Recognizing Panoramas In Video

Jason Chaves [jchaves@stanford.edu]

March 19, 2014

Abstract

Creating panoramas from a few still images is a well studied problem, with robust and efficient solutions. However, there are reasons, either from a user-standpoint or a generalizability standpoint, for which one might want to create a panorama from a video. Additionally, if a video was not taken with the exact intention of panning across a single panoramic scene, it may be desirable to automatically recognize panoramic scenes in the video. This work aims to develop a panorama-from-video program, working on top of the OpenCV Stitching module, which can automatically recognize and create different panoramas. The program must also scale efficiently in order to process the large number of frames inherent in video input. It is found that this work successfully recognizes and creates panoramas from video and uses reasonable CPU and memory resources in the process. However, there is still much room for improvement in terms of speeding up run-time compared to commercial products and possibly adding further levels of automation for deciding how to warp frames onto a common surface.

1 Introduction

There are several ways in which a camera fails to accurately capture what the human eye would see. One such dificiency is the small field-of-view of most cameras, compared to human vision. In photography, panoramas serve to increase the field-of-view of the image, creating a more realistic and satisfying view of the scene. They also have the simple appeal of allowing one to view more parts of a whole scene at once.

The general approach to creating a panorama from several input images can be broken up into the stages of Image Alignment and Image Stitching. In the Image Alignment stage, the 2D relation between the images is found, usually modeled by a homography. In the more popular feature-based approach to alignment, feature descriptors are found for several keypoints in each image, the features are matched across images, and then these correspondences are used to find the homographies between images. In the Image Stitching stage, the images are warped onto a common compositing surface using the computed homographies, and then there are steps to blend the images together to produce a uniform final panorama.

Automatic panorama recognition/generation from discrete photos is a successful procedure, as presented in [BL03, BL07]. The procedure outlined in these papers intends for the input to the algorithm to be a small number of individual photographs of the scene. However, it may be preferential from a user standpoint to be able to take a video panning across the scene and use that as input to a panorama generation program. Taking a video of a scene usually takes less time than taking individual photos, and because of the inherent continuity of taking a video, it's easier for the user to assure that they have covered the entirety of the scene of which they want to produce a panorama. The procedure proposed in [BL03, BL07] does not scale well to hundreds or thousands of input images from a video, therefore a modified approach to the problem must be taken.

The goal of this project is to build on top of existing methods of panorama creation so that video can be an efficient input medium. For instance, it may be impractical to include all frames from a video to generate the panorama, as many frames will have redundant information, and the increased number of input images will increase computation time and memory usage in a super-linear way. It may also be necessary to identify different segments in a video that can be made into different panoramas, in the case that the input video is of longer duration and covers several different scenes.

This project also aims to build on top of OpenCV's Stitching module, which implements a procedure based on [BL07]. As mentioned before, this procedure does not scale well to the number of frames one would have to process for video input. However, the desire of this work is to demonstrate how some modifications and additions to the procedure implemented in OpenCV can allow for a new form of functionality, i.e. panorama creation from video. It is also the hope that the release of the code produced by this work will be more approachable for others due to the use of the popular OpenCV library.

The ultimate goal is to then to produce a general panorama recognition program, which can efficiently take in video of practical size and duration, automatically identify panoramic scenes, and individually attempt to create the found panoramas.

2 Previous Works

2.1 Literature Review

Much of the technical approach of this work is inspired by the procedure presented in [LHG08]. The goal of their work was to automatically discover panoramas in web videos. Web videos were their focus because of the idea that one could then obtain a panorama of a scene without having to physically travel there, but their method was completely based on video as input to panorama creation.

First, in order to automatically find panoramic scenes in a video, they break the video up into segments, each of which is then assessed for the presence of a panorama. These video segments are found based on computing the color histogram of each frame and then computing the intersection of these histograms between adjacent frames. A break between segment occurs when this intersection is below some threshold.

Each video segment is assessed on three criteria before being stitched into a panorama. The requirements for a video segment being suitable for a panorama include (a) a wide field-of-view, (b) proper camera motion suitable for image stitching, and (c) high image quality of the video frames. Details about how each of these requirements were checked can be found in their paper.

For each video segment, the frames that will be incorporated into the panorama are determined in an interesting way. First, the extent of the panoramic scene in a segment is found by choosing different frames to be the central reference frame and then warping all of the other frames onto the reference by the computed homographies; the final central reference frame is chosen as the one that minimizes the composite area, to avoid choosing one that would create a highly distorted panorama. From this central refernce frame, adjacent frames from either side are added to the composite image until the measure of the total image blur and reprojection error exceeds a limit. The composite image produced by this procedure is the resultant panorama before blending, so choosing the frames in this way may help to assure that fewer artifacts will occur in the panorama since the reprojection error is re-calculated upon each addition of a frame to the composite image.

