Video Stabilization using OpenCV in Python

Prof. Milind Rane
Department of Electronics Engineering,
Vishwakarma Institute of Technology, Pune

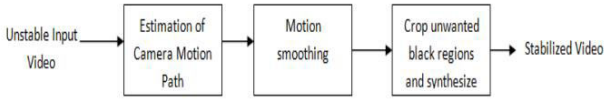
Prajwal Langde
Department of Electronics Engineering,
Vishwakarma Institute of Technology, Pune

Abstract - video stabilization is a technique to compensate for unwanted camera motion and produce a video that looks relatively stable. In this paper, an approach for video stabilization is proposed which works by estimating a trajectory built by calculating motion between continuous frames using the Shi-Tomasi Corner Detection and Optical Flow algorithms for the entire length of the video. The trajectory is then smoothed using a moving average to give a stabilized output. A smoothing radius is defined, which determines the smoothness of the resulting video. Automatically deciding this parameter's value is also discussed. The results of stabilization of the proposed approach are observed to be comparable with the state of the art YouTube stabilization.

Keywords—shaky video, point feature matching, stabilization.

I. INTRODUCTION

Video stabilization refers to a family of methods used to reduce the effect of camera motion on the final video. The motion of the camera would be a translation (i.e. movement in the x, y, and z-direction) or rotation (yaw, pitch, and roll). Most of the approaches discussed here witness a significant difference in performance on videos with an object of interest and ones without, with the latter facing a dip. In this paper, an approach based on [10] is investigated whose performance does not change for videos with objects of interest or ones without. Video stabilization generally consists of motion estimation where we use feature tracking method Harris-Stephens corner detection to identify the camera motion path, next process is motion smoothing which is used to remove undesired camera motions by using an appropriate motion model such as homograph camera motion path is smoothened. When performing video stabilization on the respective frames, due motion correction the frames may go out of bounds which creates black regions near the edges of the frames to avoid we go for cropping method. The general block diagram of video stabilization is shown.



II. DIFFERENT APPROACHES TO VIDEO STABILIZATION

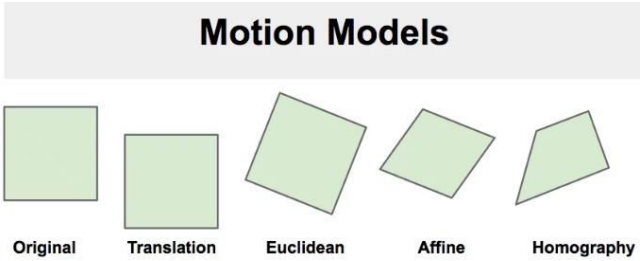
Video Stabilization approaches include mechanical, optical and digital stabilization methods. These are discussed briefly below:

1) Mechanical Video Stabilization: Mechanical image stabilization systems use the motion detected by special sensors like gyros and accelerometers to move the image sensor to compensate for the motion of the camera.

2) Optical Video Stabilization: In this method, instead of moving the entire camera, stabilization is achieved by moving parts of the lens. This method employs a moveable lens assembly that variably adjusts the path length of light as it travels through the camera's lens system.

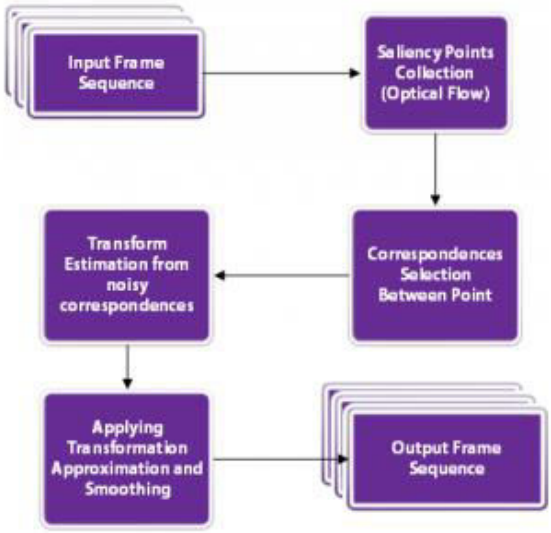
3) Digital Video Stabilization: This method does not require special sensors for estimating camera motion. There are three main steps — 1) motion estimation 2) motion smoothing, and 3) image composition. The transformation parameters between two consecutive frames are derived in the first stage. The second stage filters out unwanted motion and in the last stage the stabilized video is reconstructed.

We will learn a fast and robust implementation of a digital video stabilization algorithm in this post. It is based on a two-dimensional motion model where we apply a **Euclidean (a.k.a Similarity) transformation** incorporating translation, rotation, and scaling.



