华为视频抖动质量评价说明文档

梁观铭 2021.10

1. 研究背景

点播、直播行业的蓬勃发展，使用户生产视频(UGC)逐渐替代了专家生产和平台生产的方式，成为了主流。由于广大用户不可能全都具备专业素质和专业器材，其产出的视频往往质量较差，最明显的特征就是存在抖动。

当摄像头立杆不稳或因车辆引起地面振动时，视频画面就会发生抖动。显然，视频的抖动与拍摄者，即手机使用者的操作息息相关，那么如何评价不同用户在不同场景的不同拍摄效果呢，这就需要抖动评价算法来帮助我们分析视频抖动具体情况。

2. 抖动检测算法

a. 角点检测算法

视频抖动的本质是图像存在着微小、方向随机、频率较高的运动。首先要检测到图像帧与帧之间的运动方向，对于每一帧在自然场景下的空间位置，可以采取角点来表征，图像中的任何一个物体都通常含有独特的特征，但往往由大量的像素点构成，如图 1 所示，角点是能够准确描述这个物体的一个数量较少的点集。

使用一个固定窗口在图像上进行任意方向上的滑动，比较滑动前与滑动后两种情况，窗口中的像素灰度变化程度，如果存在任意方向上的滑动，都有着较大灰度变化，那么我们可以认为该窗口中存在角点，关于角点的具体描述可以有几种：

- 一阶导数(即灰度的梯度)的局部最大所对应的像素点；
- 两条及两条以上边缘的交点；
- 图像中梯度值和梯度方向的变化速率都很高的点；
- 角点处的一阶导数最大，二阶导数为零，指示物体边缘变化不连续的方向。

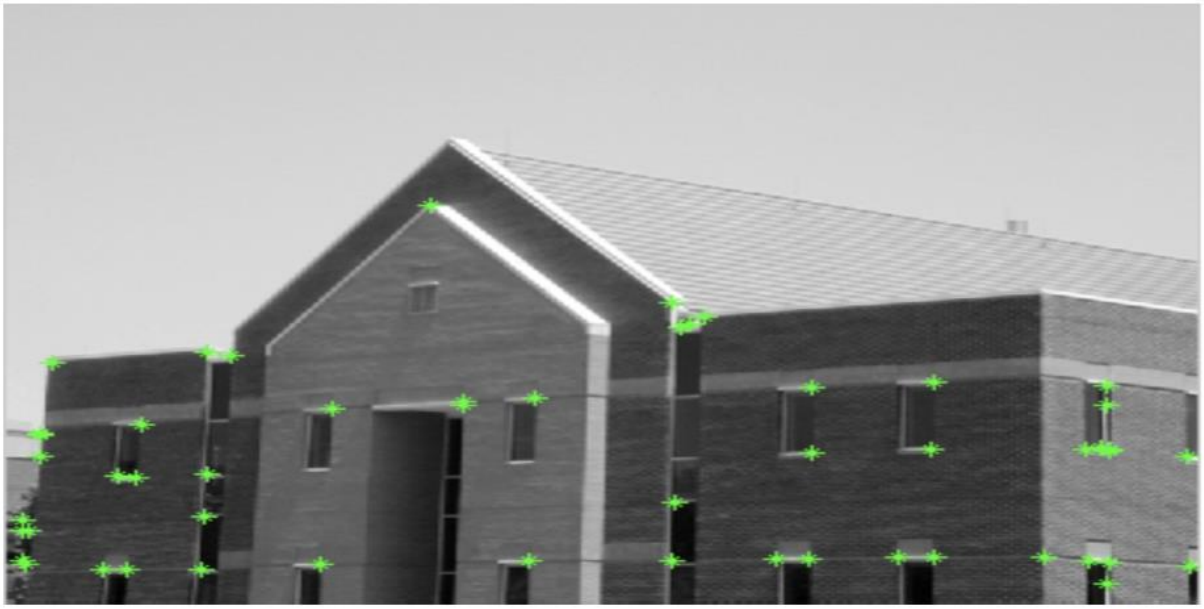
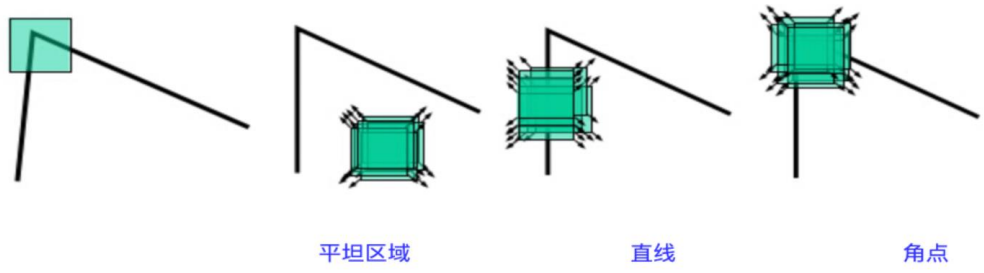


