360° Video Stabilization

Johannes Kopf
Facebook

(a) Hand-held 360° cameras (b) Full-spherical 360°×180° video (c) Original motion tracks (d) Pure rotation stabilization (e) Deformed rotation stabilization

Figure 1: *Thanks to new 360° cameras (a) recording 360° videos (b) is now fairly accessible to casual users. However, it is not easy to avoid shake with hand-held cameras (c). We present a new 360° video stabilization algorithm that first removes rotational motion (d), and further optimizes a flexible deformable model to remove even residual shake from translational motion, parallax, and rolling shutter wobble (e).*

Abstract

We present a hybrid 3D-2D algorithm for stabilizing 360° video using a deformable rotation motion model. Our algorithm uses 3D analysis to estimate the rotation between key frames that are appropriately spaced such that the right amount of motion has occurred to make that operation reliable. For the remaining frames, it uses 2D optimization to maximize the visual smoothness of feature point trajectories. A new low-dimensional flexible deformed rotation motion model enables handling small translational jitter, parallax, lens deformation, and rolling shutter wobble. Our 3D-2D architecture achieves better robustness, speed, and smoothing ability than either pure 2D or 3D methods can provide. Stabilizing a video with our method takes less time than playing it at normal speed. The results are sufficiently smooth to be played back at high speed-up factors; for this purpose we present a simple 360° hyperlapse algorithm that remaps the video frame time stamps to balance the apparent camera velocity.

Keywords: 360 video, video stabilization

Concepts: •**Computing methodologies** → Computational photography;

1 Introduction

Throughout the last century of film, we have viewed the world through a narrowly cropped view. Full-spherical 360°×180° video provides a means to break the frame and generate a completely immersive experience that transports the viewer to another world and allows her to examine all directions at once. We call this medium "360° video" for short. 360° video is particularly suitable for live action such as action sports (Figure 1b), since no angle is cropped in the recording. Viewers can pan and rotate a 360° video's perspective to watch it from different angles by dragging with the mouse or finger on a computer or mobile device, or by watching the video with a VR headset.

The level of industry support for the 360° medium has dramatically increased in the last year or two. Google and Facebook recently added support for 360° to their video sharing platforms and released reference camera designs[1,2] for professional content producers. Numerous consumer-level 360° cameras have just recently become available or will be released later this year (Figure 1a).

While recording and sharing 360° video is now fairly accessible for consumers, making it look good is not so simple. Casual 360° videos taken with a hand-held camera often look shaky (Figure 1c), and the individual frames are geometrically distorted by rolling shutter wobble. Shake in 360° video is particularly severe because it can cause discomfort ("cybersickness") when watched using VR headsets [Kennedy et al. 2010]. In this paper, we present a 360° video stabilization algorithm that can dramatically reduce shake and rolling shutter distortions in 360° videos.

Most existing video stabilization algorithms are designed specifically for narrow field-of-view video and use 2D motion models. Early work used low-dimensional similarity or homography transformations [Morimoto and Chellappa 1998]; however, these simple models cannot handle parallax. More recent work switched to more flexible models, such as mesh warps [Liu et al. 2011; Goldstein and Fattal 2012; Liu et al. 2013] or smoothed flow fields [Liu et al. 2014]. Unfortunately, this introduces the challenge of constraining the extra dimensions to avoid producing new geometric deformations to the processed video, which is the main focus of these papers. Many of these constraints, however, do not translate easily to the spherical wrap-around domain in 360° video (Section 2).

Another category of algorithms reconstructs a 3D model of the camera trajectory and scene geometry, and reason about the stabilized video in terms of 3D quantities [Buehler et al. 2001; Bhat et al. 2007; Liu et al. 2009; Kopf et al. 2014]. However, performing a full 3D reconstruction is a complex process, slow in practice, and

SA '16 Technical Papers, December 05-08, 2016, , Macao
ISBN: 978-1-4503-4514-9/16/12
DOI: http://dx.doi.org/10.1145/2980179.2982405

[1] https://www.google.com/get/cardboard/jump
[2] https://facebook360.fb.com/facebook-surround-360

ACM Reference Format
Kopf, J. 2016. 360° Video Stabilization. ACM Trans. Graph. 35, 6, Article 195 (November 2016), 9 pages.
DOI = 10.1145/2980179.2982405 http://doi.acm.org/10.1145/2980179.2982405.

not robust under certain situations such as the absence of translational motion.

The method we propose in this paper is a hybrid 3D-2D algorithm. We use a new *deformed-rotation* motion model to undo shake in the video. Slight deviations from a pure rotation allow handling some degree of translational motion, parallax, lens deformations, and rolling shutter wobble. Like most stabilization algorithms, we track feature points and perform stabilization in terms of these accumulated trajectories. We use a robust 3D analysis to estimate the true relative rotations between appropriately spaced apart key frames. For the inner frames, we switch to a 2D optimization that maximizes the smoothness of the feature point trajectories using the new deformed-rotation motion model to account for above mentioned deviations from pure rotations. We optionally re-apply a smoothed version of the estimated rotations to preserve the original video orientation (but without fast jitter).

