Tracking Point Features

Carlo Tomasi

Visual reconstruction takes as input several images of the same rigid object recorded from different viewpoints and computes a three-dimensional model of the object. A key step in this computation is to compute what is called a *point cloud* of the object, that is, a set of 3D points on the object's surface. In turn, computing a cloud of points from a set of images requires determining where the projection of each of the points is located in each of the images.

Figure 1 shows a typical visual reconstruction pipeline: Start with a large collection of images of a building or part of a city (top left in the Figure)—in this example, images are uploaded by interested photographers to a common wiki. A tree-based vocabulary of image parts (top center) is built to identify images that are in some sense similar to each other. For any two similar images, one needs to find *correspondences*, that is, pairs of points (with one point in each image) that correspond to the same point in the world. Correspondences for a sample pair of similar images are connected by white line segments in the top right panel of the Figure.

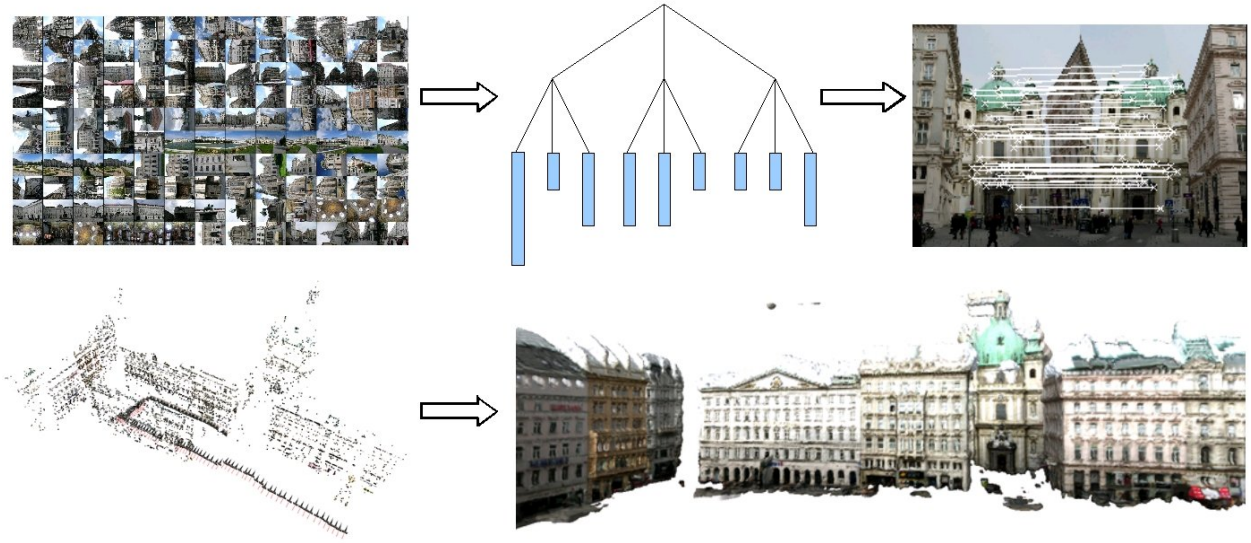

Figure 1: A typical visual reconstruction pipeline [4]. See text for an explanation.

Once correspondences are found for enough points and enough image pairs, the point cloud is computed (bottom left) through a complex optimization problem. A side-output of this computation is also the world position and orientation of each of the cameras that took the pictures. In the example, all the pictures were taken by a single person walking with a hand-held camera in front of the buildings of interest, and the panel on the bottom left of the Figure shows the position and orientation of the camera as one little arrow for every picture taken. The points of the cloud are then connected with a triangular mesh, and the pixel colors from the input images are back-projected onto it, leading to the more realistic surface rendering at the bottom right of the Figure.

It would be a relatively simple exercise in computer graphics to compute images if the shapes and colors of buildings were known, together with the positions, orientations, and parameters of the cameras. In visual reconstruction, however, one only has the images to work with, and the computation is all in reverse, from images to world. These reverse computations are called *inverse problems* in physics, and are characterized by ill-conditioning: Small errors in the images or inaccuracies in the initial computations on them lead to possibly catastrophic errors in the output reconstructions. It is rather surprising that this computation is possible even in principle. And yet the 3D reconstruction pipeline has now become commonplace, and is one of the major accomplishments of computer vision to date. It was made possible by decades of advances in the various aspects of the computation.

This note focuses on two early problems in the pipeline:

- How to determine *points of interest*, that is, image points that are promising candidates for correspondence.
- How to determine correspondences.

As usual, we will consider one of many different approaches. Our choice falls on *point-feature tracking*, because in this approach the two problems above can be addressed in a unified way. A relatively recent survey [8] discusses many alternatives, and this research area is still very thriving, especially where it tries to address larger inter-camera motions than feature tracking can handle.

If point-feature tracking is the method of choice, then the two problems above are best addressed in reverse order: First figure out how tracking works, then declare a window to be a good candidate if it lends itself well to tracking [6]. This heuristic principle leads to a definition of a good point-feature candidate that is similar to that found through different arguments [3].

1 Correspondence by Tracking

A point feature (or "feature" for short[1]) is a small image window of a predefined size. Square windows with 5 to 21 pixels to a side are common. The side-lengths are typically odd, so one can think of the central pixel in the window as "the point" one is interested in. If images of the scene—which is assumed to be static—are taken from closely spaced positions and orientations, then corresponding point features in different images are likely to be both close to each other geometrically and similar to each other photometrically, making correspondence easier to determine. Under these conditions, establishing correspondence is called *tracking*.

Variety in viewpoint position and orientation turns out to make 3D reconstruction better, so if point-feature tracking is the method of choice one typically uses only a subset of all the available images for reconstruction, and all the intervening ones for correspondence: Correspondences are found between consecutive images and extended by transitivity to any pair of them.

Not all image windows are equally good candidates for tracking. For instance, if the window covers a part of the image where intensity[2] is more or less uniform, such as the middle of a blank wall, a small motion of the window may not lead to noticeable differences in the window contents. In that case, the window's brightness distribution does not allow determining the displacement of the point feature reliably. This is called the *aperture problem* in the literature, because it arises from viewing the image through the small aperture of a feature window. A less severe version of the aperture problem arises when the feature window

[1] Do not confuse point-features with features used for image recognition.

[2] As customary, we think of gray-level images in this context, although one could also use color to some advantage.

