Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker Description of the algorithm

Jean-Yves Bouguet

Intel Corporation
Microprocessor Research Labs
jean-yves.bouguet@intel.com

1 Problem Statement

Let I and J be two 2D grayscaled images. The two quantities $I(\mathbf{x}) = I(x, y)$ and $J(\mathbf{x}) = J(x, y)$ are then the grayscale values of the two images at the location $\mathbf{x} = [x \;\; y]^T$, where x and y are the two pixel coordinates of a generic image point $\mathbf{x}$. The image I will sometimes be referenced as the first image, and the image J as the second image. For practical issues, the images I and J are discret function (or arrays), and the upper left corner pixel coordinate vector is $[0 \;\; 0]^T$. Let n_x and n_y be the width and height of the two images. Then the lower right pixel coordinate vector is $[n_x - 1 \;\; n_y - 1]^T$.
Consider an image point $\mathbf{u} = [u_x \;\; u_y]^T$ on the first image I. The goal of feature tracking is to find the location $\mathbf{v} = [u_x + d_x \;\; u_y + d_y]^T$ on the second image J such as $I(\mathbf{u})$ and $J(\mathbf{v})$ are "similar". The vector $\mathbf{d} = [d_x \;\; d_y]^T$ is the image velocity at $\mathbf{u}$, also known as the optical flow at $\mathbf{u}$. In addition to a translation component $\mathbf{d}$, let us assume that the image undergoes an affine deformation between I and J in the vicinity of the two image feature points $\mathbf{u}$ and $\mathbf{v}$. Following this assumption, let us introduce the affine transformation matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 1 + d_{xx} & d_{xy} \\ d_{yx} & 1 + d_{yy} \end{bmatrix}, \tag{1}$$

where the four coefficients d_{xx}, d_{xy}, d_{yx} and d_{yy} characterize the affine deformation of the image patch. The objective of affine tracking is then to find the vector $\mathbf{d}$ and the matrix $\mathbf{A}$ that minimize the residual function ϵ defined as follows:

$$\epsilon(\mathbf{d}, \mathbf{A}) = \epsilon(d_x, d_y, d_{xx}, d_{xy}, d_{yx}, d_{yy}) = \sum_{x=-\omega_x}^{\omega_x} \sum_{y=-\omega_y}^{\omega_y} \left(I(\mathbf{x} + \mathbf{u}) - J\left(\mathbf{A}\,\mathbf{x} + \mathbf{d} + \mathbf{u}\right)\right)^2, \tag{2}$$

where two integers ω_x and ω_y set the size of the integration window to $(2\omega_x + 1) \times (2\omega_y + 1)$. Typical values for ω_x and ω_y are 7,8,10,20 pixels (significantly larger than for translation tracking, since we need to estimate local image deformation).

2 Description of the tracking algorithm

The two key components to any feature tracker are accuracy and robustness. The accuracy component relates to the local sub-pixel accuracy attached to tracking. Intuitively, a small integration window would be preferable in order not to "smooth out" the details contained in the images (i.e. small values of ω_x and ω_y). That is especially required at occluding areas in the images where two patchs potentially move with very different velocities.

The robustness component relates to sensitivity of tracking with respect to changes of lighting, size of image motion,... In particular, in oder to handle large motions, it is intuively preferable to pick a large integration window. Indeed, considering only equation 2, it is preferable to have $d_x \leq \omega_x$ and $d_y \leq \omega_y$ (unless some prior matching information is available). There is therefore a natural tradeoff between local accuracy and robustness when choosing the integration window size. In attempt to provide a solution to that problem, we propose a pyramidal implementation of the tracking algorithm.

2.1 Image pyramid representation

Let us define the pyramid representation of a generic image I of size $n_x \times n_y$. Let $I^0 = I$ be the "zero$^{\text{th}}$" level image. This image is essentially the highest resolution image (the raw image). The image width and height at that level are defined as $n_x^0 = n_x$ and $n_y^0 = n_y$. The pyramid representation is then built in a recursive fashion: compute I^1 from I^0, then compute I^2 from I^1, and so on... Let $L = 1, 2, \ldots$ be a generic pyramidal level, and let I^{L-1} be the image at level $L - 1$. Denote n_x^{L-1} and n_y^{L-1} the width and height of I^{L-1}. The image I^{L-1} is then defined as

follows:

$$\begin{aligned} I^L(x,y) &= \frac{1}{4}I^{L-1}(2x,2y) + \\ &\frac{1}{8}\left(I^{L-1}(2x-1,2y)+I^{L-1}(2x+1,2y)+I^{L-1}(2x,2y-1)+I^{L-1}(2x,2y+1)\right) + \\ &\frac{1}{16}\left(I^{L-1}(2x-1,2y-1)+I^{L-1}(2x+1,2y+1)+I^{L-1}(2x-1,2y+1)+I^{L-1}(2x+1,2y+1)\right). \end{aligned} \tag{3}$$

For simplicity in the notation, let us define dummy image values one pixel around the image I^{L-1} (for $0 \leq x \leq n_x^{L-1}-1$ and $0 \leq y \leq n_y^{L-1}-1$):

$$\begin{aligned} I^{L-1}(-1,y) &\doteq I^{L-1}(0,y), \\ I^{L-1}(x,-1) &\doteq I^{L-1}(x,0), \\ I^{L-1}(n_x^{L-1},y) &\doteq I^{L-1}(n_x^{L-1}-1,y), \\ I^{L-1}(x,n_y^{L-1}) &\doteq I^{L-1}(x,n_y^{L-1}-1), \\ I^{L-1}(n_x^{L-1},n_y^{L-1}) &\doteq I^{L-1}(n_x^{L-1}-1,n_y^{L-1}-1). \end{aligned}$$

