一种基于光流检测的视频防抖技术

陶 文 束 伟
扬州航盛科技有限公司，江苏 扬州 225009

[摘要]随着智能手机和移动互联网的发展，视频已经成为一种重要的信息载体，视频拍摄过程中难以避免会产生画面抖动，影响用户体验。视频防抖的方法有很多种：机械、光学、电子、软件。文中从软件后处理的方法出发，基于光流法对原始抖动视频进行仿射变换估计，得到每帧画面的水平、垂直、角度偏移量参数，然后使用多项式拟合方法获得平滑的偏移量参数，最后对原始每帧图像使用平滑偏移量参数进行补偿，从而最终获得画面稳定的视频。文中选择 python 工具对整个防抖处理方案进行实现，并最终达到预期效果。

[关键词]光流法；视频防抖；特征点；仿射变换；曲线拟合；python
DOI：10.33142/sca.v6i6.9357 中图分类号：TN911.73 文献标识码：A

A Video Anti Shaking Technology Based on Optical Flow Detection

TAO Wen, SHU Wei
Yangzhou Hangsheng Technology Co., Ltd., Yangzhou, Jiangsu, 225009, China

Abstract: With the development of smartphones and mobile internet, video has become an important information carrier, and it is difficult to avoid screen shake during video shooting, which affects user experience. There are many methods for video anti shake: mechanical, optical, electronic, and software. Starting from the software post-processing method, this paper estimates the affine transformation of the original dithered video based on the optical flow method to obtain the horizontal, vertical and angular offset parameters of each frame, then uses the polynomial fitting method to obtain the smooth offset parameters, and finally uses the smooth offset parameters to compensate the original image of each frame, so as to finally obtain a stable video. The python tool was selected in the article to implement the entire anti shake processing scheme, and the expected results were ultimately achieved.

Keywords: optical flow; video stabilization; feature points; affine transformation; curve fitting; python

引言

视频起源于 19 世纪末，在 1890 年代初，法国发明家 Auguste 和 Louis Lumière 兄弟开发了一种称为 Cinématographe 的设备，它可以拍摄、处理和播放运动影像。20 世纪初，电影制作变得更加精细和专业化，出现了各种技术和创新，例如使用彩色胶片、声音录制和放映以及特殊效果。在 20 世纪后半叶，视频技术开始进入大众市场。1960 年代，便携式摄像机的出现使普通人可以拍摄自己的视频。随着录像带和录像机的发展，人们可以记录和观看自己的家庭视频。20 世纪末随着计算机技术和互联网的发展，视频在数字领域得到了进一步的发展。1990 年代末和 2000 年代初，视频压缩技术的改进和宽带互联网的普及使在线视频流行起来。2008 年以后随着智能手机和移动互联网的普及，视频已成为人们获取信息、工作学习、社交娱乐的一种重要载体。

视频内容的产生离不开视频素材的采集，很大一部分视频内容是由摄像头进行采集的，由于摄像者手持相机或手机对物体进行跟踪拍摄时，不可避免会导致镜头晃动，从而引起视频图像的抖动，如果后期不进行处理，会导致观赏效果变差。因此如何提高视频图像质量，成为行业中重要的技术课题。

1 背景介绍

1.1 视频防抖方案

常见的视频防抖技术有如下几种：

1.1.1 机械稳定方法

这种方法使用特殊的传感器，如陀螺仪和加速度计，来检测运动并通过伺服电机或机械臂往跟抖动相反的方向移动图像传感器以抵消摄像机的抖动，从而起到稳定图像的目的。这种稳定器通常作为附加设备使用，可以将相机或摄像机固定在稳定的平台上，以减少抖动。比如手持云台稳定器，可以进行 3 个轴的转动，有效减小图像抖动[1]。

1.1.2 光学稳定方法

这是一种通过光学元件在镜头内部进行微调来纠正图像抖动的技术。它可以通过检测和补偿相机的抖动来提供相对稳定的图像。这种技术最早在高端摄像机和镜头中使用，而如今大部分智能手机的主摄像头模块都有一定的光学防抖功能，这种防抖方法不管是直接改善图像质量还是为了下一步的数字图像处理，都非常有用。