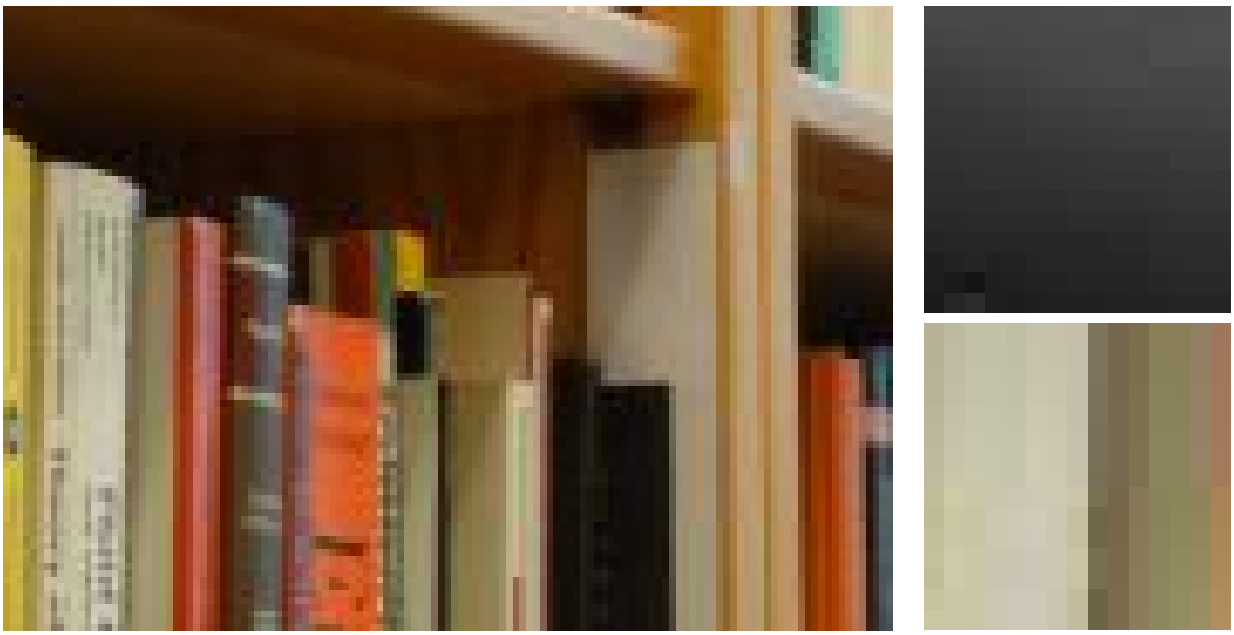

Figure 2: The 15×15 window at the top right comes from the relatively blank area at the top left of the low-resolution image detail shown on the left. Shifting that window around by a few pixels does not change its contents appreciably, and any changes that do occur may be small compared to image noise. The 15×15 window at the bottom right comes from around one of the many vertical edges in the image. Shifting the window by even a small amount horizontally, say to the right, will cause the edge in it to move to the left, so the horizontal component of motion can be reliably determined along a vertical edge. However, if the window is shifted vertically by a small amount, changes in it are minor, and may again be overwhelmed by image noise: The vertical component of motion cannot be measured reliably.

straddles a straight intensity edge. In that case, motion across the edge is clearly visible, but motion along it is not. Figure 2 illustrates.

Thus, a window that can be tracked well could be defined by requiring that its contents are sufficiently varied. This was done by Harris and Stephens [3] in 1988, and let to what is called the *Harris detector*. This note follows instead the derivation given by Shi and Tomasi [6] in 1994 for a feature tracker that generalized the vastly popular *Lucas and Kanade tracker* [5] from pure translation to affine image deformations. The main advantage of the Shi-Tomasi derivation is pedagogical brevity, as both tracking and the best windows for it are found within the same theoretical framework. Section 4 follows that derivation but maps it back to the pure-translation case of the Lucas and Kanade tracker. This choice is made both for simplicity and because the pure-translation tracker works best for very small image displacements, where one can speak of a single motion for the entire window. The detectors that result from the two alternative derivations are very similar to each other and work equally well.

In summary, Section 3 describes the Lucas and Kanade tracker, and Section 4 derives a point-feature detector very similar to the Harris detector, by following the derivation by Shi and Tomasi.

2 Tracking

Let $I(\mathbf{x})$ and $J(\mathbf{x})$ be two gray-level images of the same scene taken from slightly different viewpoints and possibly orientations, and let us focus our attention on a point feature, that is, a small, square window $W(\mathbf{x}_I)$

of odd side-length $2h+1$ pixels, centered at some point $\mathbf{x}_I$ in I. The question is, what are the coordinates

$$\mathbf{x}_J = \mathbf{x}_I + \mathbf{d}^*(\mathbf{x}_I)$$

of the corresponding window's center in image J? The two-dimensional vector $\mathbf{d}^*(\mathbf{x}_I)$ is called the *displacement* of that point feature, and the assumption is made that the motion of the camera between the two images is so small[3] that the magnitude of $\mathbf{d}^*(\mathbf{x}_I)$ is much smaller than the diameter of $W(\mathbf{x}_I)$.

A natural way to answer the question above is to measure the *dissimilarity* or *residual* between $W(\mathbf{x}_I)$ and a candidate window in J as follows:

$$\epsilon(\mathbf{x}_I, \mathbf{d}) = \sum_{\mathbf{x}} [J(\mathbf{x}+\mathbf{d}) - I(\mathbf{x})]^2 \, w(\mathbf{x}-\mathbf{x}_I) \tag{1}$$

where the double summation over $\mathbf{x} = (x_1, x_2)^T$ extends to the whole plane and $w(\mathbf{x})$ is the indicator function of the window $W(\mathbf{0})$:

$$w(\mathbf{x}) = \begin{cases} 1 & \text{if } |x_1| \leq h \text{ and } |x_2| \leq h \\ 0 & \text{otherwise} \end{cases} .$$

The residual (1) is thus the sum of squared differences between the pixels in a window around $\mathbf{x}_I$ in I and a window around $\mathbf{x}_J = \mathbf{x}_I + \mathbf{d}$ in J. One can then search for the displacement that makes the dissimilarity as small as possible:

$$\mathbf{d}^*(\mathbf{x}_I) = \arg\min_{\mathbf{d} \in R} \epsilon(\mathbf{x}_I, \mathbf{d}) \tag{2}$$

where the square $R \subseteq \mathbb{R}^2$ is centered at the origin of the plane and is called the *search range*. Do not confuse the feature window $W(\mathbf{x}_I)$ with the search range. The former is part of the image I, while the latter is a square region in the plane of all possible displacements. The assumption that the displacement is small relative to the window size implies that R has a half-size that is significantly less than h.