The new algorithm architecture described above has several advantages. The rotation compensation is accurate because it uses 3D analysis, so it can *distinguish* rotational and translational motion. The 3D reconstruction is robust because it is applied to carefully spaced key frames, and estimates only the rotational component while ignoring translation. The fixed rotation-compensated key frames provide a regularizing backbone for the 2D optimization. The resulting algorithm is extremely fast: stabilizing a video takes less time than playing it at normal speed.

Our stabilized results are sufficiently smooth to be played back at high speed-up factors. For this purpose we present a simple 360° hyperlapse algorithm that estimates the apparent camera velocity from the *after-stabilization* motion vectors in the input video, and then remaps the video frames temporally to balance the output camera velocity.

We compare our hybrid algorithm to pure 2D and 3D variants and analyze the effect of our stabilization on video bitrate. We achieve between 10%-20% bitrate reduction for near lossless compression that relies on the client to apply the inverse of the stabilization transformation. We demonstrate our algorithm on a range of challenging example videos. Please refer to the supplementary material for full results.

2 Previous Work

2.1 360° Video

360° videos are shot with an omnidirectional camera, or stitched from a collection of cameras. While we show most 360° imagery in equirectangular projection throughout this paper (since it shows all $360^\circ \times 180^\circ$ angles at once), they are usually transformed for the output display into a perspective viewport with lower-field-of-view.

When watching a 360° video on a mobile device, the viewing angle is changed by dragging the finger across the screen or by tilting the device as if it is a portal into the scene captured in the 360° video. On a computer, the mouse can be used to navigate the video. 360° videos can also be watched "in VR" with devices such as a Samsung GearVR or Google Card Board.

2.2 Video Stabilization

Video Stabilization is a topic that has been intensely studied in the past one or two decades. Almost all methods specialize on narrow field-of-view videos. Generally, algorithms track incremental motion, fit a parametric or non-parametric motion model, smooth the model, and then crop and warp the final pixels.

Most of the algorithms do not extend easily to the spherical wrap-around domain in 360° video. For example, most algorithms employ Content-Preserving Warps [Liu et al. 2009] to produce the output frame. While this algorithm has an elegant linear solution in the planar domain it becomes non-linear on the sphere, which has consequences for robustness and performance (i.e., temporal constraints have to be added to prevent jumping between different local minima in adjacent frames).

Video stabilization algorithms can be categorized into ones that reason about the model fitting and smoothing in *2D* or *3D*. Our proposed algorithm combines elements from both of these kinds of video stabilization.

2D Stabilization These algorithms fit and smooth 2D motion models. The earliest examples use low-dimensional models such as similarity or homography transformations [Morimoto and Chellappa 1998], but these cannot represent parallax. Simple models are still frequently used in modern approaches [Grundmann et al. 2011], though, due to their robustness. More recent methods tend to use more flexible motion models, such as Content-Preserving Warps [Liu et al. 2009]. However, these flexible motion models have to be carefully constrained to prevent overfitting and introduction of *new* wobble artifacts. Liu et al. [2011] smooth feature point trajectories in a low-dimensional subspace. Goldstein and Fattal [2012] use epipolar constraints. Liu et al. [2013] divide the video frame into a 4×4 grid and fit homographies with regularization constraints. The Steadyflow algorithm [Liu et al. 2014] achieves strong stabilization by smoothing dense optical flow fields.

3D Stabilization Some algorithms build 3D scene models to perform stabilization [Buehler et al. 2001; Bhat et al. 2007; Liu et al. 2009; Kopf et al. 2014]. Generally, these algorithms have higher smoothing ability, since they use a more accurate model. However, in practice 3D reconstruction is often less robust than 2D methods. It can break down in the absence of translational motion or in the presence of rolling shutter artifacts. 3D methods are typically much slower than 2D methods.

360° Video Stabilization The "Jack-In Head" system [Kasahara et al. 2015] (an earlier version of the system was called "LiveSphere") estimates relative rotation between adjacent video frames for stabilization. Because adjacent frames are very similar it is difficult for the algorithm to distinguish small translations and rotations; error accumulation results in less smooth results than we achieve with our hybrid 3D-2D algorithm that estimates rotation (more robustly) between spaced key frames and *optimizes* smoothness of the in-between frames. We compare against this kind of stabilization in Section 5.3.

