Image Motion Analysis

Carlo Tomasi

February 24, 2022

There are many reasons why analyzing the motion of objects in video is useful. Traffic monitoring, gesture recognition, American Sign Language interpretation, tracking of cardiac contractions, analysis of sports video, weather forecasting from satellite imagery, interactive video games are a sample of possible applications. Each frame of a video sequence is just the measurement of a distribution of light on a sensor, and motion is not directly encoded anywhere in a video: Our brains *compute* motion by making assumptions about the world and comparing consecutive video frames, and computer algorithms for video analysis must do that, too. This inference turns out to be surprisingly difficult, considering that our visual systems seems to make it routinely with no apparent effort. This note attempts to give some reasons why by describing how video images are formed, and pointing out some of the assumptions the inference of motion is typically based on. Algorithms for making the inference are described in later notes.

Before it is converted to digital information, the image formed by the lens of a camera is projected onto the camera's sensor, which sees a continuous distribution of light. It is useful to consider this image when analyzing image motion, because it is a real-valued function of two real variables, in contrast with the image we process on the computer, which is a discrete array of integer pixel values. We also view images as changing over time ("video"), since we are interested in image motion.

In this note we analyze image motion under two simplifying assumptions:

- The digital image array is obtained by sampling the continuous distribution of brightness on the sensor on a regular grid. The actual picture is a bit more complex because the sampling includes integration over a both time and a small area within each pixel. Appendix A sketches a more realistic model.

- The image is gray. While color images do carry more information, the three color components are often very strongly correlated to each other, and it turns out that most of the information that is useful for motion analysis is carried by image brightness.

Section 2 below introduces a fundamental assumption that is typically made when analyzing video, namely, that the appearance of a point in the scene remains constant over time, and Section 3 describes a key observability issue that makes the analysis of video motion fundamentally difficult. Section 4 categorizes, at a high level, different approaches to estimating visual motion, and Section 5 describes a first approach.

1 The Motion Field

As the world and/or the camera move, a point in the scene typically projects to different points on the sensor plane at different times, and the sensor position of the point's image is therefore described by a function of time. Consider the image brigthness $e(\boldsymbol{\xi}, s)$ at a particular point $\boldsymbol{\xi}$ on the sensor plane and time s. Point $\boldsymbol{\xi}$ is the projection of some point in the world (possibly infinitely far away), and we denote by $\mathbf{x}(\boldsymbol{\xi}, s, t)$ the trajectory that the projection of that world point traces on the sensor plane as a function of time t.

> **Notation:** An image trajectory $\mathbf{x}(\boldsymbol{\xi}, s, t)$ is a curve on the image plane, and the curve is traced as the third argument, t, varies in $\mathbb{R}$. So you can think as the trajectory as being simply $\mathbf{x}(t)$. However, we need a way to state which trajectory we are talking about among the many image trajectories of interest. Because of this, we consider a reference snapshot taken at time s, and the first two arguments, $\boldsymbol{\xi}$ and s, are our way to identify a particular trajectory: It's the one that passes through $\boldsymbol{\xi}$ in that snapshot. If you find it psychologically preferable, feel free to think of $\mathbf{x}(\boldsymbol{\xi}, s, t)$ as $\mathbf{x}_{\boldsymbol{\xi},s}(t)$. Of course, that image point may have been moving before time s, so the trajectory is often also defined for $t < s$. Figure 1 illustrates.

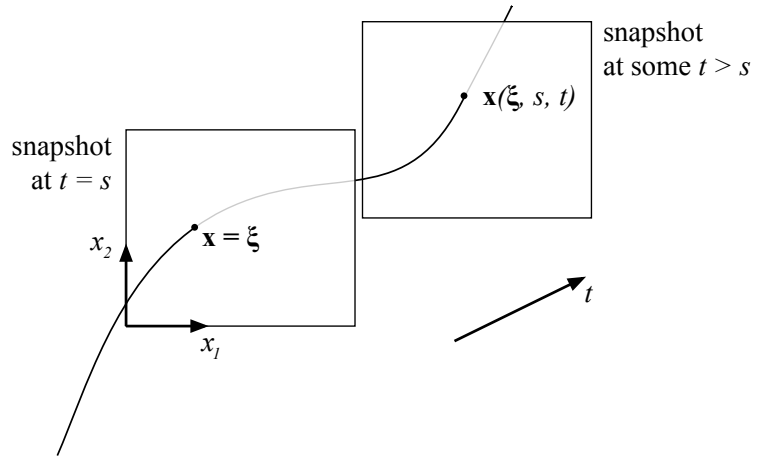

Figure 1: The trajectory $\mathbf{x}(\boldsymbol{\xi}, s, t)$ passes through image point $\boldsymbol{\xi}$ at time s.

Since the trajectory starts at $\boldsymbol{\xi}$, we have in particular

$$\mathbf{x}(\boldsymbol{\xi}, s, s) = \boldsymbol{\xi} \; . \tag{1}$$

The image velocity of $\mathbf{x}(\boldsymbol{\xi}, s, t)$ at time t is

$$\mathbf{w}(\boldsymbol{\xi}, s, t) \stackrel{\text{def}}{=} \frac{\partial \mathbf{x}(\boldsymbol{\xi}, s, t)}{\partial t} \tag{2}$$

and the velocity at time s,

$$\mathbf{v}(\boldsymbol{\xi}, s) \stackrel{\text{def}}{=} \mathbf{w}(\boldsymbol{\xi}, s, s) \tag{3}$$

is called the *motion field* at $(\boldsymbol{\xi}, s)$.