1.1.3 电子图像稳定法

这种方法通过图像处理算法对图像进行平滑处理，从而抵消相机的抖动。这种技术通常在手机摄像头和一些消费级摄像机中使用。它可以通过裁剪图像边缘或使用图像

插值来减少抖动，但可能会导致画面略微失真。

1.1.4 后期软件稳定方法

这种方法不需要特殊的传感器来估计摄像机的运动，而是使用图像处理算法来检测和纠正图像中的抖动，这种方法的好处是即使原始视频抖动严重，也可以通过后期软件处理的方法改善或解决视频抖动问题，而无须重新拍摄原始素材。比如常用的商业视频编辑软件 Adobe Premiere Pro 可以对输入视频内容进行防抖处理。

本论文描述的视频防抖技术就是属于第 4 种，通过软件的方法对原始抖动严重的视频数据进行处理，明显改善抖动现象。

1.2 软件视频防抖处理流程

通常软件对视频进行防抖处理要经过如下过程，如图 1 所示：

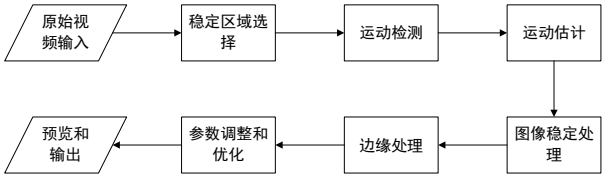

图 1 软件对视频进行防抖处理流程

1.2.1 原始视频输入

将需要进行防抖处理的视频导入到视频编辑或处理软件中。

1.2.2 稳定区域选择

根据需要，选择视频中的稳定区域。

1.2.3 运动检测

应用运动检测算法来分析视频中的运动模式和相机抖动。这可以通过运动感知算法、块匹配算法、光学流算法或其他相应的方法实现。

1.2.4 运动估计

基于运动检测的结果，估计相机抖动的运动参数，例如平移、旋转和缩放等。这些参数将用于对图像进行稳定处理。

1.2.5 图像稳定处理

根据运动估计结果，应用稳定算法来纠正视频图像的抖动。这可能涉及到平移、旋转、缩放、像素插值等变换和处理方法。

1.2.6 边缘处理

在稳定处理之后，可能会出现视频边缘的空白区域或黑边，边缘处理可以填充或修复这些区域。

1.2.7 参数调整和优化

根据实际情况，调整稳定算法的参数以获得最佳的稳定效果。包括抖动补偿的强度、平滑度和其他参数的调整。

1.2.8 预览和输出

在完成防抖处理后，可以预览处理后的视频，并检查稳定效果是否符合预期。如果满意，可以选择将处理后的视频导出为最终输出。

1.3 常用视频防抖算法

1.3.1 基于运动感知的图像稳定算法

这种算法通过分析视频帧之间的运动信息来检测相机的抖动，并根据检测到的运动模式对图像进行稳定。它可以利用运动向量估计或光流计算来获取图像的运动信息，并应用平移、旋转和缩放等变换来纠正抖动[2]。

1.3.2 基于块匹配的图像稳定算法

这种算法将视频帧划分为小块，并在连续帧之间进行块匹配，以获取图像的运动向量。通过分析运动向量，算法可以估计相机的抖动，并对图像进行相应的稳定处理[3]。

1.3.3 基于陀螺仪或传感器数据的图像稳定算法

一些相机和移动设备配备了陀螺仪或其他传感器，可以提供相机的姿态和加速度等数据。基于这些数据，可以开发算法来检测相机的抖动，并根据测量结果对图像进行稳定处理[4]。

1.3.4 基于光学流的图像稳定算法

光学流是指图像中像素随时间变化的位移模式。基于光学流的算法可以利用相邻帧之间的像素位移信息来估计相机的抖动，并对图像进行稳定处理[5]。

1.3.5 混合模型的图像稳定算法

这种算法结合了多种稳定方法，如光学稳定、电子稳定和数字滤波等。它根据视频的特点和相机的性能选择最合适的稳定方法，并进行动态调整，以达到最佳的稳定效果[6]。

2 方案设计

2.1 原始待处理视频

原始视频为手持相机走动过程拍摄的道路街景，可以看到原始画面抖动相当严重，如下图所示，两张相邻帧图像叠加在一起后图像很模糊，说明抖动严重。

图 2 原始视频画面抖动严重

2.2 算法选择

2.2.1 角点检测算法 GoodFeaturesToTrack

该算法作用是在图像中寻找具有良好跟踪性质的特征点，该算法最初由 Lucas 和 Kanade 于 1981 年提出，并被广泛应用于光流估计、特征匹配、物体跟踪等任务中。

