Project 2: Video Stabilization

William Braga
CS6475 Spring 2021
wbraga3@gatech.edu

I. Link to Results

https://gatech.box.com/s/6mll6dvbyaz783z81xui83tmyvjot69b

II. Project Goals

A. Original Scope

From the outset, my first defined goals were to just read the paper, understand the process behind it, and then to replicate it as close as possible. From my understanding of the project assignment and paper, my scope, on a high-level, then became the following: (1) track the relative camera motion between each pair of frames based on strong image features, (2) derive the absolute motion by accumulating the relative motions, (3) define the smoothing, proximity, and inclusion constraints as per the paper, (4) solve the linear program using those constraints, (5) calculate the smoothed path using the smoothing homography from the linear program, (6) plot the original camera path and the smoothed path, (7) write two video files modifying the input, one with an overlaid crop window and one that is cropped to fit that window, (8) add saliency constraints to ensure the crop window is encompassing important features of the video.

B. Scope Changes

Yes. An additional consideration was needed to account for batching the frames when solving the program, since the videos were to large to process as one unit. Also, I had to drop the saliency portion of my scope due to time constraints from various other issues I faced.

III. Project Development

A. Development Process

The first day I started coding for this project, I simply wanted to be able to read in a video as a collection of stills, do some sort of processing on them, and write out a new video. I had some issues with the OpenCV documentation (shocking!), but I was able to read in the provided skating video and max out the blue channel in each frame to ensure my I/O portion of the pipeline was working.

Next, I set out to do the feature tracking. My first implementation was homography based. I was trying to use ORB and BFMatcher like in the panorama assignment, but I found my results to not match the motion in the video. I spent quite a bit of time trying to tune and get this approach to work, but I finally settled for the optical-flow method I found through Piazza. After some small tuning of the feature and lucas-kanade parameters, the motion was matching the skating and cityscape videos.

The crux of my work and time lay in the linear programming. Like others, I used the cvxpy library. On my first jab at it, I setup the constraints in the main loop where I was solving for the homographies, only starting when t = 2 so that I could reference the two prior frames. While writing the constraints, I had to keep rechecking the confusing notation used by the paper, especially in reference to indexing into variables. The paper uses variables indexed up to t+3, which obviously I had not defined yet in my loop currently solving for t. I also needed to learn how to handle the cvxpy variables. Since they were objects with no values yet, numpy functions were not applicable to them. I had to learn the cvxpy function variants and quirks about them.

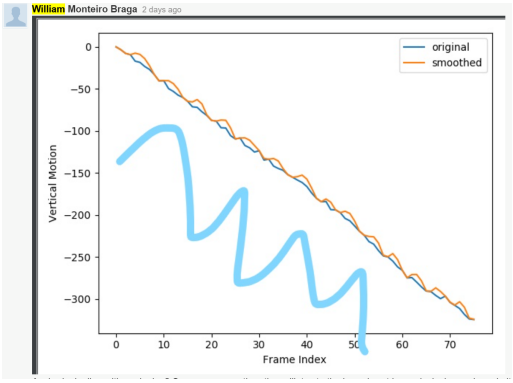
At first, I was solving the LP at each iteration of the loop. This resulted in a bizarre oscillating 'smoothed' path (see Interim Fig 1). The path seemed to repeatedly touch the camera path and bounce back as far as the proximity/inclusion constraints would allow. Thinking about the LP more, I thought it did not make much sense to only consider one pair of frames for the constraint, so I decided to attempt solving the LP once after the loop, concatenating the constraints at each iteration. This gave me an immediate positive feedback. Unfortunately, when trying to scale to the entirety of the skating video, the solver failed. This left me with needing to implement a batched approach, which compounded my indexing confusion from before (see Interim).

The last part of the pipeline I worked on was the actual cropping of the video. My first endeavors were doomed - I had a critical misunderstanding of the crop window. Having not paid enough attention, I had mixed up the math for the normal crop and the crop with saliency. I did not realize they had separate approaches. In my early implementations, I was trying to warp the entire image to fit a fixed crop window, but then finally realized that my crop window was meant to be variable and not the image.

