Video Stabilization

Guilherme Franco & Miguel Rodrigues

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole
{xfranc01, xboave00}@fit.vutbr.cz

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

December 23, 2023

Task Definition

The aim of the project is to create an application capable of compensating for unwanted camera movement.
Also known as Video Stabilization

Typical Solutions

Conventional Methods

Digital video stabilization usually employs a “three-step” approach, illustrated in Fig. 1: (i) motion estimation,(ii) motion compensation, and (iii) image warp.

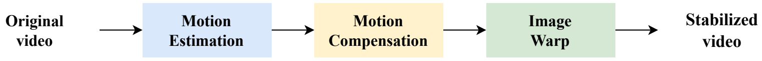
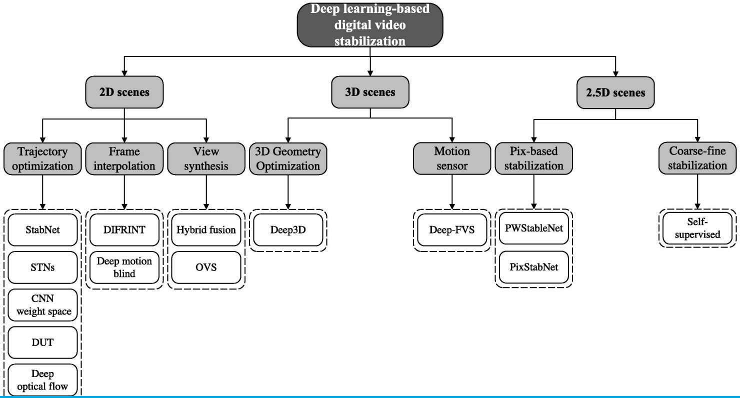


Figure: Pipeline

Typical Solutions

Deep Learning Methods

With the developments of deep learning, more and more methods based on convolutional neural networks(CNNs) have been proposed for digital video stabilization in recent years. Instead of explicitly computing a camera path, these methods model a supervised learning approach.



Our Solution

In this work, we have opted to follow a classical approach. This decision took into account the initial requirements. They state that the solution should focus on the speed of stabilization.
In this section, we will detail our solution and some of the issues that rose during the development of this work.

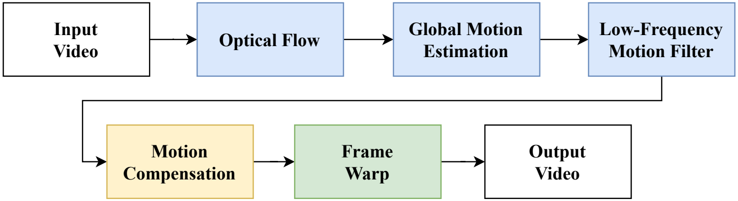
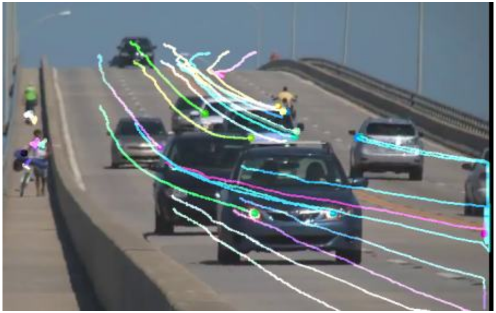


Figure: Our Pipeline

Optical Flow

The first stage of our pipeline is to compute the optical flow. Moreover, optical flow algorithms can be split into two distinct categories: dense and sparse.
During the project we started by experimenting with PyTorch's implementation of RAFT. Although RAFT is a state of the art optical flow algorithm which employs deep neural networks, it demonstrated to be very slow for our needs. For instance, processing 15 frames took around 2 minutes which was not desirable.