图 1. 一帧图像中的角点

Harris 角点检测：角点原理来源于人对角点的感性判断，即图像在各个方向灰度有明显变化。算法的核心是利用局部窗口在图像上进行移动判断灰度发生较大的变化，人眼对角点的识别通常是通过一个局部的小窗口完成的，如果在每个方向上移动这个小窗口，窗口内的灰度发生了较大的变化，那么说明该窗口内存在角点，如图 2 所示，窗口在各个方向上都没有变化，则认为窗口区域为平滑区域；窗口在某个方向上没有变化，另一个方向上有明显变化，那么，这块区域可能存在边缘；窗口在各个方向上灰度发生了较大的变化，那么，这块区域可能存在角点。Harris 角点检测正是利用了这个直观的物理现象，通过窗口在各个方向上的变化程度，决定是否为角点。



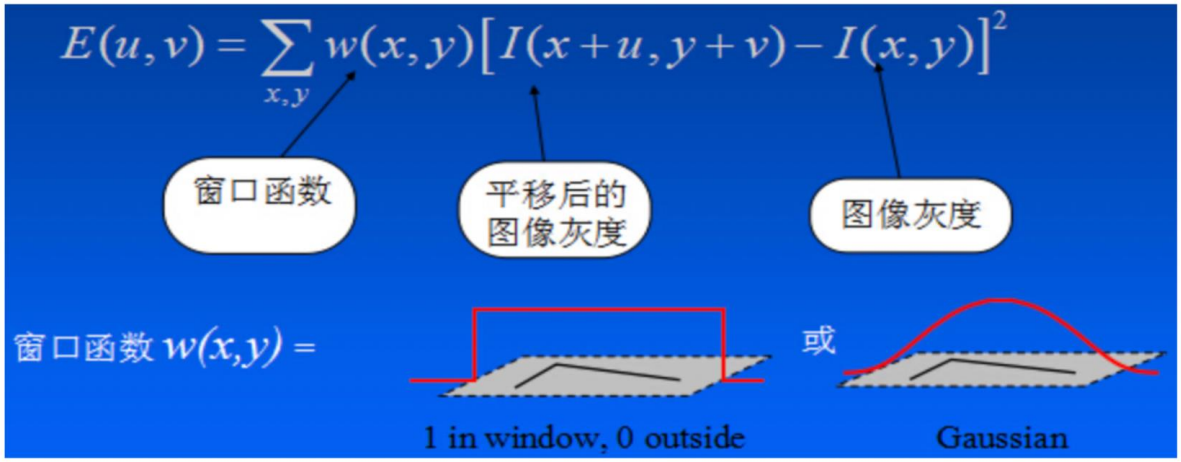


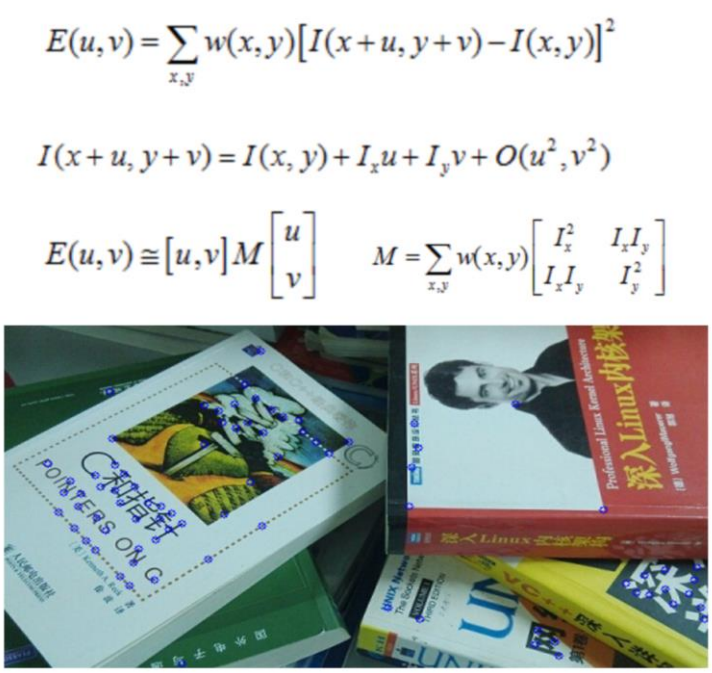
图 2. 图像上角点的灰度变化特性

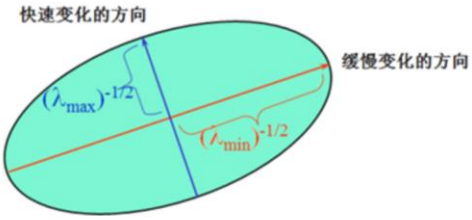
如图 3 所示，w 函数表示窗口函数，M 矩阵为偏导数矩阵。对于矩阵可以进行对称矩阵的变化，假设利用两个特征值进行替代，在几何模型中通过判断两个特征值的大小，来判定像素的属性。M 为梯度的协方差矩阵 ，在实际应用中为了能够应用更好的编程，定义了角点响应函数 R，通过判定 R 大小来判断像素是否为角点。R 取决于 M 的特征值，对于角点|R|很大，平坦的区域|R|很小，边缘的 R 为负值。Harris 角点检测算法就是对角点响应函数 R 进行阈值处理：R > threshold，即提取 R 的局部极大值。

$$E(u,v)=\sum_{x,y} w(x,y)\left[I(x+u,y+v)-I(x,y)\right]^2$$

$$I(x+u,y+v)=I(x,y)+I_x u+I_y v+O(u^2,v^2)$$

$$E(u,v)\cong[u,v]M\begin{bmatrix}u\\v\end{bmatrix}\qquad M=\sum_{x,y} w(x,y)\begin{bmatrix}I_x^2 & I_xI_y\\ I_xI_y & I_y^2\end{bmatrix}$$





