Subspace Video Stabilization

FENG LIU
Portland State University
MICHAEL GLEICHER
University of Wisconsin-Madison
and
JUE WANG, HAILIN JIN, and ASEEM AGARWALA
Adobe Systems, Inc.

We present a robust and efficient approach to video stabilization that achieves high-quality camera motion for a wide range of videos. In this article, we focus on the problem of transforming a set of input 2D motion trajectories so that they are both smooth and resemble visually plausible views of the imaged scene; our key insight is that we can achieve this goal by enforcing *subspace constraints* on feature trajectories while smoothing them. Our approach assembles tracked features in the video into a trajectory matrix, factors it into two low-rank matrices, and performs filtering or curve fitting in a low-dimensional linear space. In order to process long videos, we propose a moving factorization that is both efficient and streamable. Our experiments confirm that our approach can efficiently provide stabilization results comparable with prior 3D methods in cases where those methods succeed, but also provides smooth camera motions in cases where such approaches often fail, such as videos that lack parallax. The presented approach offers the first method that both achieves high-quality video stabilization and is practical enough for consumer applications.

Categories and Subject Descriptors: I.4.3 [**Image Processing and Computer Vision**]: Enhancement; I.4.9 [**Image Processing and Computer Vision**]: Applications; I.3.8 [**Computer Graphics**]: Applications

General Terms: Algorithms, Human Factors

Additional Key Words and Phrases: Video stabilization, video warping

ACM Reference Format:

Liu, F., Gleicher, M., Wang, J., Jin, H., and Agarwala, A. 2011. Subspace video stabilization. ACM Trans. Graph. 30, 1, Article 4 (January 2011), 10 pages. DOI = 10.1145/1899404.1899408 http://doi.acm.org/10.1145/1899404.1899408

1. INTRODUCTION

One of the most obvious differences between professional- and amateur-level video is the quality of camera motion; hand-held amateur video is typically shaky and undirected, while professionals use careful planning and equipment such as dollies or steadicams to achieve directed motion. Such hardware is impractical for many situations, so video stabilization software is a widely used and important tool for improving casual video. In this article we introduce a technique for software video stabilization that is robust and efficient, yet provides high-quality results over a wide range of videos.

Prior techniques for software video stabilization follow two main approaches, providing either high quality *or* robustness and efficiency. The most common approach is 2D stabilization [Morimoto and Chellappa 1997], which is widely implemented in commercial software. This approach applies 2D motion models, such as affine or projective transforms, to each video frame. Though 2D stabilization is robust and fast, the amount of stabilization it can provide is very limited because the motion model is too weak; it cannot account for the parallax induced by 3D camera motion. In contrast, 3D video stabilization techniques [Buehler et al. 2001; Liu et al. 2009] can perform much stronger stabilization, and even simulate 3D motions such as linear camera paths. In this approach, a 3D model of the scene and camera motion are reconstructed using Structure-From-Motion (SFM) techniques [Hartley and Zisserman 2000], and then novel views are rendered from a new, smooth 3D camera path. The problem with 3D stabilization is the opposite of 2D: the motion model is too complex to compute quickly and robustly. As we discuss in more detail in Section 2.1, SFM is a fundamentally difficult problem, and the generality of current solutions is limited when applied to the diverse camera motions of amateur-level video. In general, requiring

This work was funded in part by NSF award IIS-04016284 and a gift from Adobe Systems, Inc.
Authors' addresses: F. Liu (corresponding author), Portland State University, P.O. Box 751, Portland, OR 97207; email: fliu@cs.wisc.edu; M. Gleicher, Department of Computer Sciences, University of Wisconsin-Madison, 1210 West Dayton St., Madison, WI 53706; J. Wang, H. Jin, and A. Agarwala, Adobe Systems, Inc.

3D reconstruction hinders the practicality of the 3D stabilization pipeline.

In this article, we introduce a novel video stabilization technique that combines the advantages of 2D and 3D video stabilization. That is, our method achieves the strongly stabilized, high-quality appearance of 3D stabilization and the efficiency and robustness of 2D methods. Both 2D and 3D stabilization methods can be summarized by three steps: (1) track scene points; (2) compute where the tracked points should be located in the output to stabilize the video content; (3) render an output video which both follows those point locations and looks natural. The first tracking step is well studied in computer vision, and the content-preserving warps proposed by Liu et al. [2009] address the last step. The second step is the key challenge of stabilization: it must plan new, smooth motion trajectories that respect geometric relationships between points, so that they appear as the motion of a plausible, nondistorted view of the scene. 2D approaches enforce plausibility by limiting changes to 2D transformations, which is simple but too limiting. 3D approaches reconstruct a 3D model of the scene and use it to enforce the validity of synthesized views. However, performing 3D reconstruction is error prone and overkill for the stabilization problem.

An ideal constraint should restrict the smoothed motion trajectories to visually plausible solutions without being too restrictive, and be efficient and robust to compute. Our key insight is that we can achieve such a constraint by leveraging a well-known result in computer vision that a matrix of motion trajectories of a rigid scene imaged by a moving camera over a short period of time should approximately lie in a low-dimensional subspace [Tomasi and Kanade 1992; Irani 2002]. We extend this idea by applying the constraint to a moving window over the length of a potentially long video; that is, we efficiently compute a time-varying subspace through moving factorization. We show that we can achieve visual plausibility by performing motion smoothing in this time-varying subspace rather than directly on the original 2D trajectories. The result is the first approach to video stabilization that achieve the strongly stabilized, high-quality appearance of 3D stabilization methods, with the efficiency, robustness, and generality of 2D ones.

Our novel *subspace approach* to video stabilization consists of four steps. First, we use standard 2D point tracking and assemble the 2D trajectories of sparse scene points into an incomplete trajectory matrix. Second, we perform moving factorization to efficiently find a time-varying subspace approximation to the input motion that locally represents the trajectories as the product of basis vectors we call *eigen-trajectories* and a coefficient matrix that describes each feature as a linear combination of these eigen-trajectories. Third, we perform motion planning (or smoothing) on the eigen-trajectories, effectively smoothing the input motion while respecting the low-rank relationship of the motion of points in the scene. Fourth, the eigen-trajectories are remultiplied with the original coefficient matrix to yield a set of smoothed output trajectories that can be passed to a rendering solution, such as content-preserving warps [Liu et al. 2009], to create a final result.

Our method achieves the high-quality stabilization results seen in 3D stabilization, without computing a 3D reconstruction. On videos where SFM performs well, our results are comparable to Liu et al. [2009], but our methods are much more efficient and even allow a streaming implementation. Furthermore, our approach can handle a much wider range of inputs that are challenging for SFM, such as videos that lack parallax, or exhibit camera zoom, in-camera stabilization, or rolling shutter [Meingast et al. 2005] artifacts.

2. RELATED WORK

Two-dimensional video stabilization techniques work by estimating a 2D motion model (such as an affine or projective transform) between consecutive frames, computing new motions that remove high-frequency jitter, and applying per-frame warps to achieve the new motion [Morimoto and Chellappa 1997; Matsushita et al. 2006]. Standard 2D stabilization is robust and efficient, but can only achieve limited smoothing, since 2D warps cannot account for the parallax induced by a moving camera. While recent 2D methods have attempted more aggressive smoothing, for example, by carefully planning interpolation in a transform space [Gleicher and Liu 2008; Chen et al. 2008] or directly optimizing long trajectories [Lee et al. 2009], the inability to accommodate parallax fundamentally limits the amount of smoothing possible.