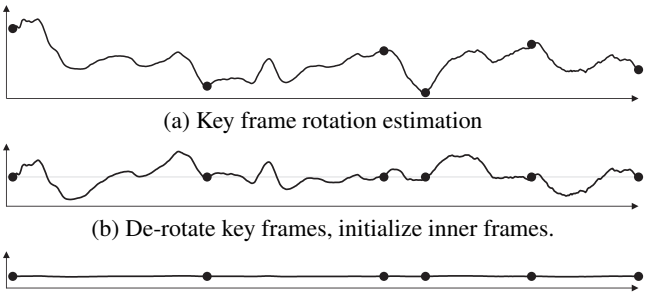
Kamali et al. [2011] describe an omnidirectional structure-from-motion algorithm and a 3D stabilization algorithm similar to Liu et al. [2009]. It shares the limitations of 3D stabilizers for narrow field-of-view videos described above.

There exists at least one commercial 360° stabilization algorithm[3]. However, it only accepts *unstitched* GoPro footage as input, and it is not clear what algorithm is used.

3 360° Video Stabilization

In this section, we describe our basic algorithm for stabilizing 360° videos. We start by tracking the motion of feature points in the video (Section 3.1). The stabilization algorithm then operates entirely on these trajectories. We propose a hybrid 3D-2D algorithm

[3] Autopano Video http://www.kolor.com/autopano-video



(a) Key frame rotation estimation

(b) De-rotate key frames, initialize inner frames.

(c) 2D optimization for inner frames.

Figure 2: *Overview of our algorithm. The graphs plot rotation (vertical axis, only one degree of freedom shown) over time (horizontal axis). (a) We first estimate the rotations of the key frames (black dots) using 3D analysis (Section 3.2), (b) then apply the inverse transformation to undo the key frame rotations, and interpolate the adjustment across the inner frames. (c) Finally, we minimize the inner frame rotations using 2D optimization that fits a deformed rotation motion model (Sections 3.3-3.4).*

architecture. The 3D reasoning estimates the relative rotations of appropriately spaced key frames (Section 3.2, Figure 2a). We then undo the relative rotation between the key frames and interpolate the adjustment for the inner frames (Figure 2b). Finally, we switch to 2D optimization of the inner frame rotations to maximize the smoothness of the stabilized feature point trajectories (Section 3.3, Figure 2c). Our motion model allows slight deviation from pure rotation at inner frames to account for residual jitter from parallax and rolling shutter wobble, etc. (Section 3.4). This architecture has several advantages:

Accuracy: The 3D analysis estimates the true rotations between key frames and does not get confused by mixed rotational/translational motion, varying feature density, non-static outlier features, etc.

Robustness: The key frames are spaced such that sufficient motion has occurred to make the 3D rotation estimation reliable. We show in Section 5.3 that our sparse 3D estimation is faster and produces smoother results than dense estimation between all successive frames.

Regularization: The fixed key frames provide a regularizing backbone for the 2D optimization of the inner frames. This constrains our deformed rotation motion model, prevents generating new wobble artifacts, and has a strong positive effect on convergence.

Speed: While we use non-linear optimization for the inner frames, the problem has a benign error function, is well initialized, and converges rapidly. We use automatic code generation to make the objective function residual and Jacobian evaluation highly efficient (tools for this are provided in a supplementary document). Our algorithm performs stabilization faster than playing the video at normal speed.

3.1 Tracking and Key Frame Generation

Like most existing stabilization algorithms, we start by tracking the motion of feature points in the video. Since our input videos use equirectangular projection (well-known from world maps, see Figure 1b for an example), which is highly distorted near the poles, we convert the frames into a less distorted cube map representation for tracking (Figure 3). We always use a cube face size of 256×256 pixels, independent of the input resolution, and only use the luma plane for tracking.

(a) Equirectangular representation

(b) Cube map representation

Figure 3: *Our input videos are represented in an equirectangular projection (a). It is highly distorted near the top and bottom, e.g., note the hand of the camera person and the rectangular structure on the roof. For tracking we convert to a cube map representation (b), which is less distorted and distributes the pixels better among all directions.*

We experimented with descriptor-based matching but found that a pyramidal Lukas-Kanade tracking algorithm [Bouguet 2000] yields longer and less noisy tracks. If a tracked point falls outside its originating cube face we simply drop that observation and end the track there. While we track points on planar cube faces, we immediately convert the 2D locations to 3D unit vectors and store them in a track table. All subsequent reasoning is done in terms of these 3D unit vectors.

Our algorithm has a notion of *key frames*, which play a very important role as we are estimating their true relative rotations and they form a regularizing backbone for the subsequent 2D optimization. We describe our heuristic for triggering new key frames during tracking below. At key frames, we spawn new tracks for subsequent tracking. We use the Shi-Tomasi algorithm to generate a list of feature points, sorted by decreasing feature strength [Shi and Tomasi 1994]. We walk through the list and accept a feature for spawning a new track only if it is more than 2° away from any previously selected or actively tracked feature.