Another work, [SPS05] focuses on redesigning the stitching pipeline in order to very efficiently create panoramas from video. Their pipeline starts by introducing a new method of doing feature matching across frames in a video. "Key" frames are found throughout the video as frames marking the boundaries between temporal regions in the video where all frames have a significant amount of overlap. Each non-key, or intermediate, frame is matched to the other intermediate frames and the key frames to its immediate left and right, and all key frames are matched together. They then use a method of compressing feature correspondences before using an optimized sparse non-linear minimization code to estimate the camera poses. Their method seems to be very thorough in taking advantage of the qualities of video that may allow for simplifying computation, while being general enough to not assume any restrictive camera motion and working well with videos that take multiple swaths over a scene.

An article entitled "Fast Panorama Stitching" hosted on Intel's Developer Zone [Intel] describes several adjustments that can be made to OpenCV's example stitching code in order to improve runtime. This article doesn't target panorama creation from video, but many of it's suggestions carry over to techniques useful for this work, such as windowed feature matching and reducing working resolutions.

All of the above works are fairly specific, either focusing on creating panoramas from video or adjusting parameters to functions in OpenCV's Stitching module. For a more introductory discussion of image alignment and stitching, one can refer to [BL05, BL07] and well as [Szeliski06].

2.2 Contributions of This Work

This work follows the approach from [LHG08] for recognizing panoramas in video, but implements the ideas in a different way. For example, video segmentation, field-of-view calculation, and image quality checks are implemented in different ways. Warping is done using the OpenCV methods which warp images onto a common geometric shape, rather than using a reference frame like in [LHG08]. Part of the reason for choosing these differing implemenations stems from the desire to still follow the general path of the original OpenCV Stitching Pipeline, while making the modifications for it work well with video input.

This work presents an approach for recognizing panoramas from video similar to that of [LHG08], but with alternative implementations of the key steps. This work also involved significant effort to assure that the program would run well on a standard personal computer (no CUDA graphics card, no OpenCL, 4 GB RAM). Lastly, part of the reason behind building on top of OpenCV's Stitching module was to make the code of this work more familiar to others who already have an OpenCV background. The code will be hosted on GitHub for open access to others.

3 Technical Approach

3.1 Overview

To give some background on the state of OpenCV's Stitching module and its procedure for panorama creation, the "stitching pipeline" from their documentation is shown below.

In this pipeline, the images are first reduced to a "work" resolution (called medium resolution in the figure) for feature finding. The SURF features are found in each image and then a "best-of-2 nearest-neighbor" matching is done, similar to the SIFT algorithm, where the best two matches for each feature are found and the best one only is kept if the ratio between the match distances is greater than a threshold, *match_conf*. Only the largest connected component of matched frames (frames with greater than 4 correspondences) is used to create a panorama. The connectedness of this graph is additionally defined by a confidence value assigned to each pair of matched frames, which must be greater than the threshold *conf_thresh* in order for two frames to still be considered matched/connected. The camera matrices are then calculated and refined through bundle adjustment. The images are warped onto a common compositing surface, such as a plane, cylinder, or sphere. Lower "seam" resolution images are used to estimate the location of seams in the panorama, i.e. where the edges of individual images are in the panorama. These seams are finally blended to produce a uniform-looking panorama.

