derivatives of $\mathbf{S}$ with respect to u and v exist at P. The *tangent plane* at P is identified by the two vectors $\mathbf{S}_u = \partial\mathbf{S}/\partial u$ and $\mathbf{S}_v = \partial\mathbf{S}/\partial v$.

The *surface normal* is the unit normal vector to the tangent plane of S at P; that is,

$$\mathbf{n}_S(P) = \frac{\mathbf{S}_u \times \mathbf{S}_v}{\|\mathbf{S}_u \times \mathbf{S}_v\|}.$$

Range images in r_{ij} form (Chapter 4) correspond to the parametrization $S(x, y) = [x, y, h(x, y)]^\top$. In this case,

$$\mathbf{n}_S(P) = \frac{(-h_x, -h_y, 1)}{\sqrt{1 + h_x^2 + h_y^2}},$$

where $h_s = \partial h/\partial s$.

It is also useful to know how to compute the area of a surface patch, $\mathbf{Q}$, from a generic parametrization, $\mathbf{Q}(u, v)$, as well as from the parametrization $\mathbf{Q}_{xy} = (x, y, h(x, y))$. If we call A_Q the former and A_{xy} the latter, we have

$$A_Q = \int\int_Q \|\mathbf{Q}_u \times \mathbf{Q}_v\| \, du \, dv$$

$$A_{xy} = \int\int_Q \sqrt{1 + h_x^2 + h_y^2} \, dx \, dy.$$

Surface Curvatures

Curvatures make useful shape descriptors as they are invariant to viewpoint and parametrization. We now want to *extend the notion of curvature of a curve to define the curvature of a surface*, in order to derive the quantities used in Chapter 4.

To begin with, recall that the curvature of a parametric curve $\alpha(t) = (x(t), y(t))$, with t a parameter, is given at each point by

$$k(t) = \frac{x'y'' - x''y'}{((x')^2 + (y')^2)^{\frac{3}{2}}},$$

where $\alpha' = d\alpha/dt$. For the common parametrization $\alpha(x) = [x, y(x)]^\top$, the curvature becomes

$$k(t) = \frac{y''}{(1 + (y')^2)^{\frac{3}{2}}}.$$

Consider a parametric surface, $S(u, v)$, and a point P on S. Assume that all first and second derivatives of S with respect to u, v exist at P. We define surface curvatures in four steps.

Step 1: Normal Curvature of a Curve on S. Consider a curve C on S going through P. We define the *normal curvature of C at P* as

$$k_n = k \cos\theta,$$

where k is the curvature of C at P, and θ is the angle formed by the surface normal at P, $\mathbf{n}_S(P)$, with the curve normal, $\mathbf{n}_C(P)$.[3]

Step 2: Normal Curvature along a Direction. It can be proven that *k_n does not depend on the particular curve C chosen*, but only on the tangent of C at P, identified by the unit vector $\mathbf{d}$. This enables us to speak of *normal curvature along a direction*. For the sake of visualization, we can choose C as the planar curve obtained by intersecting S with a plane through P and containing both $\mathbf{d}$ and $\mathbf{n}_S(P)$. Obviously, C is a cross-section of S along $\mathbf{d}$, and describes the surface shape along that direction. Notice that, in this case, $k_n = k$.

Step 3: Principal Curvatures and Directions. We could now describe the local shape of S at P by taking the normal curvatures at P in *all* directions. This is totally impractical, but fortunately unnecessary. Assume we know the maximum and minimum normal curvatures at P, k_1 and k_2 respectively, called *principal curvatures*, and the corresponding directions, $\mathbf{d}_1$ and $\mathbf{d}_2$, called *principal directions*. It can be proven that

- the principal directions are always orthogonal;
- the normal curvature along *any* direction, $\mathbf{v} = (\cos\beta, \sin\beta)$, where β is the angle from $\mathbf{d}_1$ to $\mathbf{d}$, can be computed through *Euler's formula:*

$$k_n = k_1 \cos^2\beta + k_2 \sin^2\beta;$$

- consequently, the local shape of the surface is completely specified by the principal curvatures and directions.

[3] You can think of k_n as the projection of $k\mathbf{n}_C(P)$ along the surface normal, $\mathbf{n}_S(P)$.