At time $t > s$, the point is at position

$$\mathbf{x}(\boldsymbol{\xi}, s, t) = \int_s^t \mathbf{w}(\boldsymbol{\xi}, s, \tau)\, d\tau \;+\; \mathbf{x}(\boldsymbol{\xi}, s, s)$$

(from the fundamental theorem of calculus and the definition of $\mathbf{w}$) and the vector difference

$$\mathbf{d}(\boldsymbol{\xi}, s, t) \;\stackrel{\text{def}}{=}\; \mathbf{x}(\boldsymbol{\xi}, s, t) - \mathbf{x}(\boldsymbol{\xi}, s, s) \tag{4}$$

is called the *displacement* at $\boldsymbol{\xi}$ between times s and t. Thus, the motion field is an instantaneous rate of motion change, while displacement is a finite change of position observed over a finite time interval.

The world points that are visible at a particular pixel position at time s may become *occluded* at time t, that is, either become hidden behind some world object or leave the field of view of the camera altogether. Conversely, points that are visible at time t may be occluded at time s. For the points that are occluded at either time the displacement is unobservable. All other points at one time have a *corresponding* point at the other time. If transparent objects are ignored, which is common practice, each point at one time has *at most one* corresponding point at the other time.

The motion field $\mathbf{v} : \mathbb{R}^3 \to \mathbb{R}^2$ is usually modeled as piecewise smooth (in both space and time), and its discontinuities are smooth surfaces in $\mathbb{R}^3$ called *motion boundaries*. They are typically caused by objects in the world that move in front of parts of other objects that move differently. Sometimes the term "motion boundaries" refers to the curves obtained by restricting a motion-boundary surface to a specific time s.

Eulerian and Lagrangian Viewpoint The two functions $\mathbf{v}$ and $\mathbf{w}$ are very closely related to each other (through equation 3), but they describe the motion of points in the image from two different perspectives. The motion field $\mathbf{v}$ reflects an *Eulerian* view of things: Imagine Euler sitting at a fixed point $\boldsymbol{\xi}$ on the sensor plane. He sees the image of a world point go by at time s and measures its velocity $\mathbf{v}(\boldsymbol{\xi}, s)$ on the image plane. At a different point t in time, a different world point will generally pass by at $\boldsymbol{\xi}$. Euler will still be there, measuring *that* point's motion. Thus, Euler measures the motion of all the different points that fly by the same image location $\boldsymbol{\xi}$. This is similar to a spectator at a car race sitting at $\boldsymbol{\xi}$, who sees the different cars driving by.

In contrast, the trajectory velocity $\mathbf{w}(\boldsymbol{\xi}, s, t)$ reflects a *Lagrangian* perspective: Lagrange is actually riding the image point $\mathbf{x}(\boldsymbol{\xi}, s, t)$, following it as it moves around the image with increasing t, and keeps measuring its velocity relative to the image[1]. Thus, Lagrange measures the velocity of a single point, similarly to a race car driver who observes her own car's velocity as the car position $\mathbf{x}$ changes over time.

Only at time $t = s$ is Lagrange in the same place as Euler, a very fleeting encounter of two great mathematicians.[2] At that time, Euler's and Lagrange's velocity measurements agree: When a given race car drives by the spectator, so that $\mathbf{x} = \boldsymbol{\xi}$, they are both looking at the same car and they are seeing the same velocity $\mathbf{v}(\boldsymbol{\xi}, s) = \mathbf{v}(\mathbf{x}, s)$ relative to the image.

[1] Not relative to the image point, because *that* velocity would be zero!

[2] Although Euler (1707-1783) and Lagrange (1736-1813) never met in person, they had an active correspondence by mail between 1754 and 1775 around the topic of the calculus of variations, which they each invented independently. Their lifetimes overlapped by about 48 years.

The Optical Flow The motion field is not always observable in a video, and not every change in video corresponds to the motion of visible points in the scene. For instance, a perfectly smooth, evenly colored sphere rotating around its axis produces no changes in the image: The motion field is nonzero, but it leaves no trace in the video. Conversely, a static scene lit by a moving light source (with the source placed outside the image, so it is not directly visible in it) produces changes in the image, but none of the points that are visible in the video move: The motion field is zero, but the image changes over time.

Thus, the estimation of the motion field from video suffers from a fundamental observability issue, and the term *optical flow* has been used in the literature to denote whatever aspects of the motion field are observable. For instance, with the rotating sphere, we would say that the motion field is nonzero (points in the scene truly move in the image) but the optical flow is zero (the motion is not visible). In the case of lighting change described above, the reverse is typically the case: The optical flow is nonzero (image brightness patterns change, giving the illusion of shadows sliding on objects) but the motion field is zero (no point in the scene truly moves). Thus, in some sense, the motion field is the true motion (*i.e.*, the projection of world motion onto the image plane), while the optical flow is the measured motion.

However, there is no precise definition for "optical flow," and as a result this term is often used interchangeably with "motion field." We will do so as well in these notes, but it is important to remain cognizant of this fundamental conceptual discrepancy between what we would like to measure (the noumenon, in Kant's terminology) and what we can actually measure from video (the phenomenon).

2 The Optical Flow Constraint Equation

To measure the motion field from video it is necessary to make some assumption on how frames in a video sequence relate to each other. After all, images do not move, but are merely arrays of irradiance measurements, and one needs to *interpret* the changes in the observed brightness patterns as motion by assuming that different frames are related views of the same scene. Without such an assumption, the frames could be images of entirely unrelated scenes, and the notion of "image motion" would then be meaningless.[3]

The most widely made assumption in the literature on visual motion analysis is that the appearance of any given point in the world does not change over time, at least over small temporal intervals, as the point moves across the field of view: A point in the world that looks pink and of a certain brightness now will be of the same pink and brightness a second from now if we keep looking at it. This assumption of *constant appearance* is violated as a result of changes in shading or illumination, or because the same point in the world may appear different when viewed from different viewpoints because it reflects light differently in different directions. Nonetheless, the assumption holds in a wide variety of circumstances, and is arguably one of the weakest (*i.e.*, most generic) assumptions one can make on how images in video relate to each other.