Each image point $\mathbf{x}_I$ in I will have its own displacement $\mathbf{d}^*(\mathbf{x}_I)$, and the function $\mathbb{R}^2 \to \mathbb{R}^2$ that maps image points $\mathbf{x}_I$ in I to their displacement is called the *displacement field*. For brevity, we will omit explicit mention of the dependency of $\mathbf{d}^*$ on $\mathbf{x}_I$ when there is no ambiguity.

This approach to correspondence makes the implicit assumption that the image motions contained in $W(\mathbf{x}_I)$ are all the same, so that there is a single displacement for the whole window. This is approximately true for many small motions and small windows, but not in general, and may be violated even for small motions and windows. For example, Figure 3 shows details from three consecutive frames out of a movie. If the tracking window were placed, say, around the tip of the arrow on the street sign, the window would contain two very different motions: that of the sign and that of the background. To somewhat lessen the effects of multiple motions in the same window, it is customary to replace the indicator function $w(\mathbf{x})$ with a truncated Gaussian:

$$w(\mathbf{x}) \propto \begin{cases} e^{\frac{1}{2}\left(\frac{\|\mathbf{x}\|}{\sigma}\right)^2} & \text{if } |x_1| \leq h \text{ and } |x_2| \leq h \\ 0 & \text{otherwise} \end{cases}$$

where σ is comparable to the half-width h of the window and is typically between $\sigma = h/3$ and $\sigma = h/2$. In this way, the dissimilarity (1) depends more on what happens close to the center of the window than on what happens at its periphery. Should this measure be inadequate to capture the complexity of motions in the window, one could model these with an affine geometric transformation [6], rather than just a translation.

[3] It is not really necessary for the two pictures to be taken by the same camera. Nonetheless, when small motion occurs they typically are.

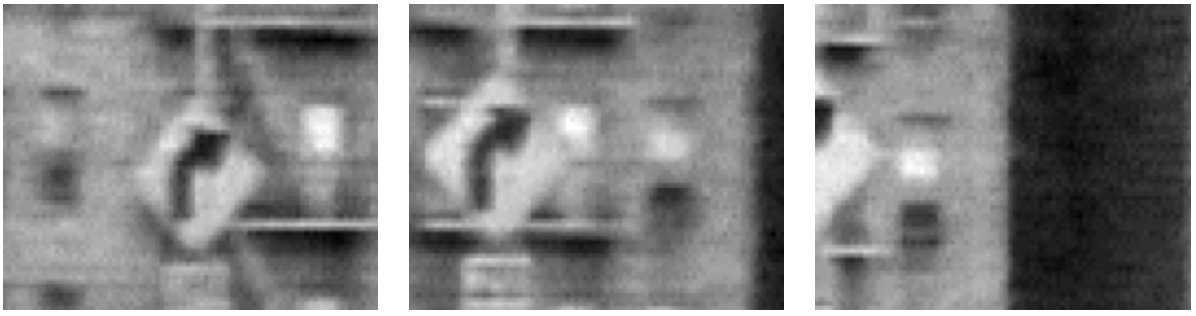

Figure 3: Details from three consecutive frames out of Woody Allen's 1979 movie *Manhattan*.

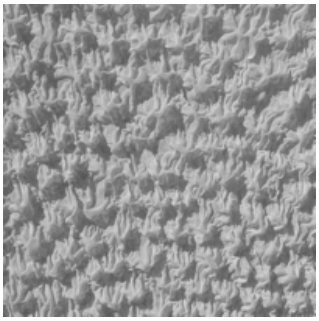

Figure 4: Detail from a fabric sample. [Image adapted from `mayang.com`.]

One can use any of a wide variety of optimization methods to solve problem (2), including direct grid search: Simply write a `for` loop that searches for $\mathbf{d}^*(\mathbf{x}_I)$ over all possible integer displacements $\mathbf{d}$ in R. The advantage of this method is that local minima are less of a problem. Consider for instance a highly textured window, like the one in figure 4. A window placed anywhere on this image is likely to be at least somewhat similar to many other windows in the same image, because of the repetitive nature of the pattern it contains. As a result, the residual (1) is likely to have many local minima. An exhaustive search will compute all of the residuals within the range R, and select the window that yields the smallest one.

The main disadvantage of exhaustive search, besides a possibly slower running time, is that the resolution of the resulting displacement $\mathbf{d}^*$ is limited to the pitch of the search grid. To make resolution better, one would have to use a fine grid, with a resulting higher computational cost.

Good resolution during tracking is important for two reasons. First, as mentioned earlier, 3D reconstruction is an ill-posed problem, so we cannot afford imprecise motion measurements. Second, small errors tend to accumulate when tracking over several frames, a problem called *drift* in the literature: When tracking a feature from image 1 to image to 2 and then to image 3, the window found by grid search in image 2 may not correspond exactly to the starting window in image 1. In turn, the window found in image 3 may not correspond exactly to that found in image 2. These small errors tend to accumulate, resulting in large drifts across distant frames. Some methods have been proposed to monitor or reduce drift [6], but accurate, sub-pixel displacements are desirable in any case, because of the ill-posedness of reconstruction.

Because of this, features are often tracked by differential methods, perhaps after grid search has provided a good starting point. The original paper by Lucas and Kanade [5] uses the Newton-Raphson method, described next.

3 The Lucas and Kanade Tracker

Let us simplify the notation for the optimization problem (2) and residual (1) as follows:

$$\mathbf{d}^* = \arg\min_{\mathbf{d}} e(\mathbf{d}) \tag{3}$$

where we abbreviate $\epsilon(\mathbf{x}_I, \mathbf{d})$ to $e(\mathbf{d})$. In this way, we only pay attention to what changes during optimization (that is, the displacement $\mathbf{d}$), and not what remains fixed (the position $\mathbf{x}_I$ of the window in image I). The choice of replacing $\mathbf{d} \in R$ with an unconstrained $\mathbf{d}$ in the minimization (3) is more controversial. In doing so, we are effectively ignoring the limited search range for $\mathbf{d}$. The advantage is that unconstrained optimization is easier than constrained optimization if one just wants a local minimum. The disadvantage is that we may get a displacement $\mathbf{d}^*$ whose norm is bigger than we are willing to accept—that is, we may move too far from $\mathbf{x}_I$. To simplify the discussion, we accept this risk but add some sanity checks: If, after minimization is complete, either the norm of $\mathbf{d}^*$ or the residual $e(\mathbf{d}^*)$ is too large, we discard the solution and declare failure. This is actually what many trackers do, because the opportunity cost of losing a few feature tracks per image is small relative to the computational cost of accounting for the constraints. Of course, optimization methods that account for constraints do exist [1].