Step 4: Classifying Local Shape. Finally, the shape classification given in Chapter 4 is achieved by defining two further quantities, the *mean curvature, H,* and the *Gaussian curvature, K*:

$$H = -\frac{k_1 + k_2}{2} \qquad K = k_1 k_2.$$

One can show that *the Gaussian curvature measures how fast the surface moves away from the tangent plane around P*, and in this sense is an extension of the 1-D curvature k. The formulae giving H and K for a range surface in r_{ij} form, $(x, y, h(x, y))$ are given in Chapter 4.

References

M.P. Do Carmo, *Differential Geometry of Curves and Surfaces*, Prentice-Hall, Englewood Cliffs (NJ) (1976).

A.6 Singular Value Decomposition

The aim of this section is to collect the basic information needed to understand the *Singular Value Decomposition* (SVD) as used throughout this book. We start giving the definition of SVD for a generic, rectangular matrix A and discussing some related concepts. We then illustrate three important applications of the SVD:

- solving systems of nonhomogeneous linear equations;
- solving rank-deficient systems of homogeneous linear equations;
- guaranteeing that the entries of a matrix estimated numerically satisfy some given constraints (e.g., orthogonality).

Definition

Singular Value Decomposition

Any $m \times n$ matrix A can be written as the product of three matrices:

$$A = UDV^\top. \tag{A.6}$$

The columns of the $m \times m$ matrix U are mutually orthogonal unit vectors, as are the columns of the $n \times n$ matrix V. The $m \times n$ matrix D is diagonal; its diagonal elements, σ_i, called *singular values*, are such that $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_n \geq 0$.

☞ While both U and V are not unique, the singular values σ_i are fully determined by A.

Some important properties now follow.

Properties of the SVD

Property 1. The singular values give you valuable information on the singularity of a square matrix, *A square matrix, A, is nonsingular if and only if all its singular values are different from zero*. Most importantly, the σ_i also tell you how close A is to be singular: the ratio

$$C = \frac{\sigma_1}{\sigma_n},$$

called *condition number*, measures the *degree of singularity of A*. When $1/C$ is comparable with the arithmetic precision of your machine, the matrix A is *ill-conditioned* and, for all practical purposes, can be considered singular.

Property 2. *If A is a rectangular matrix, the number of nonzero σ_i equals the rank of A*. Thus, given a fixed tolerance, ϵ (typically of the order of 10^{-6}), the number of singular values greater than ϵ equals the *effective rank* of A.

Property 3. *If A is a square, nonsingular matrix, its inverse can be written as*

$$A^{-1} = VD^{-1}U^\top.$$

Be A singular or not, the *pseudoinverse* of A, A^+, can be written as

$$A^+ = VD_0^{-1}U^\top,$$

with D_0^{-1} equal to D^{-1} for all nonzero singular values and zero otherwise. If A is nonsingular, then $D_0^{-1} = D^{-1}$ and $A^+ = A^{-1}$.

Property 4. The columns of U corresponding to the nonzero singular values span the range of A, the columns of V corresponding to the zero singular value the null space of A.

Property 5. The squares of the nonzero singular values are the nonzero eigenvalues of both the $n \times n$ matrix $A^\top A$ and $m \times m$ matrix $AA^\top$. The columns of U are eigenvectors of $AA^\top$, the columns of V eigenvectors of $A^\top A$. Morevoer, $A\mathbf{u}_k = \sigma_k \mathbf{v}_k$ and $A^\top \mathbf{v}_k = \sigma_k \mathbf{u}_k$, where $\mathbf{u}_k$ and $\mathbf{v}_k$ are the columns of U and V corresponding to σ_k.

Property 6. One possible distance measure between matrices can use the *Frobenius norm*. The Frobenius norm of a matrix A is simply the sum of the squares of the entries a_{ij} of A, or

$$\|A\|_F = \sum_{i,j} a_{ij}^2. \tag{A.7}$$

By plugging (A.6) in (A.7), it follows that

$$\|A\|_F = \sum_i \sigma_i^2.$$

We are now ready to summarize the applications of the SVD used throughout this book.

Least Squares

Assume you have to solve a system of m linear equations,

$$A\mathbf{x} = \mathbf{b},$$

for the unknown n-dimensional vector $\mathbf{x}$. The $m \times n$ matrix A contains the coefficients of the equations, the m-dimensional vector $\mathbf{b}$ the data. If not all the components of $\mathbf{b}$ are null, the solution can be found by multiplying both sides of the above equation for $A^\top$ to obtain

$$A^\top A\mathbf{x} = A^\top \mathbf{b}.$$

It follows that the solution is given by

$$\mathbf{x} = (A^\top A)^+ A^\top \mathbf{b}.$$

This solution is known to be optimal in the least square sense.

