EECS 442 Final Project

Implementing Auto-Directed Video Stabilization with Robust L1 Optimal Camera Paths

Eric Yang, Sam Kim, Ethan Hong, Albert Lo

I. INTRODUCTION

In today's society, many people own smartphones with video-capturing capabilities that give the user the ability to record the world around them. As phone cameras become more technologically advanced, the quality of recorded videos become comparable to those recorded by professional cinematographers. However, these cinematographers have access to tools that help stabilize video recording, something that doesn't happen with everyday recordings. For example, amateur videos on YouTube are often very shaky due to hand instability, vertical vibrations from walking, or other physical movement. Our group has chosen to implement a video stabilization method that attempts to remove these common sources of noise. We reference the research paper, "Auto-Directed Video Stabilization with Robust L1 Optimal Camera Paths" [1], in our attempt to construct stable videos from amateur recordings.

II. RELATED WORK

We found various literature on this topic, including [5], which discusses stabilization through the use of techniques that have been introduced in class, such as interpolation of neighboring frames to stabilize the video and allow for a smoother viewing experience. However, most current research techniques employed in video stabilization follows the general conventions of (1) estimating the camera path and (2) formulating and solving for an optimization problem. Other works include [3], where Liu et al. formulates a matrix factorization problem and tries to solve for the original camera path, a high-dimensional matrix, as the product of two low-dimensional matrices. Ultimately, we decided to mostly base our implementation on [1] because it utilizes concepts that we discussed in class like SIFT and forward differencing but also introduces new topics such as linear programming. In particular, it also strives for the goals set in [6], which aims to mimic cinematography techniques in providing a stabilized video that is appealingly smooth but also comparable to cinematography techniques used in modern movies.

III. METHODS

In this section, we discuss the steps in video stabilization, which are: (1) estimating the camera path, (2) smoothing the estimated camera path, and (3) reconstructing the video based on the smoothed path.

III.A. Estimating Camera Path

III.A.1. Keypoint Matching

The first step in video stabilization is to estimate the original camera path. To do this, we computed keypoint descriptors using the SIFT algorithm between pairwise adjacent frames and found similar points using Nearest Neighbor in Figure 1. The SIFT algorithm produces key points in the current frame f_t, and the next frame f_{t+1}. In addition, we used nearest neighbor threshold as specified by Lowe [2] to filter out noisy matches. More specifically, for every keypoint in frame f_t, we find the closest 2 neighbors in f_{t+1} and compare the distances of its top 2 closest neighbors. We disregard the point if the ratio of distance of the closest neighbor divided by the distance of the next closest neighbor is greater than 0.80. In other words, if the 2 closest neighbors are really close to each other, then it becomes ambiguous which match is correct and we throw the match away to decrease noise in the output.

III.B.2. 2D Affine Transformation Estimation

Once these correspondences have been computed, we estimate the affine transformation. For this, we use the Random Sample Consensus (RANSAC) algorithm. At first, we tried formulating a Least Squares Regression Problem, but the model did not accurately model the data well as it assumes that the error

terms are normally distributed and thus calculates weights based on outliers. To address this issue, we use RANSAC, which is an iterative learning model that is robust against outliers. In this algorithm, we randomly sample a subset of the data and fit a model to the subset until it satisfies an error threshold. We compute the affine transformation between pairs of adjacent frames to compute F_t and can compute the estimated path $C_t = F_1 F_2 ... F_t$ [1].

III.B. Smoothing the Estimated Camera Path

The second step in the video stabilization method that we implemented is smoothing the estimated camera path into a much steadier video that mimics the use of either video stabilization software or devices such as electronic gimbals to steady the camera. In order to stabilize the video, we attempted to calculate a B_t matrix such that $P_t = C_t B_t$, where C_t is the estimated camera path and P_t is the computed optimal path. In order to do this, we used Algorithm 1 outlined in [1], which is displayed in Figure 1.

Algorithm 1: Summarized LP for the optimal camera path.