Three-dimensional video stabilization, which was introduced by Buehler et al. [2001], instead begins by computing a 3D model of the input camera motion and scene. Image-based rendering techniques can then be used to render novel views from new camera paths for videos of static scenes [Fitzgibbon et al. 2005; Bhat et al. 2007]. Dynamic scenes are more challenging, however, since blending multiple frames causes ghosting. Zhang et al. [2009] avoid ghosting by fitting a homography to each frame; this approach cannot handle parallax, however. Liu et al. [2009] introduced content-preserving warps as a nonphysically realistic approach to rendering the appearance of new camera paths for dynamic scenes. In this method, the reconstructed 3D point cloud is projected to both the input and output cameras, producing a sparse set of displacements that guide a spatially varying warping technique.

2.1 Structure from Motion

While 3D stabilization techniques can achieve high-quality camera motions through extremely stabilized 3D camera paths, their practicality is limited by the need to perform 3D reconstruction through Structure-From-Motion (SFM). SFM is an actively researched topic in computer vision [Hartley and Zisserman 2000]. While the state-of-the-art in 3D reconstruction is advancing rapidly, there are fundamental issues that make a robust, efficient, and general solution challenging. The problem is inherently nonlinear and often has ambiguities, so most methods make restrictive assumptions about the input and/or resort to large-scale nonlinear optimization.

SFM has issues with robustness and generality because some videos simply do not contain sufficient motion information to allow for reconstruction. These issues are common in amateur-level video.

(1) *Lack of parallax.* SFM is underdetermined if the camera motion does not provide sufficient parallax, for example, if it pans rather than translates or contains large, flat regions such as a person in front of a wall. Techniques, such as Torr et al. [1999], can discover degenerate motions and switch motion models in response, but this adds yet another moving part to the system that has the potential to fail.

(2) *Camera zooming.* Differentiating between camera zoom and forward motion is a well-known problem in SFM, so many techniques assume that the camera is calibrated or that its internal parameters are fixed.

(3) *In-camera stabilization.* Most modern video cameras damp camera shake either optically or digitally, effectively changing the internal parameters of the camera on a per-frame basis. Again, most SFM techniques assume the camera is calibrated or that its internal parameters are fixed.

(4) *Rolling shutter.* Most new consumer-level video cameras employ a CMOS sensor that does not expose a frame all at once,

but rather in a time-varying fashion from top-to-bottom, causing wobble and shear in the video. This time variation causes severe problems for SFM [Meingast et al. 2005], although recent results show the possibility of removing these artifacts [Liang et al. 2008; Baker et al. 2010; Forssén and Ringaby 2010].

Efficiency is also a problem, since SFM typically requires global nonlinear optimization. Most SFM implementations are not streamable (i.e., they require random access to the entire video rather than just a window surrounding the current frame) since they need to perform multiple iterations of optimization. A few real-time SFM systems have been demonstrated, for example, Nister et al. [2004] and Davison et al. [2007]; however, they all require a calibrated video camera. Also, these systems focused on camera motion recovery rather than scene reconstruction, and thus yield very sparse 3D reconstruction which might not provide enough information for 3D video stabilization. In the end, it may be possible to create a nearly real-time, streamable SFM system that handles all of the aforesaid challenges, since all of these topics have been addressed individually. However, to the best of our knowledge no such system exists, and would certainly represent a formidable engineering challenge. In contrast, our method is simple and requires no special handling of the preceding challenges, since none of them changes the subspace properties of motion trajectories on which our technique relies. Furthermore, our method is nearly real time, performs only linear algorithms, and can be computed in a streaming fashion.

3. OUR APPROACH

We now describe our new approach to planning motion for video stabilization. We first compute a set of sparse 2D feature trajectories using the standard KLT approach [Shi and Tomasi 1994]. Given a set of 2D point trajectories, video stabilization can be split into two problems: (1) where should those points be located in the output to stabilize the video content, and (2) how do we render an output video which both follows those point locations and looks natural? To more formally define the first problem, which is our focus, we begin with a set of N input feature trajectories across F frames whose ith trajectory is $\{(x_t^i, y_t^i)\}$, where (x_t^i, y_t^i) are coordinates at frame t. These trajectories can be assembled into a trajectory matrix M.

$$\mathrm{M}_{2N\times F} = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_F^1 \\ y_1^1 & y_2^1 & \cdots & y_F^1 \\ & & \vdots & \\ x_1^N & x_2^N & \cdots & x_F^N \\ y_1^N & y_2^N & \cdots & y_F^N \end{bmatrix} \quad (1)$$

Note that this matrix is highly incomplete, since trajectories will appear and disappear over the duration of the video. We show the occupancy of a typical trajectory matrix in Figure 1.

Our task is to create a new matrix of trajectories $\hat{\mathrm{M}}$ that guides the rendering of a new, stabilized video, either by traditional full-frame warping or by content-preserving warps. This new matrix should both contain smooth trajectories and be consistent with the original 3D scene imaged by a moving camera. The latter property is hard to satisfy accurately without actually reconstructing the input geometry; however, as we show in this section we have found that visual plausibility can be achieved if we preserve the low-rank property of apparent motion while we smooth the trajectories. We first describe what happens if we do not preserve this property, and then describe our technical approach in more detail.

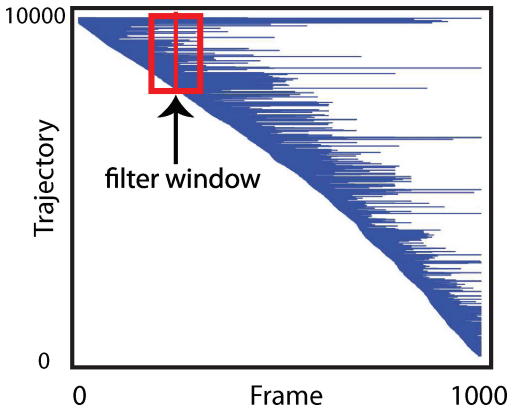


Fig. 1. A typical incomplete trajectory matrix, with each tracked 2D point moving from left to right across the matrix. Note that there are many more trajectories than frames, and the y-axis is scaled to make the matrix appear roughly diagonal. The red box indicates a typical low-pass filter kernel.