Newton's method for solving the unconstrained optimization problem (3) amounts to starting at some initial point $\mathbf{d}_0$ at iteration $t = 0$. In iteration t, find the Hessian

$$H_e(\mathbf{d}_t) = \begin{bmatrix} \frac{\partial^2 e}{\partial d_{1t}^2} & \frac{\partial^2 e}{\partial d_{1t} d_{2t}} \\ \frac{\partial^2 e}{\partial d_{2t} d_{1t}} & \frac{\partial^2 e}{\partial d_{2t}^2} \end{bmatrix}$$

and gradient

$$\nabla e(\mathbf{d}_t) = \begin{bmatrix} \frac{\partial e}{\partial d_{1t}} \\ \frac{\partial e}{\partial d_{2t}} \end{bmatrix}$$

at displacement $\mathbf{d}_t$, solve the linear, 2×2 system of *normal equations*

$$H_e(\mathbf{d}_t)\mathbf{s} = -\nabla e(\mathbf{d}_t) \tag{4}$$

for the *Newton step* $\mathbf{s}_t$, move by that step,

$$\mathbf{d}_{t+1} = \mathbf{d}_t + \mathbf{s}_t \ , \tag{5}$$

and iterate until convergence.

The rationale for Newton's step is that the function $e(\mathbf{d})$ is approximated by its second-order Taylor expansion around $\mathbf{d}_t$:

$$e(\mathbf{d}) \approx e(\mathbf{d}_t) + [\nabla e(\mathbf{d}_t)]^T(\mathbf{d} - \mathbf{d}_t) + (\mathbf{d} - \mathbf{d}_t)^T H_e(\mathbf{d}_t)(\mathbf{d} - \mathbf{d}_t) \ .$$

In other words, $e(\mathbf{d})$ is replaced by the best quadratic fit to it around $\mathbf{d}_t$. Setting the derivative of this approximation to zero yields the system (4) with

$$\mathbf{s} = \mathbf{d} - \mathbf{d}_t \ ,$$

and the step $\mathbf{s}_t$ that solves the system finds the exact minimum of the approximation. Since this may not be the exact minimum of $e(\mathbf{d})$, the process is iterated.

The convergence of Newton's method can be shown to be quadratic in the following sense:

$$\lim_{t\to\infty} \frac{|\mathbf{d}_{t+1} - \mathbf{d}^*|}{|\mathbf{d}_t - \mathbf{d}^*|^2} = \mu$$

where μ is a positive real number[4]. This rate is very fast: After several iterations (this is the limit part), the error at iteration $t+1$ is proportional to the square of the error at iteration t. In terms of digits, the number of accurate decimal digits *doubles* at every iteration!

The issue with applying Newton's method directly to problem (3) is that the argument $\mathbf{d}$ in function $e(\mathbf{d})$ is contained inside the expression $J(\mathbf{x}+\mathbf{d})$ (see equation (1)), and computing second derivatives of an image is both expensive and sensitive to noise. Instead, Raphson's variant of Newton's method first linearizes the *image* J around every point $\mathbf{x}$ in the image window, and uses the linearization in the definition of residual (1). This leads to a different quadratic approximation than the one given by the Taylor expansion of $e(\mathbf{d})$, but the rest is the same. The advantage of this different approximation is that it only requires computing first-order derivatives of the image, as opposed to the second derivatives required by Newton's method. Raphson's variant works because the residual is a sum of squares in our problem: The sum of squares of a linear(ized) function of the step $\mathbf{s}$ is a quadratic function of $\mathbf{s}$, and finding its minimum becomes a linear problem.

More specifically, let

$$J_t(\mathbf{x}) = J(\mathbf{x} + \mathbf{d}_t)$$

be the result of shifting the image J by the displacement found in iteration $\mathbf{d}_t$. The problem is now to find a step $\mathbf{s}_t$ that, when added to $\mathbf{d}_t$, yields the new displacement $\mathbf{d}_{t+1}$ (equation 5). To this end, the nonlinear, shifted image function $J_t(\mathbf{x}+\mathbf{s}) = J(\mathbf{x}+\mathbf{d}_t+\mathbf{s})$ is replaced with its first-order Taylor expansion around $\mathbf{x}$,

$$J_t(\mathbf{x}+\mathbf{s}) \approx J_t(\mathbf{x}) + [\nabla J_t(\mathbf{x})]^T \mathbf{s} \tag{6}$$

so that the residual (1) at the (unknown) point $\mathbf{d}_t + \mathbf{s}$ can be approximated as follows:

$$\begin{aligned} e(\mathbf{d}_t+\mathbf{s}) &= \sum_{\mathbf{x}} [J(\mathbf{x}+\mathbf{d}_t+\mathbf{s}) - I(\mathbf{x})]^2 \, w(\mathbf{x}-\mathbf{x}_I) \\ &= \sum_{\mathbf{x}} [J_t(\mathbf{x}+\mathbf{s}) - I(\mathbf{x})]^2 \, w(\mathbf{x}-\mathbf{x}_I) \\ &\approx \sum_{\mathbf{x}} [J_t(\mathbf{x}) + [\nabla J_t(\mathbf{x})]^T \mathbf{s} - I(\mathbf{x})]^2 \, w(\mathbf{x}-\mathbf{x}_I) \ , \end{aligned}$$

a quadratic function of $\mathbf{s}$. The gradient of e at $\mathbf{d}_t + \mathbf{s}$ then becomes

$$\nabla e(\mathbf{d}_t+\mathbf{s}) \approx 2 \sum_{\mathbf{x}} \nabla J_t(\mathbf{x}) \{ J_t(\mathbf{x}) + [\nabla J_t(\mathbf{x})]^T \mathbf{s} - I(\mathbf{x}) \} \, w(\mathbf{x}-\mathbf{x}_I) \ .$$

Setting this gradient to zero yields to the following linear system of equations in $\mathbf{s}$:

$$A\mathbf{s} = \mathbf{b} \tag{7}$$

where[5]

$$A = \sum_{\mathbf{x}} \nabla J_t(\mathbf{x}) [\nabla J_t(\mathbf{x})]^T \, w(\mathbf{x}-\mathbf{x}_I) \quad \text{and} \quad \mathbf{b} = \sum_{\mathbf{x}} \nabla J_t(\mathbf{x}) [I(\mathbf{x}) - J_t(\mathbf{x})] \, w(\mathbf{x}-\mathbf{x}_I) \ . \tag{8}$$

[4]The term "quadratic" refers to the exponent 2 in the denominator.

[5]Note that $-2\mathbf{b}$ is the gradient of e at $\mathbf{d}_t$. So by comparing with the normal equation (4), we see that $2A$ is an approximation to the Hessian of A.

