Pyramidal Implementation of the Lucas Kanade Feature Tracker

Description of the algorithm

1. 问题描述

设 I 和 J 为两张 2 维灰度图像，$I(x) = I(x, y)$和 $J(x) = J(x, y)$为这两张图片在点 $x=[x, y]^T$ 的灰度值，x 和 y 是图片上的点 x 的像素坐标。I 一般用来做第一张图片，而 J 一般用来当做第二章图片。在实际问题中，图片 I 和 J 是离散函数（或数组），定义左上角的像素坐标向量为$[0\ 0]^T$，设 n_x 和 n_y 为这两张图片的宽度和 高度。则右下角的像素坐标向量为$[n_x\text{-}1\ n_y\text{-}1]^T$。

假设一个点 $u = [u_x\ u_y]^T$ 为第一张图片 I 上的一个点。特征点追踪的目标是在第二张图片 J 找到一个点 v 使得 $v = u + d = [u_x+d_x\ u_y+d_y]^T$，就好比 $I(u)$和 $J(v)$是相似的。向量 $d=[d_x\ d_y]^T$ 是 x 点的图片速度，也就是众所周知的 x 点的光流。由于孔径问题，定义二维临近相似度的概念就是十分必要的了。设 w_x 和 w_y 是两个整数。我们定义图片速率 d（x 点的光流）是用来最小化差异函数 ε 的向量，定义如下：

$$\varepsilon(d) = \varepsilon(d_x, d_y) = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_w+w_y}^{u_y+w_y} (I(x, y) - J(x + d_x, y + d_y))^2 \qquad (1)$$

由这个定义可以得知，近似函数是通过图片的邻居大小 $(2w_x +1)\times(2w_y +1)$ 来测量的，这个图片邻居也被称为集成窗口。一般 w_x 和 w_y 的取值为 2,3,4,5,6,7 个像素。

2. 跟踪算法描述

任何特征跟踪的两个重要因素是准确度和稳定性。准确度是跟去在追踪时局部子像素的准确性来决定的。从直观上来看，为了不去平滑图片消除细节问题，一个小的集成窗口是比较合适的，也就是说 w_x 和 w_y 的值较小。尤其是在两个有着完全不同图片速度的区域，这是非常必要的。

稳定性因素需要考虑光照变化，图像运动时对尺寸变化的敏感性等问题。尤其是为了解决大矢量运动，从直观上来看，选取一个较大的集成窗口更好。确实，根据公式 1，使 $d_x \le w_x$，$d_y \le w_y$ 更好。因此在选择集成窗口大小的时候就要在准确度和稳定性中做一个折衷。为了解决这样一个问题，我们提出一个金字塔实现的经典 LK 算法。LK 光流算法的迭代实现可以提供效率高的局部跟踪准确率。

2.1 图像金字塔表示法

我们为一张图片 I 定义一个金字塔表示，大小为 $n_x \times n_y$。设 $L^0 = L$ 是第 0 层图片。这张图片的分辨率最高（原始图像），并把长宽分别定义为 $n_x^0 = n_x$、$n_y^0 = n_y$。然后用一种递

归方式建立金字塔：利用 I^0 计算 I^1，然后利用 I^1 计算 I^2，以此类推。设 L 为金字塔层数，则 I^{L-1} 为 L-1 层图片，n_x^{L-1} 和 n_y^{L-1} 分别为 I^{L-1} 的宽和高。图片 I^{L-1} 定义如下：

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

(2)

为了简化符号，我们定义不存在于图像 I^{L-1}（$0 \le x \le n_x^{L-1}-1$，$0 \le y \le n_y^{L-1}-1$）上的点：

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

公式 2 只是定义了 x 和 y 的值 $0 \le 2x \le n_x^{L-1}-1$，$0 \le 2y \le n_y^{L-1}-1$。因此 I^L 的宽度 n_x^L 和高度 n_y^L 最大时要满足以下两个条件：

$$\begin{aligned} n_x^L &\le \frac{n_x^{L-1}+1}{2}, \\ n_y^L &\le \frac{n_y^{L-1}+1}{2}. \end{aligned}$$

(3),(4)