Mathematically, the assumption of constant appearance can be expressed by taking a Lagrangian point of view: Pick some point in the world, and ride its projection $\mathbf{x}(t)$ as it moves through the image.[4] Then, the irradiance $e(\mathbf{x}(t), t)$ seen by Lagrange, who rides the point, is

[3] Let this remark sink in for a moment: Image motion is not "directly visible," but is rather the result of a computation (what we can call an "interpretation" of what we see). Eyes are not enough to see, we also need a brain.

[4] We are talking about a single point now, so we can ignore the additional arguments $\boldsymbol{\xi}$ and s to $\mathbf{x}$ to keep the

assumed to be constant:

$$\frac{de(\mathbf{x}(t),t)}{dt} \stackrel{\text{def}}{=} \lim_{\Delta t \to 0} \frac{e(\mathbf{x}(t+\Delta t), t+\Delta t) - e(\mathbf{x}(t),t)}{\Delta t} = 0 \,. \tag{5}$$

In this differential form, the constant-appearance assumption is called the *optical flow constraint equation* or the *Brightness Change Constraint Equation* (BCCE). Expanding equation 5 through the chain rule of partial differentiation yields

$$\frac{\partial e}{\partial \mathbf{x}^T} \frac{d\mathbf{x}}{dt} + \frac{\partial e}{\partial t} = 0 \,.$$

Equations 1, 2, and 3 confirm that $d\mathbf{x}/dt$ is the motion field $\mathbf{v}(t)$, so we can write

$$\frac{\partial e}{\partial \mathbf{x}^T} \mathbf{v} + \frac{\partial e}{\partial t} = 0 \,. \tag{6}$$

This is a system of up to three equations (because $\mathbf{e}$ has red, green, and blue components in a clolor image, or a single component in a gray image) in the two components of the unknown motion field $\mathbf{v}$ at time t. The derivatives of the irradiance $\mathbf{e}$ can be measured or at least estimated from video, so that *equation 6 is the mathematical basis for the computation of the motion field under the assumption of constant appearance.*

Equation 6 converts Lagrange's view to Euler's: The *partial* derivatives $\partial e/\partial \mathbf{x}^T$ and $\partial e/\partial t$ are Euler measurements, taken at a fixed position $\mathbf{x}$ in the image. For instance,

$$\frac{\partial e(\mathbf{x}(t),t)}{\partial t} = \lim_{\Delta t \to 0} \frac{\mathbf{e}(\mathbf{x}(t), t+\Delta t) - \mathbf{e}(\mathbf{x}(t),t)}{\Delta t}$$

(notice the difference with respect to equation 5, where we had a total derivative instead). The velocities measured by Lagrange and Euler happen to coincide at that point in time and are both equal to $\mathbf{v}$.

3 The Aperture Problem

For a black-and-white video, equation 6 is a single equation in the two components of $\mathbf{v}$, the unknown motion field, and does therefore not constrain the motion field uniquely. This degeneracy is called the *aperture problem*, and makes the motion field unobservable without further assumptions. The aperture problem is unavoidable with black-and-white video ($\mathbf{e} \in \mathbb{R}$), and is pervasive for color video as well, even if then equation (6) is a vector equation equivalent to three scalar equations, one for each of red, green, and blue.[5]

Since $\mathbf{v}$ appears in equation 6 through an inner product, only its component along the spatial gradient of e can be observed, assuming that this gradient is nonzero. If we let

$$\nabla e(\mathbf{x}) \stackrel{\text{def}}{=} \frac{\partial e(\mathbf{x})}{\partial \mathbf{x}}$$

notation simple.

[5] As mentioned earlier, color components are strongly correlated to each other, and derivatives of, say, the red component are close to those for the blue components. As a consequence, the three scalar equations are close to being linearly dependent on each other. Empirical observation indicates that color would indeed improve the situation only marginally in most cases.

be the spatial gradient (a column vector) of the image irradiance and we assume that $\nabla e(\mathbf{x}) \neq \mathbf{0}$ at $\mathbf{x}$, then the scalar

$$v(\mathbf{x}) \stackrel{\text{def}}{=} \|\nabla e(\mathbf{x})\|^{-1} \, [\nabla e(\mathbf{x})]^T \, \mathbf{v}(\mathbf{x})$$

is the component of the motion field along the spatial gradient of e and is called the *normal component* of the motion field.[6]

Thus, when $\nabla e(\mathbf{x}) \neq \mathbf{0}$ and the optical flow constraint equation 6 holds, only the normal component of the motion field is observable, while the component orthogonal to $\nabla e(\mathbf{x})$ is not. In addition, the earlier discussion of the difference between motion field and optical flow in the case of the rotating sphere is an example where the motion field is completely unobservable even if $\nabla e(\mathbf{x}) \neq \mathbf{0}$: In that case, the assumption of constant appearance, and therefore the optical flow constraint equation, is violated, because a fixed point on the rotating sphere changes in appearance (brightness) as it enters areas with different amounts of lighting.

The aperture problem arises even if one considers a small but non-infinitesimal patch of the image. For instance, if the patch covers a part of the image where the brightness or color pattern is more or less uniform, such as in the middle of a blank wall, a small motion inside 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. A less severe version of the aperture problem arises when the feature window straddles a straight intensity edge. In that case, motion across the edge is clearly visible, but motion along it is not. Figure 2 illustrates.