It is usually a good idea to compute the pseudoinverse of $A^\top A$ through SVD. In the case of more equations than unknowns the pseudoinverse is more likely to coincide with the *inverse* of $A^\top A$, but keeping an eye on the condition number of $A^\top A$ (**Property 1**) won't hurt.

☞ Notice that linear fitting amounts to solve exactly the same equation. Consequently, you can use the same strategy!

Homogeneous Systems

Assume you are given the problem of solving a homogeneous system of m linear equations in n unknowns

$$A\mathbf{x} = 0,$$

with $m \geq n - 1$ and $rank(A) = n - 1$. Disregarding the trivial solution $\mathbf{x} = 0$, a solution unique up to a scale factor can easily be found through SVD. *This solution is simply proportional to the eigenvector corresponding to the only zero eigenvalue of* $A^\top A$ (all other eigenvalues being strictly positive because $rank(A) = n - 1$). This can be proven as follows.

Since the norm of the solution of a homogeneous system of equations is arbitrary, we look for a solution of unit norm in the least square sense. Therefore we want to minimize

$$\|A\mathbf{x}\|^2 = (A\mathbf{x})^\top A\mathbf{x} = \mathbf{x}^\top A^\top A\mathbf{x},$$

subject to the constraint

$$\mathbf{x}^\top \mathbf{x} = 1.$$

Introducing the Lagrange multiplier λ this is equivalent to minimize the *Lagrangian*

$$\mathcal{L}(\mathbf{x}) = \mathbf{x}^\top A^\top A\mathbf{x} - \lambda(\mathbf{x}^\top \mathbf{x} - 1).$$

Equating to zero the derivative of the Lagrangian with respect to $\mathbf{x}$ gives

$$A^\top A\mathbf{x} - \lambda\mathbf{x} = 0.$$

This equation tells you that λ is an eigenvalue of $A^\top A$, and the solution, $\mathbf{x} = \mathbf{e}_\lambda$, the corresponding eigenvector. Replacing $\mathbf{x}$ with $\mathbf{e}_\lambda$, and $A^\top A\mathbf{e}_\lambda$ with $\lambda\mathbf{e}_\lambda$ in the Lagrangian yields

$$\mathcal{L}(\mathbf{e}_\lambda) = \lambda.$$

Therefore, the minimum is reached at $\lambda = 0$, the least eigenvalue of $A^\top A$. But from **Properties 4** and **5**, it follows that this solution could have been equivalently established as *the column of V corresponding to the only null singular value of A* (the kernel of A). This is the reason why, throughout this book, we have not distinguished between these two *seemingly* different solutions of the same problem.

Enforcing Constraints

One often generates numerical estimates of a matrix, A, whose entries are not all independent, but satisfy some algebraic constraints. This is the case, for example, of orthogonal matrices, or the fundamental matrix we met in Chapter 7. What is bound to happen is that the errors introduced by noise and numerical computations alter the estimated matrix, call it $\hat{A}$, so that its entries no longer satisfy the given constraints. This may cause serious problems if subsequent algorithms assume that $\hat{A}$ satisfies *exactly* the constraints.

Once again, SVD comes to the rescue, and allows us to *find the closest matrix to* $\hat{A}$, *in the sense of the Frobenius norm* (***Property 6***), *which satisfies the constraints exactly*. This is achieved by computing the SVD of the estimated matrix, $\hat{A} = UDV^\top$, and estimating A as $UD'V^\top$, with D' obtained by changing the singular values of D to those expected when the constraints are satisfied exactly.[4] Then, the entries of $A = UD'V^\top$ satisfy the desired constraints by construction.

References

G. Strang, *Linear Algebra and its Applications*, Harcourt Brace Jovanovich, Orlando (FL) (1988).

[4] If $\hat{A}$ is a good numerical estimate, its singular values should not be too far from the expected ones.

A.7 Robust Estimators and Model Fitting

This section sketches a few, introductory concepts behind *robust estimators*, in particular the so called *M-estimators*. Our limited aim is to support the discussion of Chapter 5 by explaining

- why least squares are a *maximum likelihood estimator* from the point of view of statistics,
- why least squares is skewed significantly by outliers, and
- why an estimator based on the least absolute value, as used in Chapter 5, tolerates outliers better than conventional least squares.