公式(2),(3),(4)是用来递归的构建图片 I 和 J 的金字塔表示，$\{I^L\}_{L=0,\ldots,L_m}$ 和 $\{J^L\}_{L=0,\ldots,L_m}$。L_m 是金字塔的高度。通常 L_m 的值为 2,3,4。对一般图像的大小来说，4 层以上的金字塔就没有太大意义了。举例来说，一个图片 I 的大小为 640*480，则 I^1、I^2、I^3 和 I^4 的大小分别为 320*240、160*120、80*60 和 40*30。超过四层在大多数情况下没有多大意义。金字塔表示法的主要作用是能够处理大像素移动（超过集成窗口的大小）。所以，金字塔高度的选取也要根据图像中的最大预期光流。下一部分江西的讲述了跟踪操作，让我们对概念的理解更好。最后观察公式 2，低通滤波[1/4 1/2 1/4]*[1/4 1/2 1/4]^T 是在图像二次抽样之前用来做图像反走样。但是在实际情况下（比如 C 代码中）用一个更大的反走样滤波核心来构造金字塔结构，形式上是一样的。

2.2 金字塔特征追踪

回想特征追踪的目标：根据给定图像 I 中的一个点 **u**，找到在图像 J 中相应的位置 **v=u+d** 或者迭代得到它的像素位移向量 **d**。

对于 $L=0,...,L_m$，定义 $\mathbf{u^L}=[u_x^L\ u_y^L]$，为点 **u** 金字塔每一层中对应的坐标。根据上述定义的金字塔表示公式(2),(3),(4)，计算向量 $\mathbf{u^L}$ 如下：

$$u^L=\frac{u}{2^L} \tag{5}$$

在公式(5)中的除法算子分别独立的应用于横竖坐标。特别的，$u^0=u$。

整个的金字塔跟踪过程如下：首先，在最深层（最顶层）L_m 计算光流；然后，计算得到的结果用一种对像素位移的初始化猜测传递给上一层 L_m-1。由于初始化了一个猜测，就能很精确的计算出 L_m-1 层的光流，然后再传递给 L_m-2 直到传递给最高层，也就是原始图像。

现在我们用更多数学的分析来描述一下两层 $L+1$ 和 L 之间的递归操作。假设为 L 层光流初始化一个猜测，$g^L=[g_x^L\ g_y^L]^T$ 从 $L+1$ 层计算结果得到传递给 L 层。然后，为了计算在 L 层的光流，就有必要去找到偏移向量 $d^L=[d_x^L\ d_y^L]^T$ 来最小化新的图像匹配错误函数：

$$\varepsilon^L(d^L)=\varepsilon(d_x^L,d_y^L)=\sum_{x=u_x^L-w_x}^{u_x^L+w_x}\sum_{y=u_y^L+w_y}^{u_y^L+w_y}(I^L(x,y)-J^L(x+g_x^L+d_x^L,y+g_y^L+d_y^L))^2 \tag{6}$$

可以看到无论 L 的值是多少集成窗口的大小总是不变的，注意到初始化的猜测流向量 g^L 用来预先处理第二张图片块。这样，剩余的流向量 $d^L=[d_x^L\ d_y^L]^T$ 就会很小，因此可以很容易的通过标准的 *LK* 算法计算出来。

剩余光流向量 $d^L=[d_x^L\ d_y^L]^T$ 的计算细节将在 2.3 节描述。现在让我们来假设这个向量已经计算得到。然后计算的结果传递给下一层 $L-1$，而且传递初始化猜测 g^{L-1}，表达式如下：

$$g^{L-1}=2(g^L+d^L) \tag{7}$$

通过相同的步骤计算出下一层的光流剩余向量 d^{L-1}。这个通过光流计算得到的向量来最小化错误匹配函数 $\varepsilon^{L-1}(d^{L-1})$（公式 6），直到 $L=0$。算法开始时，设置最高层初始化向量 L_m 为 0（即在最高层金字塔时没有初始化猜测）。

$$g^{L_m} = [0\ 0]^T \tag{8}$$

最后的光流结果 **d**（根据公式 1）经过光流计算得到：

$$d = g^0 + d^0 \tag{9}$$

也可以记为如下形式：

$$d = \sum_{L=0}^{L_m} 2^L d^L \tag{10}$$

金字塔实现明显的好处就是当计算一个大的整个的像素位移向量 **d** 时，每一个剩余光流向量d^L都可以保持一个很小的值。假设每一个基本的光流计算可以处理的像素移动最大为$d_{\max}$，则金字塔实现可以处理的整个像素移动就变为了$d_{\max\ final} = (2^{L_m+1} - 1)d_{\max}$。举例说明，如果金字塔深度$L_m$为 3，这就意味着最大的像素位移可以达到 15！这就是在处理大像素移动的同时也能保持集成窗口非常的小。