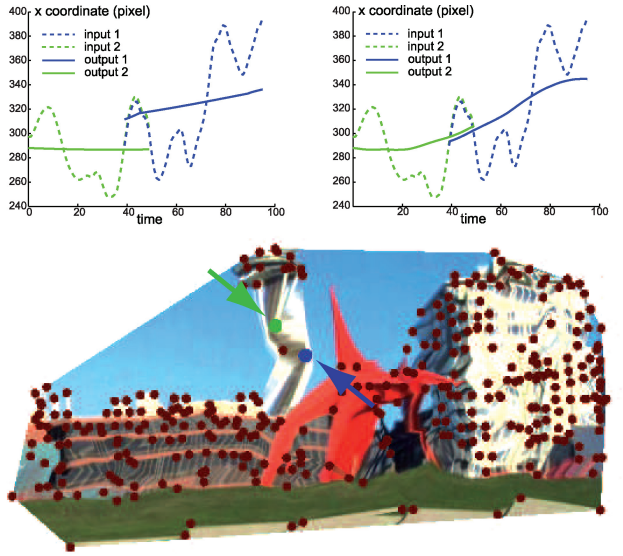
3.1 Simple Trajectory Filtering

We first assume that we wish to create a new motion by low-pass filtering the original motion across time (we address canonical motion paths, such as lines or parabolas, in Section 3.7.2). We also assume the scene is static, and consider moving scene content in Section 3.6. We would like to use large, strong smoothing kernels to achieve the strongly stabilized look of 3D video stabilization. Specifically, we employ a standard Gaussian low-pass filter with a standard deviation of $\sigma = w/\sqrt{2}$, where w is the radius (half the window size) of the filter in frames. In our experience, kernels with a radius of 20–200 frames for a 30 fps video well describe a spectrum from spline-like motions (50) to almost linear motions (200), though the effect will depend on the smoothness of the input. Our default filter radius is 50, which is much stronger than the filtering typically performed in 2D stabilization. For example, Matsushita et al. [2006] reported a typical kernel radius of 6. Ideally, this smoothing would be performed on the recovered 3D camera motion, but since we are not performing SFM we do not have access to this information.

What if we simply filtered the trajectory matrix directly, that is, $\hat{\mathrm{M}} = \mathrm{MK}$ where K is a low-pass filter kernel? This is the same as applying a low-pass filter to each trajectory individually via convolution. While such an approach does not explicitly constrain the relationships between points, the fact that the filter is linear and applied in the same way to all points implicitly preserves properties of the relationships between points. However, because the matrix M is not complete, the filtering operation is not linear; each point receives different treatment (based on its incompleteness), and therefore interpoint relationships are broken. The visual result of this naive approach is very poor; as we show in Figure 2, the geometry of the scene is clearly not respected.

One intuitive way to understand why this result is so poor is to examine what happens to nearby feature trajectories with different durations near their temporal boundaries, as shown in Figure 2. Because these trajectories have different temporal support regions for the smoothing kernel, the strength of the smoothing can differ significantly for nearby features, thus distorting local geometry.

One can imagine a number of simple solutions to this problem. One would be to simply discard the beginning and end of each feature trajectory, so that the kernel domain is always fully supported. However, since we use such large smoothing kernels this solution



(a) Filter each trajectory independently

(b) Filter the eigen-trajectories

Fig. 2. Subspace low-pass filtering. Top-left plot: A low-pass filter on two input trajectories (dashed lines) creates very different outputs (solid lines) for two similar trajectories, since their durations (and thus filter supports) are different, leading to broken geometric relationships in the rendered output (a). Top-right plot: if, instead, the trajectory matrix is completed using matrix factorization (not shown), the filter outputs are more similar, leading to a better rendered result (b). Note that these renderings are created using a 2D triangulation of the points to make the differences clearer.

is not practical, as there are often not enough trajectories that are long enough to support the warping stage (this problem most often occurs during camera panning, since features can enter and leave the field of view quickly). Another solution would be to extend each feature trajectory in duration using some sort of extrapolation or prediction. We experimented with standard extrapolation using a polynomial model, and the results were very poor; since each trajectory is extrapolated independently, geometric relationships with nearby features were again not preserved.

We next describe how subspace constraints can be used to perform this extrapolation in a fashion that better preserves relationships between feature trajectories.

3.2 Subspace Constraints

While 3D reconstruction would easily allow us to express geometric relationships between features, it is also overkill for our purposes, since we do not need to know the depths of points in the scene; we only need constraints that allow us to preserve visual plausibility. Computer vision results suggest such a possible constraint that is weaker than a full reconstruction. Tomasi and Kanade [1992] were the first to observe that when a rigid 3D scene is imaged by a moving *affine* camera, the observed motion trajectories should reside in a low-dimensional subspace. This important constraint has been used to help solve a number of problems in computer vision, including structure from motion [Tomasi and Kanade 1992], correspondence [Irani 2002], and motion segmentation [Vidal et al. 2008]. Under this subspace constraint, the trajectory matrix can be factored into a product of a camera matrix, representing each frame, and a scene matrix, representing each tracked point. If the trajectory matrix is complete, this factorization process is linear, fast, and robust. For an incomplete matrix, factorization is a nonlinear problem, but well-studied [Brand 2002; Buchanan and Fitzgibbon 2005; Chen 2008].

Unfortunately, for the more common case of perspective cameras, the rank constraint becomes more complicated. In general, motion trajectories from a perspective camera will lie on a nonlinear manifold instead of a linear subspace [Goh and Vidal 2007]. However, it is possible to approximate the manifold locally (over a short period of time) with a linear subspace. In particular, Irani [2002] showed that for instantaneous motions a trajectory matrix should have at most rank 9. In this article, we assume this property holds over a short window of frames that is at least as large as our temporal smoothing kernel. We evaluate the accuracy of this assumption in more detail in Section 4.2, but the approximation seems sufficient for the purpose of insuring plausible views for video stabilization.

3.3 Filtering with Subspace Constraints

We now show how to filter the trajectory matrix while maintaining this low-rank constraint. Consider the n trajectories that appear over a small window of the first k frames of our input sequence. Over this range of k frames, we assume that the nonlinear manifold on which the motion data lie can be locally modeled with a linear subspace of rank r. We use $r = 9$, as suggested by Irani and because we empirically found it to model the data well without overfitting or underfitting. This low-rank constraint implies that we can factor the submatrix of the first k frames into the product of two low-rank matrices

$$\mathrm{M}_{2n\times k} \approx \mathrm{W} \odot (\mathrm{C}_{2n\times r}\mathrm{E}_{r\times k}), \tag{2}$$

where W is a binary mask matrix with 0 indicating missing data and 1 indicating existing data, and $\odot$ means component-wise multiplication (we describe how we perform this factorization later). We call the r row vectors of E eigen-trajectories, in that they represent the basis vectors that can be linearly combined to form a 2D motion trajectory over this window of k frames. The coefficient matrix C represents each observed feature as such a linear combination.

This factorization provides a straightforward way to smooth the trajectory matrix while preserving its rank. We can first fill in the missing data, and then low-pass filter the complete matrix and drop the elements corresponding to the missing data. But it turns out that it is not necessary to first complete the missing data as smoothing is a linear operation which can be represented as a matrix multiplication, and matrix multiplication is associative

$$\hat{\mathrm{M}} = \mathrm{W} \odot (\mathrm{CE})\mathrm{K} = \mathrm{W} \odot \mathrm{C}(\mathrm{EK}) = \mathrm{W} \odot \mathrm{C}\hat{\mathrm{E}}, \tag{3}$$

where $\hat{\mathrm{E}} = \mathrm{EK}$. In other words, it is equivalent to first low-pass filtering the eigen-trajectories E to obtain $\hat{\mathrm{E}}$, and then obtaining a new submatrix $\hat{\mathrm{M}}_{2n\times k}$ by multiplying $\hat{\mathrm{E}}$ with the original coefficient

matrix C and dropping the elements corresponding to the missing data. We adopt the latter strategy as it is more efficient to filter the eigen-trajectories. Also, for nonlinear smoothing operations such as canonical path fitting, operating directly on the eigen-trajectories allows us to preserve the low-rank property whether or not the smoothing operation would do so if applied directly to the trajectories themselves.