III. VIDEO STABILIZATION USING POINT FEATURE MATCHING

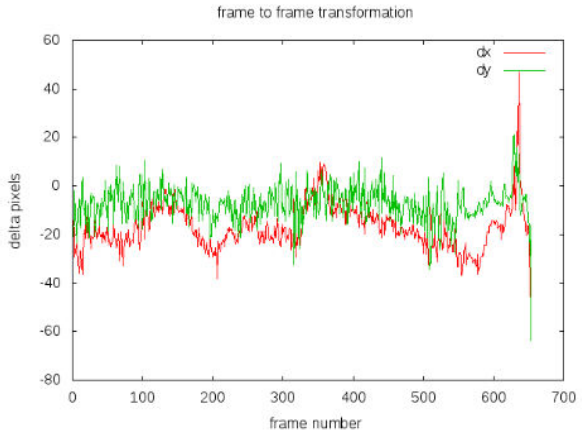
This method involves tracking a few feature points between two consecutive frames. The tracked features allow us to estimate the motion between frames and compensate for it.
The flowchart below shows the basic steps.



Block Diagram

Step 1

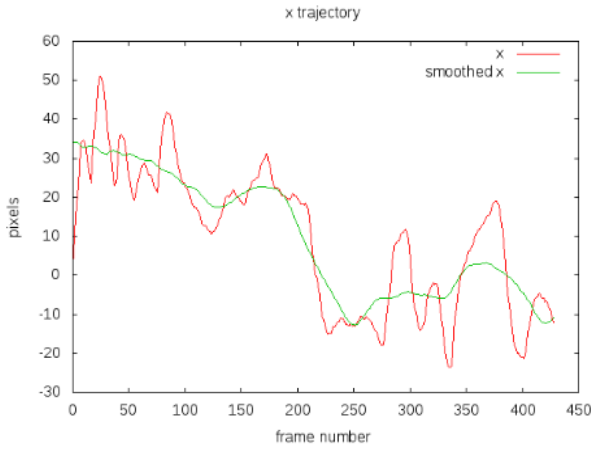
This graph shows the dx, dy transformation for previous to current frame, at each frame. Da (angle) has been omitted because it is not particularly interesting for this video since there is very little rotation. It can be seen it is quite a bumpy graph, which correlates with our observation of the video being shaky, though still orders of magnitude better than Hollywood's shaky cam effect.



Step 2

The trajectory is a rather abstract quantity that does not necessarily have a direct relationship to the motion induced by the camera. For a simple panning scene with static objects, it probably has a direct relationship with the absolute position of the image but for scenes with a forward moving camera, eg. on a car, then it's hard to see any.

The important thing is that the trajectory can be smoothed, even if it does not have any physical interpretation.



Step 3

This is the most crucial part of the algorithm. We will iterate over all the frames, and find the motion

between the current frame and the previous frame. It is not necessary to know the motion of each and every pixel. The Euclidean motion model requires that we know the motion of only 2 points in the two frames. However, in practice, it is a good idea to find the motion of 50-100 points, and then use them to robustly estimate the motion model.

3.1 Good Features to Track

The question now is what points should we choose for tracking. Keep in mind that tracking algorithms use a small patch around a point to track it. Such tracking algorithms suffer from the **aperture problem** as explained in the video below

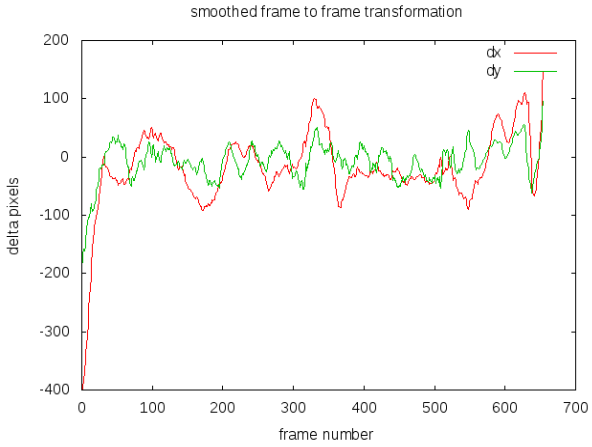
So, smooth regions are bad for tracking and textured regions with lots of corners are good. Fortunately, OpenCV has a fast feature detector that detects features that are ideal for tracking. It is called **goodFeaturesToTrack**.

3.2 Lucas-Kanade Optical Flow

Once we have found good features in the previous frame, we can track them in the next frame using an algorithm called **Lucas-Kanade Optical Flow** named after the inventors of the algorithm.
It is implemented using the function **calcOpticalFlowPyrLK** in OpenCV. In the name calcOpticalFlowPyrLK, **LK** stands for Lucas-Kanade, and **Pyr** stands for the pyramid. An image pyramid in computer vision is used to process an image at different scales (resolutions).
calcOpticalFlowPyrLK may not be able to calculate the motion of all the points because of a variety of reasons. For example, another object in the next frame could occlude the feature point in the current frame. Fortunately, as you will see in the code below, the **status** flag in **calcOpticalFlowPyrLK** can be used to filter out these values.

