Eurographics Symposium on Rendering 2014
Wojciech Jarosz and Pieter Peers
(Guest Editors)

Volume 33 (2014), Number 4

User-Assisted Video Stabilization

Jiamin Bai[1], Aseem Agarwala[2], Maneesh Agrawala[1], Ravi Ramamoorthi[1]

[1]University of California, Berkeley
[2]Adobe

Abstract

We present a user-assisted video stabilization algorithm that is able to stabilize challenging videos when state-of-the-art automatic algorithms fail to generate a satisfactory result. Current methods do not give the user any control over the look of the final result. Users either have to accept the stabilized result as is, or discard it should the stabilization fail to generate a smooth output. Our system introduces two new modes of interaction that allow the user to improve the unsatisfactory stabilized video. First, we cluster tracks and visualize them on the warped video. The user ensures that appropriate tracks are selected by clicking on track clusters to include or exclude them. Second, the user can directly specify how regions in the output video should look by drawing quadrilaterals to select and deform parts of the frame. These user-provided deformations reduce undesirable distortions in the video. Our algorithm then computes a stabilized video using the user-selected tracks, while respecting the user-modified regions. The process of interactively removing user-identified artifacts can sometimes introduce new ones, though in most cases there is a net improvement.We demonstrate the effectiveness of our system with a variety of challenging hand held videos.

Categories and Subject Descriptors (according to ACM CCS): I.8 [Computer Graphics]: Applications—Video

1. Introduction

Video stabilization algorithms have improved dramatically in recent years, and can simulate remarkably smooth camera paths. Recent algorithms are 3D-aware and go beyond single-homography motion models to produce much higher quality results. However, these algorithms all have a significant problem; they are one-button press, and offer no user control other than manipulating global parameters. If the stabilization result is flawed, or the user simply wants something else, there is no recourse. We introduce two techniques for interactive control of video stabilization.

The first step in current video stabilization methods is to track feature points that estimate scene and camera motion. The video is then stabilized by warping the frames such that the key feature points follow a smooth trajectory. However, challenging scenes such as those containing a large range of depths and dynamic objects are difficult to stabilize. One reason for failure is that selecting tracks that belong to the background becomes difficult for scenes with dynamic content. If the algorithm uses tracks on dynamic objects (Figure 1(c)) to estimate camera motion and warp the frames, the output video may be unstable. Another reason for failure is that the stabilized video is computed without any knowledge of the scene's semantic content. As a result, regular video stabilization might not yield a desirable visual appearance. For example, vertical structures in the stabilized scene shown in Figure 1(a) are tilted and a viewer might prefer these structures to remain vertical, as shown in our corrected version in Figure 1(b).

Our work directly addresses these two issues by providing two tools for the user to customize the stabilization result. Our algorithm starts with an initial baseline automatic video stabilization result [LYTS13] which the user can further improve using our tools. First, the user can improve track selection to ensure that only background tracks are selected. We cluster tracks spatio-temporally across the video so that the user can interact with tracks at the granularity of whole objects and motions, rather than individual tracks. We then plot clustered track trajectories on the warped video to allow the user to visually inspect and correct the track selection if necessary as demonstrated in Figure 1(c-d). Our clustering improves upon the previous clustering techniques by using a moving window factorization which speeds up computation by 12 times while maintaining similar clustering properties.

Second, we allow the user to provide feedback to the algorithm describing how parts of the stabilized video should

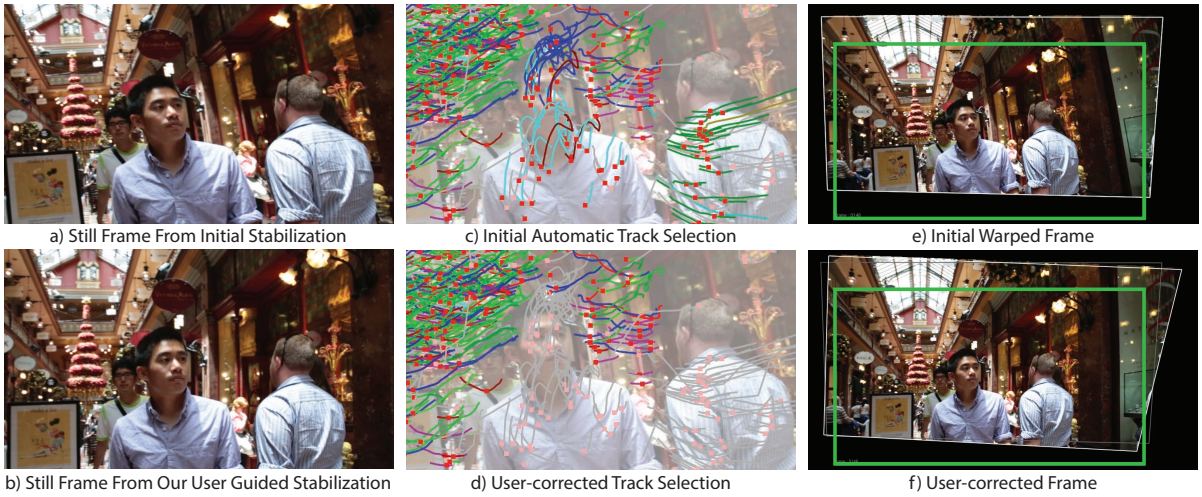


Figure 1: *Automatic video stabilization using the state-of-the-art is unsatisfactory as shown in a) as the background and subjects are heavily skewed. We visualize clusters of tracks used for stabilization c) and the user removes tracks on dynamic objects d) using mouse clicks. Tracks that are not used for the final rewarp are drawn in grey. The green outline in e) and f) shows the original frame boundaries. The distortion of the frame in e) is removed by having the user draw a quadrilateral (white lines) and its desired transformation shown in f). The new track selection and user-drawn transformations are used to re-stabilize the video to obtain the final result as shown in b). Notice that the background is rectified and that the subjects are no longer distorted.*

look. The user corrects regions at unsatisfactory frames by drawing quadrilaterals on the frame and providing her preferred location for the quadrilateral in the final stabilized video as shown in Figure 1(e-f).