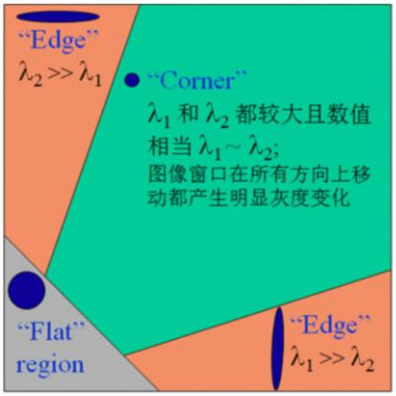


图 3. 图像上角点的判定方法

图 4. Haris 角点检测的数学表示

Shi-Tomasi 角点检测：Shi-Tomasi 算法是 Harris 算法的改进。Harris 算法最原始的定义是将矩阵 M 的行列式值与 M 的迹相减，再将差值同预先给定的阈值进行比较。后来 Shi 和 Tomasi 提出改进的方法，若两个特征值中较小的一个大于最小阈值，则会得到强角点。 Shi 和 Tomasi 的方法比较充分，并且在很多情况下可以得到比使用 Harris 算法更好的结果。

使用角点检测，可以表征图像帧的整体空间关系。

b. 光流法

光流法主要用于寻找不同图片间的特征点对应关系。特别是应用在视频中，因为对于视频，可以合理地认为当前帧中的许多点能够在下一帧中找到。

一个理想的光流算法输出应该是图中每个像素的速度预测集合，或是表示每个像素在相邻帧间相对位置的位移向量。当对图中每个像素求解时，就是密集光流法(dense optical flow)。相对的，也存在稀疏光流法(sparse optical flow)。稀疏光流法通过只追踪图中的特征点，同时达到快速和准确的目的。

Lucas-Kanade 稀疏光流法：Lucas-Kanade 算法最早在 1981 年提出，最初为密集光流法。因为本方法很容易应用在图片像素子集中，所以变成了一种重要的稀疏光流法。Lucas-Kanade 算法只依赖于围绕关键点的小窗口推断出的局部信息。这导致了 Lucas-Kanade 算法不能检测到物体的快速运动到窗口外部的点。这个缺点可以通过改进的金字塔 LK 光流法解决。

如图 5 所示，Lucas-Kanade 原理基本假设：

- 亮度恒定性：目标物体像素的强度值在帧间亮度不变。对于灰度图，即对于追踪的像素点，帧间亮度不变。
- 时间连续性：相邻帧间运动微小。
- 空间一致性：图中临近的点属于相同的表面，具有相似的移动。