Then, equation 3 is only defined for values of x and y such that $0 \leq 2x \leq n_x^{L-1}-1$ and $0 \leq 2y \leq n_y^{L-1}-1$. Therefore, the width n_x^L and height n_y^L of I^L are the largest integers that satisfy the two conditions:

$$n_x^L \leq \frac{n_x^{L-1}+1}{2}, \tag{4}$$

$$n_y^L \leq \frac{n_y^{L-1}+1}{2}. \tag{5}$$

Equations (3), (4) and (5) are used to construct recursively the pyramidal representations of the two images I and J: $\{I^L\}_{L=0,\ldots,L_m}$ and $\{J^L\}_{L=0,\ldots,L_m}$. The value L_m is the height of the pyramid (picked heuristically). Practical values of L_m are 2,3,4. For typical image sizes, it makes no sense to go above a level 4. For example, for an image I of size 640×480, the images I^1, I^2, I^3 and I^4 are of respective sizes 320×240, 160×120, 80×60 and 40×30. Going beyond level 4 does not make much sense in most cases. The central motivation behind pyramidal representation is to be able to handle large pixel motions (larger than the integration window sizes ω_x and ω_y). Therefore the pyramid height (L_m) should also be picked appropriately according to the maximum expected optical flow in the image. The next section describing the tracking operation in detail we let us understand that concept better. Final observation: equation 3 suggests that the lowpass filter $[1/4 \;\; 1/2 \;\; 1/4] \times [1/4 \;\; 1/2 \;\; 1/4]^T$ is used for image anti-aliasing before image subsampling. In practice however it is preferable to use a larger anti-aliasing filter kernel $[1/16 \;\; 1/4 \;\; 3/8 \;\; 1/4 \;\; 1/16] \times [1/16 \;\; 1/4 \;\; 3/8 \;\; 1/4 \;\; 1/16]^T$. This kernel is used in the C-code.

2.2 Pyramidal Feature Tracking

Recall the goal of affine feature tracking: for a given point $\mathbf{u}$ in image I, find its corresponding location $\mathbf{v} = \mathbf{u} + \mathbf{d}$ in image J, and the local image affine transformation matrix $\mathbf{A}$ relating image I and image J in the vicinity of $\mathbf{u}$ and $\mathbf{v}$ (see equation 2).

For $L = 0, \ldots, L_m$, define $\mathbf{u}^L = [u_x^L \;\; u_y^L]$, the corresponding coordinates of the point $\mathbf{u}$ on the pyramidal images I^L. Following our definition of the pyramid representation equations (3), (4) and (5), the vectors $\mathbf{u}^L$ are computed as follows:

$$\mathbf{u}^L = \frac{\mathbf{u}}{2^L}. \tag{6}$$

The division operation in equation 6 is applied to both coordinates independently (so will be the multiplication operations appearing in subsequent equations). Observe that in particular, $\mathbf{u}^0 = \mathbf{u}$.

The overall pyramidal tracking algorithm proceeds as follows: first, the optical flow and affine transformation matrix are comptuted at the deepest pyramid level L_m. Then, the result of the that computation is propagated to the upper level $L_m - 1$ in a form of an initial guess for the pixel displacement and the affine transformation (at level $L_m - 1$). Given that initial guess, the refined optical flow and affine transformation are computed at level $L_m - 1$, and the result is propagated to level $L_m - 2$ and so on up to the level 0 (the original image).

Let us now describe the recursive operation between two generics levels $L+1$ and L in more mathematical details. Assume that an initial guess for optical flow at level L, $\mathbf{g}^L = [g_x^L \;\; g_y^L]^T$ is available from the computations done from level L_m to level $L+1$. In addition, call $\mathbf{G}^L = [1 + g_{xx}^L \;\; g_{xy}^L; g_{yx}^L \;\; 1 + g_{yy}^L]$ the associated initial guess for the

affine transformation matrix. Then, in order to compute the optical flow and affine transformation matrix at level L, it is necessary to find the residual pixel displacement vector $\mathbf{d}^L = [d_x^L \;\; d_y^L]^T$ and the residual 2×2 affine matrix $\mathbf{A}^L = [1 + d_{xx}^L \;\; d_{xy}^L; 1 + d_{yx}^L \;\; d_{yy}^L]^T$ that minimize the new image matching error function ϵ^L:

$$\epsilon^L(\mathbf{d}^L, \mathbf{A}^L) = \epsilon^L(d_x^L, d_y^L, d_{xx}^L, d_{xy}^L, d_{yx}^L, d_{yy}^L) = \sum_{x=-\omega_x}^{\omega_x} \sum_{y=-\omega_y}^{\omega_y} \left(I_{\text{comp}}^L(\mathbf{x}) - J_{\text{comp}}^L\left(\mathbf{A}^L\,\mathbf{x} + \mathbf{d}^L\right)\right)^2, \tag{7}$$

where the images J_{comp}^L and J_{comp}^L are the centered compensated images computed based on the initial guess $\{\mathbf{g}^L, \mathbf{G}^L\}$:

$$I_{\text{comp}}^L(\mathbf{x}) = I^L\left(\mathbf{x} + \mathbf{u}^L\right), \tag{8}$$

$$J_{\text{comp}}^L(\mathbf{x}) = J^L\left(\mathbf{G}^L\,\mathbf{x} + \mathbf{g}^L + \mathbf{u}^L\right). \tag{9}$$

Observe that according to equation 7 the window of integration is of constant size $(2\omega_x+1) \times (2\omega_y+1)$ for all values of L. It is possible however to set different values for ω_x and ω_y at each level in the pyradmid. This is particularly useful when the residual translation vector is known to be small (the current guess for displacement is very close to the actual optical flow).

The details of computation of the residual optical flow $\mathbf{d}^L$ and the affine transformation matrix $\mathbf{A}^L$ will be described in the next section 2.3. For now, let us assume that this vector and matrix are computed (to close the main loop of the algorithm). Then, the result of this computation is propagated to the next level $L-1$ by passing the new initial guesses $\mathbf{g}^{L-1}$ and $\mathbf{G}^{L-1}$ of expression:

$$\mathbf{g}^{L-1} = 2\left(\mathbf{g}^{\mathbf{L}} + \mathbf{G}^L\,\mathbf{d}^L\right), \tag{10}$$

$$\mathbf{G}^{L-1} = \mathbf{G}^L\,\mathbf{A}^L. \tag{11}$$

The two expressions are derived by substituting equation 9 into equation 7. The factor two in equation 10 comes from the subsampling ratio used in the pyramid decomposation (two). There is no such scaling involved in the update of the affine transformation matrix guess because both image coordimates are subsampled using the same ratio. The next level residual optical flow $\left\{\mathbf{d}^{L-1}, \mathbf{A}^{L-1}\right\}$ is then computed through the same procedure. This vector, computed by optical flow computation (decribed in Section 2.3), minimizes the functional $\epsilon^{L-1}(\mathbf{d}^{L-1}, \mathbf{A}^{L-1})$ (substitute L by $L-1$ in equations 7, 8 and 9). This procedure goes on until the finest image resolution is reached ($L = 0$). The algorithm is initialized by setting the initial guess for optical flow vector at level L_m to zero (no initial guess is available at the deepest level of the pyramid), and the affine transformation matrix to the identity matrix:

$$\mathbf{g}^{L_m} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Rightarrow \quad g_x^{L_m} = g_y^{L_m} = 0, \tag{12}$$

$$\mathbf{G}^{L_m} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \Rightarrow \quad g_{xx}^{L_m} = g_{xy}^{L_m} = g_{yx}^{L_m} = g_{yy}^{L_m} = 0. \tag{13}$$