该算法基本原理如下：

①输入：一个灰度图像。

②首先，通过应用一个平滑滤波器（如高斯滤波器）来减少图像的噪声。

③然后，计算图像中每个像素的梯度。常用的方法是使用 Sobel 算子计算水平和垂直方向的梯度。

④对于每个像素，计算其周围像素的梯度方向和强度，可以使用窗口或卷积核来计算。常用的方法是使用 3x3 的窗口。

⑤对于每个像素，根据周围像素的梯度信息计算一个响应值（也称为角点响应值）。常用的响应函数是 Harris 角点响应函数，它基于像素周围区域的梯度矩阵。

⑥对于计算得到的所有像素的响应值，根据一定的阈值进行筛选，只保留具有较高响应值的像素作为特征点。

⑦可选地，可以使用非极大值抑制方法来进一步精确选择特征点，以避免在相邻位置选择多个相似的特征点。

⑧输出：被选中的特征点的坐标。

GoodFeaturesToTrack 算法选择具有良好跟踪性质的特征点，这些特征点通常在图像中具有较强的灰度变化、边缘或纹理信息。这些特征点在图像序列中的位置变化较小，因此适用于光流估计和物体跟踪等任务。

2.2.2 光流检测算法 calcOpticalFlowPyrLK

该算法是用于估计图像序列中特征点的光流（运动）信息。它基于 Lucas-Kanade 光流算法的改进版本，通过使用图像金字塔来提高光流估计的准确性和鲁棒性[7]。基本原理如下：

①输入：两个连续的图像帧，以及在第一个图像帧中选定的一组特征点的初始坐标。

②创建图像金字塔：通过对输入图像进行多次降采样，得到一系列不同分辨率的图像。这样做的目的是为了在不同尺度上对特征点进行跟踪，以应对物体的尺度变化。

③对于每个金字塔层级，执行以下步骤：

（1）在当前层级上，使用 Lucas-Kanade 光流算法计算特征点的光流向量。该算法通过在当前帧和下一帧之间的窗口区域内匹配特征点，并估计其位移向量。（2）根据光流向量更新特征点的坐标。（3）通过反向投影误差（back-projection error）来判断光流估计的准确性，如果误差较大，则将该特征点排除。（4）重复上述步骤，直到达到预设的迭代次数或收敛条件。

④可选地，可以通过几何校正（geometric correction）来提高光流估计的准确性。这包括在每个金字塔层级上应用图像畸变矫正和相机运动估计等技术。

⑤输出：在最高层级上的特征点的最终光流向量。

calcOpticalFlowPyrLK 算法通过使用图像金字塔来处理尺度变化，并通过迭代优化特征点的位置来估计它们的光流向量。该算法在计算速度和准确性之间取得了平衡，广泛应用于视频稳定、物体跟踪、运动分析等计算机视觉任务中。

2.2.3 局部仿射变换估计算法 estimateAffinePartial2D

该算法用于估计图像中的局部仿射变换[8]。该算法可被用于图像配准、图像拼接、物体识别等应用中，通过估计仿射变换参数来将一个图像映射到另一个图像上。

基本原理如下：

①输入：两个图像或特征点集合，其中一个图像或特征点集合是参考图像，另一个是目标图像或特征点集合。

②根据输入的图像或特征点集合，构建匹配点对。这些匹配点对是参考图像和目标图像之间的对应关系。

③选择一些匹配点对，作为局部仿射变换的输入。

④通过最小二乘法估计仿射变换的参数。常见的仿射变换模型包括平移、旋转、缩放和剪切。

⑤输出：估计得到的仿射变换参数。

estimateAffinePartial2D 算法通过选择匹配点对并使用最小二乘法来估计仿射变换参数，从而实现图像间的局部仿射变换。这种变换可以在图像配准、目标跟踪等任务中对图像进行对齐和变换。

2.2.4 多项式拟合 polyfit

该算法用于拟合数据集，通过多项式函数逼近数据点，以获得最佳拟合曲线。该算法可以用于数据分析、曲线拟合、趋势预测等应用中[9]。

基本原理如下：

①输入：包含一组数据点的数据集，其中每个数据点由自变量（x）和因变量（y）组成。

②选择要拟合的多项式的阶数（degree），即多项式的最高次幂。

③构建一个代表多项式的方程，形式为：

$$y = p_n x^n + p_{n-1} x^{n-1} + \cdots + p_1 x + p_0 \quad (1)$$

其中 p_i 是多项式系数。

④使用最小二乘法来拟合数据点，以找到最佳的多项式系数，使得拟合曲线与数据点之间的残差平方和最小化。

最小二乘法通过求解以下方程组来估计多项式系数：

$$\begin{bmatrix} x_1^n & x_1^{n-1} & \cdots & 1 \\ x_2^n & x_2^{n-1} & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ x_m^n & x_m^{n-1} & \cdots & 1 \end{bmatrix} \begin{bmatrix} p_n \\ p_{n-1} \\ \vdots \\ p_0 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \quad (2)$$