4 Estimating the Motion Field

Image displacements are not necessarily integer-valued, even in the absence of occlusions: A point visible at the center of a particular pixel in the first frame is not necessarily visible at the center of some pixel in the second frame, but may rather be visible at some position between pixel centers. Because of this, the computation of displacements cannot be framed as a partial bipartite matching (match pixels to pixels), but is rather cast as the computation of a vector field defined on either frame: For each pixel $\mathbf{i} \in \Omega$ in frame $n \in \mathbb{Z}$ one assigns a real-valued displacement $\mathbf{d}(\mathbf{i}P, nT, (n+1)T)$ or, if the point becomes occluded in frame $n+1$, no displacement. If the frame interval T is very small, this displacement may yield an acceptable approximation for a multiple of the optical flow or motion field $\mathbf{v}$:

$$\mathbf{u}(\mathbf{i}, n) \stackrel{\text{def}}{=} T\mathbf{v}(\mathbf{i}P, nT) \approx \mathbf{d}(\mathbf{i}P, nT, (n+1)T) \tag{7}$$

at that pixel and time.

Because of the aperture problem, the motion field at pixel position $\mathbf{i}$ must be estimated from the pixel values (in two or more video frames) in an area of the image that extends beyond the point $\mathbf{i}$ itself. This area is called the *support* of the computation. Two types of support are common:

- The support is a square centered at $\mathbf{i}$ and with a side of $s = 2h+1$ pixels, where h is a positive integer, so that s is odd. The square is called the *window* centered at $\mathbf{i}$. The motion field at two different points $\mathbf{i}$ and $\mathbf{i}'$ is estimated through separate computations, even when the two windows overlap.

  When a large window is used, the brightness pattern within the window is likely to be more varied than in a smaller window, and the aperture problem is less likely to arise. However,

[6] Note that $\|\nabla e(\mathbf{x})\|^{-1} \, \nabla e(\mathbf{x})$ is a unit vector in the direction of the gradient.

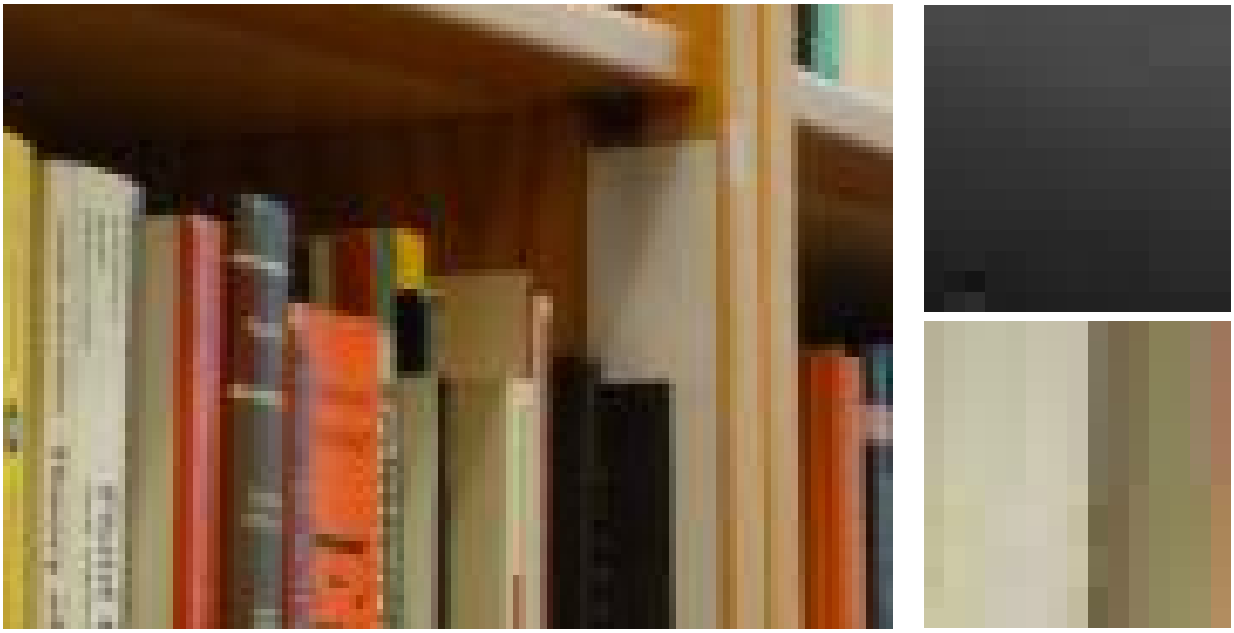

Figure 2: The 15×15 patch at the top right comes from the relatively blank area at the top left of the low-resolution image detail shown on the left. Shifting that patch around by a few pixels does not change its contents appreciably, and any changes that do occur may be small compared to image noise: Motion cannot be measured reliably at all in that area.
The 15×15 patch at the bottom right comes from around one of the many vertical edges in the image. Shifting the patch 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 patch 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.

a single displacement is reported for the entire window, even if multiple displacements occur within it. As a consequence, larger windows lead to lower variance in the results, because motion is more likely to be observable, but higher bias, because the window may contain multiple motions but only one is reported.

In addition, displacements computed from overlapping windows are potentially more similar to each other than the true displacements are, because the same motions affect the overlap area. In other words, lower variance (that is, lower sensitivity to noise) leads to lower spatial resolution in the resulting displacements.