After the user has provided edits to improve the stabilized video to her liking, we rewarp the video only using tracks that she has selected and constrain the output video to incorporate the region edits from the quadrilaterals that she has specified. The resulting video exhibits smooth motion for the selected tracks while also respecting the user-specified region constraints.

Our work is the first *user-assisted* video stabilization pipeline which allows the user to influence the video stabilization algorithm. In some cases, fixing user-specified artifacts may lead to new artifacts; however, for most examples we find there to be a net improvement in stabilization quality.

2. Previous Work

Video Stabilization. There are several video stabilization tools available today. For example, a user could stabilize the video in real-time using gyroscopic information from the camera [KJBL11]. She can also use tools such as Warp Stabilizer from Adobe during post-production [LGJA09, LGW*11] or the video stabilization tool on YouTube after uploading the input video [GKE11, GKCE12].

Video stabilization works by first estimating scene and camera motion and then re-projecting each frame such that the unwanted camera motions are minimized. RANSAC [FB81] is used to automatically select tracks on background regions to stabilize the video. If the scene is largely planar, novel camera viewpoints can be generated using homographies [HZ04, GL07, GKE11]. For general scenes, a single homography is too restrictive and re-projections using a grid mesh have better success in stabilizing such videos.

Content-preserving warps [LGJA09] reconstruct the 3D scene and plan an optimal camera path. A grid mesh which preserves the structure of the scene content is used for re-projection. However, an alternative is to use 2D feature trajectories to guide the re-projection after applying a low-rank subspace constraint for camera smoothing [LGW*11]. Epipolar geometry can also be exploited to avoid reconstructing the 3D scene to stabilize the video [GF12]. Motion in-painting can also be included in video stabilization [MOT*06] and parallax can be handled explicitly [WLHL13]. Video stabilization techniques have also been developed for light fields [SZJA09] and for videos with depth information [Sun12].

Bundles of homographies guiding cells in a grid mesh achieve state-of-the-art results by optimizing the path of each grid cell while enforcing spatial coherence [LYTS13]. We use Bundled Paths, a state-of-the-art video stabilization algorithm, as our initial baseline for user improvement because the algorithm was shown to be robust and have su-

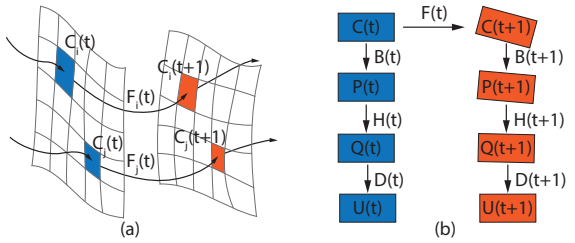


Figure 2: *Homographies $F_i(t)$ computed between grid cells $C_i(t)$ and $C_i(t+1)$ illustrated in (a). Relationships between the computed camera path $C(t)$, the smoothed camera path $P(t)$, the user-specified path $Q(t)$ and the final user-assisted stabilized video $U(t)$ are shown in (b).*

perior results compared to other available techniques. We also compare our results with other commercially available video stabilization tools such as Adobe's Warp Stabilizer and YouTube.

Motion Clustering. We reduce the amount of work the user needs to improve track selection by clustering tracks with similar trajectories and spatial locations since these tracks usually track the same object or region.

Multi-frame motion clustering seeks to group tracks with similar behavior over frames. Techniques which are based on subspaces such as agglomerative lossy clustering [RTVM10] or sparse subspace clustering [EV09] typically require some tracks which span the length of the video sequence. Since hand held videos typically do not have long track trajectories due to occlusions or large motions, these techniques are not robust enough for motion clustering.

Another approach for motion clustering is affinity-based, which measures the pairwise relationship between track trajectories. Affinities based on spatial distance, and similarity of translational motion of track trajectories can be used as features for clustering [BM10]. Affine motion similarity can also be used as a basis for affinities for clustering [FRP09]. Weights from non-negative matrix factorization of the trajectory speed and direction give reliable results for partial track data [CR09]. However, hand held videos have many short tracks and clustering can be computationally intensive. We instead use a moving window and non-negative matrix factorization of the track trajectory's speed and direction to build an affinity matrix for clustering, which greatly decreases the computation time.

3. Background

Bundled paths [LYTS13] is a state-of-the-art video stabilization algorithm which performs well with challenging videos. The algorithm divides each frame into a 16 by 16 grid and estimates camera paths throughout time for each cell. The stabilized output video is obtained by smoothing the individual paths both temporally and spatially. We use bundled paths as the baseline in our system; the user will provide inputs (Sections 4 and 5) to further improve the stabilized video. This section describes the bundled paths algorithm.

Content-preserving warps [LGJA09] guided by SURF tracks [BETVG08] are used to align neighboring frames t and $t+1$. The homography which transforms grid cell i from frame t to $t+1$ is $F_i(t)$ as illustrated in Figure 2(a). The paths are parameterized by the transformations of the grid cells over time. Therefore, the camera path $C_i(t)$ for each grid cell is defined as the sequence of homographies the cell goes through from the start of the video to frame t. Mathematically, the camera path $C_i(t)$ for a grid cell i at frame t is $C_i(t) = F_i(t-1)\cdots F_i(1)F_i(0)$.

