**导入库**  
程序开头通过“import cv2”和“import numpy as np”导入了 OpenCV 和 NumPy 两个库。OpenCV 用于图像和视频处理，而 NumPy 用于进行数值计算和矩阵操作。

**定义类 SmoothVideoProcessor**  
代码中定义了一个名为 SmoothVideoProcessor 的类，它封装了视频稳定化处理的所有功能。使用面向对象的设计将相关操作集中在一起，有助于后续扩展和维护。

**构造函数 **init****  
在构造函数中，设置了输入视频文件名（src\_filename）、输出视频文件名（dst\_filename）和一个布尔参数 enable\_perspective（决定是否使用透视变换）。同时，还初始化了一些属性，如视频帧率、帧宽、帧高、视频捕获对象、视频写入对象，以及一个初始为单位矩阵（3x3）的累积变换矩阵，用于累积每一帧的变换。

**方法 initialize\_io**  
该方法用于打开输入视频文件，并获取视频的基本属性（帧率、宽度、高度）。同时，根据这些属性创建视频写入器，用于保存处理后的视频。如果无法打开视频，会输出提示信息。

**方法 compute\_transform**  
这个方法的主要作用是计算当前帧与上一帧之间的变换，从而实现视频稳定化。具体步骤如下：

* 先使用 cv2.goodFeaturesToTrack 检测上一帧中的角点，作为易于跟踪的特征点。
* 接着利用 cv2.calcOpticalFlowPyrLK 方法，将这些角点从上一帧跟踪到当前帧，并得到新位置。
* 检查并筛选出跟踪成功的点（有效点）。如果有效点少于 4 个，则直接返回当前帧，因为无法计算变换。
* 如果 enable\_perspective 为 True，则采用透视变换（通过 cv2.findHomography 计算单应性矩阵），更新累积矩阵，并使用 cv2.warpPerspective 对当前帧进行透视变换。
* 否则，采用仿射变换（通过 cv2.estimateAffinePartial2D 计算仿射矩阵），将 2x3 仿射矩阵扩展为 3x3，更新累积矩阵，然后使用 cv2.warpAffine 进行仿射变换。 最终返回经过变换后的帧。

**方法 process\_frames**  
该方法逐帧处理视频。首先读取视频的第一帧，将其转换为灰度图像作为参考帧；然后进入循环，不断读取后续帧，对每一帧进行以下操作：

* 将当前帧转换为灰度图像；
* 调用 compute\_transform 方法计算稳定化后的帧；
* 将稳定化后的帧写入输出视频，同时用 cv2.imshow 实时显示处理后的结果；
* 更新参考帧为当前帧的灰度图像。 循环在视频结束时退出。

**方法 cleanup**  
在处理完所有帧后，此方法负责释放视频捕获和写入对象，并关闭所有 OpenCV 创建的窗口，确保资源正确释放。

**方法 execute**  
该方法是整个视频稳定化处理流程的入口。它依次调用 initialize\_io、process\_frames 和 cleanup 方法，并在处理完成后输出一条提示信息，说明输出文件的位置。

**辅助函数 smooth\_video**  
这是一个包装函数，它接受输入和输出文件名以及是否启用透视变换的参数。函数内部创建 SmoothVideoProcessor 对象并调用其 execute 方法，使得调用过程更加简单。

**主程序入口**  
最后，通过“if **name** == '**main**':”判断，代码在作为脚本运行时，会调用 smooth\_video 函数处理不同的视频文件。这里分别对两个视频文件（video\_seq\_1.avi 和 video\_seq\_2.avi）使用仿射变换和透视变换进行处理，生成对应的输出视频文件。

**总结：**  
整个程序采用面向对象的设计，将视频的打开、处理、写入和资源释放都封装到一个类中。