Image stabilization

Guilherme Franco <xfranc01@stud.fit.vutbr.cz>
Miguel Rodrigues <xboave00@stud.fit.vutbr.cz>

January 2024

1 Introduction and Motivation

For this project, we were challenged to implement a video stabilizer. This is a computer vision task whose goal is to remove the unwanted camera movement from a video.

In the digital age, video stabilization is a very important, especially among videographers who wish to deliver the best video quality to their respective audiences. Other use-cases, can be found in medicine where stable video footage is relevant to conduct sensitive procedures.

In practice, there are two types of stabilization: mechanical (with gimbals) and digital (using software). Naturally, this work focus on the latter.

2 An overview on digital video stabilization

In this section, we will provide some background on the methods and approaches employed in digital video stabilization.

2.1 Classical methods

Digital video stabilization is usually built on top of the framework illustrated in Fig. 1: (i) motion estimation, (ii) motion compensation, and (iii) image warp.

The first step is establishing a motion model and estimating the global motion vectors. In the next step, the motion model removes high-frequency shaking to produce stable camera motion. The final step is image warp. Here

images are distorted according to the smoothed transformations produces in the previous step.

Furthermore, we may also distinguish between offline and online video stabilization. In the former, there is access to all video frames beforehand. In this environment, usually, all frames are processed before proceeding to the next step. Nonetheless, in online video stabilization that is not possible, as there is only access to previous frames.

In general, classical methods comprise both 2D, 3D, and hybrid (2.5D) approaches which rely on successful feature tracking. These methods are also significantly less demanding concerning their computational cost.

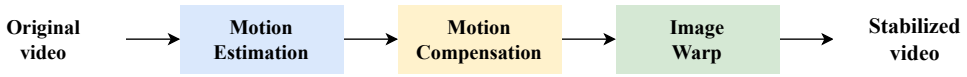


Figure 1: Classical framework for digital video stabilization [1].

2.2 Deep learning methods

With the increasing computation power and the recent developments of deep learning, methods based on convolution, neural networks (CNNs) have been recently proposed for digital video stabilization.

These methods, generally, try to compute the desired camera path by learning the transformations between frames. Several deep-learning based methods outperform some classical methods, however, the majority of them depend on stable and unstable video pairs, which may be incompatible with some use-cases for video stabilization.

Wang et al. [1] describes with great detail the topology of currently available deep learning methods.

3 Proposed 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.

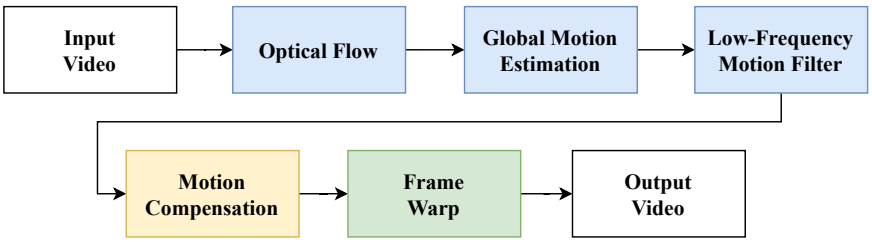


Figure 2: Pipeline of proposed solution.

3.1 Optical flow

The first stage of our pipeline is to compute the optical flow. Optical flow is the task of determining the motion between a pair of images. Moreover, optical flow algorithms can be split into two distinct categories: dense and sparse. As the name suggests, dense methods compute the pixel displacement in both axis for every frame pixel rather than a sample of points as in sparse methods. Naturally, dense methods are more computationally expensive than sparse methods but provide higher accuracy.

During the project we started by experimenting with PyTorch's implementation of RAFT [2]. 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 running on an Apple M2 chip, which was not desirable.

The solution here was to use the OpenCV's implementation of Lucas-Kanade optical flow [3] 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 [4] implemented by the `cv2.goodFeaturesToTrack()` function. The next step is to use the `cv2.calcOpticalFlowPyrLK()` function to compute the optical flow.

3.2 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.

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:

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

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

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

3.3 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.

In this context, 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 radius[1].

3.4 Motion compensation and border correction

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:

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

[1]However, in our tests we only used a radius of 50 frames.

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

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:

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

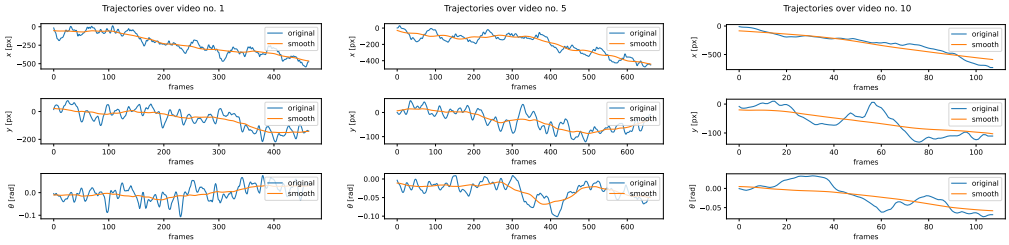
The final step of the pipeline is the application of zoom to each frame. This happens for the sake of removing some of the black borders originated after the previous step, i.e., image warp.