The stabilized camera path $P_i(t)$ is derived from the camera path $C_i(t)$ (Figure 2(b)). The algorithm computes $P_i(t)$ by optimizing an energy function $O(P(t))$such that camera path for each grid cell i is smooth temporally and is similar to the path of its neighboring grid cells,

$$\begin{aligned} O(P(t)) &= \sum_t (||P(t)-C(t)||^2) \\ &+ \sum_t (\lambda_t \cdot \sum_{r\in\Omega_t} \omega_{t,r}(C)\cdot||P(t)-P(r)||^2) \\ &+ \sum_t \sum_{j\in N(i)} ||P_i(t)-P_j(t)||^2 \end{aligned} \quad (1)$$

The first term enforces that the smooth path is similar to the original path to minimize cropping and distortion, the second term enforces smooth temporal changes, and the third term enforces spatial smoothness among the grid cells. The temporal window for smoothing, Ω_t, is set to 60 frames and $N(i)$ includes the 8 neighboring grid cells. The strength of the smoothing is controlled by λ_t. The smoothing kernel $\omega_{t,r}$ gives higher weight to paths with temporal proximity and similar transformations. It is set to the product of two Gaussians; the first Gaussian is a function of the frame distance and the second is a function of the difference in translation coefficients of the camera path $C_i(t)$. A Jacobi-based iterative solver is used to solve for the optimal camera path $P(t)$. For more details of the algorithm, please refer to the original paper [LYTS13]. We used our own implementation of the algorithm for our work.

The output video is obtained by applying the transformation $B(t)$ to the input video, which is computed as $B(t) = P(t)C^{-1}(t)$.

Our Work. We use the smooth camera path $P(t)$ as the baseline for our method, but do not assume that it is satisfactory. The user provides a set of constraints on which tracks are selected (section 4) and how regions in the video should look after stabilization (section 5). Region constraints are represented by $H_k(t)$, where some region in frame t is transformed with the homography $H_k(t)$ to obtain the user-specified path $Q(t)$. If no constraints are specified, then $H_k(t)$ is the identify transformation. Since the user does not

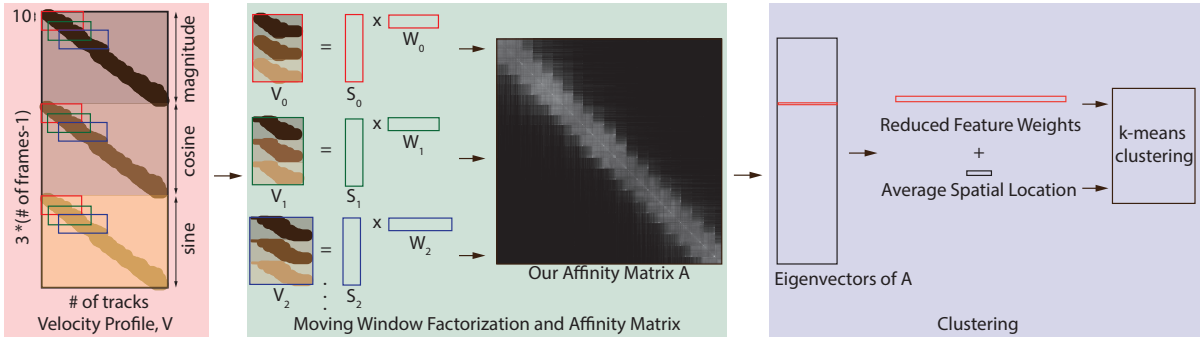


Figure 3: *There are three steps in our track clustering algorithm. The velocity profile V is computed using the magnitude and direction of the tracks. It is sparse as the tracks are short. Next, we find sub-matrices V_i by selecting only velocity profiles that exist in a window of 20 frames. The step size for the window is 10, which gives us overlapping windows and therefore dependent motion bases S_i. The affinity matrix A is computed at this step using the weights in W_i. After the affinity matrix is computed, we project the affinity vector for each track onto the 20 largest eigenvectors of A. We append the average spatial location of each track to its feature vector for k-means clustering.*

specify changes to every frame, $Q(t)$ needs to be smoothed. We compute the final user-assisted stabilized video $U(t)$ by minimizing an energy function described in equation 5 in section 5. The transformation $D(t)$ is used to correct $Q(t)$ to get the $U(t)$ and it is computed as $D(t) = U(t)Q^{-1}(t)$. Figure 2(b) illustrates the relationship between the different paths.

4. Improving Track Selection

Selecting the appropriate set of tracks to guide the stabilization algorithm is crucial to the success of the stabilized video. Tracks that accurately track key feature points should be selected. In most cases, these key points should lie on a stationary background, but for specialized uses such as de-animation [BAAR12, BAAR13], these key points may lie on a subject of interest.

Most video stabilization algorithms automatically prune tracks using RANSAC [FB81] to remove outlier tracks by fitting homographies between frames. While this approach gives acceptable results for simple scenes, selecting appropriate tracks for scenes with multiple moving objects can be difficult for an automatic algorithm.

Our solution is to have the user improve track selection by clicking on tracks to include or exclude when they are overlaid on the video. Note that we also automatically prune tracks using the technique described in Bundled Paths [LYTS13]. We cluster tracks based on affinities of their trajectories and average spatial location across time to alleviate the tedious task of selecting individual tracks for the user. A full cluster can be selected or de-selected with a single mouse click. Since clusters of tracks usually belong to the same object or region, the user can quickly ensure that selected tracks are now properly tracking the background. Tracks that should not be selected often have erratic trajectories in the warped video as their trajectories are not well explained by the camera motion, or they lie on dynamic objects (Figure 4).