2.3 迭代的光流计算（迭代的 LK 算法）

现在让我们来描述一下核心的光流计算。在金字塔中的每一层，目的都是找到向量d^L去最小化公式 6 中定义的匹配函数ε^L。由于每一种操作的执行时都是相同的，所以让我们去掉上标L然后定义新的图片A和B，如下：

$$\forall (x, y) \in [p_x - w_x - 1, p_x + w_x + 1] \times [p_y - w_y - 1, p_y + w_y + 1],$$

$$A(x, y) = I^L(x, y) \tag{11}$$

$$\forall (x, y) \in [p_x - w_x, p_x + w_x] \times [p_y - w_y, p_y + w_y],$$

$$B(x, y) = J^L(x + g_x^L, y + g_y^L) \tag{12}$$

注意到$A(x, y)$和$B(x, y)$的定义域是有些不同的，确实，$A(x, y)$的窗口大小是$(2w_x + 3) \times (2w_y + 3)$而不是$(2w_x + 1) \times (2w_y + 1)$。这个差异将会在利用中心差异算子计算$A(x, y)$的空间衍生物时体现出来。为了更清楚，我们定义偏移向量$\bar{v} = [v_x\ v_y]^T = d^L$，同时也定义坐标向量$\bar{p} = [p_x\ p_y]^T = u^L$。根据新的符号，算法目标是去找到偏移向量$\bar{v} = [v_x\ v_y]^T$，来最小化匹配函数。

$$\varepsilon(\overline{\nu}) = \varepsilon(\nu_x, \nu_y) = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (A(x,y) - B(x+\nu_x, y+\nu_y))^2 . \tag{13}$$

标准的迭代 LK 算法可以用来解决这个问题。在最好的情况下，第一个 ε 的衍生物对 $\overline{\nu}$ 求导为 0：

$$\left.\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}}\right|_{\overline{\nu}=\overline{\nu}_{\text{opt}}} = [0 \;\; 0]. \tag{14}$$

展开这个衍生物后，我们得到：

$$\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} = -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (A(x,y) - B(x+\nu_x, y+\nu_y)) . \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix} . \tag{15}$$

现在我们在 $\overline{\nu} = [0\,0]^T$ 用一阶泰勒展开式来代替 $B(x,y)$（在这里就可以有了一个好的机会来近似我们所期望的很小的位移向量）。

$$\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} \approx -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(A(x,y) - B(x,y) - \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix} \overline{\nu}\right) . \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix} . \tag{16}$$

观察到 $A(x,y) - B(x,y)$ 可以被当做是图片在点 $[x,y]^T$ 的暂时衍生物：

$$\forall (x,y) \in [p_x - \omega_x, p_x + \omega_x] \times [p_y - \omega_y, p_y + \omega_y],$$
$$\delta I(x,y) \doteq A(x,y) - B(x,y). \tag{17}$$

而矩阵 $\left(\frac{\partial B}{\partial x} \;\; \frac{\partial B}{\partial y}\right)$ 仅仅是图片的梯度向量。让我们对这个向量做一个小小的变换。

$$\nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix} \doteq \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix}^T . \tag{18}$$

观察到图片衍生物 I_x 和 I_y 可能是直接来之第一张图片 $A(x,y)$ 在 $(2w_x+1)\times(2w_y+1)$ 的临近点 p，而且无所谓 $B(x,y)$（这个观察结果的重要性将会在描述硫酸法的迭代版本时体现出来）。如果用中心差异算子来计算衍生物，两张衍生图像就会有如下表达式：

$$\forall (x,y) \in [p_x - \omega_x, p_x + \omega_x] \times [p_y - \omega_y, p_y + \omega_y],$$
$$I_x(x,y) = \frac{\partial A(x,y)}{\partial x} = \frac{A(x+1,y) - A(x-1,y)}{2}, \tag{19}$$
$$I_y(x,y) = \frac{\partial A(x,y)}{\partial y} = \frac{A(x,y+1) - A(x,y-1)}{2}. \tag{20}$$