The final solution for optical flow and affine transformation matrix $\{\mathbf{d}, \mathbf{A}\}$ (refer to equation 2) is then available after the finest optical flow computation:

$$\mathbf{d} = \mathbf{g}^{\mathbf{0}} + \mathbf{G}^0\,\mathbf{d}^0, \tag{14}$$

$$\mathbf{A} = \mathbf{G}^0\,\mathbf{A}^0. \tag{15}$$

The clear advantage of a pyramidal implementation is that each residual optical flow vector $\mathbf{d}^L$ can be kept very small while computing a large overall pixel displacement vector $\mathbf{d}$.

2.3 Iterative Affine Optical Flow Computation (Iterative Affine Lucas-Kanade)

Let us now describe the core optical flow computation. At every level L in the pyramid, the goal is finding the vector $\mathbf{d}^L$ and the matrix $\mathbf{A}^L$ that minimize the matching function ϵ^L defined in equation 7. Since the same type of operation is performed for all levels L, let us now drop the superscripts L and define the new images $\mathcal{I}$ and $\mathcal{J}$ as follows (equations 8 and 9):

$\forall \mathbf{x} \in [-\omega_x - 1, \omega_x + 1] \times [-\omega_y - 1, \omega_y + 1]$,

$$\mathcal{I}(\mathbf{x}) \doteq I_{\text{comp}}^L(\mathbf{x}) = I^L\left(\mathbf{x} + \mathbf{u}^L\right), \tag{16}$$

$\forall \mathbf{x} \in [-\omega_x, \omega_x] \times [-\omega_y, \omega_y]$,

$$\mathcal{J}(\mathbf{x}) \doteq J_{\text{comp}}^L(\mathbf{x}) = J^L\left(\mathbf{G}^L\,\mathbf{x} + \mathbf{g}^L + \mathbf{u}^L\right). \tag{17}$$

Observe that the domains of definition of $\mathcal{I}(\mathbf{x})$ and $\mathcal{J}(\mathbf{x})$ are slightly different. Indeed, $\mathcal{I}(\mathbf{x})$ is defined over a window of size $(2\omega_x + 3) \times (2\omega_y + 3)$ instead of $(2\omega_x + 1) \times (2\omega_y + 1)$. This difference will become clear when computing spatial derivatives of $\mathcal{I}(\mathbf{x})$ using the central difference operator (eqs. 28 and 29). For clarity purposes, let us change the name of the displacement vector $\overline{\nu} = [\nu_x \;\; \nu_y]^T = \mathbf{d}^L$, as well as the affine image deformation matrix $\mathcal{A} = [1 + \nu_{xx} \;\; \nu_{xy}; \nu_{yx} \;\; 1 + \nu_{yy}] = \mathbf{A}^L$. Following that new notation, the goal is to find the displacement vector $\overline{\nu}$ and the matrix $\mathcal{A}$ that minimize the matching function:

$$\varepsilon(\overline{\nu}, \mathcal{A}) = \varepsilon(\nu_x, \nu_y, \nu_{xx}, \nu_{xy}, \nu_{yx}, \nu_{yy}) = \sum_{x=-\omega_x}^{\omega_x} \sum_{y=-\omega_y}^{\omega_y} \left(\mathcal{I}(\mathbf{x}) - \mathcal{J}(\mathcal{A}\,\mathbf{x} + \overline{\nu})\right)^2. \tag{18}$$

Let $\mathcal{D}$ be the derivative matrix (Jacobian) of the error matching function ε with respect to the unknown variables ν_x, ν_y, ν_{xx}, ν_{xy}, ν_{yx} and ν_{yy}:

$$\mathcal{D} = \left[\begin{array}{cccccc} \frac{\partial \varepsilon}{\partial \nu_x} & \frac{\partial \varepsilon}{\partial \nu_y} & \frac{\partial \varepsilon}{\partial \nu_{xx}} & \frac{\partial \varepsilon}{\partial \nu_{xy}} & \frac{\partial \varepsilon}{\partial \nu_{yx}} & \frac{\partial \varepsilon}{\partial \nu_{yy}} \end{array} \right]. \tag{19}$$

At the optimimum, the matrix $\mathcal{D}$ vanishes:

$$\mathcal{D}_{\text{optimum}} = \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right]. \tag{20}$$

Let us expand the expression of the derivative matrix $\mathcal{D}$:

$$\mathcal{D} = -2 \sum_{x=-\omega_x}^{\omega_x} \sum_{y=-\omega_y}^{\omega_y} \left(\mathcal{I}(\mathbf{x}) - \mathcal{J}(\mathcal{A}\,\mathbf{x} + \overline{\nu})\right) \mathcal{D}_2(\mathbf{x}), \tag{21}$$

where the 1×6 row vector $\mathcal{D}_2(\mathbf{x})$ is:

$$\mathcal{D}_2(\mathbf{x}) = \left[\begin{array}{cccccc} \frac{\partial \mathcal{J}}{\partial x} & \frac{\partial \mathcal{J}}{\partial y} & x\frac{\partial \mathcal{J}}{\partial x} & y\frac{\partial \mathcal{J}}{\partial x} & x\frac{\partial \mathcal{J}}{\partial y} & y\frac{\partial \mathcal{J}}{\partial y} \end{array} \right]. \tag{22}$$