Input: Frame pair transforms $F_t,\ t = 1..n$
Output: Optimal camera path $P_t = C_t B_t = C_t A(p_t)$

$$
\begin{aligned}
&\text{Minimize} && c^T e \\
&\text{w.r.t.} && p = (p_1, ..., p_n) \\
&\text{where} && e = (e^1, e^2, e^3), e^i = (e^i_1, ..., e^i_n) \\
& && c = (w_1, w_2, w_3) \\
&\text{subject to} && \\
&\text{smoothness} && \begin{cases} -e^1_t \le R_t(p) \le e^1_t \\ -e^2_t \le R_{t+1}(p) - R_t(p) \le e^2_t \\ -e^3_t \le R_{t+2}(p) - 2R_{t+1}(p) + R_t(p) \le e^3_t \\ e^i_t \ge 0 \end{cases} \\
&\text{proximity} && \text{lb} \le U p_t \le \text{ub} \\
&\text{inclusion} && (0,0)^T \le \text{CR}_t p_t \le (w,h)^T
\end{aligned}
$$

Figure 1. Linear programming algorithm [1]

We can create a linear programming problem that seeks to compute the optimal intended path of each frame by utilizing the frame pair transforms F_t that were calculated in section III.A. We do this by first defining a vectorized affine transformation $p_t = (dx_t, dy_t, a_t, b_t, c_t, d_t)$ for each pair of adjacent frames and constraining the parameters (B_t is the matrix formulation of p_t). More specifically, the parameters are constrained such that they take into account the smoothness of the path, its proximity to the original path in terms of zoom and rotation, and that the cropped frame would remain within the dimensions of the original frame. The various constraints laid out in the smoothness constraints use forward differencing to account for the edges in the path calculated in section III.A and create smoother transitions throughout the video by penalizing for non-constant changes in velocity and acceleration. The proximity constraints limit the range of values for the degrees of freedom (DoF) that account for zoom and rotation in the affine transformation. By doing so, it prevents the reconstructed video from being warped too much from the original.

III.C. Reconstructing the Video Based on the Smoothed Path

The optimal camera path transform, B_t, is then used to transform each original frame to its smoothed position. This transformation is in the form of an affine transform. However, when this happens, the image may be affinely shifted, rotated, and scaled. Since each frame is transformed differently, the video will have inconsistent boundaries. Thus, we first crop the original image and apply the smoothing transform. Then, we estimate an reconstruction transform from this cropped, smoothed image to the

original video's domain, R_t. The reconstruction transform is estimated using the four corners between the cropped image and corner coordinates of the original frame. This gives us a simple result that stabilizes the center of the video. This concept is expanded in [1] to do auto-directed stabilization by adding saliency constraints, which can keep a specified point within the crop, like a face detector.

IV. RESULTS

IV.A. Estimating Camera Path

In Figure 2, shown below, we depict key point correspondences using SIFT descriptors and nearest neighbor similarity. We set the threshold at 0.8 as described in Lowe's paper [2]. By analyzing the pairwise adjacent frames for the video, we can see that this algorithm produces robust correspondences that we can use to estimate geometric transformations. Some results obtained between a pair of adjacent frames in shown in Figure 2 below.

Figure 2. SIFT similarity with nearest neighbor thresholding

In Figure 3 shown below, we attempt to plot the estimated camera path of a sample video. The vertical axis marks the path vertical or horizontal path at a given frame. The raw estimated camera (blue) path is not smooth and has regions of rapid fluctuation, which is evident in the video by sudden jerking. The optimized camera path by the algorithm described smooths the path (red), but our implementation of optimizing and visualizing this is not perfect.

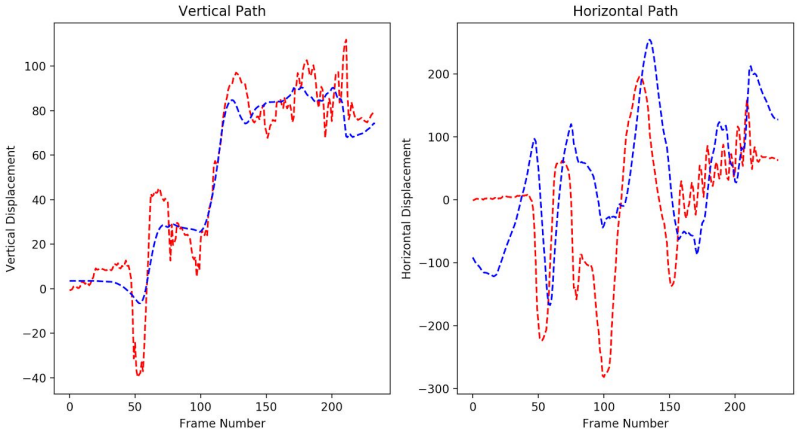

Figure 3. Camera original estimated path and optimized path (r=original; b=optimized)