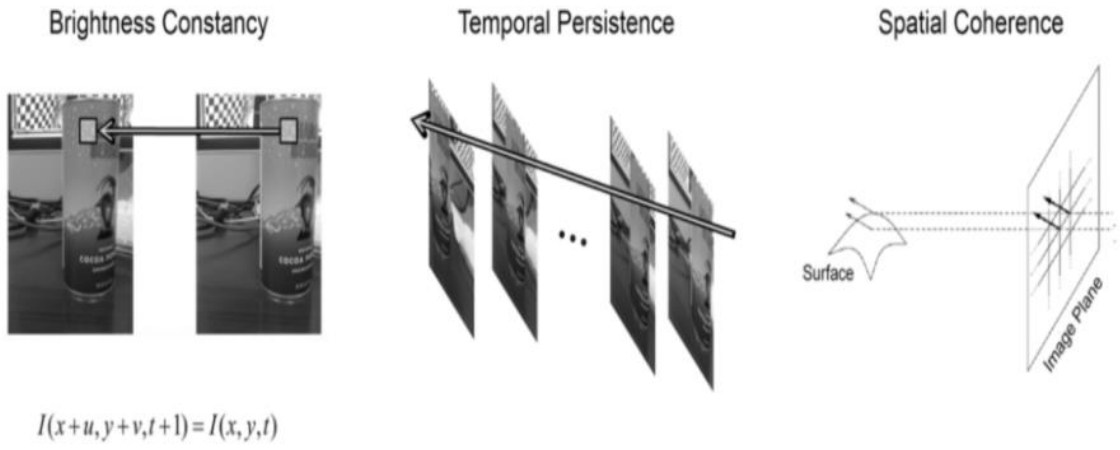


图 5. LK 光流法基本假设

根据亮度恒定假设要求跟踪部分亮度不随时间变化、时间连续假设，有以下数学假设：

$$I(x,y,t) = I(x,y,t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt = 0$$

$$I_x dx + I_y dy + I_t dt = 0$$

$$I_x u + I_y v = -I_t \Rightarrow \begin{bmatrix} I_x & I_y \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} = -I_t$$

若一个局部区域的像素运动是一致的，就可以建立邻域像素的系统方程求解中心像素的运动。如使用 5x5（窗口大小）的邻域像素亮度值，计算此像素的运动，就可建立 25 个方程：

$$A_{25\times2}\cdot d_{2\times1}=\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}=-\begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}=b_{25\times1}$$

窗口太大会违背空间一致性假设，太小又会追踪不到窗口外的点。以上方法是基于小而连贯运动的假设，但是对于大多数 30Hz 的摄像机，大而不连贯的运动是普遍存在的。所以需要更大的窗口来追踪运动，但这又会违背空间一致性假设。因此引入图像金字塔解决。最初在较大的空间尺度追踪，再在小的空间尺度上修正。

追踪方法：在图像金字塔最高层计算光流，得到的运动结果作为下层金字塔的起始点，重复这个过程直至最底层。如图 6 所示，这样就将不满足运动假定的可能性降到最低并实现更快更长的运动追踪。这个方法就叫金字塔 Lucas-Kanade 光流法。

