Video Stabilization

Alex Dahl

May 21, 2018

1 Introduction

Handheld cameras are commonplace nowadays. Most phones today have the capability to record high quality video without the need for expensive equipment. This has caused a significant increase in the amount of amateur and casual videos in the last decade. These videos are often produced without extensive planning and are filmed by directly holding the camera. This casual setup typically introduces a lot of shaking in the video which acts as a distractor from the content and, if bad enough, can make the contents indecipherable. Gimbals are tools that the camera is placed in which are able to keep it stabilized as well as smooth the motion. However these tools are ofter hundreds to thousands of dollars and are a physical item that has to be carried around, so they are often not as practical for the everyday user. Some instruments contain sensors that can be used to stabilize in that camera's hardware. The cost of these good quality sensors is dropping but it still adds a non-negligible amount to the price, sometimes adding up to 50 or 100 dollars.

Software can be used later as post-processing to stabilize the video. This removes the need for external physical devices. Both online and offline techniques exist. Online techniques allow the user to stabilized video as it is being recorded, however they can have trouble with scenes with complex motion. In such cases an offline approach is needed.

2 Previous Work

Many methods have been developed to stabilize video or image sequences. [4] tracked features between frames and computed a homography from them. [3] tracked SIFT features and showed that they make for good points to estimate motion. [9] proposed a full framework for stabilization and image inpainting. They used optical flow[15] to calculate local motion, feature-based method for global motion. The global motion was used to estimate the homography, while local motion was used for inpainting the missing areas. [7] Tracked features through time and formulated the motion estimation as a constrained non-linear least-squares optimization. Rather than working off of the recovered trajectories, [13] turned the trajectories into Bezier curves using the points as control points. An optimization of then done over those curves to fit a smooth model. These techniques work with a 2D motion model assumption, [8] recreated a 3D camera path from the video and used a least-squares optimization to do a spatially-varying warp from a desired camera path.

For online uses many people ([11], [6], [5]) have leveraged the use of Kalman filters push into real-time. [11] utilized an adaptive Kalman Filter with a 2D motion model to stabilize video, however the limited motion model lead to large accumulation errors. [6] generated short trajectories for each frame and used the filter to smooth those out, while [5] improved upon the filter and motion estimation. Other online techniques leverage the Particle filter[2] method to do it in real time [14].

Much more recently, [12] leveraged deep convolution neural networks to produce a stabilized video on the level of offline techniques but much faster. It also performs well where other feature matching techniques wouldn't due to blur or lighting conditions.

3 Method

3.1 Simple Method

The simplest method, which can be used as a baseline, is to match points between every consecutive pairs of frames and estimate the homography from those points. SIFT features are extracted for each frame. The features are matched together by minimizing the sum of squared distances between them and a modified version of RANSAC, called MSAC, is then used to estimate the homography.

If the scene is static then for each frame a transformation chain $T_t = \prod_{i=1}^{t} T_i$ is created and used to warp the frame to its stabilized position. If the scene is not static or the camera moves around, then the chain will produce large

erroneous results. Each transformation must be applied by itself to the corresponding frame at the cost of more noise. Before applying the final matrix, it is smoothed by the method in section 3.2 to remove high frequency content.

3.2 Averaging Transformations

Just finding the transformation that maps a frame to the previous one contains high frequency noise due to error in the feature selection and transformation estimation. In order to smooth it out and remove the high frequency content while keeping the motion relatively smooth a simple averaging is done. First, we need to change the structure of our transforms. The entirety of the affine transformation degrees of freedom are not needed. Shearing is not needed since you cannot create that affect by just moving a camera. The transformation can be decomposed into a scale s, an angle θ, and a translation Δ, and reformed as

$$T = \begin{bmatrix} s\cos(\theta) & -s\sin(\theta) & \Delta_x \\ s\sin(\theta) & s\cos(\theta) & \Delta_y \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

The first approach to smoothing this is to average the individual components using k neighbors on both sides of the timeline. For the transformation component this works fine, but for the rotation and scale the result is non-obvious. In order to do the averaging, we just take the components from the decomposition step and average those individually, then form the matrix from the averaged values. Alternatively any smooth curve could be fit to the parameters and then sampled at each frame to get the values.