This is a small, 2×2 system that forms the core of the Lucas and Kanade tracker. The tracker solves this system starting with some initial displacement $\mathbf{d}_0$ at iteration $t = 0$, shifts the image J_t by $\mathbf{d}_0$, then iterates by finding step $\mathbf{s}_t = \mathbf{d}_{t+1} - \mathbf{d}_t$ and repeating until convergence, every time shifting the image J_t further by the step $\mathbf{s}_t$. The overall displacement is then the sum of all the steps,

$$\mathbf{d}^* = \sum_t \mathbf{s}_t \ .$$

Since J is continually shifted during iterations of the algorithm, the image $J_t(\mathbf{x})$ becomes more and more similar to $I(\mathbf{x})$ within the feature window, and the residual decreases. In addition, the Taylor approximation (6) becomes better and better, because the step $\mathbf{s}_t$ becomes smaller and smaller.

Algorithm 1 summarizes the computation. The subscript t can be dropped everywhere in the algorithm, because old variable values are simply overwritten by new ones.

Algorithm 1. The Lucas and Kanade tracker

Input: Images I and J, window center $\mathbf{x}_I$ in I, window function $w(\mathbf{x})$, initial displacement $\mathbf{d}_0$, termination thresholds δ, ϵ, ρ, $t_{\max}$, and largest acceptable residual $e_{\max}$

$X \leftarrow \{\mathbf{x} \mid w(\mathbf{x}) > 0\}$ ▷ X is the support of the window function $w(\mathbf{x})$
$\mathbf{w} \leftarrow w(X)$ ▷ $\mathbf{w}$ is a column vector of all the nonzero values of $w(\mathbf{x})$
$X \leftarrow X + \mathbf{x}_I$ ▷ The set X now contains the window coordinates in I
$\mathbf{i} \leftarrow I(X)$ ▷ $\mathbf{i}$ is a column vector of all the image values of I on X
$\mathbf{d} \leftarrow \mathbf{d}_0$ ▷ Initialize the cumulative displacement
$\mathbf{s} \leftarrow \mathbf{d}_0$ ▷ The first shift of J is equal to the initial displacement
$t \leftarrow 0$ ▷ t is the iteration count
repeat
    $t \leftarrow t + 1$ ▷ Keep track of the number of iterations
    $X \leftarrow X + \mathbf{s}$ ▷ The set X now tracks the window coordinates in J
    $\mathbf{j} \leftarrow J(X)$ ▷ $\mathbf{j}$ is a column vector of all the image values of J on X
    $G \leftarrow \nabla J(X)$ ▷ G is a two-column matrix of all the gradients of J on X
    $A \leftarrow G^T(G \ .\star \ [\mathbf{w}, \mathbf{w}])$ ▷ The 2×2 matrix A in equation (7)
    $\mathbf{b} \leftarrow G^T((\mathbf{i} - \mathbf{j}) \ .\star \ \mathbf{w})$ ▷ The 2×1 vector $\mathbf{b}$ in equation (7)
    $\mathbf{s} \leftarrow A \setminus \mathbf{b}$ ▷ Solve for the step
    $\mathbf{d}_{\text{old}} \leftarrow \mathbf{d}$ ▷ Remember the old value of $\mathbf{d}$ to check for progress
    $\mathbf{d} \leftarrow \mathbf{d} + \mathbf{s}$ ▷ Accumulate the step $\mathbf{s}$ into the displacement $\mathbf{d}$
    done $= \|\mathbf{s}\| \le \delta$ or $e(\mathbf{d}_{\text{old}}) - e(\mathbf{d}) \le \epsilon$ ▷ Progress has slowed down beyond our thresholds
    lost $= \|\mathbf{d} - \mathbf{d}_0\| > \rho$ or $t > t_{\max}$ ▷ We are probably lost
until done or lost
if lost or $e(\mathbf{d}) > e_{\max}$ **then** ▷ We are either lost or the final residual is too large
    $\mathbf{d}^* \leftarrow$ `[NaN; NaN]` ▷ Tracker failure
else
    $\mathbf{d}^* \leftarrow \mathbf{d}$ ▷ Success
end if

Output: Locally optimal displacement $\mathbf{d}^*$ at $\mathbf{x}_I$, or a "failure" value.

Practicalities

Checks for convergence include a threshold on the size of the current step $\mathbf{s}_t$ and on the decrease in residual. Since the constraint that the overall displacement be within the range R is ignored, it is customary to check that the accumulated displacement $\mathbf{d}_t$ does not take us too far away from the initial displacement $\mathbf{d}_0$. To ensure termination in a finite amount of time, a maximum number of iterations is also imposed. Thus, iterations continue as long as

$$\|\mathbf{s}_t\| > \delta \quad \text{and} \quad e(\mathbf{d}_{t-1}) - e(\mathbf{d}_t) > \epsilon \quad \text{and} \quad \|\mathbf{d}_t - \mathbf{d}_0\| \leq \rho \quad \text{and} \quad t \leq t_{\max}$$

for suitable positive thresholds δ, ϵ, ρ. Violating either of the first two conditions signals convergence, while violating either of the last two implies failure.

To make things concrete, Algorithm 1 is written in a matrix style somewhat reminiscent of MATLAB. The set X defined in line 1 initially keeps track of the coordinates where the window function $w(\mathbf{x})$ is nonzero.[6] After X has been used to store away all the nonzero *values* in the window into the $K \times 1$ vector $\mathbf{w}$ (line 2), the coordinates in X are shifted by the center $\mathbf{x}_I$ of the window of interest in image I (line 3), and the new coordinates are used to collect the pixel values found in that window into the $K \times 1$ vector $\mathbf{i}$ (line 4). Then the set X is shifted again on line 10, and from now on X stays on top of the window in J, rather than that in I. The matrix G defined on line 12 is $K \times 2$, and each of its rows represents the gradient of J at one of the pixels in the moving window. Lines 13 and 14 assemble the 2×2 system (7), and line 17 accumulates the steps into the overall displacement.

The very first time the algorithm is run, the coordinates in X on line 4 are typically integers, because there is rarely a strong reason why the initial point $\mathbf{x}_I$ would not be on the integer pixel grid. However, the steps $\mathbf{s}$ found during optimization are generally not integer. In addition, once a point feature is tracked, say, from frame 1 to frame 2, the coordinates of its center are generally no longer integer, so when that point is subsequently tracked from frame 2 to frame 3, even the center $\mathbf{x}_I$ of the window in I (frame 2 in this case) may be non-integer. In these situations, the assignment on lines 4, 11, and 12 are implemented by bilinear interpolation.

The initial displacement $\mathbf{d}_0$ is often set to the zero vector, to reflect the assumption that motion between frames is small. An (expensive) alternative is to first do an exhaustive search for the minimum residual over a grid of displacements $\mathbf{d}$ (every pixel, or every few pixels, in both directions within a range R), and then run the Lucas and Kanade tracker starting at that minimum to refine.