We trigger key frames during tracking as follows. The first frame is always a key frame. Subsequent frames get turned into key frames based on two heuristics. (1) Points tracked with successive Lucas-Kanade alignment, as described above, can slowly drift away from their original appearance due to alignment error accumulation. Shi and Tomasi [1994] suggest verifying tracks using affine alignment back to the spawning frame; however, we found that this procedure significantly slows down the tracking operation. Instead, we simply turn the current frame into a key frame if its presentation time stamp difference to the last key frame amounts to more than 3 seconds. This effectively prevented degradation from tracking drift in our experiments. (2) To reliably estimate rotation as described in the next section, we need a sufficient number of common tracks between subsequent key frames. To ensure that this is the case and they are well distributed, we count at key frames after spawning new tracks the number of tracks within each *octant* of the sphere. As we progress we keep track of the fraction of active tracks that originated from each octant. Once the fraction for at least one octant drops below 50%, we turn the *previous* frame into a key frame (since in that frame there were still sufficient tracks).

After we have finished tracking the whole video, we cut "dangling ends" off tracks, so that each track starts and ends at a key frame.

3.2 Estimating Rotations Between Key Frames

Our next goal is to estimate the relative rotation between successive key frames. Let $K = \{k_i\}$ be the set of key frames. For each pair of successive key frames (k_i, k_{i+1}) we obtain from the feature tracks a set of matching point pairs. We use a five-point algorithm in a RANSAC procedure [Fischler and Bolles 1981] to estimate the relative rotation $\widetilde{R}_{k_i}$. We experimented with Nistér's [2004] as well as Kneip's [2012] five-point algorithms, using the implementations provided by the OpenGV library [Kneip and Furgale 2014]. Both algorithms produced very similar results, even though Nistér's algorithm theoretically requires a sufficient amount of translational motion to work robustly. However, we did not find any problems even when testing with videos where the camera was mounted on a static tripod. Kneip's algorithm [2012] guarantees to compute the correct rotation even in the case of zero translation, so using this algorithm might be preferable.

We use a relatively lenient inlier threshold to tolerate some amount of image deformation from rolling shutter and lens deformation. If the number of model inliers is below a threshold fraction of 0.5, however, we recursively split the segment and insert a new key frame in the middle. This tends to increase the number of inliers as the new key frame pairs are more similar to each other. We repeat this until the inlier threshold is satisfied or until there is no space to insert new key frames. In the rare case where the number of features in a key frame pair is below 8 matches, we do not use the five-point algorithm but instead directly find the rotation that minimizes the relative distances between matched points.

We chain the rotations to make them relative to the first key frame. Since we are interested in removing rotations from the video, we store the inverse transformation,

$$\mathbf{R}_{k_i} = \left(\prod_{j=1}^{i} \widetilde{R}_{k_j} \right)^{-1}. \tag{1}$$

Applying the $\mathbf{R}_{k_i}$ rotations to the key frames stabilizes them since it remove all relative rotation. While errors in the rotation estimation might produce a slight drift, it is barely perceivable, since the key frames are spaced on the order of seconds apart.

3.3 Optimizing the Inner Frames

We now shift our attention to the rotations of the inner frames. In the tracking phase, we computed a set of tracks $T = \{T_i\}$, each being a list of observations (3D unit vectors):

$$T_i = \left\{ \mathbf{p}_j^i \mid j \in f_i \dots l_i \right\}. \tag{2}$$

$f_i \dots l_i$ is the range of frames in which the track was observed, always starting and ending at a key frame. These tracks form visual trajectories that can be plotted on the video (Figures 1c). Our goal is to optimize the inner frame rotations such that these trajectories become as smooth as possible (Figures 1d-e). This is encoded in the following optimization problem

$$\underset{\{\mathbf{R}_j | j \in I\}}{\operatorname{argmin}} \sum_{i=1}^{|T|} \left(\sum_{j=f_i}^{l_i-1} E_{i,j}^1 + \sum_{j=f_i}^{l_i-2} E_{i,j}^2 \right). \tag{3}$$

$I = 1 \dots n \setminus K$ is the set of inner frames, i.e., we exclude the key frames from the optimization; they are fixed to the rotations estimated in Section 3.2. We consider first-order ($E_{i,j}^1$) and second-order ($E_{i,j}^2$) smoothness terms.

The first-order term encourages trajectories to be as short as possible:

$$E_{i,j}^1 = \rho \left(\left\| \mathbf{R}_j \mathbf{p}_j^i \; - \; \mathbf{R}_{j+1} \mathbf{p}_{j+1}^i \right\|_2^2 \right). \tag{4}$$

$\rho(s) = a^2 \log\left(1 + \frac{s}{a^2}\right)$ is a robust loss function to reduce sensitivity to outliers, and $a = 0.01$ sets the scale at which robustification takes place.