Let us now substitute $\mathcal{J}(\mathcal{A}\,\mathbf{x} + \overline{\nu})$ by its first order Taylor expansion about the point $[\nu_x \;\; \nu_y \;\; \nu_{xx} \;\; \nu_{xy} \;\; \nu_{yx} \;\; \nu_{yy}] = [0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0]^T$ (this has a good chance to be a valid approximation since we are expecting a small displacement vector, thanks to the pyramidal scheme):

$$\mathcal{J}(\mathcal{A}\,\mathbf{x} + \overline{\nu}) \approx \mathcal{J}(\mathbf{x}) + \mathcal{D}_2(\mathbf{x})\,\overline{v}, \tag{23}$$

where $\overline{v}$ is the extended vector of unknowns:

$$\overline{v} = \left[\begin{array}{c} \nu_x \\ \nu_y \\ \nu_{xx} \\ \nu_{xy} \\ \nu_{yx} \\ \nu_{yy} \end{array} \right] \tag{24}$$

After substitution of expression 23 into equation 21, the Jacobian matrix $\mathcal{D}$ takes the new expression:

$$\mathcal{D} \approx -2 \sum_{x=-\omega_x}^{\omega_x} \sum_{y=\omega_y}^{\omega_y} \left(\mathcal{I}(\mathbf{x}) - \mathcal{J}(\mathbf{x}) - \mathcal{D}_2(\mathbf{x})\,\overline{v}\right) \mathcal{D}_2(\mathbf{x}). \tag{25}$$

Observe that the quatity $\mathcal{I}(\mathbf{x}) - \mathcal{J}(\mathbf{x})$ can be interpreted as the temporal image derivative at the point $\mathbf{x}$:

$\forall \mathbf{x} \in [-\omega_x, \omega_x] \times [-\omega_y, \omega_y]$,

$$\delta I(\mathbf{x}) \doteq \mathcal{I}(\mathbf{x}) - \mathcal{J}(\mathbf{x}). \tag{26}$$

The matrix $\mathcal{D}_2(\mathbf{x})$ is merely a matrix function of the image gradient (in x and y) and the image coordinate vector $\mathbf{x}$. Although equation 22 says that the matrix $\mathcal{D}_2(\mathbf{x})$ should computed from the second image $\mathcal{J}$, it is possible to perform that computation from the first image $\mathcal{I}$. That assumption is valid since the residual motion between the

two images is expected to be small. The computational advantage of such a choice will become clear when describing the iterative version of the algorithm. Following this strategy, let us make a slight change of notation:

$$\nabla I = \begin{bmatrix} I_x \\ I_y \\ x\,I_x \\ y\,I_x \\ x\,I_y \\ y\,I_y \end{bmatrix} \doteq \mathcal{D}_2(\mathbf{x})^T. \tag{27}$$

where the image derivstives I_x and I_y are computed directly from the first image $\mathcal{I}(\mathbf{x})$ in the $(2\omega_x+1)\times(2\omega_y+1)$ neighborhood of the point $\mathbf{x}=[0\ \ 0]^T$ independently from the second image $\mathcal{J}(\mathbf{x})$. If a central difference operator is used for derivative, the two derivative images have the following expression:

$\forall \mathbf{x} \in [-\omega_x, \omega_x] \times [-\omega_y, \omega_y]$,

$$I_x(\mathbf{x}) = \frac{\partial \mathcal{I}(\mathbf{x})}{\partial x} = \frac{\mathcal{I}(x+1,y)-\mathcal{I}(x-1,y)}{2}, \tag{28}$$

$$I_y(\mathbf{x}) = \frac{\partial \mathcal{I}(\mathbf{x})}{\partial y} = \frac{\mathcal{I}(x,y+1)-\mathcal{I}(x,y-1)}{2}. \tag{29}$$

In practice, the Sharr operator should used for computing image derivatives (very similar to the central difference operator).
Following this new notation, equation 25 may be written:

$$\frac{1}{2}\mathcal{D} \approx \sum_{x=-\omega_x}^{\omega_x}\sum_{y=-\omega_y}^{\omega_y} \left(\nabla I^T \overline{v} - \delta I\right) \nabla I^T, \tag{30}$$

or:

$$\frac{1}{2}\mathcal{D}^T \approx \sum_{x=-\omega_x}^{\omega_x}\sum_{y=-\omega_y}^{\omega_y} \left(\left(\nabla I\, \nabla I^T\right) \overline{v} - \nabla I\, \delta I\right). \tag{31}$$

Denote

$$\mathcal{G} \doteq \sum_{x=-\omega_x}^{\omega_x}\sum_{y=-\omega_y}^{\omega_y} \nabla I\, \nabla I^T = \sum_{x=-\omega_x}^{\omega_x}\sum_{y=-\omega_y}^{\omega_y} \begin{bmatrix} I_x^2 & I_x I_y & x I_x^2 & y I_x^2 & x I_x I_y & y I_x I_y \\ I_x I_y & I_y^2 & x I_x I_y & y I_x I_y & x I_y^2 & y I_y^2 \\ x I_x^2 & x I_x I_y & x^2 I_x^2 & x y I_x^2 & x^2 I_x I_y & x y I_x I_y \\ y I_x^2 & y I_x I_y & x y I_x^2 & y^2 I_x^2 & x y I_x I_y & y^2 I_x I_y \\ x I_x I_y & x I_y^2 & x^2 I_x I_y & x y I_x I_y & x^2 I_y^2 & x y I_y^2 \\ y I_x I_y & y I_y^2 & x y I_x I_y & y^2 I_x I_y & x y I_y^2 & y^2 I_y^2 \end{bmatrix}, \tag{32}$$

and

$$\overline{b} \doteq \sum_{x=-\omega_x}^{\omega_x}\sum_{y=-\omega_y}^{\omega_y} \nabla I\, \delta I = \sum_{x=-\omega_x}^{\omega_x}\sum_{y=-\omega_y}^{\omega_y} \begin{bmatrix} I_x\,\delta I \\ I_y\,\delta I \\ x\,I_x\,\delta I \\ y\,I_x\,\delta I \\ x\,I_y\,\delta I \\ y\,I_y\,\delta I \end{bmatrix}. \tag{33}$$

