Tracking Feature Windows

Carlo Tomasi

April 6, 2019

Local motion estimation methods track one image window at a time. This leads to simple and efficient algorithms, at the cost of assuming that all motions in a window are the same motion, and that motions in different windows are unrelated to each other.

Two decisions must be made when the support for the computation of image motion is a window:

- Which windows to track
- How to track them

Because of the aperture problem, not every window can be tracked well, so the first decision above requires some care. 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 [2] in 1988, and led to what is called the *Harris detector*. This note follows instead the derivation given by Shi and Tomasi [4] in 1994 for a feature tracker that generalized the vastly popular *Lucas and Kanade tracker* [3] 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. 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 addition, the presentation is simplified by assuming that the images are black-and-white. Generalization to color images complicates the notation but is conceptually straightforward, and amounts essentially to replacing gradients with Jacobians.

In summary, after a general discussion of tracking in Section 1, Section 2 describes the Lucas and Kanade tracker, Section 3 sketches what to do when some of the displacements are large, and Section 4 derives a point-feature detector very similar to the Harris detector by following the derivation by Shi and Tomasi.

1 Tracking

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

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

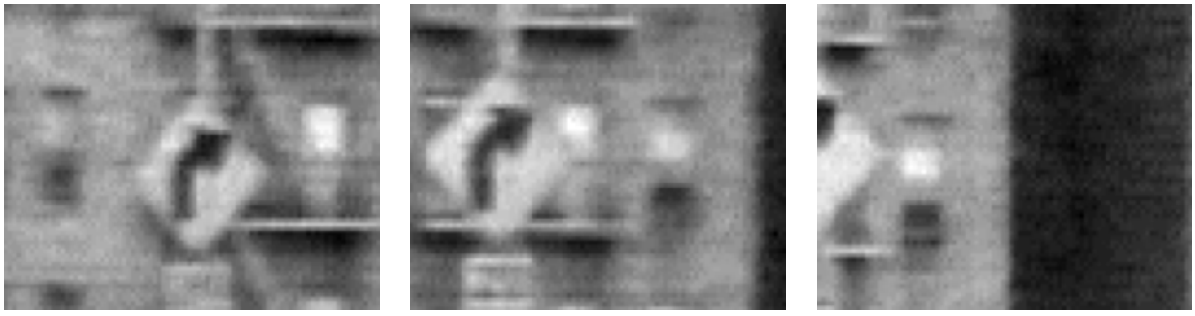

Figure 1: Details from three consecutive frames out of Woody Allen's 1979 movie *Manhattan*.

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[1] 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*:

$$L(\mathbf{x}_f, \mathbf{d}) = \sum_{\mathbf{x}} [g(\mathbf{x} + \mathbf{d}) - f(\mathbf{x})]^2 \; w(\mathbf{x} - \mathbf{x}_f) \tag{1}$$

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 $\mathbf{x}_f$ 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:

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

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 $W(\mathbf{x}_f)$ 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.

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

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

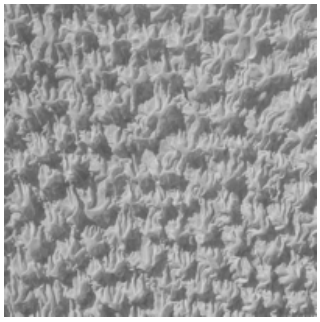

Figure 2: Detail from a fabric sample. [Image adapted from mayang.com.]

be violated even for small motions and windows. For example, Figure 1 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 [4], rather than just a translation.[2]

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 $\mathbf{d}^*(\mathbf{x}_f)$ 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 2. 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, 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,

[2]However, even an affine model would fail to do justice to the discontinuous pattern of flow fields illustrated in Figure 1.

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 differential 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.

2 The Lucas and Kanade Tracker

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

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

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 (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 $\mathbf{x}_f$. 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 (3) 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*

$$H(\mathbf{d}_t)\mathbf{s} = -\nabla L(\mathbf{d}_t) \tag{4}$$

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 $L(\mathbf{d})$ 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 (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 $L(\mathbf{d})$, the process is iterated.

We saw in a previous note that the rate of convergence of Newton's method is quadratic in the following sense:

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

where β is a positive real number[3]. 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 $L(\mathbf{d})$ is contained inside the expression $g(\mathbf{x}+\mathbf{d})$ (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* g 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 $L(\mathbf{d})$, 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 5). 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 $\mathbf{x}$,

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

so that the residual (1) 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

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

[3]Recall that the term "quadratic" refers to the exponent 2 in the denominator.

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

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

where[4]

$$A = \sum_{\mathbf{x}} \nabla g_t(\mathbf{x})[\nabla g_t(\mathbf{x})]^T \; w(\mathbf{x} - \mathbf{x}_f) \quad \text{and} \quad \mathbf{b} = \sum_{\mathbf{x}} \nabla g_t(\mathbf{x})[f(\mathbf{x}) - g_t(\mathbf{x})] \; w(\mathbf{x} - \mathbf{x}_f) \; . \tag{8}$$

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,

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

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 (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.

Practicalities

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.

Algorithm 1 summarizes the computation. The set X defined in line 1 initially keeps track of the coordinates where the window function $w(\mathbf{x})$ is nonzero.[5] 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 (7), and line 17 accumulates the steps into the overall displacement.

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

[5]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 (7)

  $\mathbf{b} \leftarrow G^T((\mathbf{i} - \mathbf{j}) \texttt{ .*} \mathbf{w})$ ▷ 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 $= \|\mathbf{s}\| \leq \delta$ 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 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 $\mathbf{x}_f$ 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 $\mathbf{x}_f$ of the window in f (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.

3 Large-Displacement Feature 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].

4 Choosing Good Windows to Track

As mentioned earlier, not every feature can be tracked well, because of the aperture problem. This section shows how this notion of "trackability" can be quantified and used to pick good features to track. 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 matrix A is far from degenerate, so it can be safely inverted.

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:

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

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 $\lambda_{\min}$. 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)

1: $X \leftarrow \text{pixels}(f)$ ▷ X is a two-column matrix of all the pixel coordinates in the image
2: $G \leftarrow \text{gradient}(f)$ ▷ 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 $\nabla f(\mathbf{x})[\nabla f(\mathbf{x})]^T$
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: $\lambda_{\min} \leftarrow \frac{1}{2}(S - D)$ ▷ Smaller eigenvalue
8: $X \leftarrow X(\lambda_{\min} \geq \lambda_0\,,\ :)$ ▷ Good enough features
9: $X \leftarrow X$ sorted by decreasing $\lambda_{\min}$ ▷ Put best features first
10: $Y \leftarrow [\ ]$ ▷ Y collects the final point-feature coordinates
11: **for** $n = 1$ to $n_{\max}$ **do** ▷ Loop for non-maximum suppression
12: **if** X is empty **then** ▷ No more good features left
13: break
14: **end if**
15: $Y \leftarrow [Y; X(1,:)]$ ▷ Pick the best feature remaining
16: $X \leftarrow X(\sim\text{overlap}(X, X(1,:)), :)$ ▷ Keep only windows that do not overlap with $X(1,:)$
17: **end for**

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

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.