The subject has a vast literature; an initial set of further readings is provided at the end of this appendix. Several robust estimators not detailed here have become popular in computer vision. If interested, you should check out at least the *least median of squares*, discussed in the review by Meer *et al.* and detailed in Rousseeuw and Leroy's book, and the *RANSAC (Random Sample Consensus)* algorithm, introduced by Fischler and Bolles and also discussed by Meer *et al.*

Least Squares as Maximum Likelihood Estimator

Consider N data points $\mathbf{p}_i = [x_i, y_i]^\top$, $i = 1, \ldots, N$, and a *model*, $y = f(x, \mathbf{a})$, where $\mathbf{a}$ is a vector of parameters, and f is a known function. Assume that the data points are observations corrupted by noise (to be characterized better in the following). The well-known estimate of the parameter vector, $\mathbf{a}_o$, such that $f(x, \mathbf{a}_o)$ interpolates the data best in the least squares sense is

$$a_o = \min_{\mathbf{a}} \sum_{i=1}^{N} | y_i - f(x_i, \mathbf{a}) |^2 . \tag{A.8}$$

We want to show briefly that $\mathbf{a}_o$ *is the parameter vector maximizing the probability that the data are a noisy version of* $f(x, \mathbf{a})$, given appropriate assumptions on the noise.

Notice that we should estimate $\mathbf{a}$ by maximizing the probability that $\mathbf{a}$ *is correct given the data*, but we cannot estimate this probability (why?). However, if we assume that the noise corrupting each data point is *additive* and *Gaussian*, with *zero mean* and standard deviation σ, and that the amounts of noise at different data points are all independent, we *can* express the probability P that, for a given $\mathbf{a}$, all data points fall within Δy of the true value:

$$P \propto \prod_{i=1}^{N} (e^{-\frac{(y_i - f(x_i, \mathbf{a}))^2}{2\sigma^2}} \Delta y). \tag{A.9}$$

In essence, *maximum likelihood estimation* follows from the assumption that *the parameter vector which maximizes P is also the most likely one to occur given the observed data*. To obtain (A.8) from (A.9) is now easy: we just notice that P is maximized by minimizing the negative of its logarithm; that is,

$$\left[\sum_{i=1}^{N} \frac{(y_i - f(x_i, \mathbf{a}))^2}{2\sigma^2} \right] - N \log \Delta y,$$

and the constants σ, N, Δy can be ignored in the minimization.

Why Least Squares Fits are Skewed by Outliers

Since we assumed that the noise corrupting the data points is Gaussian, the probability that a noisy point lies within a distance d of its corresponding true point decreases very rapidly with d. Consequently, the least squares estimator believes that most points lie within a few standard deviations of the true (unknown) model. Now suppose that the data contain even a small percentage of points that are just *way off* and presumably not consistent with the Gaussian hypothesis.[5] Points like these are called *outliers*. In the absence of further information, a least squares estimator believes that outliers too are close to the model, as the probability of an outlier being as far as it really is from the true model is practically zero. The result is that the outliers "pull" the best fit away from the true model much more than they should.

Why Absolute Value Estimators Tolerate Outliers Better

The essence of the problem with least squares is that the Gaussian distribution decreases very rapidly as d becomes larger than σ. Therefore, a solution is to adopt a noise distribution which does not vanish as quickly as a Gaussian; that is, *which considers outliers more likely to occur*. An example of such a distribution is the *double exponential*,

$$Pr\{d\} \propto e^{-|\frac{y_i - f(x_i, \mathbf{a})}{\sigma}|}.$$

In this case, the probability P becomes

$$P \propto \prod_{i=1}^{N} (e^{-|\frac{y_i - f(x_i, \mathbf{a})}{\sigma}|} \Delta y). \tag{A.10}$$

The same reasoning which took us from (A.9) to (A.8) takes us from (A.10) to the *robust* maximum likelihood estimator

$$\min_{\mathbf{a}} \sum_{i=1}^{N} | y_i - f(x_i, \mathbf{a}) | .$$

The price we pay is that, unlike (A.8), this problem cannot be solved in closed form in most cases, and numerical methods must be employed.

[5] The reasons for these points being in the data set can be diverse and, for the purpose of this brief introduction, not quite relevant.

References

M.A. Fischler and R.C. Bolles, Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, *Communications of the ACM*, Vol. 24, pp. 381-395 (1981).

P. Meer, D. Mintz and A. Rosenfeld, Robust Regression Methods for Computer Vision: a Review, *International Journal of Computer Vision*, Vol. 6, pp. 59-70 (1991).

P.J. Rousseeuw and A.M. Leroy, *Robust Regression and Outlier Detection*, Wiley, New York (1987).