Then, equation 31 becomes:

$$\frac{1}{2}\mathcal{D}^T \approx \mathcal{G}\,\overline{\nu} - \overline{b}. \tag{34}$$

Following equation 20, the optimimum solution for optical flow vector + affine transformation matrix is:

$$\overline{v}_{\text{opt}} = \begin{bmatrix} \nu_x^{\text{opt}} & \nu_y^{\text{opt}} & \nu_{xx}^{\text{opt}} & \nu_{xy}^{\text{opt}} & \nu_{yx}^{\text{opt}} & \nu_{yy}^{\text{opt}} \end{bmatrix}^T = \mathcal{G}^{-1}\,\overline{b}. \tag{35}$$

This expression is valid only if the matrix $\mathcal{G}$ is invertible. In particular, this means that the image $\mathcal{I}(\mathbf{x})$ must contain enough gradient information in both x and y directions in the neighborhood of the point $\mathbf{x}=[0\ \ 0]^T$. Notice that

in order to capture the affine deformation of the image patch, it is often necessary to choose a large window of integration. This makes the two variables x and y span a large interval of values, and therefore makes the matrix $\mathcal{G}$ better conditioned (invertible).

Once the vector $\overline{v}_{\text{opt}}$ is computed, the optimal solution $\{\overline{\nu}, \mathcal{A}\} = \{\overline{\nu}_{\text{opt}}, \mathcal{A}_{\text{opt}}\}$ that minimizes the matching function $\varepsilon(\overline{\nu}, \mathcal{A})$ (eq. 18) is:

$$\overline{\nu}_{\text{opt}} = \begin{bmatrix} \nu_x^{\text{opt}} \\ \nu_y^{\text{opt}} \end{bmatrix}, \tag{36}$$

$$\mathcal{A}_{\text{opt}} = \begin{bmatrix} 1+\nu_{xx}^{\text{opt}} & \nu_{xy}^{\text{opt}} \\ \nu_{yx}^{\text{opt}} & 1+\nu_{yy}^{\text{opt}} \end{bmatrix}. \tag{37}$$

This is the standard Affine Lucas-Kanade optical flow equation, which is valid only if the pixel displacement is small (in order to validate the first order Taylor expansion). In practice, in order to obtain an accurate solution, it is necessary to iterate multiple times on this scheme (in a Newton-Raphson fashion).

Now that we have introduced the mathematical background, let us give the details of the iterative version of the algorithm. Recall the goal: find the vector $\overline{\nu}$ and the matrix $\mathcal{A}$ that minimize the error functional $\varepsilon(\overline{\nu})$ introduced in equation 18.

Let k be the iterative index, initialized to 1 at the very first iteration. Let us describe the algorithm recursively: at a generic iteration $k \geq 1$, assume that the previous computations from iterations $1, 2, \ldots, k-1$ provide an initial guess $\overline{\nu}^{k-1} = [\nu_x^{k-1} \;\; \nu_x^{k-1}]^T$ for the pixel displacement $\overline{\nu}$ and $\mathcal{A}^{k-1} = [1+\nu_{xx}^{k-1} \;\; \nu_{xy}^{k-1}; \nu_{yx}^{k-1} \;\; 1+\nu_{yy}^{k-1}]$ for the affine transformation matrix $\mathcal{A}$. Let $\mathcal{J}_k(\mathbf{x})$ be the new compensated image according to that initial guess $\{\overline{\nu}^{k-1}, \mathcal{A}^{k-1}\}$:

$\forall \mathbf{x} \in [-\omega_x, \omega_x] \times [-\omega_y, \omega_y]$,

$$\mathcal{J}_k(\mathbf{x}) = \mathcal{J}\left(\mathcal{A}^{k-1}\,\mathbf{x} + \overline{\nu}^{k-1}\right). \tag{38}$$

The goal is then to compute the residual pixel motion vector $\overline{\eta}^k = [\eta_x^k \;\; \eta_y^k]$ and the residual affine transformation matrix $\mathcal{M}^k = [1+\eta_{xx}^k \;\; \eta_{xy}^k; \eta_{yx}^k \;\; 1+\eta_{yy}^k]$ that minimize the error function

$$\varepsilon^k(\overline{\eta}^k, \mathcal{M}^k) = \varepsilon^k(\eta_x^k, \eta_y^k, \eta_{xx}^k, \eta_{xy}^k, \eta_{yx}^k, \eta_{yy}^k) = \sum_{x=-\omega_x}^{\omega_x} \sum_{y=-\omega_y}^{\omega_y} \left(\mathcal{I}(\mathbf{x}) - \mathcal{J}_k\left(\mathcal{M}^k\,\mathbf{x} + \overline{\eta}^k\right)\right)^2. \tag{39}$$

The solution of this minimization may be computed through a one step Affine Lucas-Kanade optical flow computation (equation 35):

$$\begin{bmatrix} \eta_x^k \\ \eta_y^k \\ \eta_{xx}^k \\ \eta_{xy}^k \\ \eta_{yx}^k \\ \eta_{yy}^k \end{bmatrix} = \mathcal{G}^{-1}\,\overline{b}_k \qquad \Rightarrow \qquad \begin{cases} \overline{\eta}^k = \begin{bmatrix} \eta_x^k \\ \eta_y^k \end{bmatrix} \\ \mathcal{M}^k = \begin{bmatrix} 1+\eta_{xx}^k & \eta_{xy}^k \\ \eta_{yx}^k & 1+\eta_{yy}^k \end{bmatrix} \end{cases}, \tag{40}$$

where the 6×1 vector $\overline{b}_k$ is defined as follows (also called image mismatch vector):

$$\overline{b}_k = \sum_{x=-\omega_x}^{\omega_x} \sum_{y=-\omega_y}^{\omega_y} \begin{bmatrix} I_x(\mathbf{x})\,\delta I_k(\mathbf{x}) \\ I_y(\mathbf{x})\,\delta I_k(\mathbf{x}) \\ x\,I_x(\mathbf{x})\,\delta I_k(\mathbf{x}) \\ y\,I_x(\mathbf{x})\,\delta I_k(\mathbf{x}) \\ x\,I_y(\mathbf{x})\,\delta I_k(\mathbf{x}) \\ y\,I_y(\mathbf{x})\,\delta I_k(\mathbf{x}) \end{bmatrix}, \tag{41}$$

where the k^{th} image difference $\delta I_k(\mathbf{x})$ is defined as follows:

$\forall \mathbf{x} \in [-\omega_x, \omega_x] \times [-\omega_y, \omega_y]$,

$$\delta I_k(\mathbf{x}) = \mathcal{I}(\mathbf{x}) - \mathcal{J}_k(\mathbf{x}). \tag{42}$$

Observe that the spatial derivatives I_x and I_y (at all points in the neighborhood of $[0 \;\; 0]^T$ in the first image $\mathcal{I}$) are computed only once at the beginning of the iterations following equations 28 and 29. Therefore the 6×6 matrix

G also remains constant throughout the iteration loop (expression given in equation 32). That constitutes a clear computational advantage. The only quantity that needs to be recomputed at each step k is the vector $\overline{b}_k$ that really captures the amount of residual difference between the image patches after translation by the vector $\overline{\nu}^{k-1}$ and affine deformation by the matrix $\mathcal{A}^{k-1}$. Once the residual optical flow and affine matrix $\{\overline{\eta}^k, \mathcal{M}^k\}$ are computed through equation 40, a new affine tracking guess $\{\overline{\nu}^k, \mathcal{A}^k\}$ is computed for the next iteration step $k+1$:

$$\overline{\nu}^k = \overline{\nu}^{k-1} + \mathcal{A}^{k-1}\,\overline{\eta}^k, \tag{43}$$

$$\mathcal{A}^k = \mathcal{A}^{k-1}\,\mathcal{M}^k. \tag{44}$$

The iterative scheme goes on until the computed pixel residual $\overline{\eta}^k$ is smaller than a threshold (for example 0.01 pixel), or a maximum number of iteration (for example 100) is reached, or when the error matching function ε does not decrease significantly. Other intuitive convergence criteria may be used. It is worth noticing that it may take as many as 30 or 40 iterations before reaching convergence, depending on the amplitude of displacement between the two images. At the first iteration ($k=1$) the initial guess is initialized to zero in pixel translation, and identidy in affine transformation:

$$\overline{\nu}^0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Rightarrow \quad \nu_x^0 = \nu_y^0 = 0, \tag{45}$$

$$\mathcal{A}^0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \Rightarrow \quad \nu_{xx}^0 = \nu_{xy}^0 = \nu_{yx}^0 = \nu_{yy}^0 = 0. \tag{46}$$

Assuming that K iterations were necessary to reach convergence, the final solution for the optical flow vector $\overline{\nu} = \mathbf{d}^L$ is:

$$\overline{\nu} = \mathbf{d}^L = \overline{\nu}^K, \tag{47}$$

$$\mathcal{A} = \mathbf{A}^L = \mathcal{A}^K. \tag{48}$$

The vector $\mathbf{d}^L$ and matrix $\mathbf{A}^L$ minimize the error functional described in equation 18 (or equation 7). This ends the description of the iterative affine Lucas-Kanade optical flow computation. The solution $\{\mathbf{d}^L, \mathbf{A}^L\}$ is fed to equations 10 and 11 and this overall procedure is repeated at all subsequent levels $L-1, L-2, \ldots, 0$ (see section 2.2).

2.4 Reducing the number of variables - Simplifying the notation

It is possible to reduce the number of variables in order to make the overall algorithm more "compact". Let:

$$\mathbf{v}_k^L = \mathbf{G}^L\,\overline{\nu}^k + \mathbf{g}^L + \mathbf{u}^L, \tag{49}$$

$$\mathbf{A}_k^L = \mathbf{G}^L\,\mathcal{A}^k. \tag{50}$$

The vector $\mathbf{v}_k^L$ is then the best known location of the corresponding point on image I^L at iteration k, and the matrix $\mathbf{A}_k^L$ is the associated affine transformaion matrix (at level L, iteration k). Given this new notation, the update rule given by equations 43 and 44 becomes:

$$\mathbf{v}_k^L = \mathbf{v}_{k-1}^L + \mathbf{A}_{k-1}^L\,\overline{\eta}^k, \tag{51}$$

$$\mathbf{A}_k^L = \mathbf{A}_{k-1}^L\,\mathcal{M}^k. \tag{52}$$

Observe that the variables $\overline{\nu}^k$ and $\mathcal{A}^k$ may be dropped. In addition, observe that from equations 10,11,47,48,49 and 50, we have:

$$\mathbf{g}^{L-1} = 2\left(\mathbf{v}_K^L - \mathbf{u}^L\right), \tag{53}$$

$$\mathbf{G}^{L-1} = \mathbf{A}_K^L, \tag{54}$$

assuming a total of K iterations in the inner loop (over k), at the pyramid level L. Notice that it is then possible to drop the four variables $\mathbf{g}^L$, $\mathbf{G}^L$, $\mathbf{d}^L$ and $\mathbf{A}^L$. Indeed, the rule of propagation from pyramid level to pyramid level given originally by equations 10 and 11 becomes:

$$\mathbf{v}_0^L = 2\,\mathbf{v}_K^{L+1}, \tag{55}$$

$$\mathbf{A}_0^L = \mathbf{A}_K^{L+1}, \tag{56}$$

still assuming K iterations at pyramid level $L+1$. This equation also replaces the initialization step before the iterative loop (over k) at level L (equations 45 and 46). Observe that this notation is valid at all pyramid levels $L = L_m, \ldots, 0$ only if we assume the new notation for the global initialization of the algorithm (substituting equations 12 and 13):

$$\mathbf{v}_K^{L_m+1} = \mathbf{u}/2^{L_m+1}, \tag{57}$$

$$\mathbf{A}_K^{L_m+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{58}$$

Observe that if a global initial guess $\{\mathbf{v}_g, \mathbf{A}_g\}$ for $\{\mathbf{v}, \mathbf{A}\}$ is known prior to computation, then it is possible to account for it by replacing the global initialization step by:

$$\mathbf{v}_K^{L_m+1} = \mathbf{v}_g / 2^{L_m+1}, \tag{59}$$

$$\mathbf{A}_K^{L_m+1} = \mathbf{A}_g. \tag{60}$$

Finally, following this new notation, the final tracking solution becomes:

$$\mathbf{v} = \mathbf{v}_K^0, \tag{61}$$

$$\mathbf{A} = \mathbf{A}_K^0, \tag{62}$$

still assuming K iterations at pyramid level $L = 0$. Observe that it is now suffisant to only keep the two variables $\mathbf{v}_k^L$ and $\mathbf{A}_k^L$.