实际上，Sharr 算子是用来计算图片衍生物的（跟中心差异算子相似）。

根据新的符号，公式 15 可以重写为：

$$\frac{1}{2}\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} \approx \sum_{x=p_x-\omega_x}^{p_x+\omega_x}\sum_{y=p_y-\omega_y}^{p_y+\omega_y}\left(\nabla I^T \overline{\nu} - \delta I\right)\nabla I^T, \tag{21}$$

$$\frac{1}{2}\left[\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}}\right]^T \approx \sum_{x=p_x-\omega_x}^{p_x+\omega_x}\sum_{y=p_y-\omega_y}^{p_y+\omega_y}\left(\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}\overline{\nu} - \begin{bmatrix} \delta I\, I_x \\ \delta I\, I_y \end{bmatrix}\right). \tag{22}$$

定义：

$$G \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x}\sum_{y=p_y-\omega_y}^{p_y+\omega_y}\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad \text{and} \quad \overline{b} \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x}\sum_{y=p_y-\omega_y}^{p_y+\omega_y}\begin{bmatrix} \delta I\, I_x \\ \delta I\, I_y \end{bmatrix}. \tag{23}$$

公式 22 就可以写为：

$$\frac{1}{2}\left[\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}}\right]^T \approx G\overline{\nu} - \overline{b}. \tag{24}$$

因此根据公式 14，最佳光流向量就为：

$$\overline{v}_{opt} = G^{-1}\overline{b} \tag{25}$$

只有当矩阵 G 可逆时，这个表达式才是有效的。就等于说 $A(x,y)$ 在 x 和 y 方向的临近点 p 都包含了梯度信息。

这就是标准的 LK 光流算法公式，这个公式只有在像素位移很小的时候才有效。实际上，为了得到一个正确的解决办法，就有必要在这个流程上多次迭代。

现在我们已经介绍了数学背景，让我们给出迭代算法版本的具体细节。回想算法目标：找到向量 $\overline{v}$ 来最小化在公式 13 中提到的错误函数 $\varepsilon(\overline{v})$。

设 k 为迭代索引，在开始时初始化为 1。让我们来描述算法的循环过程：在第 $k \geq 1$ 次迭代时，假设通过前面的计算为向量偏移 $\overline{v}$ 得到了初始化猜测 $\overline{v}^{k-1} = [v_x^{k-1}\ v_y^{k-1}]^T$。设 B_k 为根据猜测 $\overline{v}^{k-1}$ 得到的新的变换图像。

$$\forall (x,y) \in [p_x-\omega_x, p_x+\omega_x]\times[p_y-\omega_y, p_y+\omega_y],$$
$$B_k(x,y) = B(x+v_x^{k-1}, y+v_y^{k-1}). \tag{26}$$

目标是计算剩余运动向量 $\overline{\eta}^k = [\eta_x\ \eta_y]^T$ 来最小化错误函数：

$$\varepsilon^k(\overline{\eta}^k) = \varepsilon(\eta_x^k, \eta_y^k) = \sum_{x=p_x-\omega_x}^{p_x+\omega_x}\sum_{y=p_y-\omega_y}^{p_y+\omega_y}\left(A(x,y) - B_k(x+\eta_x^k, y+\eta_y^k)\right)^2. \tag{27}$$

最小化的解决办法可以通过计算 LK 光流得到

$$\overline{\eta}^k = G^{-1}\overline{b}_k, \tag{28}$$

其中这个 2×1 维向量 $\overline{b}_k$（也叫做图像误匹配向量）：

$$\bar{b}_k = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} \delta I_k(x,y)\, I_x(x,y) \\ \delta I_k(x,y)\, I_y(x,y) \end{bmatrix}, \tag{29}$$

第 k 个图片差异 ϑI_k 定义如下：

$$\forall (x,y) \in [p_x-\omega_x, p_x+\omega_x] \times [p_y-\omega_y, p_y+\omega_y],$$
$$\delta I_k(x,y) = A(x,y) - B_k(x,y). \tag{30}$$