A.8 Kalman Filtering

This section is a brief account of a classic tool of optimal estimation theory, the *linear Kalman filter*. It is meant only to enable understanding of our treatment of feature-based tracking in Chapter 8. For an extensive discussion of Kalman filtering, see Maybeck's classic book.

The purpose of optimal estimation algorithms is to *produce optimal estimates of the state of a dynamic system, on the basis of noisy measurements and an uncertain model of the system's dynamics*. The theory is based on two ingredients, the notions of *system model* and *measurement model*.

The System Model

A physical system is modelled by a *state vector*, $\mathbf{x}$, often called simply the *state*, and a set of equations, called the *system model*. The state is a time-dependent vector, $\mathbf{x}(t)$, the components of which are system variables, in sufficient number to capture the dynamic properties of the system. The system model is a vector equation describing the evolution of the state in time. We indicate discrete equally spaced time instants with $t_k = t_0 + k\Delta T$, with $k = 0, 1, \ldots$ and ΔT a sampling interval, and denote with $\mathbf{x}_k$ the state $\mathbf{x}(t_k)$. We also assume that ΔT is small enough to capture the system's dynamics, so that *the state does not change much between consecutive time instants*, and a *linear* system model is an adequate approximation of the state change within ΔT. The linear system model is usually written in vector form as

$$\mathbf{x}_k = \Phi_{k-1}\mathbf{x}_{k-1} + \boldsymbol{\xi}_{k-1},$$

where $\boldsymbol{\xi}_{k-1}$ is a random vector modelling additive system noise. The subscript $k-1$ indicates that the *state transition matrix*, Φ, is a function of time, and hence accounts for more complicated dynamics than the one we considered for feature tracking in Chapter 8.

The Measurement Model

The second ingredient of estimation theory is the *measurement model*. We assume that, at any time instant t_k, a noisy measurement of the state vector (or at least of some components) is taken, and that the following, linear, relation holds between the measurements and the true system state:

$$\mathbf{z}_k = H_k\mathbf{x}_k + \boldsymbol{\mu}_k.$$

In this equation, $\mathbf{z}_k$ is the vector of measurements taken at time t_k, H_k the so-called *measurement matrix*, and $\boldsymbol{\mu}_k$ a random vector modelling additive noise, which accounts for the uncertainty associated with the measurements.

The key points and assumptions are summarized as follows:

Optimal Linear Estimation Theory (Kalman Filtering)

Assumptions

The state of a dynamic system at time t_k ($k = 1, 2 \ldots$) is described by an n-dimensional vector $\mathbf{x}_k$, the *state vector*. The evolution of the system's state is modelled by the linear equation

$$\mathbf{x}_k = \Phi_{k-1}\mathbf{x}_{k-1} + \boldsymbol{\xi}_{k-1}$$

with Φ_{k-1} a time-dependent $n \times n$ matrix called *state transition matrix*, and $\boldsymbol{\xi}_k$ a n-vector that accounts for the noise associated with the system model.

At any instant t_k, a m-dimensional vector of measurements, $\mathbf{z}_k$, is obtained. The relation between state and measurements is linear:

$$\mathbf{z}_k = H_k\mathbf{x}_k + \boldsymbol{\mu}_k$$

with H_k a time-dependent $m \times n$ matrix, and $\boldsymbol{\mu}_k$ a m-vector that accounts for the noise associated with the measurement process.

The noise terms, $\boldsymbol{\xi}_k$ and $\boldsymbol{\mu}_k$, are assumed to be white, zero-mean, Gaussian processes, with covariance matrices Q_k and R_k, respectively.

Problem Statement

Compute the best estimate of the system's state at t_k, $\hat{\mathbf{x}}_k$, taking into account the state estimate predicted by the system model at t_{k-1} *and* the measurements, $\mathbf{z}_k$, taken at t_k.

The Kalman Filter Algorithm

Optimal linear estimation theory gives you an algorithm based on four recursive equations, the celebrated *Kalman filter equations*, that returns the best estimate of the system state and its covariance at time t_k in the assumptions stated above. The equations integrate the measurements taken at t_k with the prediction of the state formulated at t_{k-1}. The resulting estimate is *optimal in a statistical sense*: over a large number of experiments, the Kalman filter's estimates would be better, on average, than those of any other filter.