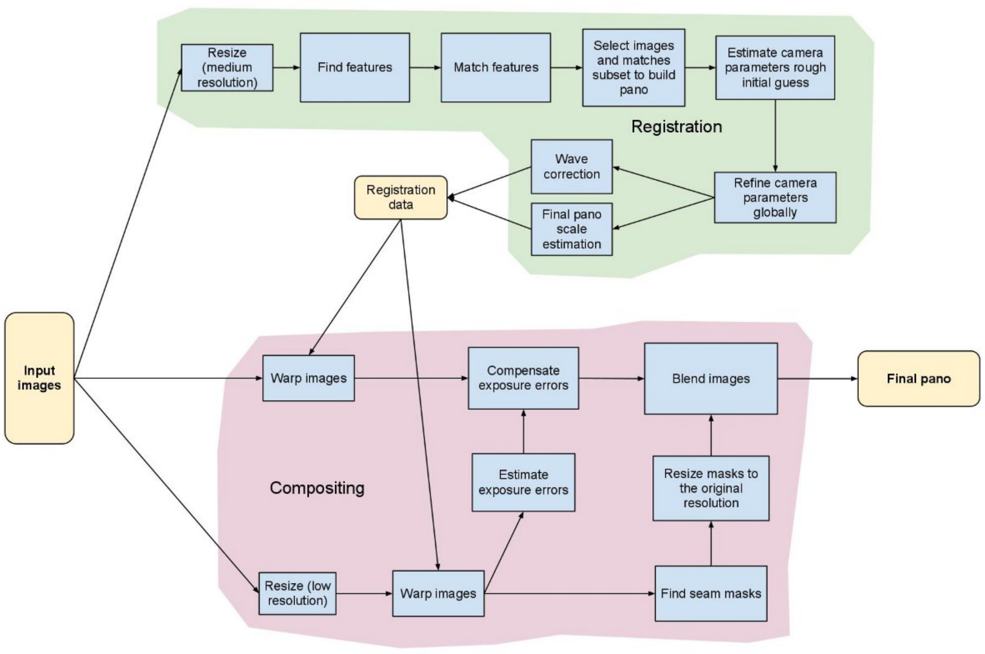

Resize (medium resolution)
Find features
Match features
Select images and matches subset to build pano
Estimate camera parameters rough initial guess
Registration
Wave correction
Registration data
Refine camera parameters globally
Final pano scale estimation
Input images
Warp images
Compensate exposure errors
Blend images
Final pano
Compositing
Estimate exposure errors
Resize masks to the original resolution
Resize (low resolution)
Warp images
Find seam masks

Figure 1: Stitching Pipeline from the documentation page for OpenCV's Stitching module

Now shown below is a modified version of the above pipeline, showing how this work has re-designed the procedure. Each of the modified/added steps shown in the figure, as well as others not shown in the figure, will be explaind in the following subsections.

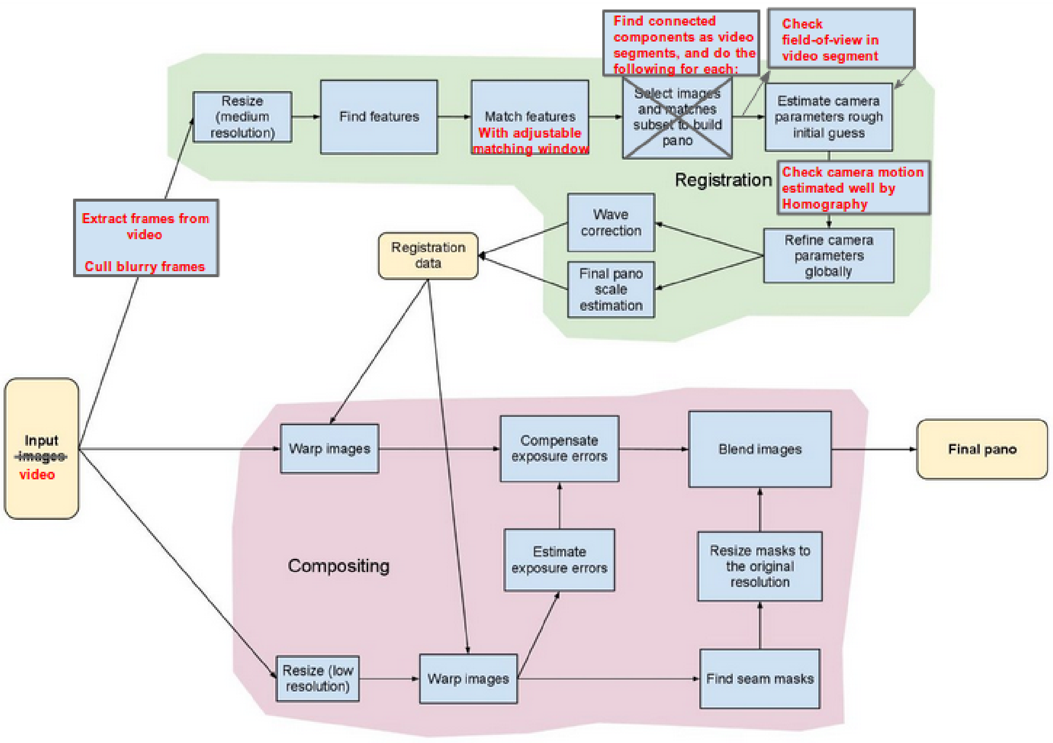

Find connected components as video segments, and do the following for each:
Check field-of-view in video segment
Resize (medium resolution)
Find features
Match features With adjustable matching window
Select images and matches subset to build pano
Estimate camera parameters rough initial guess
Registration
Check camera motion estimated well by Homography
Extract frames from video
Cull blurry frames
Registration data
Wave correction
Final pano scale estimation
Refine camera parameters globally
Input ~~images~~ video
Warp images
Compensate exposure errors
Blend images
Final pano
Compositing
Estimate exposure errors
Resize masks to the original resolution
Resize (low resolution)
Warp images
Find seam masks

Figure 2: Modified Stitching Pipeline, where red font denotes modifications implemented by this work