Using just the first-order term is not sufficient because due to its pairwise nature, it cannot reach across the locked key frames and would produce visible kinks there. So, we also use a second-order term that encourages smoothness using a discrete Laplacian operator and whose 3-tap footprint can reach across key frames:

$$E_{i,j}^2 = \rho \left(\left\| -\mathbf{R}_j \mathbf{p}_j^i + 2\mathbf{R}_{j+1} \mathbf{p}_{j+1}^i - \mathbf{R}_{j+2} \mathbf{p}_{j+2}^i \right\|_2^2 \right). \tag{5}$$

Combined additively the terms encourage short *and* smooth trajectories without kinks at key frames. We found that the exact weight balance between the terms is not critical, as long as they are within an order of magnitude of each other; so, we weigh them equally in Equation 3.

When minimizing Eq. 3, we represent all rotations using the 3-dimensional axis-angle parameterization and use Rodrigues' formula [Ayache 2007] when they are applied to vectors. Since this optimization problem is nonlinear, we require a good initialization of the free variables. We initialize the rotations of the inner frames using quaternion interpolation of the surrounding key frames:

$$\forall k_i < j < k_{i+1}\colon \; \mathbf{R}_j = \operatorname{slerp}\left(\mathbf{R}_{k_i}, \mathbf{R}_{k_{i+1}}, \tfrac{j-k_i}{k_{i+1}-k_i} \right) \tag{6}$$

We use the Ceres library [Agarwal et al. 2015] to solve this nonlinear least-squares minimization problem. We found it converges usually within 3-4 iterations. Note that the optimization does not necessarily recover the true rotations but instead the ones that produce the smoothest possible result, which is after all the primary goal of stabilization.

3.4 Residual Jitter Compensation

Solving the optimization described in the previous section removes most of the camera shake, but usually some amount of residual jitter remains (Figure 1d), due to a combination of small translational motion (e.g., bobbing up and down while walking), parallax, suboptimal lens calibration, stitching artifacts, and rolling shutter wobble.

We address this problem by adding some flexibility to the motion model so it can adapt and undo slight image deformations (Figure 1e). We want to ensure, however, that the model does not become too flexible and that it is constrained properly so it does not overfit the data and introduce artifacts instead of removing them.

Some of the phenomena listed above could be modeled directly, for example the characteristics of the rolling shutter function of a particular camera. However, this is not desirable for several reasons: (1) It would require calibrating each camera that we would like to support. In addition, we would have to know which camera each video was shot with. (2) Since 360° cameras use fisheye lenses, the sensor scanlines undergo a non-trivial transformation when converting to the projection our algorithm operates in. (3) There remains residual jitter from the other sources mentioned above.

For these reasons, we design instead a generic deformation model to handle all situations above. In this model, we distribute 6 vertices evenly on the unit sphere so that they are separated by 90° angles from each other, and define 8 congruent spherical triangles, one for

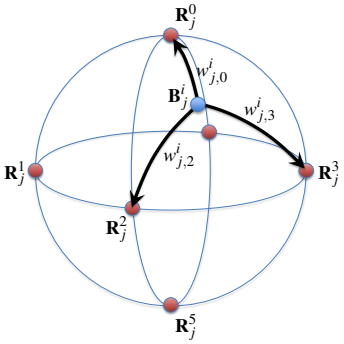


Figure 4: *Our deformed-rotation motion model for handling residual jitter. Slightly different rotations are defined at 6 vertices and interpolated using spherical barycentric coordinates.*

each octant, as shown in Figure 4. At each vertex $v \in 1\ldots6$ we store a rotation $\mathbf{R}^v$. Within each spherical triangle we interpolate the vertex rotations using spherical barycentric coordinates [Langer et al. 2006], i.e. for a track observation $\mathbf{p}_j^i$ we obtain the blended rotation

$$\mathbf{B}_j^i = \sum_{v=1}^{6} w_{j,v}^i \mathbf{R}_j^v, \tag{7}$$

where $w_{j,v}^i$ are the spherical barycentric weights. We initialize the deformed rotation model using the solution of the pure rotation optimization, i.e., $\forall v, j : \mathbf{R}_j^v = \mathbf{R}_j$.