3.3 Estimate Motion

To recap, in step 3.1, we found good features to track in the previous frame. In step 3.2, we used optical flow to track the features. In other words, we found the location of the features in the current frame, and we already knew the location of the features in the previous frame. So we can use these two sets of points to find the rigid (Euclidean) transformation that maps the previous frame to the current frame. This is done using the function **estimateRigidTransform**.
Once we have estimated the motion, we can decompose it into x and y translation and rotation (angle). We store these values in an array so we can change them smoothly.
The code below goes over steps 3.1 to 3.3. Make sure to read the comments in the code to follow along.



Step 4

In the previous step, we estimated the motion between the frames and stored them in an array. We now need to find the trajectory of motion by cumulatively adding the differential motion estimated in the previous step.

Step 4.1 : Calculate trajectory

In this step, we will add up the motion between the frames to calculate the **trajectory**. Our ultimate goal is to smooth out this trajectory.

Step 4.2 : Calculate smooth trajectory

In the previous step, we calculated the trajectory of motion. Therefore, we have three curves that show how the motion (x, y, and angle) changes over time.
In this step, we will show how to smooth these three curves.
The easiest way to smooth any curve is to use a **moving average filter**. As the name suggests, a moving average filter replaces the value of a function at the point by the average of its neighbours defined by a window. Let us look at an example.

Let's say we have stored a curve in an array c, so the points on the curve are $c[0] \ldots c[n-1]$. Let f be the smooth curve we obtain by filtering c with a moving average filter of width 5.
The k^{th} element of this curve is calculated using

$$f[k] = \frac{c[k-2] + c[k-1] + c[k] + c[k+1] + c[k+2]}{5}$$

As you can see, the values of the smooth curve are the values of the noisy curve averaged over a small window. The figure below shows an example of the noisy curve on the left smoothed using a box filter of size 5 on the right.

Step 5
We are almost done. All we need to do now is to loop over the frames and apply the transforms we just calculated.
If we have a motion specified as (x, y, θ), the corresponding transformation matrix is given by.

IV. APPLICATIONS OF VIDEO STABILIZATION

The need for video stabilization spans many domains.
It is extremely important in consumer and professional videography. Therefore, many different mechanical, optical, and algorithmic solutions exist. Even in still image photography, stabilization can help take handheld pictures with long exposure times.
In medical diagnostic applications like endoscopy and colonoscopy, videos need to be stabilized to determine the exact location and width of the problem.
Similarly, in military applications, videos captured by aerial vehicles on a reconnaissance flight need to be stabilized for localization, navigation, target tracking, etc. The same applies to robotic applications.

V. CONCLUSIONS

Pros

1. This method provides good stability against low-frequency motion (*slower vibrations*).
2. This method has low memory consumption thereby ideal for embedded devices (*like Raspberry Pi*).
3. This method is good against zooming (scaling) jitter in the video.

Cons

1. This method performs poorly against high-frequency perturbations.
2. If there is a heavy motion blur, feature tracking will fail and the results would not be optimal.
3. This method is also not good with Rolling Shutter distortion.

VI. REFERENCES

[1] Battiato, S., Gallo, G., Puglisi, G., Scellato, S.: Sift features tracking for video
stabilization. In: Image Analysis and Processing, 2007. ICIAP 2007. 14th Interna-
tional Conference on. pp. 825–830. IEEE (2007)
[2] Grundmann, M., Kwatra, V., Essa, I.: Auto-directed video stabilization with robust
l1 optimal camera paths. In: Computer Vision and Pattern Recognition (CVPR),
2011 IEEE Conference on. pp. 225–232. IEEE (2011)
[3] Horn, B.K., Schunck, B.G.: Determining optical flow. Artificial intelligence 17(1-3),
185–203 (1981)
[4] Jianbo, S., Carlo, T.: Good features to track. In: Computer Vision and Pattern
Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Confer-
ence on. pp. 593–600. IEEE (1994)
[5] Lee, K.Y., Chuang, Y.Y., Chen, B.Y., Ouhyoung, M.: Video stabilization using
robust feature trajectories. In: Computer Vision, 2009 IEEE 12th International
Conference on. pp. 1397–1404. IEEE (2009)
[6] https://www.researchgate.net/publication/320732904_Video_Stabilization_Using_Sliding_Frame_Window
[7] http://ijirae.com/volumes/Vol3/iss4/30.APAE10102.pdf
[8] https://www.learnopencv.com/video-stabilization-using-point-feature-matching-in-opencv/