3.2 Modifications to Existing Steps

Since many of the steps in the original OpenCV pipeline were still used in this work, some of them needed to be modified slightly in order to work well at video scales. The first modification was to take advantage of the sequential nature of video and the fact that temporally neighboring frames are likely to also be neighbors in the spatial domain. By this virtue, the feature matching step could be reduced from the full $\mathcal{O}(N^2)$ matching to a $\mathcal{O}(N)$ matching where each frame is only matched to its temporal neighbors. Specifically, each frame was matched to the 10 frames before it and the 10 frames after it in the video. This ± 10 frame window is robust against occasional failures to match directly adjacent frames, and is a conservative choice for the matching window size. Additionally, the default value of the relevant threshold, *match_conf*, was changed to 0.90 from 0.65 in the original OpenCV pipeline. This larger threshold helps assure that only strong matches are saved, which is important for later steps. This modification greatly reduces the runtime and memory usage of the bundle adjustment step, as well as obviously reducing the runtime of the matching step.

The second modification pertains to the step in the OpenCV pipeline where only the largest connected component is kept for the panorama creation, in order to assure only images of the same scene are included. Instead, this work breaks up the video into segments based on strongly connected components. These strongly connected components are found using the same method that was used to find the largest connected component in the original OpenCV procedure. The default value of the relevant threshold, *conf_thresh*, was changed to 0.75 from 1.0 in the original OpenCV pipeline. This value of the threshold was found as the largest value of the threshold that would still give good identification of different video segments. If the threshold is too large, the strongly connected components will include less frames, and vice versa for a smaller threshold. The optimal value of this threshold may vary with the value of *match_conf* used in the matching step.

Other more minor modifications include changing the default values of the "work" and "seam" resolutions. These values were changed to 0.1 and 0.01 Megapixels, from 0.6 and 0.1 Megapixels in the original OpenCV pipeline, respectively. This modification was particularly important for the memory usage of the program during the bundle adjustment and warping steps.

Without these modifications, the program would often run out of memory on a 4 GB RAM computer, during the bundle adjustment step or the warping step.

3.3 Image Quality Check

The goal of this check is to remove excessively blurry frames before the more computationally expensive steps in the pipeline. Removing these frames from consideration also helps to reduce the blurring introduced into the panorama during image stitching.

This check is done by calculating a contrast measure for each frame when extracting the frames from the input video, and throwing out frames with a contrast below threshold. This contrast measure is calculated as the sum of gradients of the image, using the Sobel operator. Sharper frames will have larger gradients, and therefore a larger contrast measure by this method. The threshold is chosen to be two standard deviations below the mean contrast measure in the input video. This is chosen to try to avoid culling too many frames from an input video of total poor quality.

An alternative approach to calculating blur would be to measure the power in the low-frequency region of the normalized power spectrum of the frame. More blurry frames will have more power in the low-frequency region due to a lack of sharp (high-frequency) features. The definition of the low-frequency region in the power spectrum and the threshold above which frames would be considered excessively blurry could then be calibrated to give good results.

It's also important to note that frames with a lot of motion blur are indirectly culled in this procedure by the video segmentation step. Frames with significant motion blur tend to match poorly to other frames and as a result, the break between video segments usually occurs at times of high motion blur in the video. The frames with significant motion blur are then part of video segments with only one frame, or with less frames than the minimum five I require for panorama creation, and so they are ultimately excluded from panorama creation.

3.4 Field-of-View Check

A defining quality of a panorama is a large field-of-view, at least in comparision to the field-of-view of a single frame. If a video segment has a sufficient number of frames to be considered for panorama creation, but there is little camera motion between frames, then there will be little difference in the scene captured in each frame and combining them into a panorama would produce an image not significantly different from just a single frame. To avoid spending computational resources on such poor and trivial panoramas, the video segment must pass this field-of-view check.

In this work, the field-of-view of a video segment is calculated as the maximum distance in image coordinates between any pair of matched features in different frames. For this step, a full $\mathcal{O}(N^2)$ matching is done on the frames in the video segment, so that the measured field-of-view is not limited by the matching window size. Although this full matching is expensive compared to the windowed matching discussed above, it is better to do the windowed matching over the entire input video and then just do the full matching for each of the smaller video segments.

Care is also taken to ensure that only strong feature matches are considered when looking for the maximum distance between a pair of matched features. The high value of the *match_conf* threshold helps, but an additional check is made to make sure that the reprojection error described in the next subsection is below 5 pixels for the matched pair.

The value of this measure gives the number of pixels a certain feature in the scene moved throughout the video segment. This is a lower bound estimate of the total field-of-view of the segment since there may not be any single feature which is in both the first and last frames of the segment. The default threshold set was 30 pixels, although something closer to 100 pixels could also be reasonable. Note, this 30 pixel measure is with respect to the constant ”work” resolution, so this distance in image coordinates does not need to be noramlized with respect to the input video resolution.