4.1. Track Clustering

Track trajectories are usually short for handheld videos due to occlusions and dynamic scene content. Non-negative matrix factorization (NMF) [CR09] seeks a low dimensional motion basis that explains the speed and direction of the track trajectories for the entire video while handling incomplete track trajectories. However, a full factorization of all the incomplete track trajectories can take up to 6 minutes for a 10 second video. For our user-in-the-loop approach it is crucial that the factorization runs within about a minute on standard consumer hardware.

There are three main parts to our approach for track clustering. First, we construct the velocity profile V which stores the magnitude and directions for each track k over all frames. Second, we compute an affinity matrix A which measures the similarity between track trajectories using a moving non-negative matrix factorization that is faster than factoring V directly. Finally, we reduce the dimension of the affinity vector for each track by projecting it onto the eigenvectors of A. We cluster the tracks using the reduced feature vectors and their average spatial location with k-means. Figure 3 illustrates the three steps of our approach, and we consider each one in turn.

4.1.1. Velocity Profile

The first step is to compute the velocity profile V. If the video has T frames and K total tracks, then matrix V is size $3(T-1) \times K$. The velocity profile captures the instantaneous

magnitude and direction of each track k. The k^{th} column of V is:

$$V^k = [m_1^k, \cdots, m_{T-1}^k, c_1^k, \cdots, c_{T-1}^k, s_1^k, \cdots, s_{T-1}^k]^T \quad (2)$$

where m_t^k is the magnitude of the velocity of the k^{th} track from frame t to $t+1$, while c_t^k and s_t^k are the cosine and sine of the angle of the k^{th} track. We add 1 to c_t^k and s_t^k to make the entries of V positive and we enter zeros into V to handle missing data at any particular frame. Figure 3 shows the structure of a typical velocity profile V.

4.1.2. Moving Window Factorization and Affinity Matrix

Non-negative matrix factorization seeks to factor V into its motion basis S and the corresponding weights W. The affinity matrix A of size $K \times K$ measures the similarity between all tracks in the video using their weights computed in W. However, the velocity profile is usually sparse because tracks in handheld videos are short as seen in Figure 3. Therefore the computation can be sped up by first breaking V into windows and factoring each window separately.

The entries of sub-matrix V_i correspond to velocity profiles of tracks that exist from frame $i*b$ to $(i+2)*b$, where b is the step size. For our application, we set b to 10. In other words, our window size is 20 and the windows overlap by 10 frames as we move the window. To create the windowed V_i, we simply select the appropriate rows from V (Figure 3) and discard any empty columns. We factor each V_i into its motion basis S_i and the corresponding weights W_i. The overlapping windows help to maintain coherence in the estimated motion basis S_i. We compute affinities between tracks in each sub-matrix V_i using W_i and update the overall affinity matrix A accordingly, as described later in equation 4.

We seek to factorize V_i into non-negative matrices S_i ($6b \times r$) and W_i ($r \times \hat{K}$) that minimize the following expression:

$$||V_i - S_i W_i||^2 \quad (3)$$

The matrix S_i is the motion basis for the window, and W_i contains the non-negative weights for constructing the trajectory of the tracks using the basis S_i. $\hat{K}$ is the number of tracks that exist in the window. We use alternating least-squares [GZ79] to find S_i and W_i using the the algorithm described by Cheriyadat et al. [CR09]. We set the number of terms r to be 6 for our purposes as we expect a sparse set of motion basis vectors will be sufficient to represent the motions within a short window.

We can compute the affinity matrix A after the matrices W_i are computed. Let $\Phi_{q,s}$ be the set of indices of the matrices W_i that contain track q and s. Also, let $w_i(q)$ be the column of W_i containing the weights of the q^{th} track in the i^{th} window. The affinity between two tracks q and s is computed as follows:

$$A_{qs} = \frac{1}{|\Phi_{q,s}|} \sum_{i \in \Phi_{q,s}} \exp\left(\frac{-||w_i(q) - w_i(s)||^2}{\sigma}\right) \quad (4)$$

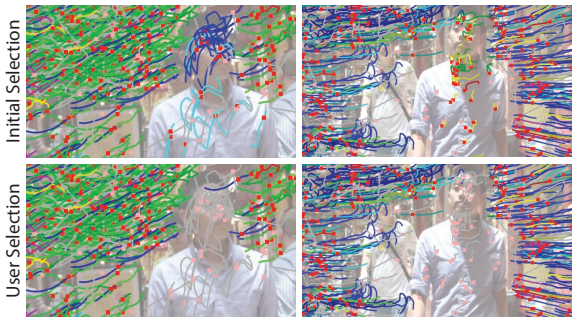


Figure 4: *We cluster and visualize the selected tracks. Each cluster of tracks is assigned a bright color and their trajectories are overlaid on the warped video. Notice how tracks on dynamic objects can have an erratic appearance. The user then removes some clusters by clicking on the clusters to toggle them off. Tracks that are not used for the next warp are shown in light grey.*

In other words, the affinity score between two tracks is the average affinity score computed over the windows where both tracks exist. We set σ to 5.0.

Our approach does not gives us the same affinity matrix as that obtained from a direct factorization of V, without breaking into windows. In particular, in our approach, tracks which do not exist on the same frame will have an affinity score of 0. This implicitly enforces tracks which do not overlap temporally to be clustered into different clusters. We compare and discuss our clustering results and performance using our approach against direct factorization in the results section.

4.1.3. Clustering

The q^{th} row of the affinity matrix is the affinity vector for the q^{th} track. Instead of using the affinity vector to cluster the tracks directly, we reduce the dimension of the affinity vector by projecting it onto the 20 largest eigenvectors of A to reduce computation costs for clustering. In practice, the reduced feature vector is the row of the set of eigenvectors corresponding to each track. This reduced feature vector for track q is combined with the average spatial location of the track across time to form a length 22 vector. (The average spatial location is normalized by the width of the video and multiplied by 15.) We use k-means clustering with the number of clusters set to be the number of frames T. Examples for our track clustering are shown in Figure 4.