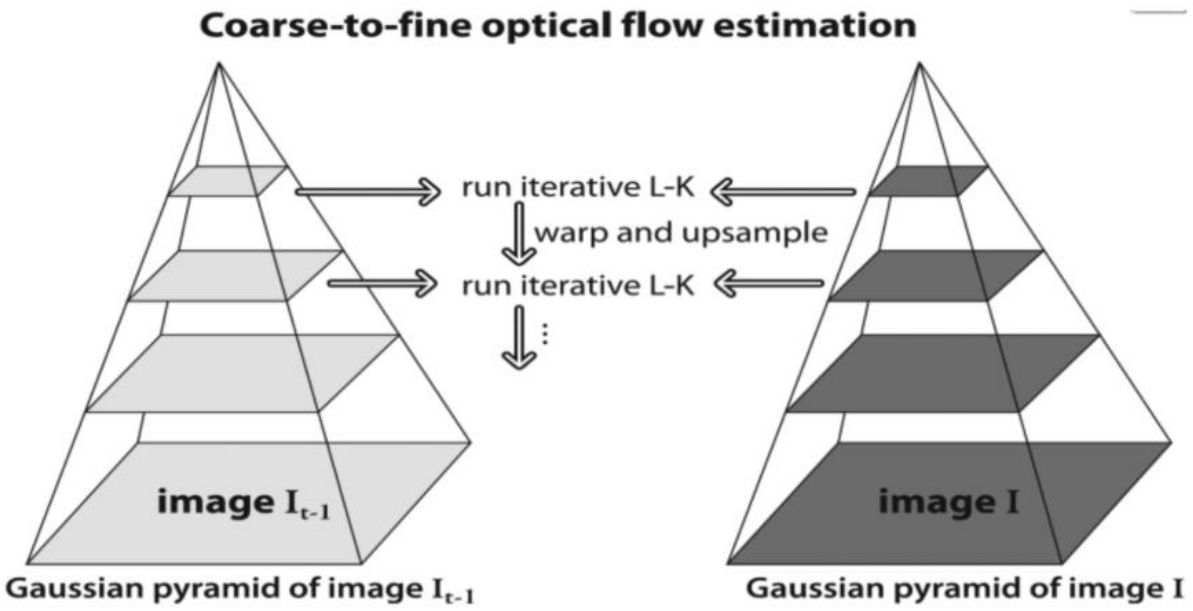


图 6. 金字塔 Lucas-Kanade 光流法

使用光流法，可以追踪前后帧之间角点的对应关系。

c. 2D 仿射变换+抖动帧判定

图像上的仿射变换, 其实就是图片中的一个像素点，通过某种变换，移动到另外一个地方。从数学上来讲， 就是一个向量空间进行一次线形变换并加上平移向量， 从而变换到另外一个向量空间的过程，即一种二维坐标（x, y）到二维坐标（u, v）的线性变换，其数学表达式形式如下：

$$\begin{cases} u = a_1 x + b_1 y + c_1 \\ v = a_2 x + b_2 y + c_2 \end{cases}$$

对应的齐次坐标矩阵表示形式为：

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

仿射变换保持了二维图形的“平直性”（直线经仿射变换后依然为直线）和“平行性”（直线之间的相对位置关系保持不变，平行线经仿射变换后依然为平行线，且直线上点的位置顺序不会发生变化）。非共线的三对对应点确定一个唯一的仿射变换。

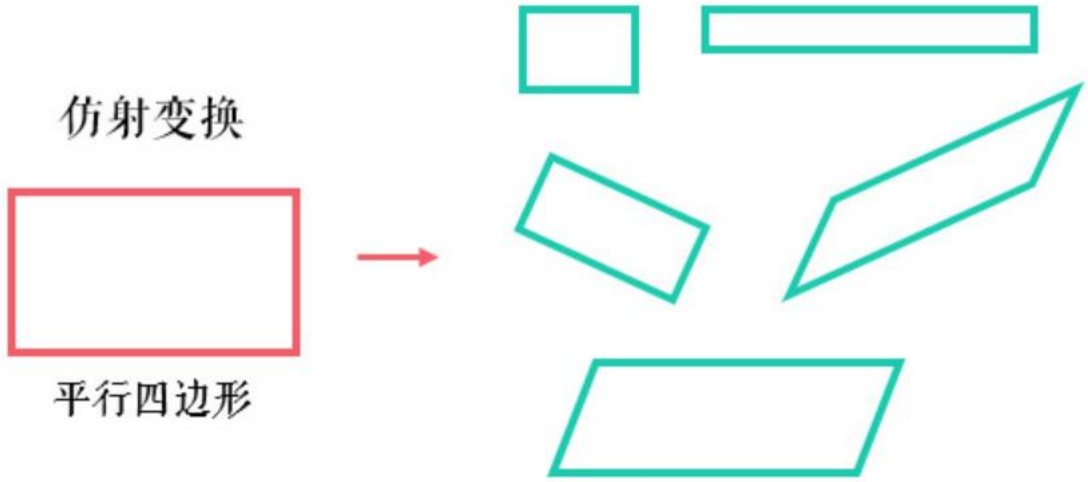
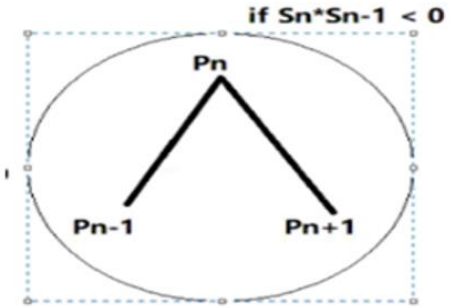


图 7. 2D 仿射变换

使用 2D 仿射变换，结合角点检测、光流追踪得到的结果，可以提取相邻帧的整体运动轨迹。对于图像帧，考虑相邻两帧的水平、垂直两个方向的相对运动位移，如图 8 所示，当相邻三图像帧出现位移方向的突变（包括 X、Y 两方向），那么对应地判定中间帧为 X 或 Y 方向上的抖动帧。



$$\begin{cases} X_{S_{n-1}} = X_{P_n} - X_{P_{n-1}} \\ Y_{S_{n-1}} = Y_{P_n} - Y_{P_{n-1}} \end{cases}$$