3.3 Robust Feature Trajectories

Rather than computing the homography on a frame-by-frame basis, it is possible to leverage the fact that features travel through multiple frames. Lee et al.[7] leveraged this with an optimization to improve results. Assume that we have a set Γ of live trajectories and a set Π of retired trajectories. Then for each frame, the SIFT features are extracted. Those features are matched against the features of the trajectories currently in Γ. Features that match are used to extend that trajectory into the current frames, while unmatched features are used to start new trajectories. Trajectories that did not get a match are retired and moved to Π.

Not all matches will be good, so an extra pruning step is applied to weed out the bad ones. First, a trajectory with bad neighborhood consistency should be pruned so a concept of neighboring trajectories is needed. When the features are extracted, Delaunay triangulation is performed using the feature locations. We then use the definition that 2 trajectories are neighbors if they share an edge in the triangulation at any overlapping frame. A cost for trajectory i at frame t can then be defined as

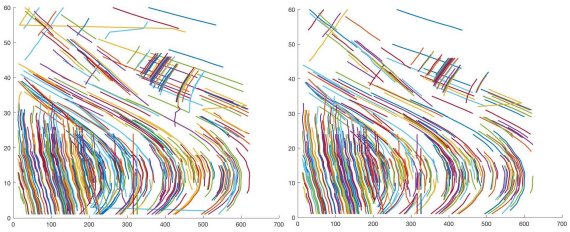


Figure 1: Comparison of valid trajectories with(left) and without(right) the weighting term. The cutoff used was 50 and 5000 respectively and both give around 500 good trajectories. Notice the left has more deviation. Trajectories shorter than 5 were discarded. X axis is x position, y axis is frame number.

$$||s(\mathbf{p}_i^{t-1}) - s(\mathbf{p}_i^t)||^2 + \lambda_t \sum_{j \in \mathbf{N}(i)} ||\mathbf{u}_i^t(\Phi^t) - \mathbf{u}_j^t(\Phi^t)||^2 \quad (2)$$

where s($\mathbf{p}$) is the SIFT descriptor for point $\mathbf{p}$, $\mathbf{u}_i^t = \mathbf{p}_i^t - \mathbf{p}_i^{t-1}$, and λ_t is a constant determining how import neighborhood consistency is. If this cost is above a threshold the trajectory is removed. The original paper included a weighting term in the summation, however I found that without that term I was more easily able to remove bad trajectories while keeping more of the good ones (shown in Figure 1).

3.4 Feature Trajectory Optimization

Not all trajectories are equal, some are very short or represent a small object moving in the foreground. Trajectories are thus weighted based on importance. Long trajectories typically come from the background or movements from objects of interest. Fast motion in the foreground leads to bad neighborhood consistency and are typically retired after only a few frames. Therefore, we want to emphasis longer trajectories over short ones. we also want trajectories that are emerging or fading out to have less importance. This is to help transition between groups of trajectories and well as reduce the affect of errors at the boundaries of trajectories. The weight of a trajectory can thus be defined as:

$$l_i^t = min(t - t_s(\xi_i), t_e(\xi_i) - t)$$

$$\tilde{w}_i^t = \frac{l_i^t}{\sum_{j \in \Pi(t)} G(|\mathbf{p}_j^t - \mathbf{p}_i^t|; \sigma_s)}$$

$$w_i^t = \frac{\tilde{w}_i^t}{\sum_{j \in \Pi(t)} \tilde{w}_i^t} \quad (3)$$

Where $t_s(\xi_i)$ and $t_e(\xi_i)$ are the start and end times for trajectory ξ_i, and $G(\cdot;\sigma)$ is a zero mean Gaussian with standard deviation σ.