4.2. Track Visualization

Each track is drawn as a red square on the frame with its entire trajectory drawn as a line on the warped video (Figure 4). Selected tracks are displayed in bright colors, with

each track cluster receiving a unique color. Tracks which are pruned and not used to guide the warp are displayed as pink squares with their trajectory in light grey.

The user interacts with the track clusters by clicking on them when the warped tracks are displayed. We find the closest track cluster to the mouse and highlight it in real-time. The user can toggle the track cluster by clicking on it to either include or exclude the cluster from guiding the warp. Once she is happy with her modifications, we stabilize the video with the user-selected tracks.

Occasionally, the user might wish finer-grain control to make edits within a cluster, or to refine the output of the clustering algorithm. We allow them to remove and turn on/off specific tracks from within a cluster.

5. Specifying Region Warps

Videos stabilized with tracks on the background might still be undesirable for a user like the example shown in Figure 1(d), where the background structures are tilted. In general, since the baseline stabilization algorithm does not have any semantic information of the content in the video, it might warp the frames in a way that the user finds undesirable. Our solution is to allow the user to directly specify how regions in the frame should deform. We then use the user specifications to guide a new optimization with new temporal weights that propagates the user edits. This new optimization tries to find a stabilized video that is close to the previous stabilized result yet conforms to the user-specified constraints as shown in Figure 1(b).

5.1. User Interface

The user scrubs through the video to find an offending frame she wishes to correct. We show the entire warped frame without cropping to allow the user to access the extreme edges of the warped frame. A green outline is overlaid to provide a visual cue on the boundaries of the initial frame size as shown in Figure 1(e-f).

The user draws a quadrilateral around the region she wishes to correct by clicking on the four corners of the region. The corners of the quadrilateral can be selected and dragged to warp the region. The region selected within the quadrilateral is deformed in real-time to provide feedback for the user as shown in Figure 1(f). Arrow keys on the keyboard translate the region to provide additional control. Note that while only one quadrilateral is drawn in Figure 1(f) which covers the entire frame, the user can draw multiple small quadrilaterals (which do not cover the entire frame) to make local corrections.

The user can specify how each edited frame will affect neighboring frames. She can choose to propagate the edit to frames both preceding and succeeding the edited frame (the default), or only propagate in one direction.

We introduce a global parameter α with which the user can specify the confidence of her region constraints. If the α value is high, her region constraints have more weight in the optimization routine when solving for a user-stabilized output. Conversely if the α value is low, her region constraints can be relaxed to generate a smoother video. We provide two recommended values for α, 1 and 10 which we found to balance well with the rest of the energy terms.

5.2. Stabilizing Video With User-Constraints

After the user is done specifying region constraints using quadrilaterals and deforming them, our system derives the user-specified camera path $Q(t)$. It also propagates the constraints smoothly across time and we achieve this by introducing a window function which modulates the temporal term in the video stabilization energy function. We also introduce a data term that controls which frames are penalized if they deviate from the user-specified path $Q(t)$. The following parts detail how $Q(t)$ is computed, how the energy functions are modified and how the final stabilized video $U(t)$ is computed.

5.2.1. User-Specified Camera Path

The user-specified camera path $Q(t)$ incorporates the region constraints the user has drawn. Therefore, we first identify which grid cells are affected by each region constraint. Grid cells which have more than 50% of their area within the drawn quadrilateral inherit any modifications the user makes to the quadrilateral. The k^{th} deformation the user makes on frame t is represented by a homography $H_k(t)$. We compute $H_k(t)$ by computing the homography that takes the original four corners of the quadrilateral to the corners of the modified quadrilateral.

If the user adds no constraints, then $Q(t) = P(t)$. However, if the user modifies regions by deforming them with the k^{th} quadrilateral, then for grid cell i in frame t, the user-specified camera path is $Q_i(t) = H_k(t)P_i(t)$.

5.2.2. Constraint Propagation

Since the user only specifies region constraints on a sparse set of frames, our algorithm must propagate the region constraints smoothly over time. Each region constraint is linearly interpolated to its neighboring frames as shown in Figure 5. The linear interpolation is performed for a maximum of 60 frames centered at the edited frame t, illustrated by the frames after the blue arrow in Figure 5.

We embed this interpolation in the original energy function in equation 1. We introduce a new window function σ_t which modulates the existing temporal filter $\omega_{t,r}$ in equation 1 by replacing $\omega_{t,r}$ with $\sigma_t \cdot \omega_{t,r}$. Our window function σ_t is a linear interpolation weight as shown in Figure 5. We set σ_t to 1 for the entire window by default. We use the corresponding linear interpolation weights as illustrated in

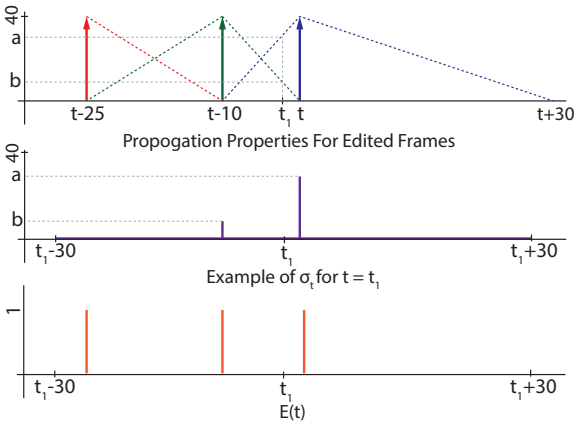


Figure 5: ***Top****: The user adds three region constraints on frame t, $t-10$ and $t-25$. The dotted lines show how the the region constraints should be linearly propagated. The support of the interpolation is determined by the proximity of other constraints. If there are region constraints that are within 30 frames, such as the red-green and green-blue pair, the linear interpolation is performed within the frames between them. Otherwise, the interpolation is performed for 30 frames like the blue arrow to the right. The user can also specify one-sided propagation like the red arrow on the left.* ***Middle:*** *We show an example of our window function σ_t for frame t_1.* ***Bottom:*** *The function $E(t)$ controls if the new data term is applied for each frame, allowing frames to follow the user-specified path $Q(t)$ or not.*

Figure 5 if there are region constraints on frames within the window.