2.5 Making tracking more robust with respect to changes in illumination

In practice, when tracking features on long sequences, changes in illumination may very often induce changes in brightness and contrast of the feature patches. To make tracking more robust with respect those changes due to illumination, it is preferable to make a slight modification to the global cost function ϵ to minimize:

$$\epsilon(\mathbf{d}, \mathbf{A}, \lambda, \delta) = \epsilon(d_x, d_y, d_{xx}, d_{xy}, d_{yx}, d_{yy}, \lambda, \delta) = \sum_{x=-\omega_x}^{\omega_x} \sum_{y=-\omega_y}^{\omega_y} \left(I(\mathbf{x}+\mathbf{u}) - (\lambda\, J\,(\mathbf{A}\,\mathbf{x} + \mathbf{d} + \mathbf{u}) + \delta)\right)^2, \tag{63}$$

where λ and δ are two additional scalar coefficients accounting for changes in image contrast and image brightness respectively. The set of coefficients that minimize that new cost function give then the solution for tracking. Of course, a new set of optimization equations could be expanded by adding the two new variables λ and δ to the global set of unknowns. That would naturally lead to a new Jacobian iteration scheme with a matrix $\mathcal{G}$ of size 8×8 (instead of 6×6). However, we believe that step of formally expanding the vector of unknowns is not necessary. In effect, the values λ and δ can be easily computed in a closed form way if the remaining tracking coefficients $\{\mathbf{d}, \mathbf{A}\}$ are known. Therefore, one approach to include illumination changes to tracking is to insert an image normalization step within the iteration process, just before updating the set of tracking coefficients. More precisely, this normalization must take place right after warping of the second image J, or right after Equation (17). This normalization step consists of scaling (by λ) and translating (by δ) the second warped image $\mathcal{J}$ so that $\mathcal{I}$ and $\mathcal{J}$ have same mean brightness and variance. Of course, since this step is done independently from the computation of $\mathbf{d}$ and $\mathbf{A}$, the normalization coefficients λ and δ are easily computable in a closed form fashion.

2.6 Summary of the pyramidal affine tracking algorithm

Let is now summarize the entire affine tracking algorithm in a form of a pseudo-code. Since $\mathbf{v}_k^L$ and $\mathbf{A}_k^L$ are the only two variables that are continuously updated throughout the algorithm, we take the liberty of dropping the indices L and k and substitute them by the (varying) variables $\mathbf{v}$ and $\mathbf{A}$. The final value of $\{\mathbf{v}, \mathbf{A}\}$ (after the iterations at level $L = 0$) is the solution of the tracking problem. In addition, for clarity purposes, we drop the index k in the iterative loop. Following these final changes, the overall algorithm may be summarized by the following compact pseudo-code (note that $\mathbf{0}$ is the 1×2 row vector of zero entries, and $\mathbf{I}_2$ is the 2×2 identity matrix):

Goal: Let $\mathbf{u}$ be a point on image I. Find its corresponding location $\mathbf{v}$ on image J, and the matrix $\mathbf{A}$ describing the affine transformation between the two images, in the vicinity of $\mathbf{u}$ (on image I) and $\mathbf{v}$ (on image J).

Global initial guess for $\{\mathbf{A}, \mathbf{v}\}$:

$$\mathbf{v}_g \leftarrow \mathbf{u} \quad \text{(or other if specified)}$$
$$\mathbf{A}_g \leftarrow \mathbf{I}_2 \quad \text{(or other if specified)}$$

Build pyramid representations of I *and* J: $\{I^L\}_{L=0,\dots,L_m}$ and $\{J^L\}_{L=0,\dots,L_m}$ (eqs. 3,4,5)

Initialization of affine tracking:

$$\mathbf{v} \leftarrow \mathbf{v}_g / 2^{L_m+1}$$
$$\mathbf{A} \leftarrow \mathbf{A}_g \quad \text{(eqs. 59,60)}$$

for $L = L_m$ **down to** 0 **with step of** -1

Translation of first image: $\mathcal{I}(\mathbf{x}) \leftarrow I^L\left(\mathbf{x} + \mathbf{u}/2^L\right)$ (eq. 6,16)

Derivative of $\mathcal{I}$ *with respect to* x: $I_x(\mathbf{x}) \leftarrow \dfrac{\mathcal{I}(x+1,y) - \mathcal{I}(x-1,y)}{2}$ (eqs. 28,16)

Derivative of $\mathcal{I}$ *with respect to* y: $I_y(\mathbf{x}) \leftarrow \dfrac{\mathcal{I}(x,y+1) - \mathcal{I}(x,y-1)}{2}$ (eqs. 29,16)

Spatial gradient matrix:

$$\mathcal{G} \leftarrow \sum_{x=-\omega_x}^{\omega_x} \sum_{y=-\omega_y}^{\omega_y} \begin{bmatrix} I_x^2 & \cdots & y\, I_x\, I_y \\ \vdots & \ddots & \vdots \\ y\, I_x\, I_y & \cdots & y^2\, I_y^2 \end{bmatrix} \quad \text{(eq. 32)}$$

Initial guess from previous level $L+1$: $\mathbf{v} \leftarrow 2\,\mathbf{v}$ (eq. 55,56)

Repeat:

Warping of second image: $\mathcal{J}(\mathbf{x}) \leftarrow J^L(\mathbf{A}\,\mathbf{x} + \mathbf{v})$ (eqs. 17,38,49,50)

Image normalization (optional): $\mathcal{J} \leftarrow \lambda\,\mathcal{J} + \delta$, such that $\mathcal{I}$ and $\mathcal{J}$ have same mean and variance (sec. 2.5)

Image difference: $\delta I(\mathbf{x}) \leftarrow \mathcal{I}(\mathbf{x}) - \mathcal{J}(\mathbf{x})$ (eq. 42)

Image mismatch vector:

$$\overline{b} \leftarrow \sum_{x=-\omega_x}^{\omega_x} \sum_{y=-\omega_y}^{\omega_y} \begin{bmatrix} I_x\, \delta I \\ \vdots \\ y\, I_y\, \delta I \end{bmatrix} \quad \text{(eq. 41)}$$