可以看到空间衍生物 I_x 和 I_y（在所有点的邻居 $\bar{p}$ 中）只在迭代开始时通过公式 19 和 20 只计算一次，因此 2×2 的矩阵 G 在迭代过程中也始终不变。这就带来了一个很明显的好处。只有一个参数需要在迭代中反复计算，就是 $\bar{b}^k$，而且这个参数携带了很多图片块根据向量 $\bar{v}^{k-1}$ 移动之后的差异。一旦剩余光流 $\bar{\eta}^k$ 用公式 28 计算出来之后，一个猜测新的像素位移猜测 $\bar{v}^k$ 就可以计算出来了：

$$\bar{v}^k = \bar{v}^{k-1} + \bar{\eta}^k.$$

这个迭代过程一直进行直到 $\bar{\eta}^k$ 小于一个阈值（比如说 0.03 个像素）或达到了最大的迭代次数（比如 20）之后停止。平均来说，5 次迭代就足够达到收敛了，在第一次迭代时（k=1）初始化猜测置为 0：

$$\bar{v}^0 = [0 \;\; 0]^T.$$

假设为达到收敛进行 K 次迭代是必须的，则最后的光流向量 $\bar{v} = d^L$ 为：

$$\bar{v} = \mathbf{d}^L = \bar{v}^K = \sum_{k=1}^{K} \bar{\eta}^k.$$

这个向量用来最小化错误函数在公式 13 中描述（或公式 6）。也用来终止迭代 LK 光流计算。向量 d^L 是用在公式 7，整个过程在每一个子层 $L-1, L-2, ..., 0$ 重复进行（见 2.2 节）。

2.4 金字塔追踪算法总结

见原文档。

2.5 子像素计算

在计算的过程中很明显要将所有计算都在亚像素精度层进行。因此要可以计算在整形像素之间计算亮度值就很必要了。为了在亚像素位置计算图像亮度，我们提出使用双线性插值的方法。

设 L 为金字塔的一层，假设我们需要一个图像值 $I^L(x,y)$ 且 x 和 y 不是整形。设 x_o 和 y_o 是 x 和 y 的整形部分（小于 x 和 y ），设 α_x 和 α_y 为保留值（0 到 1 之间）：

$$x = x_o + \alpha_x$$

$$y = y_o + \alpha_y$$

我们观察一下亚像素计算。根据 2.4 节给出的总结。当在点 (x,y) 的邻居 $[p_x - w_x, p_x + w_x] \times [p_y - w_y, p_y + w_y]$ 计算两个图像的衍生物 $I_x(x,y)$ 和 $I_y(x,y)$ 时，就有必要在点 (x,y) 的邻居 $[p_x - w_x - 1, p_x + w_x + 1] \times [p_y - w_y - 1, p_y + w_y + 1]$ 得到亮度值。当然中心 **p** 坐标不一定是整形，称 p_{x_o} 和 p_{y_o} 是 p_x 和 p_y 的整形部分。然后我们可以得到：

$$\begin{aligned} p_x &= p_{x_o} + p_{x_\alpha}, \\ p_y &= p_{y_o} + p_{y_\alpha}, \end{aligned}$$

p_{x_α} 和 p_{y_α} 是小数部分。因此为了通过双线性插值在点 (x,y) 的邻居 $[p_x - w_x - 1, p_x + w_x + 1] \times [p_y - w_y - 1, p_y + w_y + 1]$ 计算 $I^L(x,y)$，就有必要在一个整数框 $[p_x - w_x - 1, p_x + w_x + 2] \times [p_y - w_y - 1, p_y + w_y + 2]$ 使用原始亮度序列的值 $I^L(x,y)$。

2.6 跟踪靠近图片边缘的特征点

观察到以下内容是很有用的：可能我们要处理一些离图片边界很近的点，我们就要把这些点超出图片的集成窗口按比例缩小。如果金字塔层数很大的话，这个理解就变得更有意义了。确实，如果我们为了追踪图片总是使用完整的 $(2w_x + 1) \times (2w_y + 1)$，那么就会在整个图片宽度 w_x 处存在禁止区。如果 L_m 为金字塔的高度，那么就意味着有一个宽度为 $2^{L_m} w_x$ 的禁止区在整张原始图片上。如果 L_m 、 w_x 和 w_y 的值比较下，就可能不会带来一个影响很大