We integrate the new motion model into the stabilization optimization problem as follows:

$$\underset{\{\mathbf{R}_j^v \mid j\in I,\ v\in 1\ldots6\}}{\operatorname{argmin}} \sum_{i=1}^{|T|} \left(\sum_{j=f_i}^{l_i-1} E_{i,j}^{rs\text{-}1} + \sum_{j=f_i}^{l_i-2} E_{i,j}^{rs\text{-}2} \right) + \mu \sum_{j=1}^{n} E_j^{reg} \tag{8}$$

The first- and second-order smoothness terms, $E_{i,j}^{rs\text{-}1}$ and $E_{i,j}^{rs\text{-}2}$, are similar to the pure rotation version, except we replace all occurrences of rotations, $\mathbf{R}_j, \mathbf{R}_{j+1}, \mathbf{R}_{j+2}$ with their blended versions $\mathbf{B}_j^i, \mathbf{B}_{j+1}^i, \mathbf{B}_{j+2}^i$. We set the balancing coefficient $\mu = 0.1$. The regularization term, E_j^{reg}, prevents the deformations from becoming too strong:

$$E_j^{reg} = \sum_{v,w\in 1\ldots6} d\left(\mathbf{R}_j^v, \mathbf{R}_j^w\right), \tag{9}$$

where d is a measure of closeness of rotations. An appropriate choice for d would be the required angle of rotation to get from one orientation to the other. Since it is simpler to optimize we instead use the Euclidean distance of the axis-angle representation coefficients, which is a good approximation for reasonably close orientations.

A second layer of regularization is provided by the fact that the fixed key frames use pure rotation. This effectively prevents the rolling shutter compensation from drifting too far off, since the key frames are interpolated.

3.5 Implementation Details

Analytic derivatives: Non-linear optimization algorithms depend on being able to evaluate the Jacobians of the objective functions with respect to the parameter values. The objective functions in Equations 3 and 8 are highly complex, and it is difficult to work out the derivative expressions manually. For that reason, the Ceres library provides an automatic differentiation technique using operator overloading [Agarwal et al. 2015]. However, this approach is not as efficient as would be desirable, since it cannot make full use of compiler optimizations and avoid redundancies inherent in computing the various expressions.

To improve this situation, we wrote a Python script that automatically generates highly efficient C++ code for the objective function residuals and their Jacobians. We use the SymPy package [SymPy Development Team 2016] to express the residuals symbolically and take their derivatives. We then use term rewriting techniques to recursively identify common subexpressions in these larger terms, and collect them so they can be evaluated at once. Finally, we use automatic code generation to translate the optimized expressions into C++ code. The analytic derivatives generated in this way provide a $2\times - 3\times$ performance improvement over the automatic differentiation code.

We provide the final C++ code as well as the Python script that was used to generate it in a supplementary document. It is a useful piece of code, as it can be easily adapted to other objective functions.

Image warping: In order to render an output frame using backwards warping, we need to *invert* the deformed rotation model. We use iterative search [Yang et al. 2011]. However, evaluating Equation 7 and various transformations multiple times per pixel requires a significant amount of computation. Since the resulting warp function is quite smooth, we evaluate the warp coordinates only for 1 in every 8×8 pixels, and interpolate the remaining coordinates bilinearly. This looks visually nearly identical, but improves the warping speed dramatically.

4 Extensions

In the previous section, we described our basic algorithm for removing shake in 360° videos. Here we describe several interesting extensions that are layered on top of the basic algorithm.

4.1 Reapplying Smoothed Rotations

The algorithm described in the previous section removes *all* rotation from the input video. This seems appropriate for VR experiences where any induced acceleration (such as rotation) can cause discomfort [Kennedy et al. 2010].

However, in a video such as MOUNTAIN BIKING (Figure 7), where the camera makes *slow* turns, it causes the direction of forward motion direction to drift away from the viewer's "front" direction. In the case where a video is not watched in VR but on a computer or mobile device, we would like to preserve the overall camera front direction and just remove the high-frequency jitter.

We achieve this by adding back a smoothed version of the raw rotation estimates that we subtracted from the video when stabilizing it. We compute the smoothed rotations by low-pass filtering their quaternion representation temporally, smoothing each of the 4 quaternion components independently and normalizing the result. Figure 5 shows the raw rotation estimates and the smoothed curve that is reapplied to the video. An interesting extension would be to consider L1-smoothness constraints based on cinematography principles as proposed by Grundmann et al. [2011].

4.2 Two-pass Tracking

Tracking through very shaky sections of a video is difficult, since the appearance of patches changes significantly due to perspective distortion and tracks drift across different cube map faces (which we don't support). We can improve the result in this situation

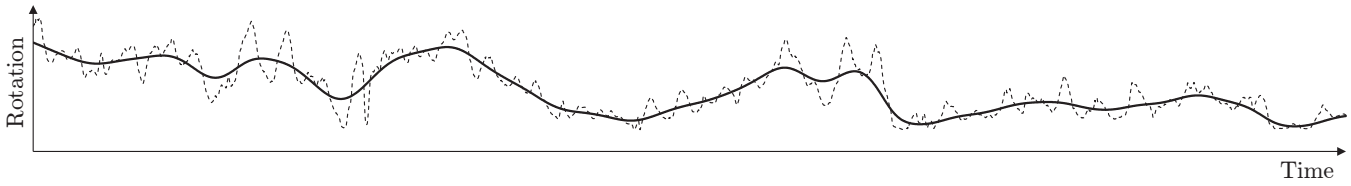


Figure 5: *Smoothing the stabilizing rotation estimates to preserve the overall camera forward direction while removing the residual jitter. The dashed curve shows the stabilizing rotations computed in Section 3, and the solid curve shows the smoothed version (only one of the 4 quaternion components is shown).*