3.5 Camera Motion Model Check

Image alignment is done by estimating a homography between matched frames. For this alignment to be sufficiently accurate to produce non-distorted panoramas, the motion of the camera between frames must be sufficiently approximated by a homography.

This work uses the reprojection error to measure how well the camera motion is approximated by a homography. For each pair of matched frames, a homography can be computed between them since there are at least 4 correspondences. From this homography, one can project the feature keypoints from the first frame to the second frame. But there are also the feature keypoints found in the second frame which have been matched to those in the first frame. The distance between the matched feature keypoint in the second frame and the homography-mapped feature keypoint from the first to second frame is the reprojection error. If the camera motion is well modeled by a homography, then these different estimates of the same point in the second frame will be at the same location, otherwise they may be at slightly different positions in the second frame. The average reprojection error is calculated over all matched features and if it is larger than the threshold, then the video segment cannot be stitched properly and panorama creation is aborted for that segment.

A threshold of 0.5 pixels on the average reprojection error was found to correspond well to when a video segment would stitch well or not. This threshold should vary if the *match_conf* threshold is varied. I.e. if a lower value of *match_conf* is used, than less confident matches will be saved and will lead to an increase in the average reprojection error due to more occasional incorrect matches. Also, the use of a lower value of *match_conf* and the resultant increase in the average reprojection error calculated for the same video segment may degrade the distinguishibility of segments with proper vs improper camera motion.

3.6 Frame Reduction Based On Overlap

Within a video segment that captures a panoramic scene, there is a lot of spatial overlap between the frames. Because of all this redundant information, not all the frames in the segment are needed in order to create a panorama. In fact, including many redundant frames in panorama creation can have adverse affects, either by the inclusion of frames with some blur/artifacts or by an accumulation of alignment error when stitching all the frames together, which could lead to a degraded final panorama. It would therefore be a good idea to try to reduce the number of frames taken from a video segment to be used in the panorama creation, not only to avoid these issues, but to also reduce the runtime and memory usage during the panorama creation procedure.

This work implemented a simple-minded method for doing frame reduction. In this method, the full $\mathcal{O}(N^2)$ matching from the field-of-view step is used, and a list is made where each entry contains the indices of the two frames in the matched pair and the percentage overlap between the two frames. This overlap is calculated by transforming the first frame by the computed homography and then computing the number of overlapping pixels between that and the original second frame. This number, divided by the minimum area of the two frames (the transformed first frame and the original second frame), gives the percentage overlap which is recorded. This list is sorted in descending order by the overlap value and processed in this order. For each entry in the list with an overlap greater than some minimum overlap, if neither of the frames in the entry have already been marked for removal, then one of them is chosen to be removed based on a metric. This metric is a sum of the difference in contrast measures of the two frames and the difference in the number of matches to other frames for each of the two frames in the entry. The goal of this metric is to decide to remove the more blurry of the two frames, unless the difference is contrast is small and the slightly more blurry frame matches better to the other frames in the video segment.

This greedy algorithm tries to remove frames with the greatest amount of overlap first and it does not try to re-calculate the number of matches with remaining (ie not marked for removal) frames for each frame. An issue with this greedy method is that it does not ensure that the remaining frames will form a connected component in terms of matched features. A heuristic way to avoid such an issue is to set the minimum overlap to be rather high, say 98% overlap, so that only very highly overlapping pairs are pruned and it's less likely that the remaining frames will have too little overlap to create a connected component. Even with such a high minimum overlap, frame reductions such as 120 frames to 30 frames are achievable.

A more robust method for frame reduction would be to copy the graph of matched frames, set the edge weights to be the percentage overlaps, and then use an algorithm such as Dijkstra's algorithm to find the shortest path between the first and last frames in the video segment. This has the advantage of assuring that the remaining frames form a connected component by feature matching. It also introduces a constraint on the video segment in that since the start and end frames must be chosen, frame reduction by this method may cut out parts of the video segment if the panning of the camera isn't strictly in one dimension. For example, if the scene is covered in multiple swaths, i.e. multiple pans across the scene from an "S"-like motion of the camera, the shortest path between the first and last frames may exclude a large chunk of the video segment, since frames far from each other temporally may have large spatial overlap. Overall, this method is still likely a good option for frame reduction, but it should be noted to the user that it may impose this constraint on the video segment.

3.7 Additional Features Added

In the interest of making this program adjustable and user-friendly, I've added a few additional command-line options, on top of those used in the original OpenCV stitching example. Among these are the options to downsample the number of frames taken from the video, to skip to a specific segment found in a video (in case a different warping surface would be better for just that segment), to skip the three checks I've discussed above (in case one wants to see what the resultant panorama would look like despite these requirements), to use all the frames in a video rather than find segments (in case one wants to try to force the creation of a very wide panorama), to skip bundle adjustment (in case one finds it's not necessary and takes an excessive amount of time), and to set a memory limit on the program to avoid crashing the computer if it tries to use up too much memory.