Optical Flow

The solution we found was to use OpenCV's implementation of Lucas-Kanade optical flow which is a sparse optical flow algorithm. However, before that, it is necessary to get image features to be tracked. These are the input of the algorithm. In our implementation we used the Shi-Tomasi corner detection method implemented by the `cv2.goodFeaturesToTrack()` function.
The next step is to use the `cv2.calcOpticalFlowPyrLK()` function to compute the optical flow.

Dense vs Sparse Optical Flow

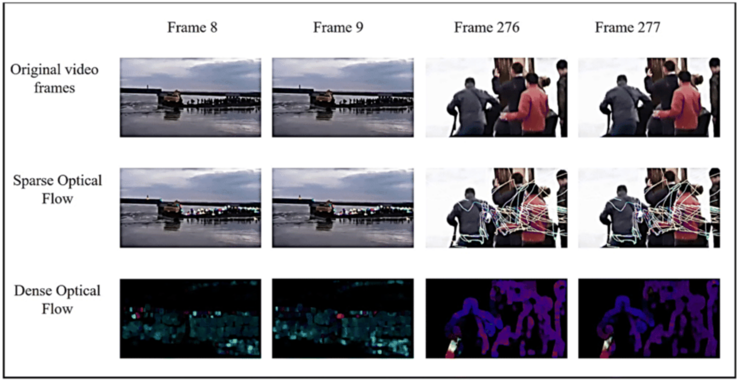


Figure: Comparison of optical Flows

Global Motion Estimation

After computing the optical flow, it is time to compute the transformations between frames. In our work, we have assumed that the motion model of the camera is **euclidean**, thus admitting only **translation and rotation**.
This transformation is obtained by estimation using matched features from both frames. OpenCV's `cv2.estimateAffinePartial2D()` function was used to compute such transformations. Moreover, it has the advantage of removing outliers using RANSAC.

Global Motion Estimation

Despite performing well with the provided input videos, this method has its own drawbacks as it may not be capable of detecting features between frames with low brightness.
The result of the global motion estimation is the triple (x, y, θ). To obtain the values from the estimation matrix T' we consider:

$$x = T'_{01} \tag{1}$$

$$y = T'_{02} \tag{2}$$

$$\theta = \arctan \frac{T'_{10}}{T'_{00}} \tag{3}$$

Motion Smoothing

The main goal of this stage is to remove the unwanted camera movement. Furthermore, we can describe the unwanted camera movement as high-frequency movement. A nice analogy is to think of the movement as being a signal, thus the solution to this problem is to apply a low-pass filter to remove the unwanted camera movement and keep the slow and steady movement.

Motion Smoothing

- The first step is to compute the trajectory of the camera in each of its transformation components.
- The transformation gives us the translation and rotation between two frames (points in time), or simply, the velocity of the camera.
- This means that, in order to obtain the camera trajectory, we need to integrate these points. This may be achieved by calculating the cumulative sum of each component over every frame.

In our solution we used a simple moving average filter with a customizable frame window

Motion compensation

After filtering the trajectories of each component it is time to apply the motion compensation on each frame. The smoothed transformations can be described by the following formula:

$$F_{smooth} = F_{original} + (J_{smooth} - J_{original}) \tag{4}$$

In the formula above, the F represents a transformation between consecutive frames and J the trajectory value at those frames.

Motion compensation

The following step consists in warping the frame to the right place using a new transformation matrix. In OpenCV, this is achieved by a call to `cv2.warpAffine()`. This function receives a matrix T that can be constructed from (x, y, θ) and has the following shape:

$$T = \begin{bmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \end{bmatrix} \tag{5}$$

The final step of the pipeline is the application of zoom to each frame. This happens for the sake of removing some of the black borders originated after the previous step, i.e., image warp.