5.2.3. Warping With Constraints

We would like to reoptimize the stabilized video to find a smooth user-assisted stabilized video $U(t)$ which respects the user-specified camera path $Q(t)$. We use a similar energy function $O(U(t))$ to that in equation 1 used to find the initial stabilized video,

$$\begin{aligned} O(U(t)) &= \sum_t (\tfrac{1}{5}(1-E(t))||U(t)-C(t)||^2) \\ &+ \sum_t (5\alpha E(t)||U(t)-Q(t)||^2) \\ &+ \sum_t (\alpha\lambda_t \cdot \sum_{r\in\Omega_t} \sigma_t \cdot \omega_{t,r}(C)\cdot||U(t)-U(r)||^2) \\ &+ \sum_t \sum_{j\in N(i)} ||U_i(t)-U_j(t)||^2 \end{aligned} \tag{5}$$

The energy function we optimize is similar to equation 1. We reduce the weight of the first term which enforces the final stabilized video to follow the estimated camera motion $C(t)$. We introduce a new term so that the final stabilized video will follow the user-specified path $Q(t)$. Note that the temporally varying function $E(t)$ controls when the final stabilized video should be similar to $Q(t)$. Also, $\omega_{t,r}$ is replaced with $\sigma_t \cdot \omega_{t,r}$ and with $P(t)$ replaced with $U(t)$. The value α (default value is 10) is the global variable which controls the confidence of the user's region constraints.

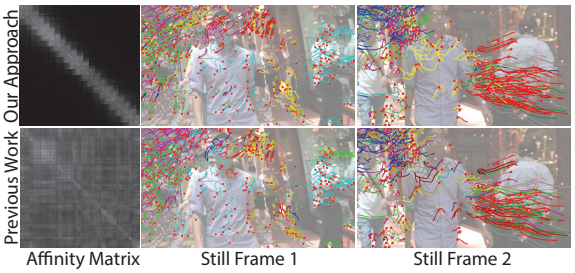


Figure 6: *We compare the clustering result of our method against using the direct factorization of V. Tracks in the same clusters are drawn with the same color. Notice that while the affinity matrices are different, the quality of the track clusters are similar. In particular, tracks belonging to the dynamic objects are rarely clustered with the background tracks. In still frame 1, our result correctly clusters the tracks on the arm with the tracks on the torso. Also, in still frame 2, our result clusters the tracks on the torso as a single cluster. Our clustering method is also 12 times faster.*

We set $E(t)$ to have a value of 0 for all frames initially. For each frame t that the user provides an edit, we set $E(t)$ to 1.

The update rule for the Jacobi-based iterative solver [BS97] is:

$$\begin{aligned} U_i^{(\xi+1)}(t) &= \tfrac{1}{\gamma'}(\tfrac{1}{5}(1-E(t))C_i(t)+5\alpha E(t)Q_i(t) \\ &+ \sum_{\substack{r\in\Omega_t \\ r\neq t}} 2\lambda_t\omega_{t,r}\sigma_t U_i^{(\xi)}(t) + \sum_{\substack{j\in Ni \\ j\neq i}} 2U_j^{(\xi)}(t)) \end{aligned} \tag{6}$$

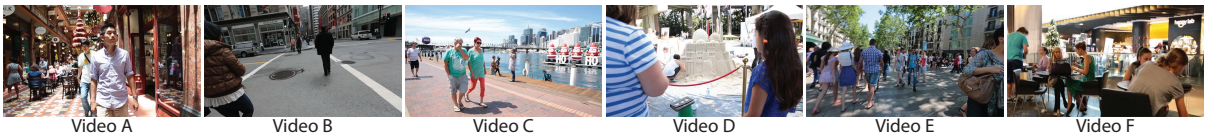
where

$$\gamma' = 2\lambda_t\sigma_t \sum_{\substack{r\in\Omega_t \\ r\neq t}} \omega_{t,r} + 2N(i) + \frac{1}{5}(1-E(t)) + 5\alpha E(t)$$

The $(\xi+1)^{th}$ iteration of the user-specifed camera path $U_i^{(\xi+1)}(t)$ is computed with the update rule in equation 6. The initial solution $U^{(0)}(t)$ is initialized with $Q(t)$ and we use 15 iterations. The transformation $D(t)$ is used to correct $Q(t)$ and it is computed as $D(t) = U(t)Q^{-1}(t)$.