With a set of good trajectories and weighting, a set of transformations $\mathbf{T} = \{T_t | 1 \leq t \leq n\}$ can be fitted. We want to consider both the roughness of the trajectories and the image quality loss after being transformed, thus we want to minimize the following objective function:

$$\arg\min_{\mathbf{T}} E_{roughness} + \lambda_d E_{degradation} \quad (4)$$

The roughness of a trajectory is related to its acceleration. The acceleration of a trajectory at frame t is defined as

$$\mathbf{a}_i^t(\mathbf{T}) = T_{t+1}\mathbf{p}_i^{t+1} - 2T_t\mathbf{p}_i^t + T_{t-1}\mathbf{p}_i^{t-1} \quad (5)$$

Now we can measure the roughness by summing up the values for all trajectories. We also want to make sure we measure the roughness with respect to the original frame rather than the transformed frame.

$$E_{roughness}(\mathbf{T}) = \sum_{\xi_i \in \Pi} \sum_{t=2}^{n-1} w_i^t ||T_t^{-1}\mathbf{a}_i^t(\mathbf{T})||^2 \quad (6)$$

$E_{degradation}$ is a measure of how the transform will reduce image quality and can be broken up into 3 parts: distortion, scaling, and amount of image uncovered.

$$E_{degradation}(\mathbf{T}) = \sum_{t=1}^{n} E_{distortion}(T_t) + \lambda_s E_{scale}(T_t) + \lambda_u E_{uncovered}(T_t) \quad (7)$$

$E_{distortion}$ is a measure of how the image gets distorted by the transformation. With our definition of the transforms with 4 degrees of freedom, this term is very simple. Translation and rotation do not have any affect on image quality, and scaling is accounted for with the E_{scale} term, so trivially $E_{distortion} = 0$. Large scaling factors results in very zoomed-in video. This is not ideal, as that is a rather large loss of quality and information, so large scaling is heavily penalized.

$$\mathtt{E}_{scale}(T_t) = \begin{cases} (s_t - 1)^4 & s_t > 1 \\ 0 & \text{otherwise}s \end{cases}$$

We also want to penalize the area that gets uncovered by the transformation. Calculating the exact area that gets uncovered is rather expensive, so an estimation is used.

$$\Psi(d) = \sqrt{d^2 + \delta^2} - \delta$$

$$c(v, e) = \begin{cases} len(e)\Psi(dist(v, e)) & \text{if } v \text{ is outside } e \\ 0 & \text{otherwise} \end{cases}$$

$$E_{uncovered}(T_t) = \sum_{v \in \mathcal{V}, e \in \mathcal{E}} c(T_t^{-1}v, e)^2 \quad (8)$$

where δ is a small number, typically set to 0.001, $len(e)$ is the length of edge e, and $dist(v, e)$ is the distance of point v to edge e. The optimization is solved using the Levenberg-Marquardt method. Rather than optimizing based on the full matrices, the variables are the 4 parameters for each frame. For an initial guess, the equivalent of a scaled identity transform is used, with the scale set to 1.01 and everything else to 0.

4 Results

Everything was implemented in MATLAB, using built-in functions for feature extraction. To speed up computation, SURF was used instead of SIFT for features. The equations are all equivalent and the features produced are very similar.

4.1 Transformation Smoothing

Figure 2 shows the x translation results when using 2 and 4 neighbors. The more neighbors are included the more smoothed it will be. There is a tradeoff because if the path is too smoothed, the video itself won't be smoothed, but too little and there is a lot of high frequency noise. Figure 3 shows the difference between the components when averaging the matrix elements and the components directly. When the initial error on the scale component is increased, the deviation of the resulting angle component increases linearly. Similarly when the angle error is increased, the deviation of the scale component increases linearly.