When image motion is large, the tracker can be run on a truncated image pyramid: First find motion at a set of points in the coarser levels of the pyramid, then track points at each finer level using the properly scaled motions at the coarser level to initialize. Specifically, when tracking point $\mathbf{x}_I^{(\ell)}$ at level ℓ, let $\mathbf{x}_I^{(\ell+1)}$ be the coordinates of the corresponding point at the next-coarser level $\ell + 1$. Let $\mathbf{d}_0^{(\ell+1)}$ be the displacement found at the point among the tracked points at level $\ell + 1$ that is nearest to $\mathbf{x}_I^{(\ell+1)}$. Finally, scale the displacement $\mathbf{d}_0^{(\ell+1)}$ to account for the scale difference between the two levels and use the scaled displacement to initialize the Lucas and Kanade tracker at $\mathbf{x}_I^{(\ell)}$. This scheme works rather well for relatively large motions and with only a modest number of levels in the pyramid [7].

[6] This set could be implemented as a $K \times 2$ matrix in MATLAB if there are K nonzero pixels in the window.

4 Good Features to Track

As mentioned in the introduction, not every feature can be tracked well, because of the aperture problem. The optimization problem addressed by the Lucas and Kanade tracker can be solved reliably when the system (7) at the core of the algorithm has a well-defined solution, that is, when the system is well-conditioned. This means that if the ideal matrix $\tilde{A}$ and ideal vector $\tilde{\mathbf{b}}$ in the system are affected by some error, the solution $\mathbf{s}$ does not change much.

More quantitatively, suppose that the ideal system

$$\tilde{A}\mathbf{s} = \tilde{\mathbf{b}}$$

is perturbed into the actual system

$$A\mathbf{s} = \mathbf{b}$$

where

$$A = \tilde{A} + \epsilon E \quad \text{and} \quad \mathbf{b} = \tilde{\mathbf{b}} + \epsilon\, \mathbf{e}$$

for any matrix E and vector $\mathbf{e}$ of suitable size. Then if $\tilde{\mathbf{s}}$ and $\mathbf{s}$ are the solution to the ideal system and the solution to the perturbed one, Appendix A shows that the relative error

$$\rho_{\mathbf{s}} = \frac{\|\mathbf{s} - \tilde{\mathbf{s}}\|}{\|\tilde{\mathbf{s}}\|}$$

on the solution can be bounded as follows:

$$\rho_{\mathbf{s}} \;\leq\; \kappa_2(\tilde{A})\,(\rho_{\mathbf{b}} + \rho_A) \tag{9}$$

where

$$\kappa_2(\tilde{A}) = \frac{\sigma_{\max}(\tilde{A})}{\sigma_{\min}(\tilde{A})} \geq 1 \quad , \quad \rho_{\mathbf{b}} = |\epsilon|\,\frac{\|\mathbf{e}\|}{\|\tilde{b}\|} \quad \text{and} \quad \rho_A = |\epsilon|\,\frac{\|E\|}{\|\tilde{A}\|}\;.$$

The scalars $\rho_{\mathbf{b}}$ and ρ_A measure the relative errors on $\tilde{b}$ and $\tilde{A}$. The quantity $\kappa_2(\tilde{A})$ is called the *condition number* of $\tilde{A}$, and is equal to the ratio between the greatest and the smallest singular value of A. The convention is made that

$$\kappa_2(\tilde{A}) = \infty \quad \text{whenever} \quad \sigma_{\min}(\tilde{A}) = 0\;.$$

In the definitions above and in Appendix A, the Euclidean norm $\|\mathbf{z}\|_2$ is used to measure the magnitude of vectors, and the 2-norm is used for matrices,

$$\|B\|_2 = \sup_{\mathbf{z}} \frac{\|B\mathbf{z}\|_2}{\|\mathbf{z}\|_2} = \sup_{\|\mathbf{z}\|_2=1} \|B\mathbf{z}\|_2 = \sigma_{\max}(B)\;,$$

the largest singular value of B. If a different norm is used, the result above merely changes by a constant factor, since matrix norms can always be bounded by suitable multiples of each other from above and from below [2]. We omit the subscript 2 from the norms for brevity.

Equation (9) states that if the condition number of the ideal matrix $\tilde{A}$ is small then the solution to the perturbed system (7) is not very different, in relative terms, from the solution of the ideal system. Of course, we do not have the ideal matrix, but we take the condition number $\kappa_2(A)$ of the actual matrix as an estimate of that of $\tilde{A}$. In addition, the matrix A keeps changing during execution of the Lucas and Kanade algorithm

(see line 13 in Algorithm 1). So we take the matrix $A(\mathbf{x}_I)$ computed around point $\mathbf{x}_I$ in image I to be an estimate of the matrix A for the moving window in J.

In summary, we say that a point feature centered at $\mathbf{x}_I$ in image I is a *good feature to track* if the condition number of this matrix is below some predetermined threshold $\kappa_{\max}$:

$$\kappa_2(A_I(\mathbf{x}_I)) \leq \kappa_{\max} \quad \text{where} \quad A_I(\mathbf{x}_I) = \sum_{\mathbf{x}} \nabla I(\mathbf{x})[\nabla I(\mathbf{x})]^T \, w(\mathbf{x} - \mathbf{x}_I) \; .$$

The expression of $A_I(\mathbf{x}_I)$ is a convolution of the matrix image

$$\Gamma(\mathbf{x}) = \nabla I(\mathbf{x})[\nabla I(\mathbf{x})]^T = \begin{bmatrix} \gamma_{11}(\mathbf{x}) & \gamma_{12}(\mathbf{x}) \\ \gamma_{21}(\mathbf{x}) & \gamma_{22}(\mathbf{x}) \end{bmatrix}$$

with the window function $w(\mathbf{x})$, a fact that can be used to advantage for efficient computation, especially after noticing that $w(\mathbf{x})$ is separable. The image $\Gamma(\mathbf{x})$ is a matrix image in the sense that it has a 2×2 matrix at every pixel, so you can think of it as four separate, scalar images. This matrix is both symmetric,

$$\gamma_{12}(\mathbf{x}) = \gamma_{21}(\mathbf{x})$$

(so you only need to compute and store three images, not four), and rank-deficient (that is, its rank is at most 1), because it is the product of a column vector and a row vector. However, the symmetric matrix $A_I(\mathbf{x}_I)$ is generally not rank-deficient, because it is a sum of many (rank-deficient) matrices.