- The support is the entire image. In this case, the motion is estimated jointly at all pixel positions, and the assumption of a single displacement within the support is of course unwarranted even approximately. Instead, each pixel is assigned a (possibly different) motion field vector, and some form of regularization is used to address the lack of observability implied by the aperture problem. Specifically, the estimated motion field is made to be spatially smooth by levying a penalty on differences between field values at neighboring pixel positions. Again, a larger penalty leads to lower variance but tends to smooth the motion field estimates, with the effect being particularly felt (high bias) along motion discontinuities.

Methods with local support are described next, and some global-support methods are discussed in a subsequent note.

5 Local-Support Motion Estimation

Motion estimation methods with local support track one image window at a time. This leads to a simple and efficient algorithm called the *Lucas-Kanade tracker* [3], which assumes that all motions in a window are the same motion. After a general discussion of window tracking in Section 5.1, Section 5.2 describes the Lucas and Kanade tracker. Practical aspects are discussed in Section 5.3.

5.1 Tracking Windows

Let $f(\mathbf{x})$ and $g(\mathbf{x})$ 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 $W(\mathbf{x}_f)$ of odd side-length $2h+1$ pixels, centered at some point $\mathbf{x}_f$ in f. The question is, what are the coordinates

$$\mathbf{x}_g = \mathbf{x}_f + \mathbf{d}^*(\mathbf{x}_f)$$

of the corresponding window's center in image g? The two-dimensional vector $\mathbf{d}^*(\mathbf{x}_f)$ 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[7] that the magnitude of $\mathbf{d}^*(\mathbf{x}_f)$ is much smaller than the sidelength of $W(\mathbf{x}_f)$.

A natural way to answer the question above is to measure the *dissimilarity* or *residual* between $W(\mathbf{x}_f)$ and a candidate window in g with the following *loss function*:

$$L(\mathbf{x}_f, \mathbf{d}) = \sum_{\mathbf{x}} [g(\mathbf{x}+\mathbf{d}) - f(\mathbf{x})]^2 \; w(\mathbf{x}-\mathbf{x}_f) \tag{8}$$

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

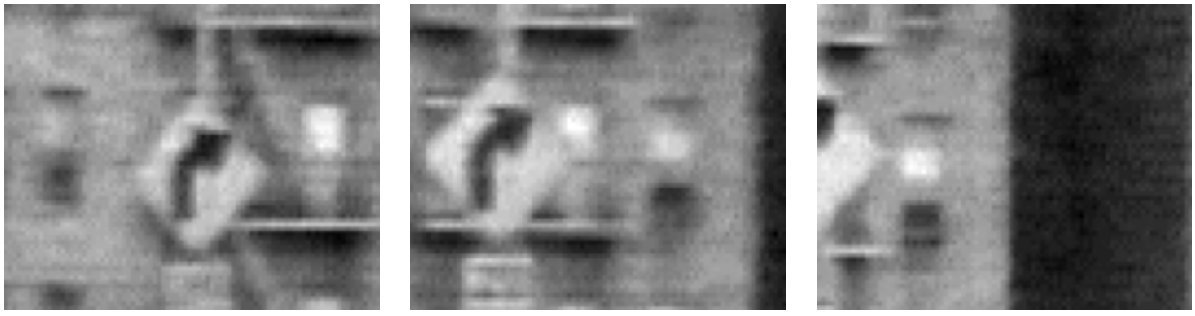

Figure 3: Details from three consecutive frames out of Woody Allen's 1979 movie *Manhattan*.

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 (8) is thus the sum of squared differences between the pixels in a window around $\mathbf{x}_f$ in f and a window around $\mathbf{x}_g = \mathbf{x}_f + \mathbf{d}$ in g. One can then search for the displacement that makes the dissimilarity as small as possible:

$$\mathbf{d}^*(\mathbf{x}_f) = \arg\min_{\mathbf{d} \in R} L(\mathbf{x}_f, \mathbf{d}) \tag{9}$$

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 $W(\mathbf{x}_f)$ with the search range. The former is part of the image f, 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.

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

This approach to correspondence makes the implicit assumption that the image motions contained in $W(\mathbf{x}_f)$ 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 (8) depends more on what happens close to the center

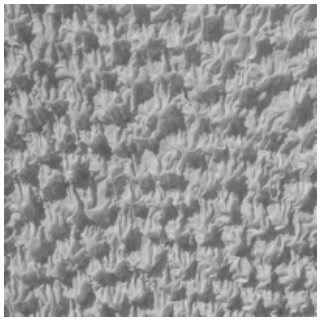

Figure 4: Detail from a fabric sample. [Image adapted from mayang.com.]

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 [4], rather than just a translation.[8]

One can use any of a wide variety of optimization methods to solve problem (9), including direct grid search: Simply write a `for` loop that searches for $\mathbf{d}^*(\mathbf{x}_f)$ 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 (8) 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, some computer vision problems that require image motion information as input, such as 3D reconstruction, are ill-posed problems, 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 [4], but accurate, sub-pixel displacements are desirable in any case, because of the ill-posedness of problems like reconstruction.

Because of these reasons, features are often tracked by gradient-based optimization methods, perhaps after grid search has provided a good starting point. The original paper by Lucas and Kanade [3] uses the Newton-Raphson method, described next.

[8] However, even an affine model would fail to do justice to the discontinuous pattern of flow fields illustrated in Figure 3.