Throughout the project, I faced several more minor bugs like frequently mixing x and y with respect to OpenCV's functions, extraneous or missing inversions of the homography matrix, and indexing/shape issues. The ones described above, however, detail the more critical hurdles I had to face due to misunderstandings of important project details or lack of knowledge in certain areas. They make up the major narrative of my learning to make the stabilizing pipeline.

B. Interim Results

While writing the LP solver - once I had batching implemented, I had to deal with many insane interim results. My graphs were all over the place. Broadly, they could be categorized in three groups: smoothed path seems misaligned/independent of camera motion, smoothed path and camera motion are the same, smoothed path unnecessarily jerks to a new position. When my path seemed mostly independent to the camera motion, that usually suggested an error in my constraints. One related issue I had to face with these ones is which direction to look when smoothing. At some point, I had tried looking 'forward' to new frames when minimizing the derivatives, which is the direction the paper notation suggests. Later on, I changed it back to the more intuitive backwards reference and that lent me better results.
When debugging smoothed paths that overlaid the camera paths, I found that my LP solver was often silently failing to produce results. Usually, the issue was an unbounded constraint that made my homography default to the identity matrix. These were often caused by my indexing issues. When implementing batching, I had mistakenly thought I needed to limit my variables to the size of the batch and reinstantiate them after solving (instead of making it the number of frames large). The overhead introduced made my code much more susceptible to going out-of-bounds or failing to account for all my variables.
The jerking is unfortunately a problem that I was not able to fully solve. The cause is mostly, if not wholly, attributed to the batch size. The beginning of each batch slingshots the homography since the old constraints are gone. My solution was to always retain the previous batch's constraints (throwing away those that were two or more timesteps back). This improved its performance on some videos but the issue is still present and very glaring (see fig 3c, fig 4b/c).



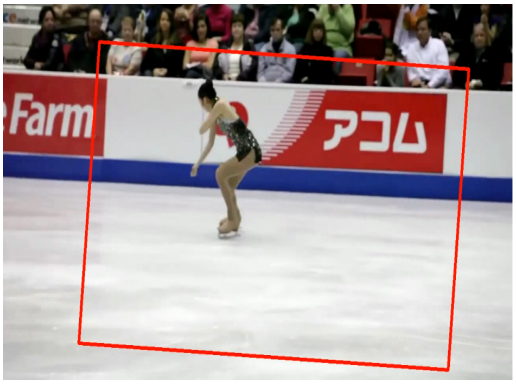
(a) Oscillating Failed Result - Piazza

Fig. 1: Interim Result

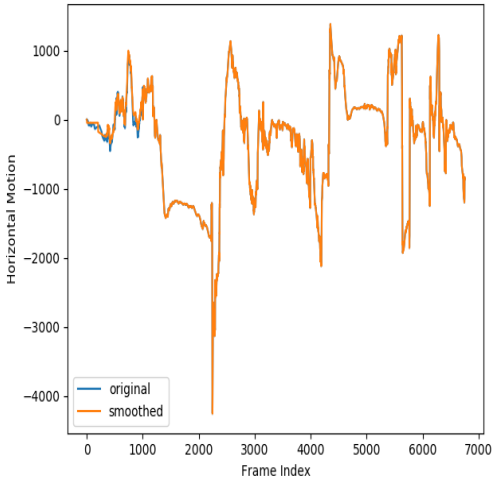
C. Discussion

Briefly on saliency, with more time, I would have liked to try some different methods of adding more weight to certain features like face detection or background/foreground separation.
On the main LP pipeline, I am glad that I was able to produce a coherent output video. Many of my earlier attempts ended with the image twirling about and eventually leaving the frame entirely. The way the videos were cropped generally shows an intention on attempting to keep relevant content in focus, albeit not always in a smooth manner. I am also happy with all the progress I made in the smoothing and batching. I expected that my end result may not have any deviation or coherence to the camera motion, and I am pleased to see some middle ground has been stuck.
Now, onto what is not finished on the main pipeline... The videos are obviously clunky. I think intention (by the LP to smooth) is there but not implemented correctly. I believe this is a combination of another constraint issue I could not find and potentially an error with the actual cropping. The batch jerk is particularly egregious but my attempts to mitigate it were not successful enough apart from the test skyscape video. This is definitely where I would want to invest more time and effort. I really obsessed over these results, and although I am glad to see the tangible progress I made, I can't help but be disheartened that I was not able to really figure out where my code went wrong.
If I were to redo the project, I would have made some key differences. Although I put an effort to comment, refactor, and abstract out code, I would emphasize it even more. It surprised me to see how quickly my code turned into entwined, soggy spaghetti. One reason for this was my trying to strictly abide by the notation in the paper. On a redo, I would ditch the notation for descriptive names. In terms of testing, I would try again to use pickle. I made a brief effort to use it to store the frames read in by OpenCV, but it ran out of memory. While debugging, it was to tempting to just run my code after each modification hoping for instant gratification, when really I should have laid some more groundwork for faster runtimes and smarter unit testing.