图 8. 抖动帧判定

d. opencv 实现

对上述使用到的算法，采用 opencv 中集成的方法实现如下：

- 角点检测：cv2.goodFeaturesToTrack，指定超参，在一幅灰度图上找到一定数目的角点；
- 光流追踪：cv2.calcOpticalFlowPyrLK，根据相邻两帧灰度图和前一帧的角点预测得到当前帧角点的所在位置；
- 2D 仿射变换：cv2.estimateAffinePartial2D，根据相邻两帧角点的位置关系给出旋转角、平移 dx、dy。

```
feature_params = dict(maxCorners=100, qualityLevel=0.1, minDistance=7, blockSize=7)
lk_params = dict(winSize=(15, 15), maxLevel=3, criteria=(cv2.TERM_CRITERIA_EPS|cv2.TERM_CRITERIA_COUNT, 10, 0.02))
prev_pts = cv2.goodFeaturesToTrack(prev_gray ,**feature_params)
curr_pts, status, err = cv2.calcOpticalFlowPyrLK(prev_gray, curr_gray, prev_pts, None, **lk_params)
m ,_ = cv2.estimateAffinePartial2D(prev_pts, curr_pts, False)
```

3. 抖动评价算法

本项目通过抖动检测算法得到相邻帧间相对位移 dx、dy，分别表示横、纵向图像帧的运动，对于连续三图像帧，有两个方向上各两段相对位移：

$$\begin{cases} dx_1 = x_{this} - x_{prev} \\ dx_2 = x_{post} - x_{this} \\ dy_1 = y_{this} - y_{prev} \\ dy_2 = y_{post} - y_{this} \end{cases}$$

对于同一方向上两段相对位移，有以下抖动幅度判定，level 表示单方向上的抖动强度，抖动强度的阈值由主观试验确定（对于大于 50*baseJitterUnit 的抖动舍弃，避免引入极大值影响整体判断）：

$$\begin{cases} threshold_{i+1}^{x} > -dx_1 * dx_2 \geq threshold_{i}^{x} & \Rightarrow level_x = i+1 \\ threshold_{i+1}^{y} > -dy_1 * dy_2 \geq threshold_{i}^{y} & \Rightarrow level_y = i+1 \\ threshold = [0, 0.5, 1, 2, 5, 10, 50] * baseJitterUnit & \\ 0 \leq i \leq 5, \quad 1 \leq level \leq 6 & \end{cases}$$

其中 baseJitterUnit 表示最小抖动单元，为确保同一视频在不同分辨率下有相似的抖动评分，baseJitterUnit 与视频分辨率有关，同时与抖动强度阈值是强相关的，在这里粗略设定如下：

$$\begin{cases} baseJitterUnit_x = w * w/50000 \\ baseJitterUnit_y = h * h/50000 \\ resolution\ ratio : w * h \end{cases}$$

对于连续三图像帧，根据以上公式可以判断得到其中间帧的抖动等级（为非抖动、或 1-6 级抖动），同时，对视频所有抖动帧的抖动幅度进行加权平均，可以得到视频的抖动分数（x，y 两个方向平均抖动幅度），对应地，设立六个等级作为视频最终的抖动评分，score 记录了以平均抖动幅度作为参考的抖动等级阈值（由主观试验确认），其中 jitterRank 表示抖动等级：1-6，数值越大，表示视频越抖。

$$\begin{cases} \bar{jitterRange} = \frac{\sum jitterRange}{n\ frame} \\ score = [0, 0.5, 4.5, 12.5, 24.5, 32.5, +\infty] \\ jitterRank = i + 1\ \ if\ score_{i+1} > \bar{jitterRange} \geq score_i \\ 0 \leq i \leq 5 \end{cases}$$

注：上述抖动强度对应单帧图像相对前后帧的抖动情况（仅用于统计视频的抖动帧强度分布情况）；抖动分数对应整个视频的平均抖动幅度，抖动等级对应整个视频的抖动情况（由抖动分数落在的阈值区间决定）。

4. 抖动视频数据集

抖动视频数据集包括验证、测试两种：

- 验证数据集：包括不同分辨率、不同场景、不同光照条件、不同抖动强度的视频共 20 个，用于验证算法可靠性。
- 测试数据集：包括分辨率为 1440x1080 的四种场景下的视频共 24 个，四个场景分别为①室内（人造景物）光照充足，②室内（人造景物）光照不足，③室外（自然景物）光照充足，④室外（自然景物）光照不足；每个场景下视频各六个，用于说明算法的普适性。

对于验证数据集，我们选取了不了解研究背景的若干名观众对 20 个不同分辨率、不同场

景、不同光照条件、不同抖动强度的视频进行观看，并且按照主观感受进行视频抖动程度的排序；同时，根据 20 个视频的抖动评价算法输出结果，即抖动分数，我们计算了 SROCC 指标为 0.91。