⑤输出：拟合多项式的系数，它描述了最佳拟合曲线。

polyfit 算法通过最小二乘法来估计多项式系数，从而得到最佳拟合曲线。该算法能够灵活地逼近不同形状的数据集，并用多项式函数对数据进行建模和预测。

2.2.5 仿射变换算法 warpAffine

该算法用于对图像进行仿射变换。通过对图像进行线性变换和插值操作来实现图像的平移、旋转、缩放、剪切等变换操作。该算法在计算机视觉和图像处理领域广泛应用于图像配准、图像校正、目标跟踪等任务中。

基本原理如下：

①输入：待变换的源图像和变换矩阵。

②创建一个空白目标图像，用于存储变换后的图像。

③对于目标图像的每个像素点（x’，y’），根据变换矩阵计算对应的源图像坐标（x，y）：

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (3)$$

其中，变换矩阵包含 6 个参数（a，b，c，d，e，f），表示了仿射变换的平移、旋转、缩放和剪切等操作。

④根据插值方法，在源图像中找到与计算得到的坐标最接近的像素点，并将其值赋给目标图像的对应像素点。常用的插值方法有最近邻插值、双线性插值和三次样条插值。

⑤重复步骤 3 和步骤 4，直到为目标图像的每个像素点计算出值。

⑥输出：变换后的目标图像。

warpAffine 算法通过变换矩阵对源图像进行坐标映射和插值操作，从而实现对图像的仿射变换。这种变换可以改变图像的位置、角度、尺度和形状，从而实现图像的校正、调整和对齐等操作。

2.3 方案设计

由于原始视频为手持相机走动过程中拍摄的，抖动主要来自于手的水平、垂直、角度三个维度的波动，本方案主要分两步进行：完整的处理流程图如下图所示：

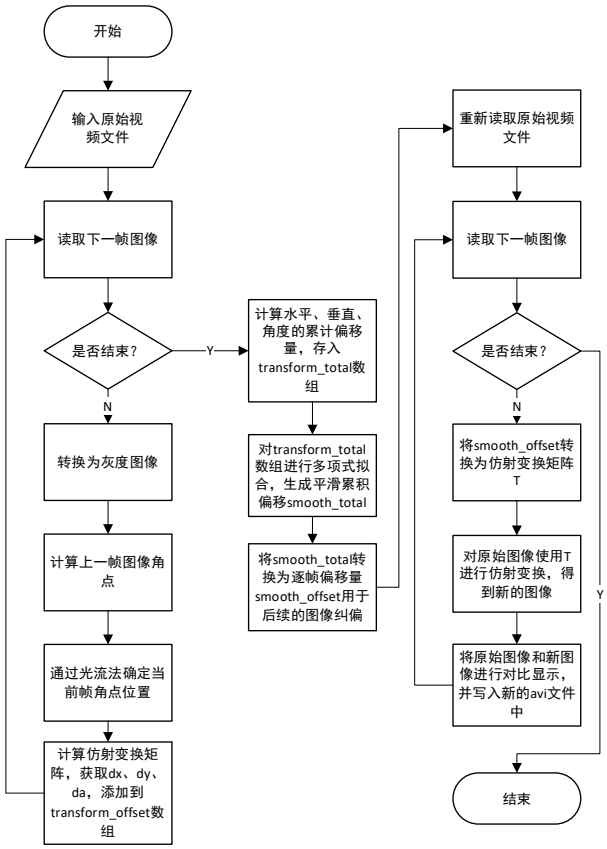

图 3　防抖处理完整流程

第一步，逐帧读取所有视频图像，先利用 goodFeaturesToTrack 函数找出良好的特征点，然后利用光流法计算这些特征点的目标位置，跟之前位置对比获得水平、垂直、角度偏移量，对总偏移量用多项式拟合的方法，获得平滑的偏移量参数。