4 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.

To test our solution we used videos from the DeepStab [5] dataset. This dataset contains 60 different videos with the corresponding stable and unstable versions.

For analysis and visualization purposes, the original and smoothed trajectories are be plotted over the course of the video, as seen in Fig. 3. This provides insights into the effectiveness of the stabilization algorithm.



(a) Video no. 1 **(b)** Video no. 5 **(c)** Video no. 10

Figure 3: Trajectories obtained over different videos.

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. 4. Given this, our solution processes, on average, a frame in 17 milliseconds running on an Apple M2 chip.

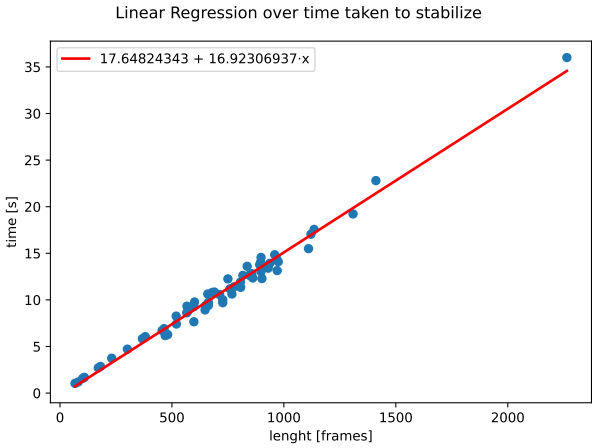


Figure 4: Line fitting between the amount of frames and the time to process. Each point represents a unstable video from the DeepStab dataset.

We collected some qualitative metrics as well. We showed 3 students from our university two sets of videos; one containing unstable videos and the other containing the same videos after being put through our solution. Every Single student stated that the videos after going through our solution were more stable, albeit a bit blurrier. Students were: João Coelho, João Ferreira, and Pedro Perdigão.

5 Conclusions and future work

In this work, we implemented a video stabilizer using OpenCV in Python. Even though the results were pretty decent there are some improvements and experiments that we would like to conduct.

5.1 Problems with the current solution

As stated in subsection 3.2, 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 [6], or other optical flow methods.

In our opinion, another experiment worth giving a shot is to discard optical flow computation altogether and obtain transformations using the `cv2.findTransformECC()` function from OpenCV. This function simply computes the transform between two images using the ECC criterion [7]. It is commonly used in the context of image registration.

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.

5.2 Advanced techniques

In the last years, video stabilization has seen several advancements. Although there is a big push on methods which employ deep neural architectures, we would like to point out a method developed by Google.

Google's technique [8] uses principles from cinematography to compute an ideal camera path. In essence, the technique tries to find an optimal solution for a convex linear programming problem. Furthermore, they assume that the camera path shall only be composed of constant (stationary camera), linear (constant velocity) or parabolic (constant acceleration) segments. Naturally, our solution is very primitive when compared with such technique.

References

[1] Y. Wang, Q. Huang, C. Jiang, J. Liu, M. Shang, and Z. Miao, "Video stabilization: A comprehensive survey," *Neurocomputing*, vol. 516, pp. 205–230, 2023.

[2] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *Computer Vision–ECCV 2020: 16th European*

Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16, pp. 402–419, Springer, 2020.

[3] J.-Y. Bouguet, "Pyramidal implementation of the lucas kanade feature tracker," 1999.

[4] J. Shi *et al.*, "Good features to track," in *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, pp. 593–600, IEEE, 1994.

[5] M. Wang, G. Yang, J. Lin, S. Zhang, A. Shamir, S. Lu, and S. Hu, "Deep online video stabilization with multi-grid warping transformation learning," *IEEE Transactions on Image Processing*, pp. 1–1, 2018.

[6] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*, pp. 2564–2571, Ieee, 2011.

[7] G. D. Evangelidis and E. Z. Psarakis, "Parametric image alignment using enhanced correlation coefficient maximization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 10, pp. 1858–1865, 2008.

[8] M. Grundmann, V. Kwatra, and I. Essa, "Auto-directed video stabilization with robust l1 optimal camera paths," in *CVPR 2011*, pp. 225–232, IEEE, 2011.

A Task and effort distribution

- Guilherme Franco was responsible for research, report writing, and programming (50% of effort).
- Miguel Rodrigues was responsible for research, report writing, and programming (50% of effort).

B Code repository

The code related to this project can be found at `https://github.com/mbrdg/video-stabilization`.