The result of the aforesaid factorization is a set of eigen-trajectories that can take any partial trajectory through the first k frames and complete it to a full trajectory. The final step before smoothing is to extend this approach across the duration of the entire video. Remember that while the entire matrix M may not be well modeled with a low-rank subspace because the data lie on a nonlinear manifold, we assume that it is over a range of k frames. This property implies that for any range of k frames in the matrix in Figure 1, a local factorization can be computed that well models the portion of the matrix over the k frames that has existing values. To support our low-pass filter, the factorization only needs to be able to extend each trajectory forwards and backwards in time by the radius of the smoothing kernel; thus, the factorization does not need to be accurate everywhere, but only locally near the original trajectory values (e.g., near the diagonal of Figure 1). Given a factorization of the first k frames, we need to propagate it forwards in a fashion that is consistent with the existing factorization, and explains the new data as well. In essence, we need to track a time-varying subspace. We do so in a greedy, moving fashion as we now describe in detail.

3.4 Moving Factorization

We employ a moving factorization approach that is customized to our application and designed to be efficient, scalable, and streamable. In short, we perform factorization in a fixed window of k frames, and move that window forward δ frames at each step (we use values $k = 50$ and $\delta = 5$).

Our algorithm starts by factoring the first k frames. Fortunately, for our application there should be a reasonable number of complete trajectories that already span all k frames and describe the subspace. We therefore take these m complete feature trajectories and use them to assemble a trajectory matrix $\mathrm{M}^0_{2m\times k}$, which is a complete submatrix of $\mathrm{M}_{2n\times k}$ defined in Eq. (2). Note that m must at least be as large as $r/2$ to make the subspace constraint meaningful, and in practice should be much larger. In the rare cases where $m < 2r$, we reduce k until there are a sufficient number of trajectories. We then factor M^0 as follows.

$$\mathrm{M}^0_{2m\times k} = C_{2m\times r} E_{r\times k} \tag{4}$$

The factorization is calculated by truncating the output of SVD [Golub and Van Loan 1996] to the rows, columns, and values corresponding to the largest r singular values, and then distributing the square root of each singular value to the left and right matrices.

Given a factorization of a window M^0, we compute the factorization M^1 of the next window (moved forward δ frames) in the same fashion; note that M^1 is also a complete matrix. Since the factorization windows are highly overlapped, the corresponding trajectory matrices M^0 and M^1 are also highly overlapped. As shown in Figure 3, by matrix permutation, M^0 and M^1 can be reorganized as $\mathrm{M}^0 = \begin{bmatrix} A^{00} & A^{01} \\ A^{10} & A^{11} \end{bmatrix}$ and $\mathrm{M}^1 = \begin{bmatrix} A^{11} & A^{12} \\ A^{21} & A^{22} \end{bmatrix}$, respectively, where A^{11} is shared between M^0 and M^1. Note that the factorization of $A^{11} = C^1 E^1$ was already computed when M^0 was factored, so we

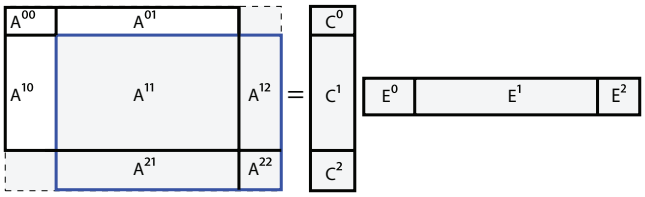


Fig. 3. Moving matrix factorization. The factorization for an additional δ frames is computed by keeping C^0, C^1, E^0, and E^1 fixed and computing C^2 and E^2. The blue box indicates matrix M^1.

keep these values fixed and process M^1 as follows.

$$\mathrm{M}^1 = \begin{bmatrix} A^{11} & A^{12} \\ A^{21} & A^{22} \end{bmatrix} = \begin{bmatrix} C^1 \\ C^2 \end{bmatrix} \begin{bmatrix} E^1 & E^2 \end{bmatrix} \tag{5}$$

We wish to estimate C^2 and E^2 in a fashion that is both consistent with the already computed factorization (i.e., C^1 and E^1) and the new data (i.e., A^{12}, A^{21}, and A^{22}). We do so in a least-squares fashion by minimizing

$$\min_{C^2,E^2} \|C^2E^1 - A^{21}\|_F^2 + \|C^2E^2 - A^{22}\|_F^2 + \|C^1E^2 - A^{12}\|_F^2, \tag{6}$$

where $\|\cdot\|_F$ stands for the matrix Frobenius norm. Note that this is a bilinear optimization problem. However, since we want an efficient solution we choose to solve it approximately in a linear fashion. Since the factorization window moves forward a small number of frames per iteration, the size of A^{22} is significantly smaller than A^{21} and A^{12}. We therefore solve this problem by first estimating C^2 as the projection of A^{21} onto E^1,

$$C^2 = A^{21}{E^1}^T(E^1{E^1}^T)^{-1}, \tag{7}$$

and then solve for E^2 as follows.

$$E^2 = \left(\begin{bmatrix} C^1 \\ C^2 \end{bmatrix}^T \begin{bmatrix} C^1 \\ C^2 \end{bmatrix}\right)^{-1} \begin{bmatrix} C^1 \\ C^2 \end{bmatrix}^T \begin{bmatrix} A^{12} \\ A^{22} \end{bmatrix} \tag{8}$$

We find that this linear solution to the bilinear problem (6) is nearly as accurate as those obtained through nonlinear optimization techniques such as Levenberg-Marquardt.

Note that all the matrices in Eqs. (4)–(8) are complete. The final step in performing the moving factorization is to handle the missing data by computing the coefficients for those trajectories that were too short to be included in matrix M^1. We compute the coefficients for any trajectory whose duration ends in the current factorization window and whose coefficients are not already computed by projecting it onto the eigen-trajectories, as in Eq. (7).

3.5 Algorithm Summary

In summary, our algorithm for subspace video stabilization proceeds as follows.

(1) Estimate 2D feature trajectories from the input video.
(2) Assemble a feature trajectory matrix, and factor it in a moving fashion into two low-rank matrices, a coefficient matrix and an eigen-trajectories matrix.
(3) Smooth the eigen-trajectories and obtain smooth output feature trajectories by multiplying with the original coefficient matrix.
(4) Warp the input video with the guidance of the new feature trajectories using content-preserving warps [Liu et al. 2009].