第二步，对视频中的每帧图像使用第一步计算出来的平滑偏移量参数进行仿射变换，得到偏移量稳定的图像，再将这些图像重新打包成新的视频文件。

3 实验结果

3.1 实验环境

由于 Python 开源免费，生态完善，拥有功能丰富的库，并且运行效率较高。非常适合用来进行图像算法类方案的开发和验证，如下是本实验用到的工具的版本：

Pychram：2023.1.2

Python：3.11.3

OpenCV：4.7.0

Numpy：1.24.3

Matplotlib：3.7.1

3.2 实验过程

3.2.1 设置全局变量

```
VIDEO_FILE = '1.avi'
SMOOTHING_RADIUS = 30
HORIZONTAL_BORDER_CROP = 20
POLYFIT_DEG = 1
CANVAS_BLANK = 50
```

3.2.2 预读第一帧图像，获取基本参数

```
cap = cv2.VideoCapture（VIDEO_FILE）
ret， frame_old = cap.read（）
gray_old = cv2.cvtColor（ frame_old ，cv2.COLOR_BGR2GRAY）
frame_count = np.int32[cap.get（cv2.CAP_PROP_FRAME_COUNT）]
frame_hight = np.int32[cap.get（cv2.CAP_PROP_FRAME_HEIGHT）]
frame_width = np.int32[cap.get（cv2.CAP_PROP_FRAME_WIDTH）]
fps = cap.get（cv2.CAP_PROP_FPS）
```

3.2.3 逐帧读取视频图像，并计算相邻帧之间的偏移量

```
while cap.isOpened（）:
ret， frame = cap.read（）
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack（gray_old， 200，0.01， 30）
p1 ， st ， err = cv2.calcOpticalFlowPyrLK（gray_old， gray， p0， None）
good_old = p0[st == 1]; good_new = p1[st == 1]
T = cv2.estimateAffinePartial2D （good_old，good_new）[0]
dx = T[0， 2]; dy = T[1， 2]; da = math.atan2
```

```
(T[1, 0], T[0, 0])
    transform_offset[k - 1] = np.array([dx, dy, da])
    gray_old = gray
    k += 1
    3.2.4 计算累积偏移量
    index = np.arange(frame_count)
    x = 0; y = 0; a = 0
    for i in index[: -1]:
    x += transform_offset[i, 0]
    y += transform_offset[i, 1]
    a += transform_offset[i, 2]
    transform_total[i] = np.array([x, y, a])
    3.2.5 对累积偏移量进行拟合,获得平滑累积偏移量
    smooth_total = np.zeros(transform_total.shape)
    f1 = np.polyfit(index[:-1], transform_total[:, 0], POLYFIT_DEG)
    p1 = np.poly1d(f1)
    smooth_total[:, 0] = p1(index[: -1])
    f2 = np.polyfit(index[:-1], transform_total[:, 1], POLYFIT_DEG)
    p2 = np.poly1d(f2)
    smooth_total[:, 1] = p2(index[: -1])
    f3 = np.polyfit(index[:-1], transform_total[:, 2], POLYFIT_DEG)
    p3 = np.poly1d(f3)
    smooth_total[:, 2] = p3(index[: -1])
    3.2.6 将原始累积偏移量和平滑处理后的累积偏移量可视化
    plt.plot(transform_total[:, 0], 'r', transform_total[:, 1], 'g', transform_total[:, 2], 'b')
    plt.plot(smooth_total[:, 0], 'r', smooth_total[:, 1], 'g', smooth_total[:, 2], 'b')
    plt.show()
    3.2.7 将平滑累积偏移量转换为平滑逐帧偏移量
    smooth_offset = np.zeros(transform_offset.shape)
    for i in index[: -1]:
    diff_x, diff_y, diff_a = smooth_total[i] - transform_total[i]
    smooth_offset[i] = transform_offset[i] + [diff_x, diff_y, diff_a]
    3.2.8 对每帧图像用平滑逐帧偏移量进行补偿,以消除图像抖动
    vert_border = np.int32(HORIZONTAL_BORDER_CROP * frame_hight / frame_width)
    out_size = ((frame_width + CANVAS_BLANK * 2) * 2, frame_hight + CANVAS_BLANK * 2)
    while True:
    ret, frame = cap.read()
    T[0, 0] = math.cos(smooth_offset[k, 2])
    T[0, 1] = -math.sin(smooth_offset[k, 2])
    T[1, 0] = math.sin(smooth_offset[k, 2])
    T[1, 1] = math.cos(smooth_offset[k, 2])
    T[0, 2] = smooth_offset[k, 0]
    T[1, 2] = smooth_offset[k, 1]
    rows, cols = frame.shape[: 2]
    frame_smooth = cv2.warpAffine(frame, T, (cols, rows))
    canvas = np.zeros([rows + CANVAS_BLANK * 2, (cols + CANVAS_BLANK * 2) * 2, 3], frame.dtype)
    canvas[CANVAS_BLANK : -CANVAS_BLANK, CANVAS_BLANK: cols + CANVAS_BLANK] = frame
    canvas[CANVAS_BLANK: -CANVAS_BLANK, cols + CANVAS_BLANK * 3: -CANVAS_BLANK] = frame_smooth
    canvas = cv2.putText(canvas, 'original', (int(cols / 2) + CANVAS_BLANK - 60, rows + CANVAS_BLANK + 30),
    C-v2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 1)
    canvas = cv2.putText(canvas, 'stabilized', (int(cols / 2) + CANVAS_BLANK * 3 + cols - 60, rows + CANVAS_BLANK + 30),
    cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 1)
    cv2.imshow('original vs stabilized', canvas)
```