Affine Lucas-Kanade:

$$[\eta_x \;\; \eta_y \;\; \eta_{xx} \;\; \eta_{xy} \;\; \eta_{yx} \;\; \eta_{yy}]^T \leftarrow \mathcal{G}^{-1}\,\overline{b} \quad \text{(eq. 40)}$$

Update of tracking solution:

$$\begin{bmatrix} \mathbf{A} & \mathbf{v} \\ \mathbf{0} & 1 \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{A} & \mathbf{v} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} 1+\eta_{xx} & \eta_{xy} & \eta_x \\ \eta_{yx} & 1+\eta_{yy} & \eta_y \\ 0 & 0 & 1 \end{bmatrix} \quad \text{(eq. 40,51,52)}$$

Until $\sqrt{\eta_x^2 + \eta_y^2} <$ resolution threshold (or other intuitive convergence criteria). Better criteria: Check that all the four corners of the window of integration do not move by more than the resolution threshold.

end of for-loop on L

Solution: The corresponding point is at location $\mathbf{v}$ on image J, and the matrix $\mathbf{A}$ describes the affine transformation between the two images, in the vicinity of $\mathbf{u}$ (on image I) and $\mathbf{v}$ (on image J).

2.7 Re-parameterization of the affine matrix to enable reduced tracking models

In some cases of affine tracking, not all the affine parameters d_{xx}, d_{xy}, d_{yx}, d_{yy} can be estimated. When this happens, it is useful to re-parameterize the affine transformation matrix $\mathbf{A}$ such that a subset of the parameters can always be estimated. This is also known as model reduction. One possible parameterization is:

$$\mathbf{A} = \begin{bmatrix} 1+d_{xx} & d_{xy} \\ d_{yx} & 1+d_{yy} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1+a & 0 \\ 0 & 1+b \end{bmatrix}, \tag{64}$$

where four variables a, b, s and θ are the new affine parameters. The angle θ accounts for the change in orientation of the feature patch, the scalars a and b account for change in scale, and the scalar s accounts for skew. This new parameterization has the advantage of being very flexible in the sense that tracking can be achieved while estimated only a subset of the parameters (for example only θ while enforcing $a = b = s = 0$). First, observe that the mapping between the two parameter spaces is straight forward:

$$d_{xx} = \cos\theta\,(1+a) - 1 \tag{65}$$
$$d_{xy} = (1+b)\;(s\,\cos\theta - \sin\theta) \tag{66}$$
$$d_{yx} = \sin\theta\,(1+a) \tag{67}$$
$$d_{yy} = (1+b)\;(s\,\sin\theta + \cos\theta) - 1 \tag{68}$$

The tracking equations can easily be adapted to this new parameterization. Let $\mathbf{H}$ be the derivative of the "old" parameter vector $[d_x\ d_y\ d_{xx}\ d_{xy}\ d_{yx}\ d_{yy}]^T$ with respect to the "new" parameter vector $[d_x\ d_y\ \theta\ s\ a\ b]^T$:

$$\mathbf{H}(\theta, s, a, b) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -(1+a)\,\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & -(1+b)\;(s\,\sin\theta + \cos\theta) & (1+b)\,\cos\theta & 0 & s\,\cos\theta - \sin\theta \\ 0 & 0 & (1+a)\,\cos\theta & 0 & \sin\theta & 0 \\ 0 & 0 & (1+b)\;(s\,\cos\theta - \sin\theta) & (1+b)\,\sin\theta & 0 & s\,\sin\theta + \cos\theta \end{bmatrix} \tag{69}$$

Since each tracking iteration is done under the assumption of small motion, only the derivative matrix at the "no-motion" point $[d_x\ d_y\ \theta\ s\ a\ b]^T = [0\ 0\ 0\ 0\ 0\ 0]^T$ needs to be considered:

$$\mathbf{H}_0 = \mathbf{H}(0,0,0,0) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{70}$$

At each tracking iteration, the residual motion vector $\overline{v'} = [\delta d_x\ \delta d_y\ \delta\theta\ \delta s\ \delta a\ \delta b]^T$ in the parameter space $\{d_x, d_y, \theta, s, a, b\}$ is given by:

$$\overline{v'} = \left(\mathbf{H}_0^T\,\mathcal{G}\,\mathbf{H}_0\right)^{-1}\,\mathbf{H}_0^T\,\overline{b} \tag{71}$$

which is an alternate expression to Eq. (40). The 6×6 matrix $\mathcal{G}$ and the 6×1 vector $\overline{b}$ are given by Equations (32) and (33) respectively. Before going to the next tracking iteration, the affine matrix $\mathbf{A}$ and the position vector $\mathbf{v}$ needs to be updated following the same procedure as described in the summary algorithm presented in Section 2.6 (Eq. 40, 51, 52). For that purpose, the residual 2×2 affine matrix is computed using Equation (64):

$$\begin{bmatrix} \mathbf{A} & \mathbf{v} \\ \mathbf{0} & 1 \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{A} & \mathbf{v} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \delta d_x \\ 0 & 1 & \delta d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\delta\theta & -\sin\delta\theta & 0 \\ \sin\delta\theta & \cos\delta\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \delta s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1+\delta a & 0 & 0 \\ 0 & 1+\delta b & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{72}$$

There is no real advantage of using this formalism if all six parameters d_x, d_y, θ, s, a and b are always estimated. Indeed, in that case, the tracking equation (71) reduces to $\overline{v'} = \mathbf{H}_0^{-1}\,\mathcal{G}^{-1}\,\overline{b}$. In effect, in the limit, both parameterizations converge to the same numerical solution. The real advantage comes when one wants to use a reduced affine model for tracking. For example, if only rotation and skew can be estimated, then the two scale variables a and b may be taken out of the affine model, effectively taking the last two columns of $\mathbf{H}_0$ out, and assuming $a = b = 0$. The real advantage comes from the fact that the resulting matrix to invert $\mathbf{H}_0^T\,\mathcal{G}\,\mathbf{H}_0$ is of dimension less than 6×6, making the estimation potentially more robust. Any combination of reduced models is then possible (such as rotation and scale only).