3.6 Dynamic Scene Content

The low-rank constraints of which we take advantage do not hold for moving objects in the scene. So, we need to remove as many of these outlier trajectories as possible before performing factorization and content-preserving warps (a small number of remaining outliers can be treated by the factorization algorithm as noise). We remove such trajectories using three strategies. First, we have found that trajectories on moving objects are typically much shorter than those on rigid objects. Therefore, trajectories shorter than 20 frames are removed entirely. Also, trajectories shorter than k frames do not influence the eigen-trajectories, since factorization is only computed on trajectories that are complete over the window. Second, we rule out motion outliers using the fundamental matrix constraint [Hartley and Zisserman 2000]. We estimate a fundamental matrix between every 5 frames within a RANSAC loop [Fischler and Bolles 1981], and remove a trajectory when it deviates from the epipolar constraint by more than 1 pixel for more than 1/3 of the duration our algorithm has processed. Third, after factorization is computed we remove trajectories with large factorization error, which we classify as any trajectory whose per-frame error ever exceeds 3 pixels. We could recompute factorization after removing these outlier trajectories, but we found the improvement in results not worth the computational expense.

3.7 Smooth Motion Planning

Once the eigen-trajectories are computed using moving factorization, the final task before rendering is to smooth the eigen-trajectories to simulate smooth camera motion. Note that, unlike 3D video stabilization, there is no need to linearize the non-Euclidean space of camera orientations [Lee and Shin 2002], since in our case apparent motion is already represented with a linear approximation. This fact greatly simplifies our motion planning compared to Liu et al. [2009]. We support several approaches to motion planning, including simple low-pass filtering, automatic polynomial path fitting, and interactive spline fitting.

3.7.1 *Low-Pass Filtering.* The simplest approach to smooth camera motion is to just run a low-pass filter over the eigen-trajectories. The advantage of this method is that it works for any length of video, and fits within a streaming framework where only a window of frames around the current frame need be accessible. Specifically, our technique only requires access to a window of $\max(k, 2w)$ frames centered at the current frame, where k is the factorization window size and w is the radius of the smoothing kernel. We offer a range of kernel sizes, though our default is $w = 50$. We can support much larger kernels than 2D video stabilization since our technique is not based on a 2D motion model, and can therefore account for parallax in the input.

3.7.2 *Polynomial Path Fitting.* As observed by Liu et al. [2009], some of the most dramatic cinematographic effects are created by moving a camera along a very simple path, such as a line or parabola. In our case, we cannot achieve such motions exactly since we do not know the 3D camera projection matrices. However, we have found that we can achieve qualitatively similar results by fitting polynomial models to the eigen-trajectories.

Our method currently supports three polynomial motion models: constant, linear, and quadratic. We represent a polynomial motion model for the eigen-trajectories as $\hat{E}_t = \sum_{j=0}^{d} K_j t^j$, where $\hat{E}_t$ is a vector containing the values of the new eigen-trajectories at frame t, d is the degree of the polynomial, and each K_j is an unknown r-element vector that is the coefficient for the polynomial term. Degree-d polynomial eigen-trajectories lead to degree-d polynomial feature trajectories: $(x_t^i, y_t^i) = (C_{2i} \sum_{j=0}^{d} K_j t^j, C_{2i+1} \sum_{j=0}^{d} K_j t^j)$.

Our method computes the K_j coefficients of this polynomial model for the output eigen-trajectories as the best polynomial approximation of the input eigen-trajectories. Specifically, we minimize the displacement between the new position and the original position of every feature point.

$$\min_{\hat{E}} \|\mathrm{W} \odot (\mathrm{C}\hat{\mathrm{E}} - \mathrm{CE})\|_F^2 \tag{9}$$

The optimal polynomial eigen-trajectories can be computed by solving the aforesaid linear system. Note again that the result of this process creates a 2D polynomial path for each output trajectory, which is different than fitting a 3D polynomial to the camera's motion; however, we have found the visual result to be similar. Also, this approach to planning camera motion requires access to the whole video, and cannot be computed in a streaming fashion.

3.7.3 *Interactive Spline Fitting.* Polynomial path fitting is often not appropriate for long videos, since their motion cannot be well modeled with a single polynomial. In these cases, we provide an interface that allows the user to select several keyframes. We then fit a standard spline to the eigen-trajectories with knots at the keyframes. The user can choose between quadratic and cubic splines. Since the feature trajectories are linear combinations of the eigen-trajectories, a quadratic (or cubic) spline for the eigen-trajectories leads to a quadratic (or cubic) spline for each output feature trajectory. This motion planning approach also cannot be computed in a streaming fashion.

4. EXPERIMENTAL VALIDATION

Before evaluating the visual quality of our results, we first describe three numerical experiments that experimentally validate the properties of our algorithm: the improvement in quality from eigen-trajectory smoothing, the accuracy of our factorization scheme, and robustness to rolling shutter artifacts.

4.1 Eigen-Trajectory Smoothing

The main contribution of our article is a new smoothing approach that incrementally factorizes the input trajectories and then smoothes the resultant eigen-trajectories. To evaluate the improvement in quality created by this technique, we took our overall pipeline summarized in Section 3.5 and replaced these two steps (i.e., steps 2 and 3) with a simple Gaussian smoothing of each trajectory individually. The Gaussian filter kernels were normalized at the beginning and end of the trajectory durations. This experiment is similar to the one shown in Figure 2, but with the output rendered using content-preserving warps rather than triangulation. We applied the experiment on two video sequences, and used smoothing kernel radii of 40, 50, and 60; the resultant videos are included as supplemental materials accessible at our project Web site.[1] It is clear from the results that smoothing each trajectory individually introduces significantly more distortion than smoothing the eigen-trajectories, especially when we smooth the trajectories aggressively.

4.2 Factorization Accuracy

Our incremental factorization approach is an approximation in several ways. For one, our assumption that motion trajectories over k

[1] http://www.cs.pdx.edu/∼fliu/project/subspace_stabilization

frames can be described with a low-rank subspace is only approximately true. Second, our factorization approach is greedy, starting from the first k frames and moving forward using a linear approximation to the bilinear fitting; some sort of nonlinear optimization may yield more accurate results.

We evaluated the error of our factorization approach by computing the mean factorization error, that is, the difference between the original trajectory and the same trajectory reconstructed from the subspace. (Note that we compute this error before removing any large-error outlier trajectories, as described in Section 3.6.) For a diverse set of 70 videos resized to 640×360, the mean error per video ranged from 0.08 to 0.26 pixels. Next, we experimented with several iterative global optimization techniques such as the damped Newton method [Buchanan and Fitzgibbon 2005] and Levenberg-Marquardt [Chen 2008]. We found that these methods significantly reduced factorization error (by a factor of 4, on average), at the expense of much longer computation times. However, this improvement in error did not significantly improve the visual quality of our results, perhaps because the error is already subpixel. It is also worth noting that we do not need as precise a factorization as some other applications, just one that leads to visual plausibility. We therefore choose to use our efficient, streamable approach.

4.3 Rolling Shutter Video

Most recent consumer-level video cameras have CMOS sensors that can only be read-out in a sequential fashion from top to bottom. The result is that a video frame does not record a snapshot of time; instead, time varies across the frame, resulting in aliasing that can be seen as skewing or wobbling of objects in the video. This property of video poses serious challenges for both SFM and traditional 2D video stabilization [Liang et al. 2008; Meingast et al. 2005].