5.2 The Lucas and Kanade Tracker

Let us simplify the notation for the optimization problem (9) and residual (8) as follows:

$$\mathbf{d}^* = \arg\min_{\mathbf{d}} L(\mathbf{d}) \tag{10}$$

where we abbreviate $L(\mathbf{x}_f, \mathbf{d})$ to $L(\mathbf{d})$. 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 $\mathbf{x}_f$ of the window in image f). The choice of replacing $\mathbf{d} \in R$ with an unconstrained $\mathbf{d}$ in the minimization (10) 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 $\mathbf{x}_f$. 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 $L(\mathbf{d}^*)$ 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 (10) amounts to starting at some initial point $\mathbf{d}_0$ at iteration $t = 0$. In iteration t, find the Hessian

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

and gradient

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

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

$$H(\mathbf{d}_t)\mathbf{s} = -\nabla L(\mathbf{d}_t) \tag{11}$$

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

$$\mathbf{d}_{t+1} = \mathbf{d}_t + \mathbf{s}_t \ , \tag{12}$$

and iterate until convergence.

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

$$L(\mathbf{d}) \approx L(\mathbf{d}_t) + [\nabla L(\mathbf{d}_t)]^T(\mathbf{d} - \mathbf{d}_t) + \frac{1}{2}(\mathbf{d} - \mathbf{d}_t)^T H(\mathbf{d}_t)(\mathbf{d} - \mathbf{d}_t) \ .$$

In other words, $L(\mathbf{d})$ is replaced by the best quadratic fit to it around $\mathbf{d}_t$. Setting the derivative of this approximation to zero yields the system (11) 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 $L(\mathbf{d})$, the process is iterated.

The issue with applying Newton's method directly to problem (10) is that the argument $\mathbf{d}$ in function $L(\mathbf{d})$ is contained inside the expression $g(\mathbf{x}+\mathbf{d})$ (see equation (8)), 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* g around every point $\mathbf{x}$ in the image window, and uses the linearization in the definition of residual (8). This leads to a different quadratic approximation than the one given by the Taylor expansion of $L(\mathbf{d})$, 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

$$g_t(\mathbf{x}) = g(\mathbf{x} + \mathbf{d}_t)$$

be the result of shifting the image g 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 12). To this end, the nonlinear, shifted image function $g_t(\mathbf{x}+\mathbf{s}) = g(\mathbf{x}+\mathbf{d}_t+\mathbf{s})$ is replaced with its first-order Taylor expansion around $\mathbf{x}$,

$$g_t(\mathbf{x}+\mathbf{s}) \approx g_t(\mathbf{x}) + [\nabla g_t(\mathbf{x})]^T \mathbf{s} \tag{13}$$

so that the residual (8) at the (unknown) point $\mathbf{d}_t + \mathbf{s}$ can be approximated as follows:

$$\begin{aligned} L(\mathbf{d}_t+\mathbf{s}) &= \sum_{\mathbf{x}} [g(\mathbf{x}+\mathbf{d}_t+\mathbf{s}) - f(\mathbf{x})]^2 \; w(\mathbf{x}-\mathbf{x}_f) \\ &= \sum_{\mathbf{x}} [g_t(\mathbf{x}+\mathbf{s}) - f(\mathbf{x})]^2 \; w(\mathbf{x}-\mathbf{x}_f) \\ &\approx \sum_{\mathbf{x}} [g_t(\mathbf{x}) + [\nabla g_t(\mathbf{x})]^T \mathbf{s} - f(\mathbf{x})]^2 \; w(\mathbf{x}-\mathbf{x}_f) \; , \end{aligned}$$

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

$$\nabla L(\mathbf{d}_t+\mathbf{s}) \approx 2 \sum_{\mathbf{x}} \nabla g_t(\mathbf{x}) \{ g_t(\mathbf{x}) + [\nabla g_t(\mathbf{x})]^T \mathbf{s} - f(\mathbf{x}) \} \; w(\mathbf{x}-\mathbf{x}_f) \; .$$

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

$$A\mathbf{s} = \mathbf{b} \tag{14}$$

where[9]

$$A = \sum_{\mathbf{x}} \nabla g_t(\mathbf{x}) [\nabla g_t(\mathbf{x})]^T \; w(\mathbf{x}-\mathbf{x}_f) \quad \text{and} \quad \mathbf{b} = \sum_{\mathbf{x}} \nabla g_t(\mathbf{x}) [f(\mathbf{x}) - g_t(\mathbf{x})] \; w(\mathbf{x}-\mathbf{x}_f) \; . \tag{15}$$

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 g_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 g_t further by the step $\mathbf{s}_t$. The overall displacement is then the sum of all the steps,

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

[9]Note that $-2\mathbf{b}$ is the gradient of L at $\mathbf{d}_t$. So by comparing with the normal equation (11), we see that $2A$ is an approximation to the Hessian of A.

Since g is continually shifted during iterations of the algorithm, the image $g_t(\mathbf{x})$ becomes more and more similar to $f(\mathbf{x})$ within the feature window, and the residual decreases. In addition, the Taylor approximation (13) becomes better and better, because the step $\mathbf{s}_t$ becomes smaller and smaller.

Algorithm 1 in Appendix B summarizes the computation. As mentioned earlier, not every window can be tracked well, because of the aperture problem. Appendix B.1 shows how to pick good windows to track.

5.3 Practicalities

This Section discusses practical aspects of iterative window tracking, namely, how to handle real-valued image coordinates; how to initialize the iteration; how to check for termination; and how to track windows that are smaller that their displacements.

Interpolation The very first time the algorithm is run, the initial point $\mathbf{x}_f$ is 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 $\mathbf{x}_f$ of the window in f (frame 2 in this case) may be non-integer. Because of this, images are sampled on non-integer grids by bilinear interpolation.

Initialization 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.

Termination Check Checks for convergence include a threshold on the size of the current step $\mathbf{s}_t$ and on the decrease in residual. In addition, since the constraint that the overall displacement be within the range R is ignored in the problem formulation, 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

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

for suitable positive thresholds δ, ϵ, ρ, $t_{\max}$. Violating either of the first two conditions signals convergence, while violating either of the last two implies failure.