Notably in practice the error between the 2 methods is very small, not enough to cause any

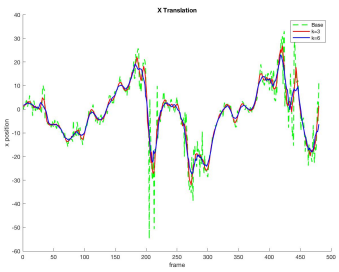


Figure 2: X translation for the component smoothed transformation using 6(k=3) and 12(k=6) neighbors.

visual differences. This is because the majority of the transformation is centered on the translation. When the camera is shaking most of the movement is translational rather than scaling or rotational. In the theme park example, the x and y components of translation ranged from -52.25—33 and -16.5—23.74 respectively, while scale and rotation only ranged between 0.9659—1.083 and -0.0541—0.0447. The maximum error $Er = ||S_c - S_m||_F$, where S_c is the averaged components matrix, S_m is the averaged matrix elements matrix, and $|| \cdot ||_F$ is the Frobenious norm, was 0.000885.

4.2 Feature Trajectories

Unfortunately the optimization for the feature trajectories has proved quite difficult. To help the optimization along I have taken the Jacobian of most of the optimization function with respect to the 4∗number of frames variables. The only part not included is the partial derivative of $E_{uncovered}$ as that function was too difficult for me to derive. The optimization functions already estimate the Jacobian using finite differencing, however this is expensive and can be error prone. I have verified that the provided Jacobian is close in accuracy to the objective function without the uncovered estimation using built-in function options to check them and that it is exponentially faster to compute.

While the scaling and angle parts of the optimization seem to do well, the translation part is lacking. Most of the values for translation are in the [-1,1] range, producing very little noticeable smoothing. I have currently found a set of non-default parameters of damping cutoff of 1e12, damping change value of 5, and relative step cutoff of 1e-7 that start to converge to good values. However the optimization takes around 16000 iterations to converge and it's a second and a half per iteration. While I am confident that I could work on tuning this to get good results, there is not enough available time as each test would take over 6.5 hours. I have also tried other optimization methods, such as MATLAB's interior-point, sqp, and active-set algorithms. With the exception of interior-point which immediately exits from local minima, these algorithms produced larger changes but were highly erroneous, often warping most of the image out of view.

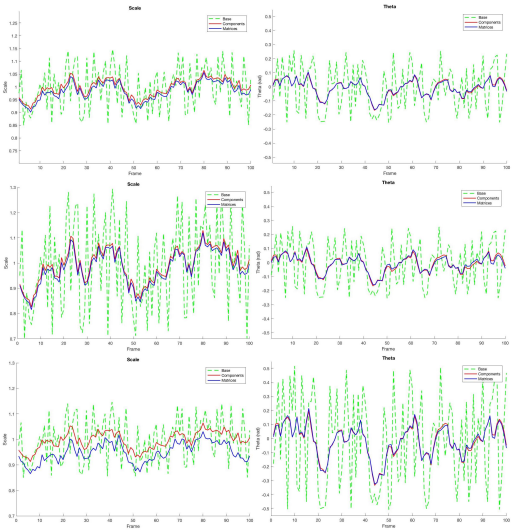
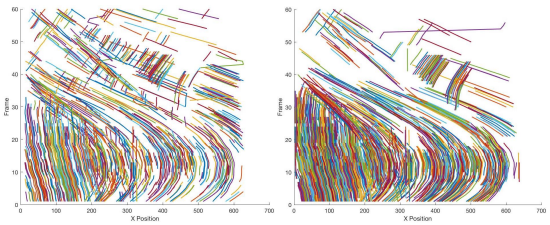


Figure 3: Random error is added to each component. Component is from averaging components, matrices is from averaging individual matrix elements. (Top) Base error added. (Middle) Error on the scale is doubled. (Bottom) Error on the angle is doubled.