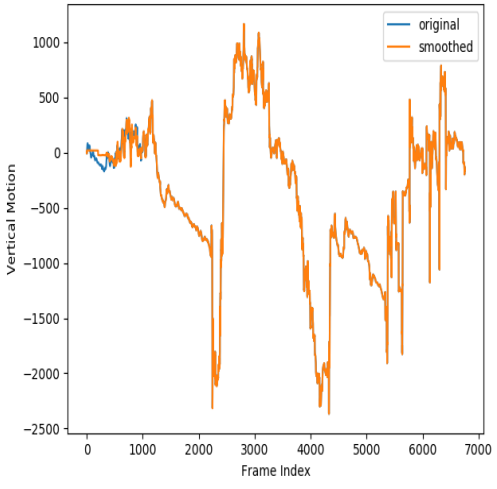
IV. Results



(a) Sample Video Optimal Crop Window

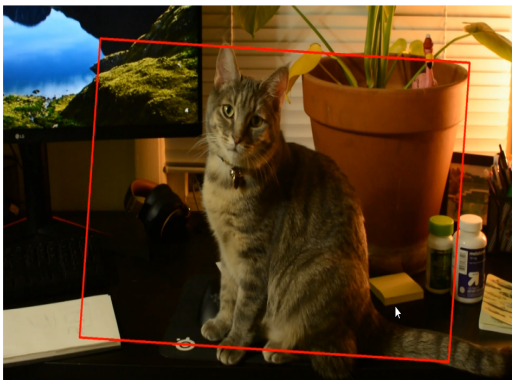


(b) Motion in x

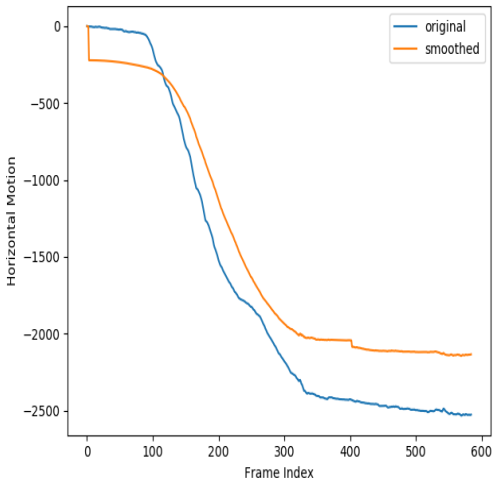


(c) Motion in y

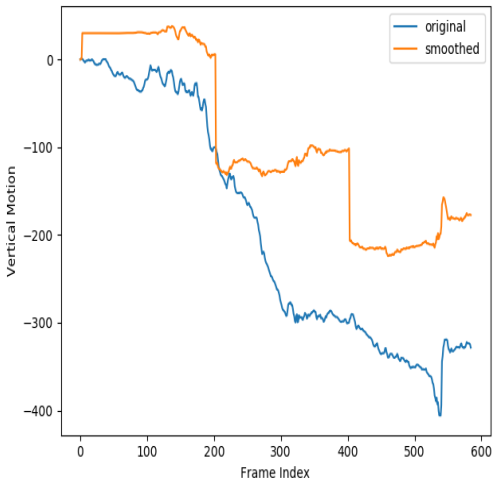
Fig. 2: Sample Video Results



(a) Optimal Crop Window

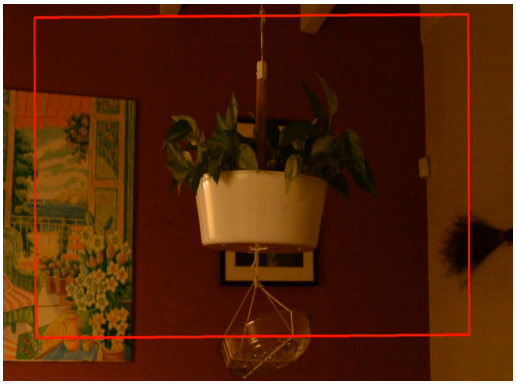


(b) Motion in x

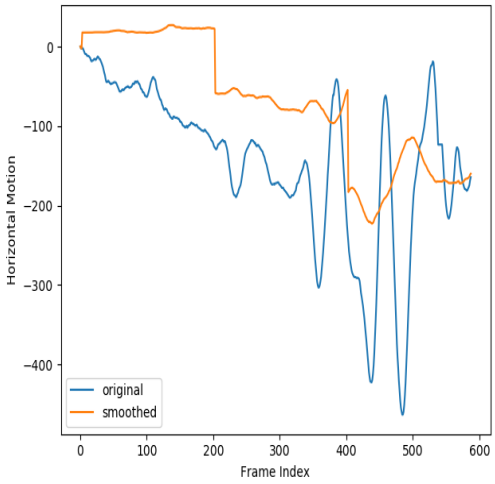


(c) Motion in y

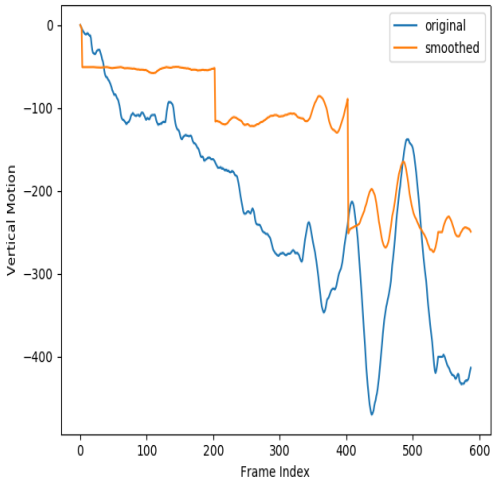
Fig. 3: Original Video 1 Results



(a) Optimal Crop Window



(b) Motion in x



(c) Motion in y

Fig. 4: Original Video 2 Results

V. Code Discussion

My image I/O is done through cv2.VideoCapture. I use read() to take in each frame of the video and write using cv2.VideoWriter to link stills together into a full video. My motion plots are generated using matplotlib. I also use os functions and filename processing lines from previous assignments To get the relative motion homographies, I use openCV's optical flow pipeline. First, I call cv2.goodFeaturesToTrack() on my first image to find pixels with strong corner values. I pass feature parameters specifying how to conduct the search. Next, cv2.calcOpticalFlowPyrLK uses the Lucas-Kanade algorithm to find matching features in the second image. It also takes parameters including the window size. Optical flow returns both the points and the status of them. Points can be filtered out as outliers if status is not 1 at that index. Finally, the function cv2.estimateAffinePartial2D creates an affine matrix specifying the transformation between the second and first image. I concatenate a [0, 0, 1] third row to transform it into a homography matrix and multiply it to my accumulated camera motion matrix.

The linear programming library I used is cvxpy (functions denoted cp.*). I define my parametized homography p and slack variables e as cp.Variable objects. This means that their value is still unknown, but they can be placeholders in operations with other known variables/matrices. They can also be used as placeholders in inequality statements, essential when it comes to defining the constraints. When the LP is solved, their values can be retrieved. When defining the constraints, I alternate between the parametized and matrix form of the homography. To make the matrix, I index into p and use the cp.bmat() function. This defines a matrix with To compare my computed R_t values to the slack e variables, I call cp.vec() on R_t. I use the same functions for the corners in my inclusion constraints. I then use cp.Minimize() to define my objective object and plug that and my constraints into a cp.Problem() object. Finally, solve() can be called on the problem to perform the convex optimization and solve for the variables.