It may help intuition to look at the structure of $A_I(\mathbf{x}_I)$ for some special image windows, and relate this structure to our earlier discussion of the aperture problem. If the image intensity function is constant within the window, the gradient $\nabla I(\mathbf{x})$ is everywhere zero in the window and so is $\Gamma(\mathbf{x})$. As a consequence, $A_I(\mathbf{x}_I)$ is zero as well, and its condition number is infinity (by our convention on κ_2). If $A_I(\mathbf{x}_I)$ straddles a vertical edge, or even a collection of vertical edge elements, then the vertical component of the gradient is zero everywhere in the window,

$$\nabla I(\mathbf{x}) = \begin{bmatrix} g_1 \\ 0 \end{bmatrix} \quad \text{so that} \quad \Gamma(\mathbf{x}) = \begin{bmatrix} \gamma_{11}(\mathbf{x}) & 0 \\ 0 & 0 \end{bmatrix}$$

and therefore

$$A_I(\mathbf{x}_I) = \begin{bmatrix} a_{11}(\mathbf{x}_I) & 0 \\ 0 & 0 \end{bmatrix} \; .$$

This is a rank-1 matrix, and its condition number is infinite again. Similar considerations hold for edges in different orientations, in the sense that $A_I(\mathbf{x}_I)$ is rank-1, although it will generally not have zero entries unless the edges are aligned with the coordinate axes.

These degenerate cases correspond to occurrences of the aperture problem. They also show that if $\sigma_{\min}(A_I(\mathbf{x}_I)) \approx 0$, that is, when the matrix $A_I(\mathbf{x}_I)$ is close to being rank-deficient, the condition number does not matter, because a small condition number (that is $\kappa_2(A_I(\mathbf{x}_I)) \approx 1$) could be achieved by two nonzero but very small singular values that are approximately equal to each other. This suggests modifying the check for a good feature to the following:

$$\kappa_2(A_I(\mathbf{x}_I)) \leq \kappa_{\max} \quad \text{and} \quad \sigma_{\min}(A_I(\mathbf{x}_I)) \geq \sigma_0$$

for some positive threshold σ_0.

It is only when gradients in the window centered at $\mathbf{x}_I$ have different orientations that the condition number of $A_I(\mathbf{x}_I)$ becomes finite and relevant. When the variety of orientations and strengths of the edges are high, the condition number is close to 1, and the window is a good feature to track.

When the window centered at $\mathbf{x}_I$ is such a feature, chances are that windows that overlap with it are good features as well, because they contain closely related pixel values. So the good features are selected by *non-maximum suppression*: Pick a best point feature in the image, that is, a point feature with smallest condition number[7]. Then remove all good point features that overlap with it and repeat, either until no good point features remain or enough features have been collected. Algorithm 2 summarizes the computation of good features to track. The function $\mathrm{overlap}(X, \mathbf{x}^T)$ on line 15 takes a matrix X of pixel coordinates and a single row vector $\mathbf{x}^T$ with the coordinates of one point, and returns the row indices for X that correspond to windows that overlap with the window centered at $\mathbf{x}^T$.

Algorithm 2. Finding good features to track

Input: Image I, window function $w(\mathbf{x})$, thresholds $\kappa_{\max} > 1$ and $\sigma_0 > 0$ on the maximum condition number and on the smallest $\sigma_{\min}$ allowed, and maximum number $n_{\max}$ of features needed (can be set to infinity)

1: $X \leftarrow \mathrm{pixels}(I)$ ▷ X is a two-column matrix of all the pixel coordinates in the image
2: $G \leftarrow \mathrm{gradient}(I)$ ▷ G is a two-column matrix of all the gradients in the image
3: $\Gamma \leftarrow [G(:,1)\ .\hat{}\ 2,\ G(:,1)\ .*\ G(:,2),\ G(:,2)\ .\hat{}\ 2]$ ▷ Distinct entries of $\nabla I(\mathbf{x})[\nabla I(\mathbf{x})]^T$
4: $A \leftarrow \Gamma * w$ ▷ Convolution of $\Gamma(\mathbf{x})$ with the window function $w(\mathbf{x})$
5: $S \leftarrow A(:,1) + A(:,3)$ ▷ A piece of the formula for eigenvalues
6: $D \leftarrow \sqrt{(A(:,1) - A(:,3))\ .\hat{}\ 2 + 4\,A(:,2)\ .\hat{}\ 2}$ ▷ Discriminant
7: $\sigma_{\max} \leftarrow \frac{1}{2}(S + D)$ ▷ Compute singular values
8: $\sigma_{\min} \leftarrow \frac{1}{2}(S - D)$ ▷ (this is done in closed form for a 2×2 matrix)
9: $X \leftarrow X(\sigma_{\max} \le \kappa_{\max}\sigma_{\min}\ \&\ \sigma_{\min} \ge \sigma_0\ ,\ :)$ ▷ Good enough features
10: $X \leftarrow X$ sorted by increasing $\sigma_{\max}/\sigma_{\min}$ ▷ Put best features first
11: $Y \leftarrow [\,]$ ▷ Y collects the final point-feature coordinates
12: **for** $n = 1$ to $n_{\max}$ **do** ▷ Loop for non-maximum suppression
13: **if** X is empty **then** ▷ No more good features left
14: break
15: **end if**
16: $Y \leftarrow [Y; X(1,:)]$ ▷ Pick the best feature remaining
17: $X \leftarrow X(\sim\mathrm{overlap}(X, X(1,:)), :)$ ▷ Keep only windows that do not overlap with $X(1,:)$
18: **end for**

Output: Two-column matrix Y of coordinates for at most $n_{\max}$ non-overlapping, good point features

[7]Or with largest $\sigma_{\min}$, or some combination of these two criteria. Which sorting criterion to use is not critical, as all features remaining at this point are good enough for tracking.

References

[1] S. Boyd and L. Vandeberghe. *Convex Optimization*. Cambdrige University Press, 2004.

[2] G. Golub and C. Van Loan. *Matrix Computations*, chapter 7, page 320. Johns Hopkins University Press, Baltimore, 3 edition, 1996.

[3] C. Harris and M. Stephens. A combined corner and edge detector. In M. M. Matthews, editor, *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, 1988.

[4] A. Irschara, C. Zach, and H. Bischof. Towards wiki-based dense city modeling. In *Workshop on Virtual Representations and Modeling of Large-Scale Environment, IEEE International Conference on Computer Vision*, pages 1–8, 2007.

[5] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI81)*, pages 674–679, 1981.

[6] J. Shi and C. Tomasi. Good features to track. In *1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 593–600, Seattle, WA, USA, 1994. IEEE Comput. Soc. Press.

[7] C. Tomasi and T. Kanade. Shape and motion from image streams – a factorization method. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 90:21, pages 9795–9802, November 1993.