Large-Displacement Window Tracking 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 $\mathbf{x}_f^{(k)}$ at level k, let $\mathbf{x}_f^{(k+1)}$ be the coordinates of the corresponding point at the next-coarser level $k+1$. Let $\mathbf{d}_0^{(k+1)}$ be the displacement found at the point among the tracked points at level $k+1$ that is nearest to $\mathbf{x}_f^{(k+1)}$. Finally, scale the displacement $\mathbf{d}_0^{(k+1)}$ to account for the scale difference between the two levels and use the scaled

displacement to initialize the Lucas and Kanade tracker at $\mathbf{x}_f^{(k)}$. This scheme works rather well for relatively large motions and with only a modest number of levels in the pyramid [5].

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

Appendices

A Image Sampling

Every point $\mathbf{x} \in \mathbb{R}^2$ on the sensor sees some pattern of colors $\mathbf{e}(\mathbf{x}, t)$ at time $t \in \mathbb{R}$. This function is called the *irradiance*. While irradiance is generally a spectral distribution of wavelengths, the camera uses three sets of filters to record the amount of light in each of three bands of wavelengths roughly centered around red, green, and blue light. Accordingly, we encode the irradiance by a triple of nonnegative real numbers, $\mathbf{e} \in \mathbb{R}^3$ with $\mathbf{e} \geq 0$.

The output of the camera is an *image sequence* (or *video*)

$$\mathbf{f} \ : \ \Omega \times \mathbb{Z} \to \mathbb{N}^3 \ ,$$

a discrete three-dimensional array $\mathbf{f}(\cdot, n)$ of nonnegative integers at each of a discrete sequence of times n. The *image domain* $\Omega \subset \mathbb{N}^2$ for the discrete space variable $\mathbf{i}$ is a finite rectangular grid of nonnegative integers. While every video has a beginning, and many have an end, it is often mathematically simpler to view every video as being extended indefinitely into the past and future. This is why n is modeled as an integer, $n \in \mathbb{Z}$, rather than a natural number ($n \in \mathbb{N}$).

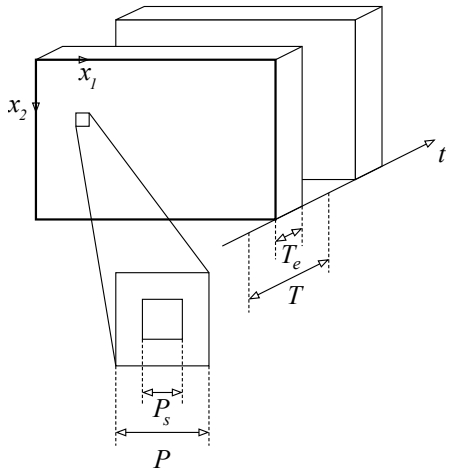

Figure 5: A pixel integrates image irradiance over both space and time.

Each *pixel value* $\mathbf{f}(\mathbf{i}, n)$ is the integral of irradiance over a small, rectangular volume of space and time:

$$\mathbf{f}(\mathbf{i}, n) \ = \ Q \left(\int_{nT - T_e/2}^{nT + T_e/2} \left[\iint_{\mathbf{i}P - P_s/2}^{\mathbf{i}P + P_s/2} \mathbf{e}(\mathbf{x}, t) \ d\mathbf{x} \right] dt \ + \ \boldsymbol{\nu}(\mathbf{i}, n) \right) \ . \tag{16}$$

In this expression, the positive real number P is the *pixel pitch*, that is, the distance between adjacent pixels, which we assume to be the same in the vertical and horizontal direction. The positive real number P_s is the *pixel size*, that is, the part of the pixel pitch over which each pixel senses light. The positive real number T is the *frame interval*, namely, the time elapsed between a frame and the next, and the positive real number $T_e \leq T$ is the *exposure time*, that is, the part of T over which incoming light is recorded. Finally, $\boldsymbol{\nu}$ is measurement noise, and the function $Q(\cdot)$

quantizes the measured values, that is, it rounds them to integers. This function also clips values to be between 0 and 255. Figure 5 illustrates.

B Pseudo-Code for the Lucas-Kanade Algorithm

The time subscript t is dropped everywhere in the algorithm listing below because old variable values are simply overwritten by new ones.

The set X defined in line 1 initially keeps track of the coordinates where the window function $w(\mathbf{x})$ is nonzero.[10] After X has been used to store 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 $\mathbf{x}_f$ of the window of interest in image f (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 g, rather than that in f. The matrix G defined on line 12 is $K \times 2$, and each of its rows represents the gradient of g at one of the pixels in the moving window. Lines 13 and 14 assemble the 2×2 system (14), and line 17 accumulates the steps into the overall displacement.

B.1 Choosing Good Windows to Track

Because of the aperture problem, only some image windows can be tracked well. These windows could be defined by requiring that their contents are sufficiently varied. This was done by Harris and Stephens [2] in 1988, and led to what is called the *Harris detector*.

This Appendix follows instead the derivation given by Shi and Tomasi [4] in 1994 for a feature tracker that generalized the Lucas and Kanade tracker from pure translation to affine image deformations. The main advantage of the Shi-Tomasi derivation is pedagogical brevity, as both the tracking method and criteria for determining the best windows to track are developed within the same theoretical framework.

However, this Appendix simplifies the treatment by reverting 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 optimization problem addressed by the Lucas and Kanade tracker can be solved reliably when the system (14) at the core of the algorithm has a well-defined solution, that is, when the matrix A is far from degenerate, so it can be safely inverted. Distance from degeneracy can be measured by the condition number of A.