In practice, we've found that our approach works well on rolling shutter videos. While we cannot make strong theoretical arguments for why rolling shutter videos preserve the subspace constraint, we believe that the artifacts appear as structured noise to our algorithm, which tends to be robust to noise. However, to confirm the performance of our algorithm on rolling shutter videos, we constructed an experiment to challenge it. We took 30 videos shot with a 3-CCD camera (and thus free of rolling shutter) and approximately simulated a rolling shutter. Then, we compared factorization error before and after the rolling shutter.

For our experiments we modeled rolling shutter similar to Liang et al. [2008] by shifting each track according to its scanline. That is, for each feature trajectory at frame t we calculate its new position, $\tilde{p}_t = (x_t, y_t)$ by shifting it in time by λ and interpolating its position at consecutive frames

$$\tilde{p}_t = (1 - \lambda)p_t + \lambda p_{t+1}, \tag{10}$$

where p_t and p_{t+1} are its position at time t and $t + 1$, respectively. These two coordinates are obtained from the tracking results of the nonrolling shutter video, and λ is the shift in time. The value of λ depends on the camera and vertical coordinate, that is, $\lambda = \kappa(y_t/H)$ where H is the height of the video frame. The parameter κ depends on the amount of rolling shutter introduced by the camera, and is typically less than 1. In our experiments, we set it as 1, which might exaggerate the rolling shutter effect.

We then performed factorization on the original trajectory matrix and the simulated rolling shutter matrix. We considered only the first 100 frames of each sequence and only used trajectories that spanned that entire duration, thus yielding a complete matrix that we can factorize using SVD. We found that the mean factorization errors for these rolling shutter matrices are reasonably close to the original matrices: on average, the rolling shutter effect increases the factorization error by 13.5%. Note that rolling shutter can also negatively impact 2D tracking, since it introduces local distortion, and our experiments do not measure this impact. Also, our method only addresses stabilization in the presence of rolling shutter wobble introduced by camera shake. We do not perform general rolling shutter artifact removal, or handle more structured artifacts such as the shear introduced by a fast, intentional panning motion.

These empirical experiments show that the effect of rolling shutter on the subspace constraint is relatively minor and can be treated as structured noise. We thus took another 40 videos using a rolling shutter camera, and performed our incremental factorization approach. The mean reconstruction error for these rolling shutter videos was 0.165 pixels, compared to the 0.135 error of nonrolling shutter videos. We find that, in general, our method produces visually good results for rolling shutter sequences.

5. RESULTS

We show a number of results in our project Web site. We tested our approach on 109 video sequences, from 5 to 180 seconds, captured by a variety of people and cameras in many different scenes. Of these videos, 48 were captured with a 3-CCD camera without a rolling shutter, and 61 were captured with a CMOS HD camera with a rolling shutter. We also compared our results to the 3D stabilization approach of Liu et al. [2009] and to the publicly available 2D stabilization features of iMovie '09[2] and Deshaker.[3]

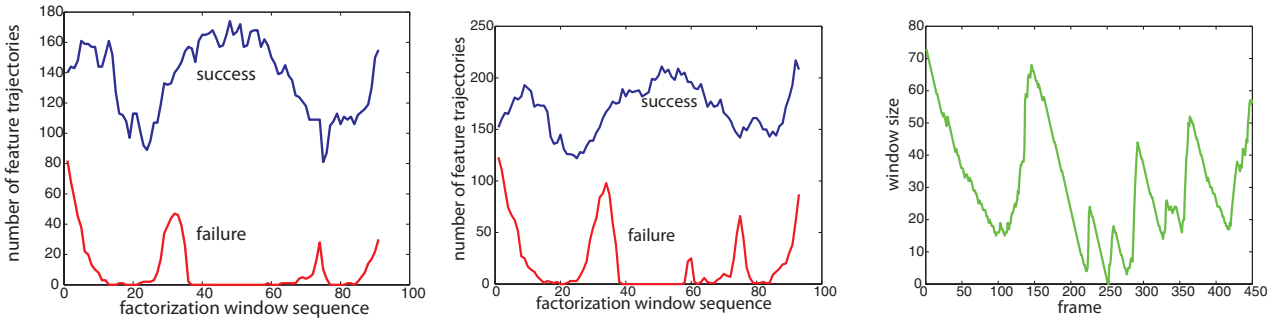
Our first set of experiments evaluates our method on the 32 videos used by Liu et al. [2009] to successfully demonstrate 3D video stabilization. Note that these sequences were specifically chosen to be friendly to SFM: they were shot by a person walking continuously to provide sufficient parallax, and exhibit no zooming or in-camera stabilization in order to keep the internal parameters of the camera fixed. We found that our method produced qualitatively similar results for 25 of these videos, and better results for 7 of these videos. Our comparative ratings are subjective, and were produced by two students who watched the video and came to agreement on the ratings. We consider two results as similar if the camera stability and the degree of artifacts are similar, even if the exact motions and field of view of the results are slightly different. We include several such results in the supplemental video.

Next, we ran our system on 15 new videos captured with camera motions known to challenge SFM. Like Liu et al. [2009] we use the Voodoo[4] camera tracker, and consider its output failed since Voodoo produced a clearly incorrect reconstruction. The types of camera motions that we found to be challenging for SFM include: (1) sequences that include periods of both walking, and panning while standing still, (2) sequences with significant rolling shutter, (3) sequences with in-camera stabilization left on, and (4) sequences with changing focal length. We include several examples in the accompanying video, along with comparisons to 2D stabilization; we also provide three videos that Voodoo failed to reconstruct, along with the results from our method and Liu et al. [2009], in the supplemental materials. Note that other SFM implementations may perform better or worse than the one we tested, but all of the preceding are well-known problems for SFM.

[2]http://www.apple.com/ilife/imovie/

[3]http://www.guthspot.se/video/deshaker.htm

[4]http://www.digilab.uni-hannover.de



(a) window size: 50 (b) window size: 40 (c) maximal window size at each frame

Fig. 4. Plots (a) and (b) show the number of feature trajectories covering each factorization window for two example videos (with window sizes $k = 50$ and $k = 40$, respectively). Our method succeeded for the blue line and failed for the red one. Plot (c) shows the maximal size that a window at each frame can have to guarantee that at least 20 trajectories cover it for the failure case video.

We are particularly interested in how our method works on rolling shutter videos since more and more new video cameras exhibit this artifact. Out of 61 rolling shutter videos, we found that 46 of our results were completely successful, 12 were moderately successful, and 3 were not successful. We consider a result moderately successful if it suffers from slight artifacts, like bouncing or wobbling locally, but is clearly preferable to the input. Rolling shutter is particularly challenging for 2D stabilization, since full-frame warps cannot remove the local wobbling it introduces. We ran iMovie '09 and Deshaker (which has a special feature designed for handling rolling shutter) on our examples. We found that our algorithm performs significantly better than iMovie for 47 out of 61 videos, and moderately better than iMovie for 11 out of 61 videos. Our algorithm performs significantly better than Deshaker for 42 out of 61 videos, and moderately better than Deshaker for 16 out of 61 videos. We consider our results significantly better than those produced by iMovie or Deshaker when their results suffer obvious high-frequency vibration, while ours do not. We include several examples in the supplemental video. For the remaining 3 out of the 61 videos, our method fails entirely since the videos are heavily dominated by scene motion. For these 3 videos, both iMovie '09 and Deshaker produce results that are visually worse than the input.