视频	抖动分数	人工排序	算法排序	d^2
v001.mp4	1.33	13	10	9
v002.mp4	0.15	8	6	4
v003.mp4	0.8	12	8	16
v004.mp4	2.41	9	11	4
v005.mp4	0.14	1	5	16
v006.mp4	3.53	10	12	4
v007.mp4	3.65	6	13	49
v008.mp4	0	1	1	0
v009.mp4	0.02	1	4	9
v010.mp4	12.66	14	15	1
v011.mp4	24.81	16	17	1
v012.mp4	0	1	1	0
v013.mp4	0	1	1	0
v014.mp4	13.6	17	16	1
v015.mp4	59.74	19	19	0
v016.mp4	0.19	6	7	1
v017.mp4	43.72	18	18	0
v018.mp4	83.97	20	20	0
v019.mp4	5.69	14	14	0
v020.mp4	1.29	11	9	4
				119
			SROCC	0.91

图 9. 算法在验证集上的表现

对于测试数据集，我们选取了对每种（共 4 种）场景下六个不同抖动强度的视频，其主观等级已经通过主观实验按序确定，经抖动评价算法测试，输出的结果符合主观感受分布，且各等级间视频抖动强度区分明显。

5. 算法结果

对输入的待检测视频，首先根据角点检测得到每一帧关键点，然后根据光流追踪得到相邻帧的关键点对应关系，再通过 2D 仿射变换可以得到相邻帧间的运动关系。根据帧序列的相对运动情况，可以得到每一帧的抖动强度，再通过综合完整视频的每一帧抖动情况，可以得到各项抖动相关的统计结果。