I also implemented an idea of partitioned stitching, but found that it didn't work well, at least in how I approached it. The idea of partitioned stitching is that if a video segment is particularly long, then it would take less computational resources to break the segment into smaller pieces, stitch each of those smaller pieces, and then stitch all the intermediate composite images into a final panorama. This is similar to a $\mathcal{O}(log(n))$ divide-and-conquer approach to a problem. I found that the intermediate composite images were often distorted due to their projection onto the warping surface and so feature matching or image alignment would often fail on the set of intermediate images. Perhaps this method would work better if the smaller pieces were even smaller than what I tried (40 frames long) or if a different warping was used.

4 Results

Evaluating the results of a panorama creation program in an objective fashion is a difficult thing to do. This work used videos recorded by a consumer Nikon camera, as well as some of the YouTube videos used in [LHG08], to evaluate the program.

The first assessment was to simply feed the videos taken for this work into the program and observe the output panoramas. These outputs would then be judged qualitatively by the quality of the panoramas and by the separation of the videos into different panoramic scenes. The figures below show some sample panoramas produced by the program. In general, it was found that the program would correctly separate the different panoramic scenes when more than one was intentionally included in the video. It was also found that in some videos that were intended to include a single large-degree panoramic scene, the program would split it into multiple segments. This could usually be fixed by the user lowering the *match_conf* and *conf_thresh* values by command-line arguments. When a video segment passed all of the checks, the resulting panorama never had any noticeable artifacts, such as ghosting.

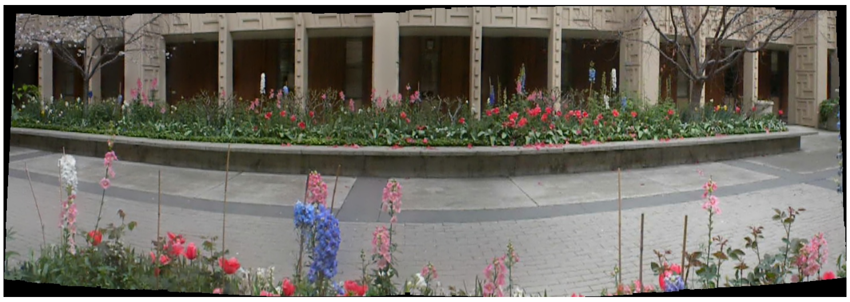


Figure 3: Panorama produced by this work. Video taken by a consumer camera contained just this one panoramic scene.

These videos were also fed into a commercial program, ArcSoft's PanoramaMaker, for comparison. This commercial program does not attempt to automatically recognize different panoramic scenes. It attempts to create a single panorama from the entire video, unless the user specifies just a certain section of the video to be used. For videos where there was only one panoramic scene, the panoramas produced by PanoramaMaker were nearly identical to those produced by this work. For videos with multiple panoramic scenes, leaving PanoramaMaker to use the entirety of the video to make a panorama resulted in very good large-degree, and even 360-degree, panoramas, as shown below. If the program developed by this work is forced to attempt to make a panorama such as these, ignoring video segments and using the whole video, the resulting panoramas contain ghosting artifacts. The run-time of PanoramaMaker

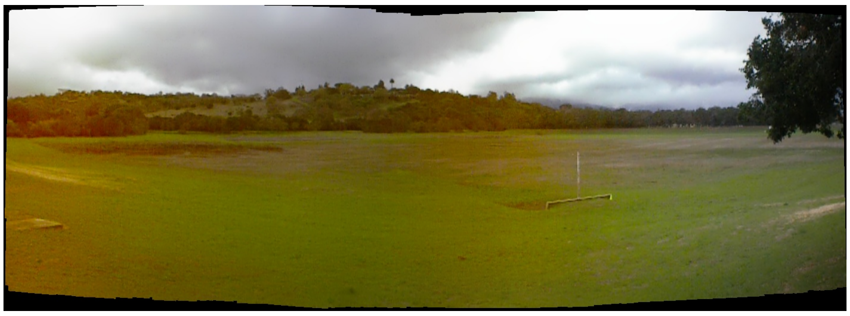
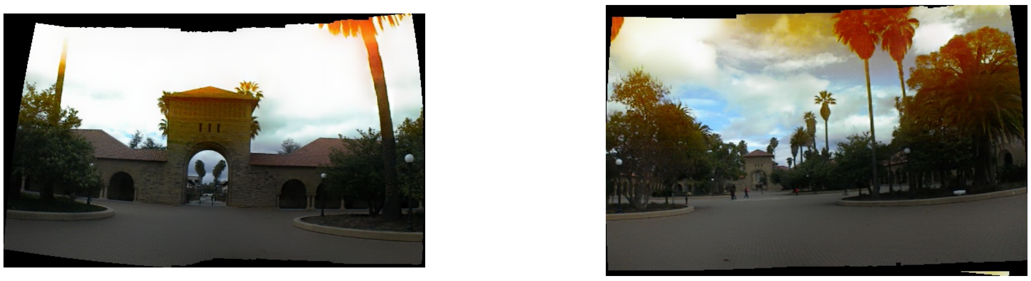


Figure 4: Panorama produced by this work. Video taken by a consumer camera contained just this one panoramic scene.