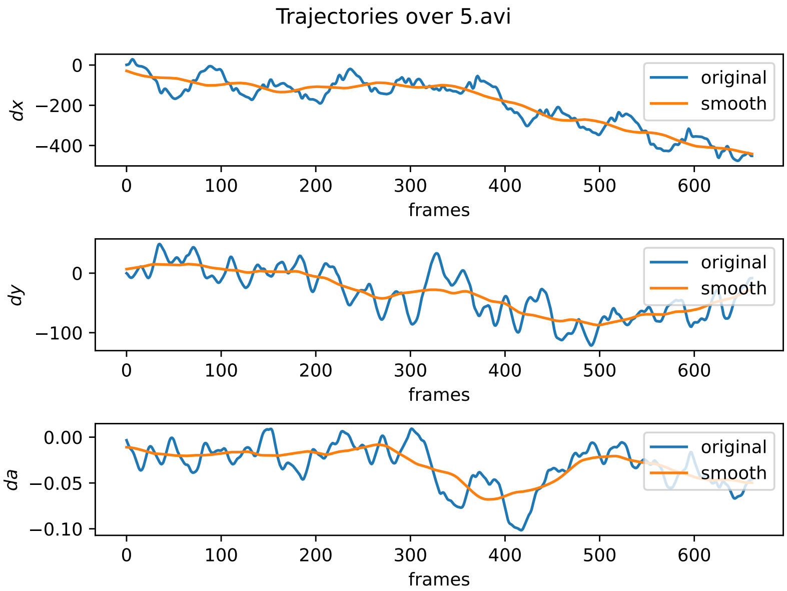
Results

The effectiveness of our video stabilization algorithm is demonstrated through the comparison of original and stabilized video frames. The smoothing of trajectories ensures that the stabilized video maintains natural motion while reducing undesired shaking.

Results

For analysis and visualization purposes, the original and smoothed trajectories can be plotted over the course of the video. This provides insights into the effectiveness of the stabilization algorithm.

Results



Trajectories over 5.avi
dx
dy
da
frames
original
smooth
0
−200
−400
0
−100
0.00
−0.05
−0.10
0
100
200
300
400
500
600

Results

Another relevant metric is the time taken by the program. Our implementation iterates through all video frames, thus it is expected that longer videos would take longer to process, which can be confirmed in Fig. 7. Given this, our solution processes, on average, a frame in 17 milliseconds running on an Apple M2 chip.

Results

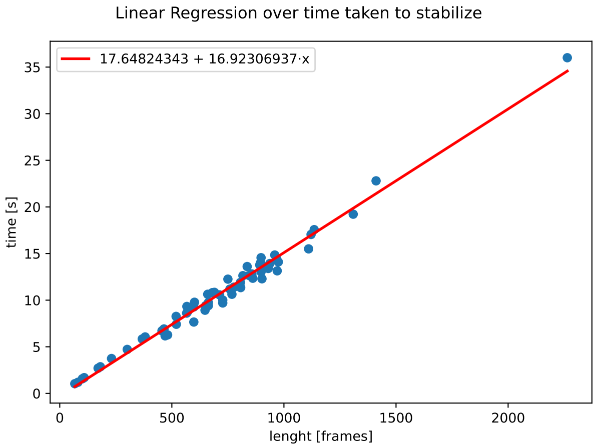


Figure: Line fitting between the amount of frames and the time to process. Each point represents a unstable video from the DeepStab dataset.

Problems with our solution

Our solution has some problems concerning videos where features are indistinguishable between consecutive frames.

- A simple way to overcome this may be using different and more advanced feature detection methods, e.g. ORB, or other optical flow methods.
- Another experiment worth giving a shot is to discard optical flow computation altogether and obtain transformations using the `cv2.findTransformECC()` function from OpenCV.

Problems with our solution

Another problem with our solution concerns the motion blur and wobbling. Some of the videos we have tested become significantly blurry and a wobbling effect is noticeable after applying stabilization. Although we tried to apply a Wiener filter to deblur, several issues emerged along the way. To the best of our knowledge, we did not find any implementation that was both efficient and able to operate on colored images.

Thank You For Your Attention !