的限制。但是这会使大的集成窗口变得非常麻烦。如果 $w_x = w_y = 5$ 个像素，$L_m = 3$ 就会导致在图片上有 40 个像素的禁止区！

为了防止上述事情的发生，我们提出让追踪点的集成窗口可以部分落在图片外面（在任何层都可以）。在这种情况下跟踪过程不变，除非在所有表达式中出现的总和应该在所有可行的图片邻居比例中完成，也就是说邻居的部分有足够的值来计算 $I_x(x,y)$ 、$I_y(x,y)$ 和 $\vartheta I_k(x,y)$。观察到这样做，可用的总和区域可能通过 LK 算法的迭代过程中有很多。确实，一次有一次的迭代，通过变换向量 $[g_x^L + v_x^{k-1}\ g_y^L + v_y^{k-1}]^T$ 变化后图片差异 $\vartheta I_k(x,y)$ 可能也有很多。而且观察到当计算误匹配向量 $\bar{b}^k$ 和梯度矩阵 G 时，总和区域必须是明确的。因此，在这种情况下，G 矩阵必须在每一步的迭代过程中重新计算，但是差异块 $I_x(x,y)$ 和 $I_y(x,y)$ 可以在迭代循环开始前只计算一次。

当然如果中心点 p 落到了图片 *I* 外，或者相应的跟踪点落到了图片 *J* 外，就应该合理的说明点丢失，而不去追踪它。

2.7 声明特征丢失

有两种情况会引起特征点丢失。第一种情况是很直观的：点落在图片外。我们已经在 2.6 节讨论了这种问题。第二种情况的丢失是当追踪点周围图片块在图片 I 和 J 上变化的太大了。观察到这种条件就需要更多一点的挑战来精确判断。举个例子，如果一个特征点的错误函数大于一个阈值（公式 1），那么他可能就丢失了。这就带来了一个问题，如何来界定这个阈值。这在多张图片的完整序列中追踪一个点就会变得尤其敏感。确实，如果基于连续图像的追踪完成，则跟踪的图片块隐含的在每一次跟踪中初始化。结果是，这个点可能通过一序列图片后移动了很多，但是在两张连续图片上的位移很小。这个位移问题在处理比较长的序列时是一个很经典的问题。一个方法是在一个序列中跟踪一个特征点的同时为特征块保持一个修正的引用（使用第一张图片的特征）。根据这个技术，函数 $\varepsilon(d)$ 就有了更多的意义。然而，解决了这个问题就会有其他问题产生：特征可能就会消失的太快了。我们预想的解决这个问题的一个方向是用仿射图像匹配来决定丢失跟踪。

目前知道的最好的技术是联合传统的跟踪方法来计算匹配，用仿射图像匹配来决定错误跟踪。跟多的内容请参考 Shi 和 Tomasi 的文章。

3 特征选取

截止到目前为止，我们已经描述了跟踪的过程，就是在图片 I 上的一个点 u，找到它在图片 J 上的点 v。但是我们还没有描述如何在 I 上选取点 u。这个过程叫做特征选取。一旦跟踪的数学模型给出，解决特征选取的问题就会变得很直观了。确实，跟踪的中心步骤就是计算光流向量 $\bar{\eta}^k$。在那个过程中，矩阵 G 需要是可逆的，或者用别的话来说，G 的最小的特

征向量必须是足够大的（大于一个阈值）。这样的像素点就是好追踪的。

因此，选取的过程如下：

1. 在图像 I 中的每一个像素点计算矩阵 G 和它最小的特征向量 λ_m
2. 称 $\lambda_{\max}$ 是 λ_m 在整张图片上的最大值
3. 保留图片上的像素值的 λ_m 的个数大于一个比例的 $\lambda_{\max}$，这个比例可以是 10%或者 5%。
4. 通过这些点，保留局部最大的像素（如果一个像素的 λ_m 比周围其他像素的 λ_m 都大时，保留）
5. 保留这些像素的子序列就可以使在任何苏昂素对之间的最小距离比任何给出的阈值距离大（比如，10 或 5 个像素）。

通过这些处理，保留下来的像素一般都是容易去跟踪的。他们就是用来做跟踪的特征点。

算法的最后一步由执行像素间的成对的最小化距离组成，如果跟踪引擎可以处理很大数量的点也可能省略掉。这都要看特征跟踪器的计算性能。

最后观察到这一点很重要：为了选取特征没有必要去选一个很大的集成窗口（也是为了计算矩阵 G）。事实上，一个 3×3 的窗口就很有效率了。但是对跟踪来说，窗口就有点太小了。

参考文献

[1] Jianbo Shi and Carlo Tomasi, \Good features to track", Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn., pages
593{600, 1994.