[8] T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: A survey. *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2007.

A The Sensitivity of the Solution to a Linear System to Errors in the Coefficients

Suppose that the ideal matrix $\tilde{A}$ and ideal vector $\tilde{\mathbf{b}}$ in the linear system

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

are affected by errors. This Appendix examines the relationship between the relative magnitude of these errors and the relative magnitude of the error in the solution to the system. The factor by which the former errors are amplified to yield the latter is called the *condition number* of the matrix $\tilde{A}$, and can be computed from the SVD of it.

To measure errors in $\tilde{A}$ and $\tilde{\mathbf{b}}$ with a single scalar ϵ, we write these errors as

$$A - \tilde{A} = \epsilon E \quad \text{and} \quad \mathbf{b} - \tilde{\mathbf{b}} = \epsilon\, \mathbf{e}$$

where E and $\mathbf{e}$ are a 2×2 matrix and a 2×1 vector. So the real, perturbed system

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

can be written as

$$(\tilde{A} + \epsilon E)\, \mathbf{x} = \tilde{\mathbf{b}} + \epsilon\, \mathbf{e}\ .$$

Let us assume that each of these two system admits a single solution, so that the matrices A and $\tilde{A}$ are invertible.[8] Then, the solution to the perturbed system can be viewed as a function of ϵ,

$$\mathbf{x}(\epsilon) = (\tilde{A} + \epsilon E)^{-1}\, (\tilde{\mathbf{b}} + \epsilon\, \mathbf{e})\ ,$$

and in particular $\mathbf{x}(0)$ is the solution to the ideal system $\tilde{A}\mathbf{x} = \tilde{\mathbf{b}}$.

To study the sensitivity of $\mathbf{x}$ with respect to ϵ we approximate the function $\mathbf{x}(\epsilon)$ with its Taylor expansion truncated after the linear term. Straightforward matrix calculus yields

$$\begin{aligned}
\dot{\mathbf{x}}(0) &= \left[-(\tilde{A} + \epsilon E)^{-1} E\, (\tilde{A} + \epsilon E)^{-1}\, (\tilde{\mathbf{b}} + \epsilon\, \mathbf{e}) \;+\; (\tilde{A} + \epsilon E)^{-1} \mathbf{e}\right]_{\epsilon=0} \\
&= -\tilde{A}^{-1} E \tilde{A}^{-1} \tilde{\mathbf{b}} \;+\; \tilde{A}^{-1}\mathbf{e} = \tilde{A}^{-1}\, (\mathbf{e} - E\tilde{A}^{-1}\tilde{\mathbf{b}}) \\
&= \tilde{A}^{-1}\, (\mathbf{e} - E\, \mathbf{x}(0))
\end{aligned}$$

so that to first order

$$\mathbf{x}(\epsilon) \approx \mathbf{x}(0) + \dot{\mathbf{x}}(0)\epsilon = \mathbf{x}(0) + \epsilon \tilde{A}^{-1}\, (\mathbf{e} - E\, \mathbf{x}(0))\ .$$

Rearranging terms and using standard norm inequalities then yield the following bound on the relative magnitude of the error on $\mathbf{x}(\epsilon)$ for $\epsilon \to 0$:

$$\rho_{\mathbf{x}} = \frac{\|\mathbf{x}(\epsilon) - \mathbf{x}(0)\|}{\|\mathbf{x}(0)\|} \leq |\epsilon|\, \|\tilde{A}^{-1}\| \left(\frac{\|\mathbf{e}\|}{\|\mathbf{x}(0)\|} + \|E\|\right)\ .$$

Since $\tilde{\mathbf{b}} = \tilde{A}\mathbf{x}(0)$, we have

$$\|\tilde{\mathbf{b}}\| \leq \|\tilde{A}\|\, \|\mathbf{x}(0)\|$$

and therefore

$$\rho_{\mathbf{x}} \leq |\epsilon|\, \|\tilde{A}^{-1}\|\, \|\tilde{A}\| \left(\frac{\|\mathbf{e}\|}{\|\tilde{A}\|\, \|\mathbf{x}(0)\|} + \frac{\|E\|}{\|\tilde{A}\|}\right) \leq\ \|\tilde{A}^{-1}\|\, \|\tilde{A}\| \left(|\epsilon|\, \frac{\|\mathbf{e}\|}{\|\tilde{\mathbf{b}}\|} + |\epsilon|\, \frac{\|E\|}{\|\tilde{A}\|}\right)\ .$$

[8] This assumption will be removed later.

To summarize,

$$\rho_{\mathbf{x}} \;\leq\; \kappa_2(\tilde{A})\,(\rho_{\mathbf{b}} + \rho_A) \tag{10}$$

where

$$\rho_{\mathbf{x}} = \frac{\|\mathbf{x}(\epsilon) - \mathbf{x}(0)\|}{\|\mathbf{x}(0)\|} \quad , \quad \kappa_2(\tilde{A}) = \|\tilde{A}^{-1}\|\,\|\tilde{A}\| \quad , \quad \rho_{\mathbf{b}} = |\epsilon|\,\frac{\|\mathbf{e}\|}{\|\tilde{\mathbf{b}}\|} \quad \text{and} \quad \rho_A = |\epsilon|\,\frac{\|E\|}{\|\tilde{A}\|}\;.$$

The scalars $\rho_{\mathbf{b}}$ and ρ_A measure the relative errors on $\tilde{b}$ and $\tilde{A}$, and $\kappa_2(\tilde{A})$ is called the *condition number* of $\tilde{A}$.

Equation (10) states that the condition number of the matrix $\tilde{A}$ of the ideal system $\tilde{A}\mathbf{x} = \tilde{\mathbf{b}}$ is the scalar that multiplies the sum of relative errors on the coefficients of the system to yield a bound on the relative error on the solution of the system. Since the two-norm of a matrix is its largest singular value and the singular values of the inverse are the reciprocals of the singular values, the two-norm of $\tilde{A}^{-1}$ is the reciprocal of the smallest singular value of $\tilde{A}$, and therefore

$$\kappa_2(\tilde{A}) = \frac{\sigma_{\max}(\tilde{A})}{\sigma_{\min}(\tilde{A})} \quad \text{and} \quad \kappa_2(\tilde{A}) \geq 1\;.$$

If either A or $\tilde{A}$ is not invertible, then the solution of the corresponding linear system is underdetermined, and the sensitivity of the error in the solution is unbounded. The result (10) therefore holds in all cases if we make the natural convention that

$$\sigma_{\min}(\tilde{A}) = 0 \;\;\Rightarrow\;\; \kappa_2(\tilde{A}) = \infty\;.$$