3.3 实验结果

运行 python 代码，输出水平、垂直、角度偏移量及拟合后的平滑曲线，如下图所示：

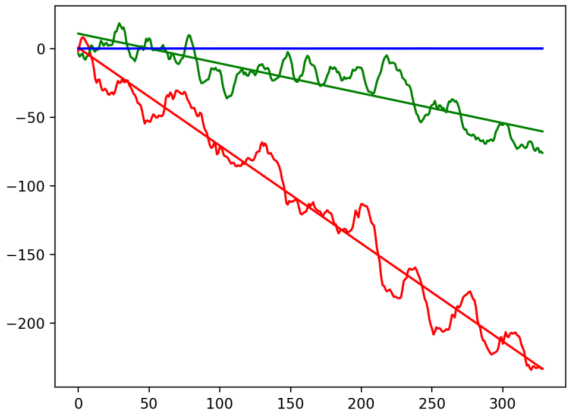

图 4 原始偏移量 vs 平滑后的偏移量

其中，红色曲线为水平方向偏移量，绿色曲线为垂直方向偏移量，蓝色曲线为角度偏移量，三种颜色的直线部

分是一次多项式拟合后的偏移量，拟合效果良好。

然后输出视频对比效果预览：

图 5　原始视频 vs 防抖处理后的视频

左侧为原始视频，右侧为防抖处理后的视频，为了更好地显示仿射变换的效果，右侧图像的边缘未经裁剪处理，可以看到，防抖处理后的视频效果非常明显。

4 结束语

视频防抖处理方法有很多种，本文针对移动手持拍摄的街景视频的抖动特点，使用光流检测方法计算出相邻帧之间的水平、垂直、角度偏移量，然后使用多项式拟合方式，获得平滑的偏移量参数，最后对原始视频每帧图像进行纠偏。最终效果显示使用该方案防抖处理后的视频稳定性良好，达到了预期的效果。

[参考文献]

[1]唐佳林,郑杰锋,李熙莹,等.基于特征匹配与运动补偿的视频稳像算法[J].计算机应用研究,2018,35(2):4.
[2]黄石磊,陈书立,刘驰,等.基于视觉感知的运动目标跟踪算法[J].计算机应用研究,2013(7):2199-2201.
[3]王旸,冯驰.基于块匹配的电子图像稳定算法[J].咸阳师范学院学报,2006(4):36-38.
[4]赵赛,康宝生,王力.基于 MEMS 陀螺仪的电子稳像算法[J].西北大学学报自然科学版,2018,48(3):355-362.
[5]孙辉,赵红颖,熊经武,金宏.基于光流模型的图像运动估计方法[J].光学精密工程,2002(5):443-447.
[6]张敏,赵猛,贾云得,等.基于自适应高斯混合模型的图像稳定方法[J].北京理工大学学报,2004(10):897-900.
[7]肖军,朱世鹏,黄杭,等男.基于光流法的运动目标检测与跟踪算法[J].东北大学学报：自然科学版,2016,37(6):770-774.
[8]曾文锋,李树山,王江安.基于仿射变换模型的图像配准中的平移、旋转和缩放[J].红外与激光工程,2001(1):18-20.
[9]陈光,任志良,孙海柱.最小二乘曲线拟合及 Matlab 实现[J].兵工自动化,2005(3):107-108.

作者简介：陶文（1985.11—），男，江苏扬州人，汉族，硕士，工程师，主要从事汽车子嵌入式软件开发；束伟（1983.8—），男，江苏扬州人，汉族，硕士，工程师，主要从事汽车子嵌入式软件开发。