For the trajectory selection I have also tried using KAZE[1] and MSER[10] features. MSER produced only a fraction of the number of features and did not yield enough good trajectories to be useful. KAZE performed rather well, with more good trajectories than SIFT. Figure 4 shows the resulting trajectory paths for comparison. KAZE had more longer trajectories with a low cost. The downside to KAZE is that it is much more expensive to calculate, taking around 7.5x longer.



(a) SURF Features (b) KAZE Features

Figure 4: Difference in good trajectories using SIFT and KAZE as features. Cutoffs were set to 5000 and 100 respectively, giving around 2000 trajectories for both. Trajectories shorter than 4 were discarded.

References

[1] Alcantarilla, P. F., Bartoli, A., and Davison, A. J. Kaze features. In *Computer Vision – ECCV 2012* (Berlin, Heidelberg, 2012), A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds., Springer Berlin Heidelberg, pp. 214–227.

[2] Arulampalam, M. S., Maskell, S., Gordon, N., and Clapp, T. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing 50*, 2 (Feb 2002), 174–188.

[3] Battiato, S., Gallo, G., Puglisi, G., and Scellato, S. Sift features tracking for video stabilization. In *14th International Conference on Image Analysis and Processing (ICIAP 2007)* (Sept 2007), pp. 825–830.

[4] Censi, A., Fusiello, A., and Roberto, V. Image stabilization by features tracking. In *Proceedings 10th International Conference on Image Analysis and Processing* (1999), pp. 665–667.

[5] Dong, J., and Liu, H. Video stabilization for strict real-time applications. *IEEE Transactions on Circuits and Systems for Video Technology 27*, 4 (April 2017), 716–724.

[6] Dong, J., Xia, Y., Yu, Q., Su, A., and Hou, W. Instantaneous video stabilization for unmanned aerial vehicles. *Journal of Electronic Imaging 23* (2014), 23 – 23 – 10.

[7] Lee, K.-Y., Chuang, Y.-Y., Chen, B.-Y., and Ouhyoung, M. Video stabilization using robust feature trajectories. In *2009 IEEE 12th International Conference on Computer Vision* (Sept 2009), pp. 1397–1404.

[8] Liu, F., Gleicher, M., Jin, H., and Agarwala, A. Content-preserving warps for 3d video stabilization. *ACM Trans. Graph. 28*, 3 (July 2009), 44:1–44:9.

[9] Matsushita, Y., Ofek, E., Ge, W., Tang, X., and Shum, H.-Y. Full-frame video stabilization with motion inpainting. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 7 (July 2006), 1150–1163.

[10] Nistér, D., and Stewénius, H. Linear time maximally stable extremal regions. In *Computer Vision – ECCV 2008* (Berlin, Heidelberg, 2008), D. Forsyth, P. Torr, and A. Zisserman, Eds., Springer Berlin Heidelberg, pp. 183–196.

[11] Wang, C., h. Kim, J., y. Byun, K., Ni, J., and j. Ko, S. Robust digital image stabilization using the kalman filter. *IEEE Transactions on Consumer Electronics 55*, 1 (February 2009), 6–14.

[12] Wang, M., Yang, G., Lin, J., Shamir, A., Zhang, S., Lu, S., and Hu, S. Deep online video stabilization. *CoRR abs/1802.08091* (2018).

[13] Wang, Y. S., Liu, F., Hsu, P. S., and Lee, T. Y. Spatially and temporally optimized video stabilization. *IEEE Transactions on Visualization and Computer Graphics 19*, 8 (Aug 2013), 1354–1361.

[14] Yang, J., Schonfeld, D., Chen, C., and Mohamed, M. Online video stabilization based on particle filters. In *2006 International Conference on Image Processing* (Oct 2006), pp. 1545–1548.

[15] yves Bouguet, J. Pyramidal implementation of the lucas kanade feature tracker. *Intel Corporation, Microprocessor Research Labs* (2000).