On the supplemental video we also demonstrate several additional challenging scenes. These include a very long video, and a video with large amounts of scene motion. We also show examples of different camera path planning, such as low-pass filtering, polynomial paths, and splines.

Overall, of the 109 videos we tested we consider 86 as completely successful, 20 as moderately successful because they exhibit moderate bouncing or other small artifacts, and 3 as failures. The three failure cases are due to excessive shake and scene motion. Our method could not produce any results for these videos since our pipeline could not proceed without the output trajectories from the moving factorization step.

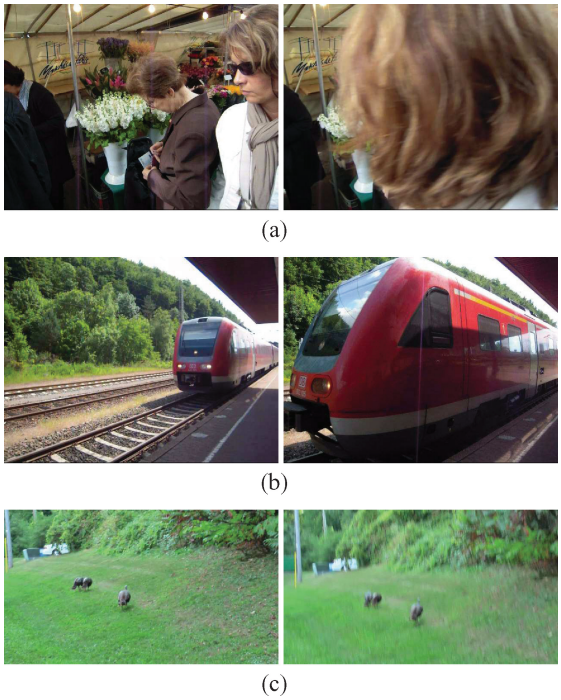
5.1 Limitations

To further evaluate the limitations of our method, we decided to collect an additional 30 "stress-test" videos that were likely to be challenging. Our experience with the first 109 videos suggested that videos with large amounts of scene motion, excessive shake, or strong motion blur were the most difficult for our method, so we intentionally captured videos with these properties. As expected, 13 of these 30 videos were failure cases.

Of those 13 failure cases, 10 failed because there were not enough long trajectories to cover an entire factorization window. In this case, the moving factorization fails to complete and we are unable to even produce an output. To further understand this type of failure, we counted the number of feature trajectories that cover each factorization window for each of the 10 sequences. In this test, the factorization window was moved forward 5 frames at each step, and we tried two different window sizes: $k = 50$ and $k = 40$. For each of the 10 videos, there were several windows with zero trajectories long enough to cover the entire window. We also randomly selected 10 of the successful videos, and found that the minimum number of trajectories covering each window was 80. Figure 4(a) and 4(b) show several plots of the number of feature trajectories covering each window for a successful video (blue) and a failure case (red). There are several possible reasons where there might be an insufficient number of long trajectories: dramatic camera motions that push trajectories out of the field of view, strong motion blur, geometry with little texture, or large moving objects that pass through and occlude the tracked geometry. Two frames of several examples are shown in Figure 5 and included as supplemental materials. One solution to this problem would be to reduce the window size (though it must be at least as large as the rank r). However, for many failure cases there are "bottleneck" frames where trajectories suddenly terminate en masse; in this case, even a very short window cannot be covered by enough trajectories. Figure 4(c) shows the maximum size a window at each frame can have before the number of covering trajectories drops below 20; in this case, even a window size of zero is not sufficient.

Figure 6 shows another common type of failure which accounted for 3 of the 13 failures. In this failure type, while there are enough long feature trajectories, a significant portion of them are on a large moving object that dominates the scene. Our method for removing dynamic outliers, described in Section 3.6, will not succeed if the object dominates the scene. In this case, a single subspace cannot account for both the trajectories in the face and the background, leading to distortions in the result. We include the input and output of this failure case in the supplemental materials.

Finally, like most stabilization techniques, more aggressive stabilization leads to smaller output field of views, since our results are cropped to the largest possible rectangle. Motion blur is also not removed by our method. Our technique could be combined with others that address these issues [Matsushita et al. 2006; Chen et al. 2008].



(a) (b) (c)

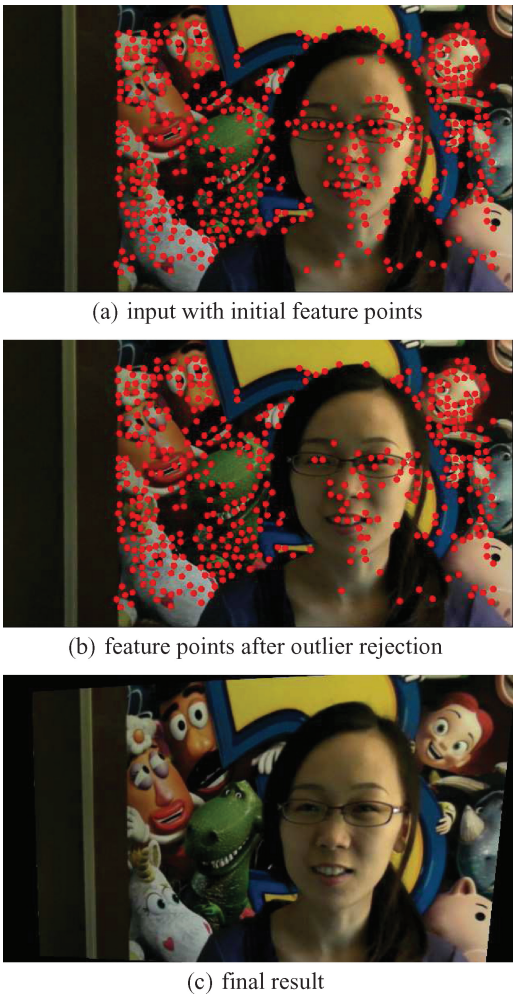
Fig. 5. Failure cases caused by an insufficient number of long feature trajectories. In examples (a) and (b), large objects (a person and a train) pass quickly through the field of view and terminate the trajectories behind them. In example (c), several frames exhibit large motion blur, thus breaking the continuity of the trajectories.

5.2 Performance

The computation of our method consists of three major parts: feature tracking, matrix factorization, and content-preserving warping. We did our experiments on a machine with a 3.16 GHz Intel Dual Core CPU and 3GB of memory, although our implementation does not take advantage of multicore processing. The KLT tracker we used achieves 7 fps when it is tuned to track roughly 500 feature points per frame on the input videos, which are resized to 640×360; note that others have developed GPU-accelerated real-time versions [Sinha et al. 2006]. Our incremental factorization method achieves 500 fps. Our implementation of content-preserving warps [Liu et al. 2009], which builds a 64×36 grid mesh for each frame and solves a linear system, achieves 10 fps; however, we used a generic sparse matrix representation and believe that a customized one would allow us to construct the linear system much more quickly. Overall, our implementation currently achieves 4 fps, and we believe that with the use of parallelization and the GPU we can reach real-time performance. In comparison, Liu et al. [2009] report that their running time was dominated by 3D scene reconstruction using Voodoo; in our experiments, Voodoo takes between 4 and 10 hours for a video of 600 frames (20 seconds).