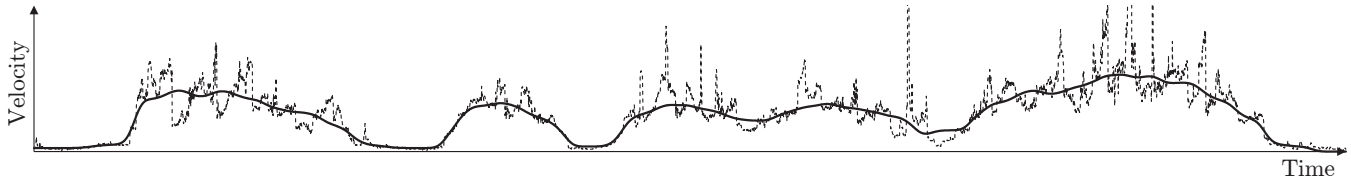


Figure 6: *Estimated velocity curve for the* DOWNHILL SKIING *video. The sections where the skier took breaks stand out clearly. The dashed curve shows the raw velocity estimates, and the solid curve shows the smoothed version.*

by repeating the tracking after rotational stabilization. In the second tracking phase, we apply the stabilizing rotational adjustments when converting the input video to the cube map representation used for tracking. On the DOWNHILL SKIING video, we found that this increases the average track length from 62 to 78 frames. We notice that for particularly shaky inputs, two-pass tracking can give a slight but noticeable boost in smoothness (see Section 5.3).

4.3 Hyperlapse

Since our stabilized 360° videos are so smooth, they provide excellent source material for creating speed-up hyperlapse videos [Kopf et al. 2014]. Since 360° videos are always full-frame, there is no need to blend bits and pieces from multiple frames, which seems to have been the source of most technical challenges in Kopf et al.'s system.

Creating 360° hyperlapse simply by dropping all but every n-th frame in a sequence stabilized with our algorithm already looks great. However, a common element of hyperlapse videography is a smoothly balanced camera velocity. We can modify the apparent camera velocity by remapping the video frame time stamps in the stabilized video.

First, we need to estimate the camera velocity at every frame. Instead of turning to 3D reconstruction and all its problems, we use a simple 2D approximation: a robust average of the after-stabilization motion vector magnitudes. This simple measure would not work with narrow field-of-view video, because the motion vectors behave differently when the camera is pointed forward vs. to the side, which would bias the velocity estimate. In 360° video, however, we always see all angles, so we do not suffer from this problem at all.

We start by computing the median motion vector magnitude for each frame:

$$v_j = \text{median}\left\{\cos^{-1}\left(\mathbf{p}^i_j \mathbf{p}^i_{j+1}\right) \;\middle|\; f_i \le j < l_i\right\}. \qquad (10)$$

This estimate is very noisy (dashed curve in Figure 6). We process it further by removing outliers using a running temporal median (radius 4 seconds) followed by a low pass filter ($\sigma = 1$ second). This smoothed velocity estimate is shown as the solid curve in Figure 6. It can be used to simulate a constant camera velocity by remapping the video frame time stamps inversely proportional to the cumulative velocity function. If it is desired to reapply smoothed orientation, as described in Section 4.1, it is important to consider these modified time stamps when low-pass filtering the orientation curve. Please refer to the supplementary material for example results.

5 Results

We tested our algorithm on eight hand-held 360° video sequences of different activities with varying amounts of shake. The full input and stabilized sequences are provided in the supplementary material. Figure 7 shows a sample frame from each sequence.

5.1 Performance

We implemented our algorithm *without* GPU acceleration to make it amenable for execution on standard cloud hardware. Figure 8 summarizes the performance of our algorithm. The timings were measured when processing the MARKET WALK sequence at 1080p input/output resolution on an Intel Core i7-5930K CPU at 3.50GHz. The sequence is 310 seconds (9,292 frames) long and processing took 201 seconds, i.e., stabilization took less time than playing the video at normal speed. In the table, we break the process down into different algorithm stages and report timings *per frame*. We do not include the final video decoding and encoding steps, since they are fundamental parts of a video transcoding pipeline, and highly dependent on codec parameters. We found that these timings do not vary significantly for different sequences. The total memory consumption during the process was at all times less than 2 GB.