6. Results

We test our system on a collection of very challenging videos and compare our user stabilized result against the other methods. Specifically, we compare against our baseline [LYTS13], YouTube's video



ID	# frames	# tracks	$C(t)$	$P(t)$	$U(t)$	Direct A	Our A	Total Clustering	# Cluster Edit	# Region Const.
A	207	16144	38.9 s	102.9s	106.5 s	225.2 s	14.85 s	30.1 s	24	23
B	298	19385	57.6 s	146.1 s	153.9 s	156.0 s	17.78 s	44.6 s	49	23
C	167	11050	28.9 s	79.2 s	84.5 s	86.0 s	10.4 s	19.1 s	10	23
D	203	15001	34.8 s	103.9 s	103.6 s	253.36 s	13.5 s	28.6 s	23	24
E	312	23936	54.6 s	163.1 s	162.9 s	371.3 s	25.4 s	60.5 s	68	36
F	147	10186	23.2 s	73.8 s	73.7 s	60.3 s	8.7 s	15.0 s	2	23

Table 1: *Our computation times for our examples as well as the number of user edits. Notice that our method for computing the affinity matrix is on average 12 times faster. Our average clustering time is 33 seconds. Computing the baseline video stabilization takes 0.67 seconds per frame (1.5fps). Optimizing for our user-assisted video takes 0.52 seconds per frame (1.9fps). On average the user removes 29 clusters and provides 25 region constraints.*

Figure 7: *Comparision of initial baseline with result using user-selected tracks. The face and torso of the subject have less horizontal stretch in the top example. The walls of the buildings on the left are less skewed in the bottom example.*

stabilizer [GKE11, GKCE12], and Adobe's warp stabilizer [LGJA09, LGW*11]. We use a MacBook Pro 2013 2.8 Ghz i7 with 16GB of RAM. Our code is unoptimized in C and Matlab. OpenGL is used to display the video and user interface.

User Input and Performance. Depending on the complexity of the video, and desired control over the final output video, the user typically takes around 10 minutes to select tracks and provide constraints. Our selection of 6 videos contains severe camera shake with wide field-of-view and large depth range. This makes our videos very challenging for video stabilization algorithms. We remove 29 track clusters and provide 25 region constraints on average for our videos. The region constraints are distributed over the length of the entire video. Our results have smoother camera motion with less background distortion when compared to the other methods. For detailed comparisons, please see the main video and supplementary videos.

It takes 0.67 seconds per frame (1.5fps) to stabilized the video using the initial baseline ($C(t)$ and $P(t)$) and 0.52 seconds per frame (1.9fps) to optimize with user constraints ($U(t)$). Our computation for the affinity matrix is on average 12 times faster than using direct factorization. Our total clustering time on average is 33 seconds. Table 1 has the detailed breakdown of the times and edits.

Quality of Track Clustering. Factoring the velocity profile matrix directly can be computationally intensive since the size of V can be up to 1000×20000 for our examples. Our method clusters small overlapping windows of V_i instead and computes affinities based on local weights W_i from local motion bases S_i as described in Section 4.1. While the affinities computed with the two techniques are different, the clustering results are similar as shown in Figure 6.

Notice that in both methods, the tracks on dynamic objects are rarely clustered with the background tracks. In still frame 1, our method clusters tracks on the subject's arm on the right with the torso while the previous work clusters them with the background. In still frame 2, our method successfully clusters tracks on the torso as a single cluster.

In addition, our approach assigns tracks that do not exist in the same frame an affinity score of 0. As a result, the likelihood of tracks that do not exist on the same frame being assigned the same cluster is much lower than the previous approach.

Figure 8: *Before region constraints were applied (top row), the frame is skewed. The region constraint is drawn on frame 150 and it corrects for this deformation. The red arrows show the change in the quadrilateral drawn in white for the region constraint. After the video is stabilized with the constraints, note that the neighboring frames are corrected appropriately as the background is upright and there are fewer distortions.*

Quality of Stabilization. There are two ways the user can improve video stabilization in our framework. First by improving track selection, and second, by adding additional region constraints. In Figure 7, the use of user-selected tracks reduces distortion in both examples. In the top example, the horizontal stretch of the subject is reduced after using the appropriate tracks for stabilization. In the bottom example, the walls of the building on the left, after using user-selected tracks, have less skew than the initial baseline. For video comparisons please see the supplementary videos.

When proper track selection is not sufficient to improve the stabilization results, the user can correct distortions directly as shown in the example in Figure 8. The user drawn constraints are on the right and are applied to frame 150. Notice that after stabilizing the video with the user constraints, the neighboring frames inherit the user modifications properly. However, in some cases, slight wobbles are introduced in the video after correcting for distortions.

In Figure 9, we compare video stills from the baseline stabilization against corresponding frames after the user-specified constraints are used to stabilize the video. Notice that in both examples, the stabilized video with user-constraints has milder distortions as vertical structures in the scene remain vertical. For better comparison for all results, please see our main video and supplementary videos.

Since the quality of the stabilized video is strongly correlated with the user constraints, the more constraints the user provides across the frames, the better the stabilized video will be. Our results are generated with a set of user constraints across the video to correct for distortion and camera paths. Note that if the user only wishes to fix major artifacts in isolated regions, she only would have to provide constraints in those regions.

Limitations. While our method allows the user to have control over how the stabilized video will look, our system will not generate good results if the quality of the user constraints is bad. In particular, if the user constraints significantly deviate from the camera path and are in conflict, the output video will be jerky. Our system does not infer the intent of the user. For example, if the user wants to level the horizon in the stabilized video, she would still have to manually correct regularly spaced frames as opposed to just specifying her intent. However, there is also a drawback to our method; in some cases, new artifacts are introduced in the effort to reduce existing artifacts.

7. Conclusion and Future Work

We have demonstrated that our system can improve video stabilization by allowing the user to customize the result to her preferences. The two modes of interactions, track selection and region specification, directly relate to how automatic stabilization algorithms work. With a set of correctly selected tracks and region constraints, the output video exhibits smoother motion while respecting the user's intent.

One avenue for future work is to explore how more tools can be used for improving stabilization. For example, higher level controls such as automatic vertical and horizontal rectification could be explored to minimize the user interaction needed. Similarly, the user could also potentially specify constraints by providing exemplar frames and the algorithm could compute the transformations necessary to match scene properties such as vanishing points. Finally, there is definitely room for improvement as our technique might introduce unwanted artifacts such as small wobbles in the final result.