V. DISCUSSION

We implemented a video-stabilization algorithm that can estimate the original camera path, smooth it to create a new optimal path, and then apply the new path to reconstruct a stable video. Our reconstructed video, however, is not perfect. It is subject to wobble effects and has a hard time focusing

on the main subject of the video. In [1], the authors discuss further techniques in handling wobble effects and focusing the video. We were also unable to achieve auto-directed stabilization through saliency constraints due to its complexity. We have, however, achieved decent stabilization for videos moving about a centered object, like holding a shaky camera in place or tracking a centered object.

An issue we ran into was with the plotting of the original and optimized path. Looking at the reconstructed video, we saw that there was a discrepancy between the reconstructed video and the plotted path, which led us to think that there was something we misunderstood about the representation of the path of the camera.

Moreover, we ran into various issues when trying to solve for the linear programming problem that did not allow us to try these additional methods. More specifically, none of us had any experience solving linear programming problems and we were unsure of which Python packages to use. As a result, we ended up writing 3 different implementations based on PuLP and CVXOPT before settling on the best result [7, 8, 9]. We, however, believe that we still have kinks in the implementation to hammer out. More specifically, we originally formulated the problem of estimating the camera path between adjacent frames as least-squares regression problem. This led to a very noisy path that did not smooth well and thus did not produce a stable video. We saw the greatest improvement occur when we switched to using the RANSAC algorithm, which gave us a vastly smoother path. We believe that changes like these, especially in the preprocessing stage, will improve the output of our implementation. Thus, we would like to improve on both our path estimation and path optimization aspects of the project in the future to produce more stable results.

VI. REFERENCES

[1] M. Grundmann, V. Kwatra, I. Essa. (2011). "Auto-directed Video Stabilization with Robust L1 Optimal Camera Paths." *IEEE.* (Online article)
https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/37041.pdf
[2] D. Lowe. (2004). "Distinctive Image Features from Scale-Invariant Keypoints." *International Journal of Computer Vision.* (Online article).
https://link-springer-com.proxy.lib.umich.edu/article/10.1023%2FB%3AVISI.0000029664.99615.94
[3] F. Liu, M. Gleicher, J. Wang, H. Jin, A. Agarwala. (2011). "Subspace Video Stabilization." *ACM Transactions on Graphics.* (Online article). https://dl.acm.org/citation.cfm?id=1899408
[4] M. Grundmann, V. Kwatra, I. Essa. (2012). "Video Stabilization on Youtube." *Google Research Blog.* (Online article).
https://research.googleblog.com/2012/05/video-stabilization-on-youtube.html
[5] Y. Matsushita, E. Ofek, W. Ge, X. Tang, and H.-Y. (2006) Shum. "Full-frame video stabilization with motion inpainting." *IEEE Trans. Pattern Anal. Mach. Intell.* (Online article).
http://ieeexplore.ieee.org/document/1634345/. p. 225, 231
[6] M. L. Gleicher, F. Liu. (2008). "Re-cinematography: Improving the camerawork of casual video." *ACM Trans. Mult. Comput. Commun*. *Appl*. (Online article).
https://pdfs.semanticscholar.org/230b/b1fae2c7b498b18b37984f1b03cc31476e65.pdf. p. 225, 226
[7] S. Mitchell et al. (2011). "PuLP". *Python Hosted.* (Online document). https://pythonhosted.org/PuLP/
[8] M. Andersen, J. Dahl, L. Vandenberghe. (2016). "CVXOPT." (Online document). http://cvxopt.org/
[9] Michael Grant and Stephen Boyd. (2013) CVX: Matlab software for disciplined convex programming, version 2.0 beta. (Online article). http://cvxr.com/cvx.
[10] M. Grant, S. Boyd. (2008). Graph implementations for nonsmooth convex programs, Recent Advances in Learning and Control. (Lecture notes). http://stanford.edu/~boyd/papers/graph_dcp.html.