(a) First Panorama

(b) Second Panorama

Figure 5: Two panoramas taken from the same video, produced by this work.

was also compared to that of this work's program. Where this work took on the order of 15 minutes to process a video, PanoramaMaker took about 40 seconds for the same video. It's clear that the commercial software is superior in it's efficiency and ability to create large-degree panoramas compared to the program developed in this work, although the commercial software does not include automatic panorama recognition to discover multiple panoramas in longer videos.

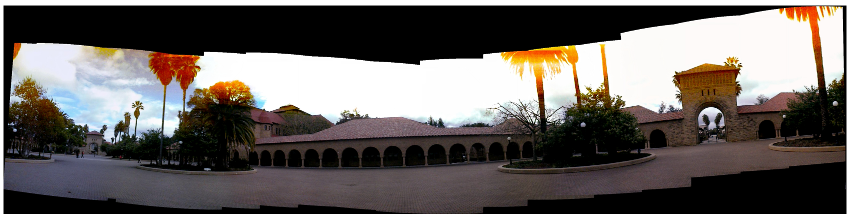


Figure 6: Panorama produced by ArcSoft PanoramaMaker, using same video as the one used for Figure 5

Another assessment was to use some of the YouTube videos used in [LHG08] and compare the panoramas produced by each work. Since these YouTube videos were of a low-resolution, the *match_conf* and *conf_thresh* thresholds had to be lowered in order to get sizeable video segments. They were lowered to 0.8 and 0.6, respectively. For the Notre Dame video, which contains one panoramic scene, both works produced clean and full panoramas. For the West Lake video, which contained two panoramic scenes according to [LHG08], this work found four panoramic scenes. The first and third video segments matched the two panoramas produced in [LHG08]. The second segment was actually quite short, and the forth included a panning scene over distant buildings, which was not included in the panoramas produced by the other work. Unfortunately, the last two video segments identified by this work were not successfully stitched into panoramas because of excessive time being spent on the bundle adjustment step for these segments. While it's a success to see similar panorama identification between the two works, it's clear that there still exists an issue in this work's program which led to an unusual hang-up on the bundle adjustment step.

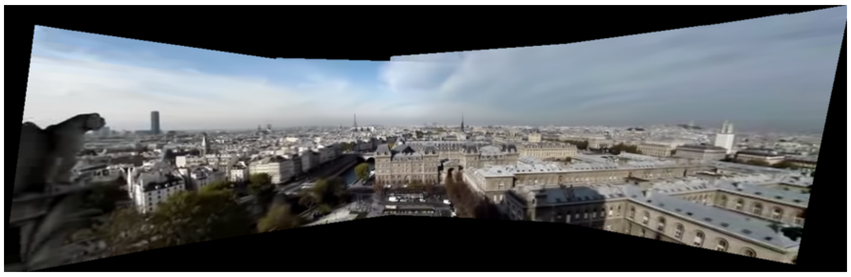


Figure 7: Panorama presented by [LHG08], produced from Notre Dame YouTube video from [LHG08]

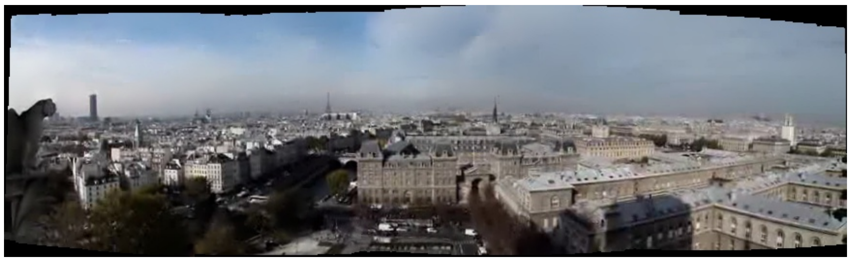


Figure 8: Panorama produced by this work, produced from Notre Dame YouTube video from [LHG08]

5 Conclusion

This work has been successful in developing a program which can identify panoramic scene from video, check these captured scenes for the required criteria suggested in [LHG08], and produce high-quality panoramas from a variety of different input videos. There is the occasional need for the user to change some parameters, such as the warping surface, *match_conf* threshold, ”work” resolution , by command-line arguments. However, it can be made fairly easy for the user to recognize when such changes are possible. For instance, the warp surface should be changed to ”cylindrical” if the panorama is of larger field-of-view and was previously output as an arch with the default option. The value of *match_conf* can be lowered to about 0.8 if the detected video segment are too short or the ”work” resolution can be increased to about 0.3 Megapixels if too few features are found for each frame; these are usually only necessary when the input video is of pretty low resolution/quality.