The Kalman filter equations are characterized by the *state covariance matrices*, P_k and P'_k, and the *gain matrix*, K_k. P'_k is the covariance matrix of the k-th state estimate, $\hat{\mathbf{x}}'_k = \Phi_{k-1}\hat{\mathbf{x}}_{k-1}$, predicted by the filter *immediately before* obtaining the measurement $\mathbf{z}_k$. P_k is the covariance matrix of the k-th state estimate, $\hat{\mathbf{x}}_k$, computed by the filter *after*

integrating the measurement, $\mathbf{z}_k$, with the prediction, $\mathbf{x}'_k$. The covariance matrices are a quantitative model of the *uncertainty* of $\mathbf{x}'_k$ and $\mathbf{x}_k$. Finally, K_k establishes the relative importance of the prediction, $\hat{\mathbf{x}}'_k$, and the state measurement, $\hat{\mathbf{x}}_k$.

The Kalman Filter Algorithm

Let Q_k and R_k be the covariance matrices of the noise processes ξ_k and μ_k. The Kalman filter equations are

$$P'_k = \Phi_{k-1} P_{k-1} \Phi_{k-1}^\top + Q_{k-1} \tag{A.11}$$

$$K_k = P'_k H_k^\top (H_k P'_k H_k^\top + R_k)^{-1} \tag{A.12}$$

$$\hat{\mathbf{x}}_k = \Phi_{k-1} \hat{\mathbf{x}}_{k-1} + K_k (\mathbf{z}_k - H_k \Phi_{k-1} \hat{\mathbf{x}}_{k-1}) \tag{A.13}$$

$$P_k = (I - K_k) P'_k (I - K_k)^\top + K_k R_k K_k^\top \tag{A.14}$$

In A.13, the term $\Phi_{k-1}\hat{\mathbf{x}}_{k-1} = \hat{\mathbf{x}}'_k$ is often called *prediction*, the term $(\mathbf{z}_k - H_k \Phi_{k-1} \hat{\mathbf{x}}_{k-1})$ *innovation*.

☞ Notice that the term $(H_k P'_k H_k^\top + R_k)$ in A.12 is the covariance of the innovation.

The (A.11–A.14) estimate the state and its covariance recursively: they assume known initial estimates of the state covariance matrix, P_0, and of the state, $\hat{\mathbf{x}}_0$. The entries of P_0 are usually set to high, arbitrary values. The value of $\hat{\mathbf{x}}_0$ depends of the initial knowledge of the state; if none is available, an arbitrary value is used.

Let us briefly go through the four steps. First, the state covariance matrix at t_k, P'_k, is estimated according to the system model (A.11). Second, the *gain* of the Kalman filter is computed by (A.12), before reading the measurements. Third, the *optimal state estimate* at time t_k, $\hat{\mathbf{x}}_k$, is formed by (A.13), which integrates the state predicted by the system model $(\Phi_{k-1}\hat{\mathbf{x}}_{k-1})$ with the discrepancy of prediction and observation $(\mathbf{z}_k - H_k \Phi_{k-1} \hat{\mathbf{x}}_{k-1})$ in a sum weighted by the gain matrix, K_k. Finally, the *new state covariance matrix*, P_k, is evaluated through (A.14).

A simpler expression for P_k (A.14) is

$$P_k = (I - K_k H_k) P'_k,$$

but this can lead to serious numerical problems: if the entries of both $(I - K_k H_k)$ and P'_k are small, the accumulation of round-off errors in time can destroy the symmetry and positive definiteness of the covariance matrices, two essential assumptions of the Kalman filter. Violating these assumptions results in unstable state estimates. For this reason, one prefers (A.14), which guarantees both properties for P_k (why?).

☞ Notice that *the state covariance matrix does not depend on the measurements*. Therefore, if the time dependence of Φ_k, H_k, Q_k and R_k is known, P_k can be computed off-line and then approximated by a stepwise function. This property can be crucial for real-time implementations of the Kalman filter.

It can be proven that the Kalman filter is the optimal estimator in the assumptions of linear system and white, Gaussian noise. The filter's assumptions may seem restrictive, but are actually justified for many applications. Interestingly, if the noise is *not* Gaussian, *the Kalman filter can still be proven to be the best linear unbiased filter*. Check the classic book by Maybeck for more on this subject.

Uncertainty

What is the uncertainty on the state vector estimates computed by the Kalman filter? The answer is embedded in the state covariance matrix, which tells us *which region of the state space contains the true state with a given probability*. We illustrate briefly this important idea for the 2-D case.