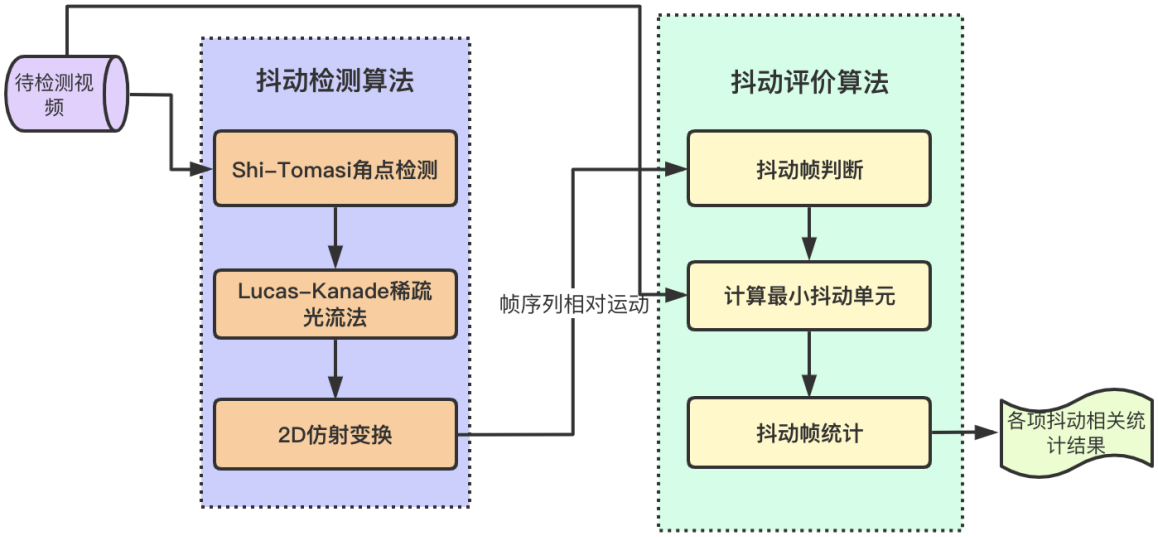


图 10. 抖动质量评价算法

一个典型的输出样例如图 11 所示，其中包括：

- 横、纵向抖动频率：抖动帧数/总帧数；
- 横、纵向抖动帧序列；
- 横、纵向抖动帧强度分布：共六种，如图 12 所示；
- 抖动分数：完整视频横、纵向平均抖动强度之和；
- 抖动等级：根据抖动分数及其参考阈值确定，共六种；
- 抖动主导轴：抖动占比较高的方向；
- 最抖动/平缓区间：通过任意帧（时间）的滑窗确定，描述视频局部抖动情况。

```
视频：  ourdoor004.mp4
x抖动阈值基数：  18.432
y抖动阈值基数：  5.91872
分辨率：  960 * 544
角点检测算法参数：  {'maxCorners': 100, 'qualityLevel': 0.1, 'minDistance': 7, 'blockSize': 7}
光流追踪算法参数：  {'winSize': (15, 15), 'maxLevel': 3, 'criteria': (3, 10, 0.02)}
x抖动频率：  109 / 331
y抖动频率：  81 / 331
x抖动帧序列：  [[26, 140, 207, 236, 327], [25], [53, 58, 124, 144, 145, 245], [110, 138, 208, 209, 254], [3, 126, 135, 141, 143, 210, 251, 283], [1, 10, 14, 15, 16, 18, 30, 33, 34, 35, 40, 42, 47, 50, 57, 60, 61, 66, 67, 71, 72, 75, 76, 78, 80, 81, 82, 83, 84, 85, 93, 96, 99, 102, 108, 111, 113, 115, 117, 118, 119, 120, 121, 123, 128, 130, 147, 148, 149, 150, 151, 152, 156, 161, 166, 167, 170, 173, 175, 176, 179, 181, 184, 187, 191, 197, 202, 205, 213, 214, 215, 218, 219, 220, 225, 230, 240, 250, 267, 271, 272, 273, 277, 308]]
x抖动六分化计数：  5 1 6 5 8 84
y抖动帧序列：  [[126], [90, 119], [80, 92, 191, 295], [3, 52, 179, 187, 188, 234, 306, 321], [0, 43, 84, 102, 184, 190, 251], [7, 9, 12, 13, 14, 20, 25, 30, 33, 37, 39, 50, 58, 60, 62, 66, 68, 70, 75, 82, 96, 111, 117, 133, 139, 144, 149, 152, 154, 155, 156, 161, 162, 163, 166, 169, 177, 192, 196, 198, 199, 202, 210, 214, 218, 219, 220, 221, 230, 233, 250, 252, 259, 263, 264, 268, 270, 271, 282]]
y抖动六分化计数：  1 2 4 8 7 59
x抖动分数 / 抖动帧数(总帧数)：  62675.072766876176  /  109  (  331  )
y抖动分数 / 抖动帧数(总帧数)：  14340.946378110879  /  81  (  331  )
x平均抖动强度：(占抖动帧) 575.00066758602
y平均抖动强度：(占抖动帧) 177.04872071741826
x平均抖动强度：(占全部帧) 189.3506730117105
y平均抖动强度：(占全部帧) 43.32612198825039
抖动分数为：  (越大表示越抖)  230.96593621319647
187.9577205882353 + 43.01139705882353 = 230.97
抖动等级为：  6 ，抖动主导轴为：  x
x轴最抖的是：  57 到  86 帧，平均抖动分数为：  399.4277847417196 抖动等级为：  6
y轴最抖的是：  193 到  222 帧，平均抖动分数为：  99.3947957702637 抖动等级为：  6
综合最抖的是：  57 到  86 帧，平均抖动分数为：  475.9305730888367 抖动等级为：  6
x轴最平稳的是：  279 到  308 帧，平均抖动强度为：  4.274289449055974 抖动等级为：  2
y轴最平稳的是：  284 到  313 帧，平均抖动强度为：  0.7201426188150966 抖动等级为：  2
综合最平稳的是：  279 到  308 帧，平均抖动强度为：  7.92283020019529 抖动等级为：  3
```

图 11. 抖动质量评价算法输出结果 1

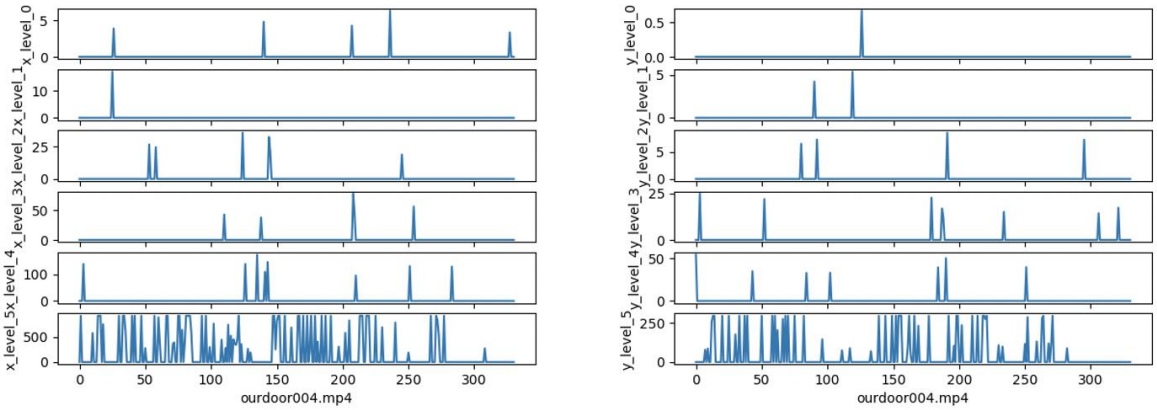


图 11. 抖动质量评价算法输出结果 2