The Results section above shows several examples of the high-quality panoramas produced by this work; as mentioned before, no ghosting or other artifacts were observed in panoramas that went through all of the criteria checks (i.e. the –force option wasn’t used). All of these were produced in reasonable running times on the order of 20 minutes, and with reasonable memory-usage on a personal computer. However, in comparison to the commercial software, ArcSoft PanoramaMaker, there is still much room for improvement in terms of reducing run-times to the minute-scale and improving stitching capabilites for large field-of-view panoramas.

There are some other methods that could be looked into and possibly included in a program such as the one developed in this work. As mentioned in [Szeliski06], direct pixel-based methods of image alignment may be advantageous for video input, even if they generally perform worse than feature-based methods on the traditional discrete image formulation of creating panoramas. [LHG08] and other works create the composite panorama by picking a reference image and warping all other images onto it, rather than using a geometric surface like in the OpenCV methods. This approach to warping could be more general in producing good looking panoramas over a larger range of fields-of-view.

Possible additional features for such a program include an automatic selection of a warping method, automatic rectangular cropping, or even add real-time panorama view-finding as in [B05] and [S98]. Lastly, videos ability to capture motion can be used to create panoramic activity synopsis as in [LHG08] or panoramic video textures as in [A05].

References

[1] [A05] Aseem Agarwala et al. Panoramic Video Textures. In ACM SIGGRAPH 2005 conference proceedings.

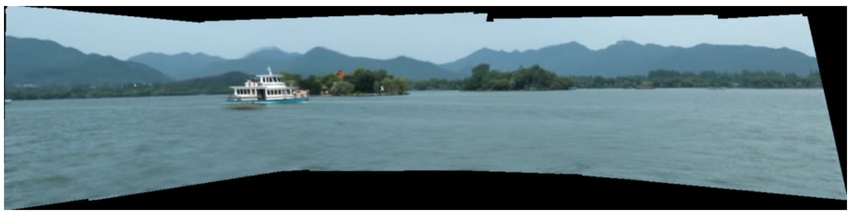


Figure 9: Panorama presented by [LHG08], first panorama produced from West Lake YouTube video from [LHG08]

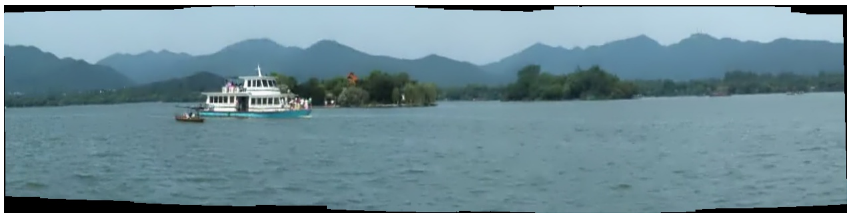


Figure 10: Panorama produced by this work, first panorama produced from West Lake YouTube video from [LHG08]

[2] [B05] P. Baudisch et al., “Panoramic viewfinder: Providing a real-time preview to help users avoid flaws in panoramic pictures,” in OZCHI 2005, (Canberra, Australia), November 2005.

[3] [BL03] M. Brown and D. Lowe. Recognising panoramas. In Proceedings of the 9th International Conference on Computer Vision (ICCV03), volume 2, pages 1218–1225, Nice, October 2003.

[4] [BL07] M. Brown and D. Lowe. Automatic Panoramic Image Stitching using Invariant Features. International Journal of Computer Vision, 74(1), pages 59-73, 2007.

[5] [Intel] Brandon Jones. http://software.intel.com/en-us/articles/fast-panorama-stitching, 01/10/2014.

[6] [LHG08] Feng Liu, Yu-hen Hu and Michael Gleicher. Discovering panoramas in web videos. ACM Multimedia 2008, Vancouver, Canada, October 2008. pp. 329-338.

[7] [S98] H. S. Sawhney et al. ”VideoBrushTM: Experiences with Consumer Video Mosaicing.” IEEE Xplore. N.p., n.d. Web.

[8] SPS05 [SPS05] Drew Steedly, Chris Pal, Richard Szeliski. Efficiently Registering Video into Panoramic Mosaics. (ICCV05).

[9] [Szeliski06] R. Szeliski. Image Alignment and Stitching: A Tutorial. Foundations and Trends in Computer Graphics and Vision Vol. 2, No 1 (2006) 1–104