Let $\mathbf{x} = [x_1, x_2]^\top$ be a 2-D state vector. The Kalman filter computes the optimal state estimate, $\hat{\mathbf{x}}_k$, as the maximum of the conditional probability density of $\mathbf{x}_k$ given the past state estimates, $\hat{\mathbf{x}}_1, \ldots \hat{\mathbf{x}}_{k-1}$, the past measurements, $\mathbf{z}_1, \ldots \mathbf{z}_{k-1}$, and the current measurement, $\mathbf{z}_k$. This density function is assumed Gaussian, so its maximum coincides with its mean. The important consequences in practice are:

- the region of the plane centered around $\hat{\mathbf{x}}_k$ which contains the *true* state with a given probability c^2 is the ellipse

$$(\mathbf{x} - \hat{\mathbf{x}}_k)(P_k)^{-1}(\mathbf{x} - \hat{\mathbf{x}}_k)^\top \leq c^2; \tag{A.15}$$

- the axes of this ellipse are $\pm c\sqrt{\lambda_i}\mathbf{e}_i$, $i = 1, 2$, where λ_i and $\mathbf{e}_i$ are the eigenvalues and eigenvectors, respectively, of P_k;
- the variable $(\mathbf{x} - \hat{\mathbf{x}}_k)(P_k)^{-1}(\mathbf{x} - \hat{\mathbf{x}}_k)^\top$ has a chi-square distribution, so that the probability of the region described by the ellipse is $(1 - \alpha)$, where c^2 is the upper (100α)-th percentile of a chi-square distribution with two degrees of freedom (in general, with a number of degrees of freedom equal to the dimension of the state vector).

In feature tracking, you use these *uncertainty ellipses* to reduce the search for the feature in the next frame, as follows. At t_{k-1}, the filter's prediction of the state at t_k is $\hat{\mathbf{x}}'_k = \Phi_{k-1}\hat{\mathbf{x}}_{k-1}$, with covariance P'_k. You diagonalize P'_k to get its eigenvectors,[6] consider the components of the state vector giving the feature's position $[x_1, x_2]^\top$, build an uncertainty ellipse (A.15) (centered in $\hat{\mathbf{x}}'_k$ and with a desired probability $(1 - \alpha)$), and look for the feature in the k-th frame *only* within that ellipse. Once your feature detector measures the feature's position at t_k, you compute $\hat{\mathbf{x}}_k$ and P_k, which allow you to build the uncertainty ellipse containing the true feature at t_k with the probability level chosen.

[6] This can always be done, because covariance matrices are always symmetric.

References

W.S. Cooper, Use of Optimal Estimation Theory, in Particular the Kalman Filter, in Data Analysis and Signal Processing, *Review of Scientific Instrumentation*, Vol. 57, pp. 2862-2869 (1986).

P.S. Maybeck, *Stochastic Models, Estimation, and Control*, Vol. I, Academic Press, New York (1979).

R. A. Johnson and D. W. Wichern, *Applied Multivariate Statistical Analysis*, Third edition, Prentice-Hall, Englewood Cliffs (1992).

A.9 Three-dimensional Rotations

This section collects some useful facts about *3-D rotations* and their representation. We limit ourselves to a few formulae involved by the discussions in this book, most importantly Chapter 11.

General Properties of Rotation Matrices

A rigid rotation of a 3-D vector $\mathbf{v}$ onto a 3-D vector $\mathbf{v}'$ can be represented by a linear transformation, defined by a 3×3 matrix R:

$$\mathbf{v}' = R\mathbf{v},$$

subject to the constraints

$$RR^\top = R^\top R = I \qquad \det(R) = 1,$$

with I the identity matrix. The first constraint tells you that the *inverse* of R equal its *transpose*. The second that *the transformation preserves the relative orientation of the reference frame*, and therefore its right- or left-handedness.

A matrix R for which $RR^\top = I$ is called *orthogonal*. The orthogonality property can be better appreciated by assuming that $\mathbf{v}$ and $\mathbf{v}'$ are expressed in two orthogonal reference frames, defined by the unit vectors $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$, and $\mathbf{e}'_1$, $\mathbf{e}'_2$, $\mathbf{e}'_3$ respectively. It can easily be seen that the generic entry r_{ij} of R is *the cosine of the angle formed by the base vector*, $\mathbf{e}_i$, *with the rotated base vector* $\mathbf{e}'_j$:

$$r_{ij} = \mathbf{e}_i^\top \mathbf{e}'_j.$$

Therefore we have that

$$\sum_{j=1}^{3} r_{ij} r_{kj} = \sum_{j=1}^{3} r_{ji} r_{jk} = \begin{cases} 0 & i \neq k \\ 1 & i = k \end{cases} \tag{A.16}$$

that is, *the rows (and columns) of R are mutually orthogonal unit vectors*.