My stabilize function only uses two methods. It invokes cv2.perspectiveTransform() to find the corners of the cropping rectangle after warping and cv2.warpPerspective() to actually use the homography to transform the image into the crop bounds. I call warpPerspective() with a canvas size equal to the cropping rectangle. To make the red crop lines, I just call cv2.line() with the corners.

VI. Computational Pipeline

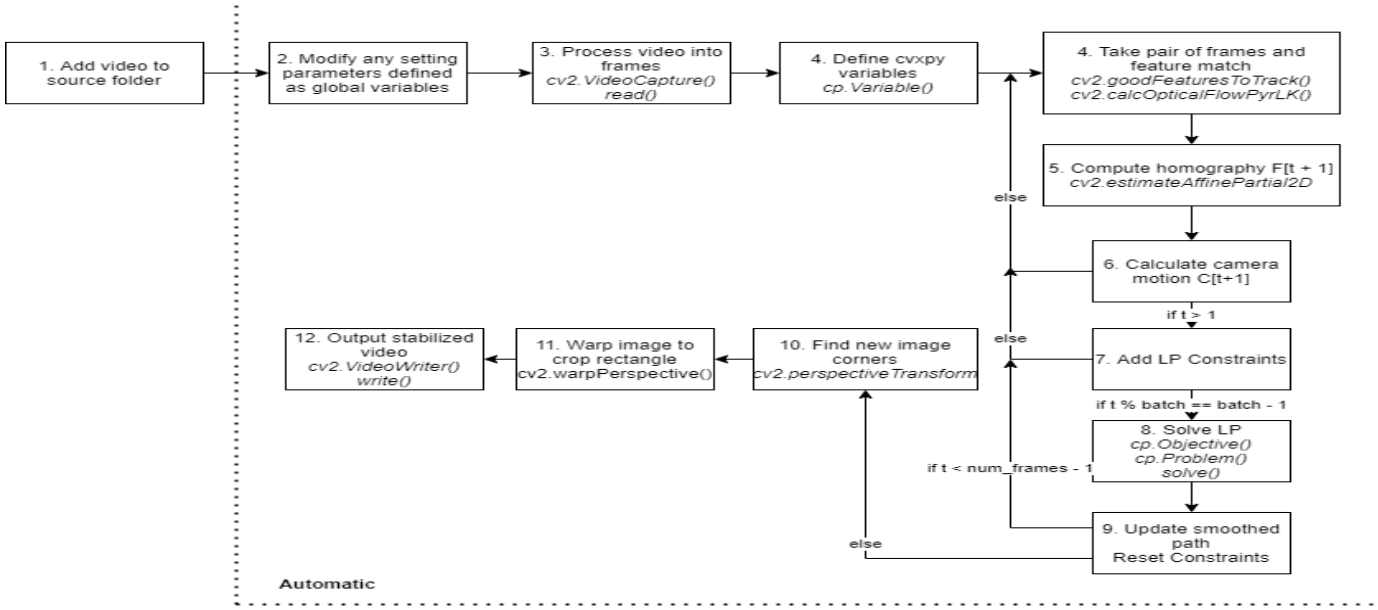


Fig. 5: Computational Pipeline

REFERENCES

[1] Matthias Grundmann, et al., “Auto-Directed Video Stabilization with Robust L1 Optimal Camera Paths” https://www.cc.gatech.edu/cpl/projects/videostabilization/stabilization.pdf

[2] Stack Overflow, ”How do I list all files of a directory”, https://stackoverflow.com/questions/3207219/how-do-i-list-all-files-of-a-directory

[3] Learn OpenCV, ”How to find frame rate of frames per second” https://learnopencv.com/how-to-find-frame-rate-or-frames-per-second-fps-in-opencv-python-cpp/

[4] OpenCV, ”Getting started with videos” https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html

[5] CVXPY, https://www.cvxpy.org/install/

[6] Piazza, https://piazza.com/class/kjlitjmm1bi3it

[7] CS 6475, A3 Panoramas, https://github.gatech.edu/omscs6475/assignments/tree/master/A3-Panoramas

[8] OpenCV, ”Optical Flow”, https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html