6. CONCLUSION

In this article, we have provided a technique for video stabilization that can achieve aggressive, high-quality stabilizations on a wide range of videos in a robust and efficient way. We can achieve the appearance of smooth camera motions without creating 3D reconstructions, allowing our approach to operate efficiently and to work in situations where reconstruction is challenging. Therefore, we believe our approach is sufficiently practical to be used in consumer-level video processing tools.



(a) input with initial feature points

(b) feature points after outlier rejection

(c) final result

Fig. 6. A failure case caused by a large moving object (a face). Image (a) shows the initial set of trajectories, while (b) shows the set after outlier rejection. Our method cannot remove all the feature points in the dynamic region, leading to distortions in the result ((c)).

Our technique is based on the observation that accurate scene reconstruction is not necessary if our goal is merely to improve the quality of video. For video processing applications, visually plausible results are sufficient. By aiming for this simpler goal, we can devise methods that avoid solving challenging computer vision problems. In this article, we have successfully applied this strategy to address an important issue for video users: video stabilization. In the future, we hope to apply the strategy to other video processing tasks.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful feedback.

REFERENCES

BAKER, S., BENNETT, E., KANG, S. B., AND SZELISKI, R. 2010. Removing rolling shutter wobble. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2392–2399.

BHAT, P., ZITNICK, C. L., SNAVELY, N., AGARWALA, A., AGRAWALA, M., COHEN, M., CURLESS, B., AND KANG, S. B. 2007. Using photographs to enhance videos of a static scene. In *Proceedings of the 18th Eurographics Workshop on Rendering*. 327–338.

BRAND, M. 2002. Incremental singular value decomposition of uncertain data with missing values. In *Proceedings of the European Conference on Computer Vision*. 707–720.

BUCHANAN, A. M. AND FITZGIBBON, A. 2005. Damped Newton algorithms for matrix factorization with missing data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 316–322.

BUEHLER, C., BOSSE, M., AND MCMILLAN, L. 2001. Non-Metric image-based rendering for video stabilization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 609–614.

CHEN, B.-Y., LEE, K.-Y., HUANG, W.-T., AND LIN, J.-S. 2008. Capturing intention-based full-frame video stabilization. *Comput. Graph. Forum 27,* 7, 1805–1814.

CHEN, P. 2008. Optimization algorithms on subspaces: Revisiting missing data problem in low-rank matrix. *Int. J. Comput. Vis. 80,* 1, 125–142.

DAVISON, A. J., REID, I. D., MOLTON, N. D., AND STASSE, O. 2007. MonoSLAM: Real-time single camera SLAM. *IEEE Trans. Patt. Anal. Mach. Intell. 26,* 6, 1052–1067.

FISCHLER, M. A. AND BOLLES, R. C. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commu. ACM 24,* 6, 381–395.

FITZGIBBON, A., WEXLER, Y., AND ZISSERMAN, A. 2005. Image-Based rendering using image-based priors. *Int. J. Comput. Vis. 63,* 2, 141–151.

FORSSÉN, P.-E. AND RINGABY, E. 2010. Rectifying rolling shutter video from hand-held devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 507–514.

GLEICHER, M. L. AND LIU, F. 2008. Re-cinematography: Improving the camerawork of casual video. *ACM Trans. Multimedia Comput. Comm. Appl. 5,* 1, 1–28.

GOH, A. AND VIDAL, R. 2007. Segmenting motions of different types by unsupervised manifold clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–6.

GOLUB, G. H. AND VAN LOAN, C. F. 1996. *Matrix Computations* 3rd Ed. Johns Hopkins University Press.

HARTLEY, R. I. AND ZISSERMAN, A. 2000. *Multiple View Geometry in Computer Vision*. Cambridge University Press.

IRANI, M. 2002. Multi-frame correspondence estimation using subspace constraints. *Int. J. Comput. Vis. 48,* 1, 39–51.

LEE, J. AND SHIN, S. Y. 2002. General construction of time-domain filters for orientation data. *IEEE Trans. Vis. Comput. Graph. 8,* 2, 119–128.

LEE, K.-Y., CHUANG, Y.-Y., CHEN, B.-Y., AND OUHYOUNG, M. 2009. Video stabilization using robust feature trajectories. In *Proceedings of the IEEE International Conference on Computer Vision*. 1397–1404.

LIANG, C. K., CHANG, L. W., AND CHEN, H. H. 2008. Analysis and compensation of rolling shutter effect. *IEEE Trans. Image Process. 17,* 8, 1323–1330.

LIU, F., GLEICHER, M., JIN, H., AND AGARWALA, A. 2009. Content-preserving warps for 3D video stabilization. *ACM Trans. Graph. 28,* 3, Article No. 44.

MATSUSHITA, Y., OFEK, E., GE, W., TANG, X., AND SHUM, H.-Y. 2006. Full-frame video stabilization with motion inpainting. *IEEE Trans. Patt. Anal. Mach. Intell. 28,* 7, 1150–1163.

MEINGAST, M., GEYER, C., AND SASTRY, S. 2005. Geometric models of rolling-shutter cameras. In *Proceedings of the 6th International Workshop on Omnidirectional Vision, Camera Networks, and Non-Classical Cameras*. 12–19.

MORIMOTO, C. AND CHELLAPPA, R. 1997. Evaluation of image stabilization algorithms. In *Proceedings of the DARPA Image Understanding Workshop*. 295–302.

NISTER, D., NARODITSKY, O., AND BERGEN, J. 2004. Visual odometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 652–659.

SHI, J. AND TOMASI, C. 1994. Good features to track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 593–600.

SINHA, S., FRAHM, J.-M., POLLEFEYS, M., AND GENC, Y. 2006. GPU-based video feature tracking and matching. In *Proceedings of the Workshop on Edge Computing Using New Commodity Architectures*.

TOMASI, C. AND KANADE, T. 1992. Shape and motion from image streams under orthography: a factorization method. *Int. J. Comput. Vis. 9,* 2, 137–154.

TORR, P. H. S., FITZGIBBON, A. W., AND ZISSERMAN, A. 1999. The problem of degeneracy in structure and motion recovery from uncalibrated image sequences. *Int. J. Comput. Vis. 32,* 1, 27–44.

VIDAL, R., TRON, R., AND HARTLEY, R. 2008. Multiframe motion segmentation with missing data using PowerFactorization and GPCA. *Int. J. Comput. Vis. 79,* 1, 85–105.

ZHANG, G., HUA, W., QIN, X., SHAO, Y., AND BAO, H. 2009. Video stabilization based on a 3D perspective camera model. *Vis. Comput. 25,* 11, 997–1008.

Received May 2010; revised October 2010; accepted October 2010