However, the matrix A keeps changing during execution of the Lucas and Kanade algorithm (see line 13 in Algorithm 1). So in order to have a single estimate of the condition number we take the (unique) matrix $A_f(\mathbf{x}_f)$ computed around point $\mathbf{x}_f$ in image f to be an estimate of the (changing) matrix A for the moving window in g. In summary, we say that a point feature centered at $\mathbf{x}_f$ in image f is a *good feature to track* if the smaller eigenvalue of $A(\mathbf{x}_f)$ is above some predefined threshold:

$$\lambda_{\min}(A(\mathbf{x}_f)) \geq \lambda_0 \quad \text{where} \quad A_f(\mathbf{x}_f) = \sum_{\mathbf{x}} \nabla f(\mathbf{x}) [\nabla f(\mathbf{x})]^T \; w(\mathbf{x} - \mathbf{x}_f) \; .$$

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

Algorithm 1. The Lucas and Kanade tracker

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

$X \leftarrow \{\mathbf{x} \mid w(\mathbf{x}) > 0\}$ ▷ 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 $w(\mathbf{x})$

$X \leftarrow X + \mathbf{x}_f$ ▷ The set X now contains the window coordinates in f

$\mathbf{i} \leftarrow f(X)$ ▷ $\mathbf{i}$ is a column vector of all the image values of f on X

$\mathbf{d} \leftarrow \mathbf{d}_0$ ▷ Initialize the cumulative displacement

$\mathbf{s} \leftarrow \mathbf{d}_0$ ▷ The first shift of g 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 g

  $\mathbf{j} \leftarrow g(X)$ ▷ $\mathbf{j}$ is a column vector of all the image values of g on X

  $G \leftarrow \nabla g(X)$ ▷ G is a two-column matrix of all the gradients of g on X

  $A \leftarrow G^T(G \texttt{ .*} [\mathbf{w}, \mathbf{w}])$ ▷ The 2×2 matrix A in equation (14)

  $\mathbf{b} \leftarrow G^T((\mathbf{i} - \mathbf{j}) \texttt{ .*} \mathbf{w})$ ▷ The 2×1 vector $\mathbf{b}$ in equation (14)

  $\mathbf{s} \leftarrow A \setminus \mathbf{b}$ ▷ 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 $= \|\mathbf{s}\| \leq \delta$ or $L(\mathbf{d}_{\text{old}}) - L(\mathbf{d}) \leq \epsilon$ ▷ 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 $L(\mathbf{d}) > L_{\max}$ **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 $\mathbf{x}_f$, or a "failure" value.

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

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

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 $A_f(\mathbf{x}_f)$ 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 $A_f(\mathbf{x}_f)$ 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 $\nabla f(\mathbf{x})$ is everywhere zero in the window and so is $\Gamma(\mathbf{x})$. As a consequence, $A_f(\mathbf{x}_f)$ is zero as well, and so is its smaller eigenvalue. If $A_f(\mathbf{x}_f)$ 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,

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

and therefore

$$A_f(\mathbf{x}_f) = \begin{bmatrix} a_{11}(\mathbf{x}_f) & 0 \\ 0 & 0 \end{bmatrix} .$$

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

Non-Maximum Suppression When the window centered at $\mathbf{x}_f$ is a good feature to track, chances are that windows that overlap with it are good features as well, because they contain closely related pixel values. Thus, in order to avoid the unnecessary expense of tracking multiple points with heavily overlapping windows (that is, to avoid tracking essentially the same point multiple times), the good features can be selected by *non-maximum suppression*: Pick a best point feature in the image, that is, a point feature with largest λ_{min}. 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 overlap$(X, \mathbf{x}^T)$ on line 16 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 f, window function $w(\mathbf{x})$, threshold $\lambda_0 > 0$ on the smallest $\lambda_{\min}$ allowed, and maximum number $n_{\max}$ of features needed (can be set to infinity)

$X \leftarrow \text{pixels}(f)$ ▷ X is a two-column matrix of all the pixel coordinates in the image

$G \leftarrow \text{gradient}(f)$ ▷ G is a two-column matrix of all the gradients in the image

$\Gamma \leftarrow [G(:,1)\ .\hat{}\ \ 2,\ G(:,1)\ .*\ G(:,2),\ G(:,2)\ .\hat{}\ \ 2]$ ▷ Distinct entries of $\nabla f(\mathbf{x})[\nabla f(\mathbf{x})]^T$

$A \leftarrow \Gamma * w$ ▷ Convolution of $\Gamma(\mathbf{x})$ with the window function $w(\mathbf{x})$

$S \leftarrow A(:,1) + A(:,3)$ ▷ A piece of the formula for eigenvalues

$D \leftarrow \sqrt{(A(:,1) - A(:,3))\ .\hat{}\ \ 2 + 4\,A(:,2)\ .\hat{}\ \ 2}$ ▷ Discriminant

$\lambda_{\min} \leftarrow \frac{1}{2}(S - D)$ ▷ Smaller eigenvalue

$X \leftarrow X(\lambda_{\min} \geq \lambda_0\,,\ :)$ ▷ Good enough features

$X \leftarrow X$ sorted by decreasing $\lambda_{\min}$ ▷ Put best features first

$Y \leftarrow [\,]$ ▷ Y collects the final point-feature coordinates

for $n = 1$ to $n_{\max}$ **do** ▷ Loop for non-maximum suppression

  if X is empty **then** ▷ No more good features left

    break

  end if

  $Y \leftarrow [Y; X(1,:)]$ ▷ Pick the best feature remaining

  $X \leftarrow X(\sim\text{overlap}(X, X(1,:)), :)$ ▷ Keep only windows that do not overlap with $X(1,:)$

end for

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