In conclusion, we believe that our paper is a first step towards video stabilization algorithms that leverage user guidance and expertise to solve for a stabilized output, beyond the capabilities of fully automatic methods.

Figure 9: *Comparision of initial baseline with result using user constraints. In all examples on the left, the baseline stabilization is not satisfactory as the scene is distorted and the horizon is not level. The same frames in our result are less distorted as the structures are vertical.*

Acknowledgements

We would like to thank the reviewers for their detailed comments. This work is supported in part by ONR PECASE grant N00014-09-1-0741, NSF grants CCF-0643552, IIS-1016920, an A*STAR NSS PhD fellowship, gifts and software from Adobe, funding from Nokia, and the Intel Science and Technology Center for Visual Computing.

References

[BAAR12] BAI J., AGARWALA A., AGRAWALA M., RAMAMOORTHI R.: Selectively de-animating video. *ACM Transactions on Graphics* (2012). 4

[BAAR13] BAI J., AGARWALA A., AGRAWALA M., RAMAMOORTHI R.: Automatic cinemagraph portraits. *Computer Graphics Forum (EGSR 2013)* (2013). 4

[BETVG08] BAY H., ESS A., TUYTELAARS T., VAN GOOL L.: Speeded-up robust features (surf). *Comput. Vis. Image Underst. 110*, 3 (June 2008), 346–359. 3

[BM10] BROX T., MALIK J.: Object segmentation by long term analysis of point trajectories. In *European Conference on Computer Vision (ECCV)* (Sept. 2010), Lecture Notes in Computer Science, Springer. 3

[BS97] BRONSHTEIN I. N., SEMENDYAYEV K. A.: *Handbook of Mathematics (3rd Ed.)*. Springer-Verlag, London, UK, UK, 1997. 7

[CR09] CHERIYADAT A., RADKE R. J.: Non-negative matrix factorization of partial track data for motion segmentation. In *ICCV* (2009), pp. 865–872. 3, 4, 5

[EV09] ELHAMIFAR E., VIDAL R.: Sparse subspace clustering. In *In CVPR* (2009). 3

[FB81] FISCHLER M. A., BOLLES R. C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM 24*, 6 (June 1981), 381–395. 2, 4

[FRP09] FRADET M., ROBERT P., PÉREZ P.: Clustering point trajectories with various life-spans. In *Proceedings of the 2009 Conference for Visual Media Production* (Washington, DC, USA, 2009), CVMP '09, IEEE Computer Society, pp. 7–14. 3

[GF12] GOLDSTEIN A., FATTAL R.: Video stabilization using epipolar geometry. *ACM Trans. Graph. 32*, 5 (2012). 2

[GKCE12] GRUNDMANN M., KWATRA V., CASTRO D., ESSA I.: Calibration-free rolling shutter removal. In *International Conference on Computational Photography [Best Paper]* (2012). 2, 8

[GKE11] GRUNDMANN M., KWATRA V., ESSA I.: Auto-directed video stabilization with robust l1 optimal camera paths. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2011)* (2011). 2, 8

[GL07] GLEICHER M. L., LIU F.: Re-cinematography: Improving the camera dynamics of casual video. In *Proceedings of the 15th International Conference on Multimedia* (New York, NY, USA, 2007), MULTIMEDIA '07, ACM, pp. 27–36. 2

[GZ79] GABRIEL K. R., ZAMIR S.: Lower Rank Approximation of Matrices by Least Squares with Any Choice of Weights. *Technometrics 21*, 4 (1979), 489–498. 5

[HZ04] HARTLEY R. I., ZISSERMAN A.: *Multiple View Geometry in Computer Vision*, second ed. Cambridge University Press, ISBN: 0521540518, 2004. 2

[KJBL11] KARPENKO A., JACOBS D., BAEK J., LEVOY M.: Digital video stabilization and rolling shutter correction using gyroscopes. In *In Stanford CS Tech Report* (2011). 2

[LGJA09] LIU F., GLEICHER M., JIN H., AGARWALA A.: Content-preserving warps for 3d video stabilization. *ACM Trans. Graph. 28*, 3 (July 2009), 44:1–44:9. 2, 3, 8

[LGW*11] LIU F., GLEICHER M., WANG J., JIN H., AGARWALA A.: Subspace video stabilization. *ACM Trans. Graph. 30*, 1 (Feb. 2011), 4:1–4:10. 2, 8

[LYTS13] LIU S., YUAN L., TAN P., SUN J.: Bundled camera paths for video stabilization. *ACM Trans. Graph. 32*, 4 (July 2013), 78:1–78:10. 1, 2, 3, 4, 7

[MOT*06] MATSUSHITA Y., OFEK E., TANG X., MEMBER S., YEUNG SHUM H.: Full-frame video stabilization with motion inpainting. *IEEE Trans. Patt. Anal. Mach. Intell* (2006), 1150–1163. 2

[RTVM10] RAO S., TRON R., VIDAL R., MA Y.: Motion segmentation in the presence of outlying, incomplete, or corrupted trajectories. *IEEE Transactions on Pattern Analysis and Machine Intelligence 32*, 10 (2010), 1832–1845. 3

[Sun12] SUN J.: Video stabilization with a depth camera. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Washington, DC, USA, 2012), CVPR '12, IEEE Computer Society, pp. 89–95. 2

[SZJA09] SMITH B., ZHANG L., JIN H., AGARWALA A.: Light field video stabilization. In *Computer Vision, 2009 IEEE 12th International Conference on* (Sept 2009), pp. 341–348. 2

[WLHL13] WANG Y.-S., LIU F., HSU P.-S., LEE T.-Y.: Spatially and temporally optimized video stabilization. *IEEE Transactions on Visualization and Computer Graphics 19*, 8 (Aug. 2013), 1354–1361. 2