As discussed in section A.6, this property makes the numerically estimation of a rotation matrix a rather tricky task. Even a slight perturbation of the entries of R, due to noise or small errors, destroys the orthogonality property and affects the estimation of the rotation parameters.

Two Useful Parametrizations

A 3×3 orthogonal matrix has nine elements which have to satisfy the six orthogonality constraints (A.16). This reduces the number of *degrees of freedom* of a 3-D rotation to $9 - 6 = 3$ and tells you that describing a rotation matrix through its nine entries is redundant and not really natural; in most cases it is useful and simpler to represent a rotation by means of more *natural* parametrization. In what follows we limit ourselves to the two parametrizations used in the book: *rotations around the coordinate axes* and *axis and angle*. Further parametrizations, well-known in computer vision, include *Euler angles*, *quaternions*, and *pitch,roll,yaw angles* (see References).

Rotations Around the Coordinate Axes. We can express a 3-D rotation as the result of three consecutive rotations around the coordinate axes, $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$, by angles α, β, and γ respectively. The angles are then the three free parameters of R, and each rotation is expressed as a rotation matrix, R_j, rotating vectors around $\mathbf{e}_j$, that is,

$$R_1(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

$$R_2(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

$$R_3(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The matrix R describing the overall rotation is the product of the R_j:

$$R = R_1 R_2 R_3 =$$

$$\begin{bmatrix} \cos\beta\cos\gamma & -\cos\beta\sin\gamma & \sin\beta \\ \sin\alpha\sin\beta\cos\gamma + \cos\alpha\sin\gamma & -\sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & -\sin\alpha\cos\beta \\ -\cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma & \cos\alpha\sin\beta\sin\gamma + \sin\alpha\cos\gamma & \cos\alpha\cos\gamma \end{bmatrix}$$

☞ The order of multiplication matters. Different sequences give different results, with the same triplet of angles. Notice that there are six ways to represent a rotation matrix as a product of rotations around three different coordinate axes.

The recovery of α, β, and γ from R is easy (and is left to you as an exercise). However, if the matrix R has been obtained as the output of some numerical computation, you should always make sure that the estimated R is *really* orthogonal; section A.6 gives you the recipe that we employed throughout the book.

Axis and Angle. According to *Euler's theorem*, any 3-D rotation can be described as a rotation by an angle, θ, around an axis identified by a unit vector $\mathbf{n} = [n_1, n_2, n_3]^\top$. The corresponding rotation matrix, R, can then be obtained in terms of θ and the components of $\mathbf{n}$, which gives you a total of four parameters. The redundancy of this

parametrization (four parameters for three degrees of freedom) is eliminated by adding the constraint that $\mathbf{n}$ has unit norm, that is, by dividing each n_i by $\sqrt{n_1^2 + n_2^2 + n_3^2}$.

The matrix R in terms of θ and $\mathbf{n}$ is given by

$$R = I\cos\theta + (1-\cos\theta)\begin{bmatrix} n_1^2 & n_1n_2 & n_1n_3 \\ n_2n_1 & n_2^2 & n_2n_3 \\ n_3n_1 & n_3n_2 & n_3^2 \end{bmatrix} + \sin\theta \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix} \quad \text{(A.17)}$$

Conversely, both θ and $\mathbf{n}$ can be obtained from the eigenvalues and eigenvectors of R. The three eigenvalues of R are 1, $\cos\theta + i\sin\theta$, and $\cos\theta - i\sin\theta$, where i is the imaginary unit. The unit vector, $\mathbf{n}$, is proportional to the eigenvector of R corresponding to the eigenvalue 1; the angle θ can be obtained from either of the two complex eigenvalues. To resolve the ambiguity in the sign of both θ and $\mathbf{n}$, you can check the consistency of (A.17).

References

S. L. Altmann, *Rotations, Quaternions, and Double Groups*, Oxford University Press, Oxford (1986).

O. D. Faugeras and M. Hebert, The Representation, Recognition, and Locating of 3-D Shapes from Range Data, *International Journal of Robotic Research*, Vol. 5, pp. 27-52 (1986).

G. A. Korn and T. M. Korn, *Mathematical Handbook for Scientists and Engineers*, second edition, McGraw-Hill, New York (1968).

Index