5.2 Effect of Stabilization on Video Bit Rate

It is not surprising that stabilization can have a beneficial effect on video bit rate consumption. There are some non-obvious insights for 360° video, however. First, because the stabilized videos are not cropped, we can trivially recover the original shaky sequence from the stabilized result. Second, since any 360° viewer already applies rotational view transformations, we can do this without any additional computation (at least if we use pure rotation stabilization). We found that the bit rate reductions can be substantial.

We analyzed the bit rate consumption when encoding the video into H.264/MPEG-4 AVC format using the x264 library[4]. Figure 9 summarizes our findings. We analyzed two sequences: CABLE CAR (blue colors) and DOWNHILL SKIING (orange colors). The first

[4]http://www.videolan.org/developers/x264.html

Figure 7: *Some sequences we have run our stabilization on. The supplemental material contains the full videos.*

Stage	Time / frame
Video decoding (ffmpeg, luma only)	7.91ms
Equirect to cube conversion	0.73ms
Pyramid construction	0.87ms
Feature generation (at key frames)	0.10ms
Translational Lucas-Kanade tracking	2.19ms
Rotation estimation (at key frames)	0.10ms
Rotation optimization (inner frames, Eq. 3)	2.20ms
RS optimization (inner frames, Eq. 8)	1.84ms
Warp coordinate computation	3.46ms
Frame warping	2.20ms
Total	**21.60ms**

Figure 8: *Per-frame performance break down of our 360° video stabilization algorithm. We can stabilize videos faster than real-time (i.e., playback rate).*

scene is mostly static and has a low amount of shake, while the latter contains very fast, shaky motion. We stabilized both sequences with the pure rotation (light shade) and deformed rotation (dark shade) motion models. For each experiment we encoded the first 600 frames of the unprocessed, pure rotation, and deformable rotation model into an H.264 stream using a 30 frames long group of pictures (GOP) structure, i.e., each stream consists of 20 GOPs. We then compare the size of each pure-rotation and deformed-rotation GOP against the corresponding unstabilized GOP and report the average size fraction in the figure.

In the left half of the figure we fix the encoding parameter preset to "very slow" and vary the constant rate factor (CRF, i.e., quality setting), while in the right half we fix the CRF to 21 and vary the preset meta-parameter instead. For both sequences and all settings, we found a considerable reduction in bit rate. To our surprise the savings for the *less shaky* CABLE CAR sequence were far more substantial, which is likely due to the fact that the scene is mostly static and becomes very redundant after stabilization and rolling shutter compensation, while the DOWNHILL SKIING sequence remains fast moving even with stabilization.

5.3 Comparisons and Quantitative Evaluation

While we provide qualitative results and comparisons in the supplementary material (i.e., videos to watch), we look here into some quantitative experiments. We can use the first-order and second-order penalties in Equations 4 and 5 to numerically evaluate the smoothness of a given result. In order to do this we collect all track observations and evaluate the first-order and second-order penalties. Figure 10 shows the distribution functions for these quantities for different variants of our stabilization algorithm. We compare pure rotation (Section 3.3) and deformed rotation (Section 3.4) motion models, both in 1-pass and 2-pass tracking mode. In Figure 11 we show aggregate statistics in tabular form.

In addition to the variants mentioned above we also compare against running the 3D rotation analysis (Section 3.2) densely on all successive frames. This corresponds roughly to a reimplementation of the Jack-In Head (LiveSphere) method [Kasahara et al. 2015]. The evaluation shows that this results in significantly less smooth results (please refer to the supplementary material for a video comparison).

5.4 Limitations

Very strong rolling shutter deformations and, in particular, *high frequency* rolling shutter deformations with frequency greater than the frame rate cannot be not represented by our low frequency deformed-rotation model. While we do not always completely succeed in removing these deformations, our output still tends to be more stable and watchable than the input in these cases.

Another issue of the deformed-rotation model is that it might occasionally introduce slight wobbling in the result. This effect is relatively mild and can be observed in our result videos.

Our method does not always yield good results on "produced" videos. For example, the algorithm as described here does not handle shot boundaries: it may track and stabilize the video across the shot boundary with unpredictable results. It would be relatively straight-forward to add code to detect and handle this situation appropriately.

Another common failure mode with produced videos occurs if they contain static overlays such as logos or text. In this case our stabilizer might stabilize the shaky background motion, which will in turn make the overlays appear shaking.

6 Conclusions

We have presented a method for 360° video stabilization. Our method is a hybrid 3D-2D method. It uses 3D analysis to estimate the rotation between key frames that are appropriately spaced such that we can reliably estimate the true rotations. For the in-between frames we solve a 2D optimization problem that maximizes the smoothness of tracked feature point trajectories. A flexible deformed-rotation motion model enables reducing residual jitter caused by small translational motion, parallax, rolling shutter wobble, etc. Our novel stabilization architecture provides more ac-

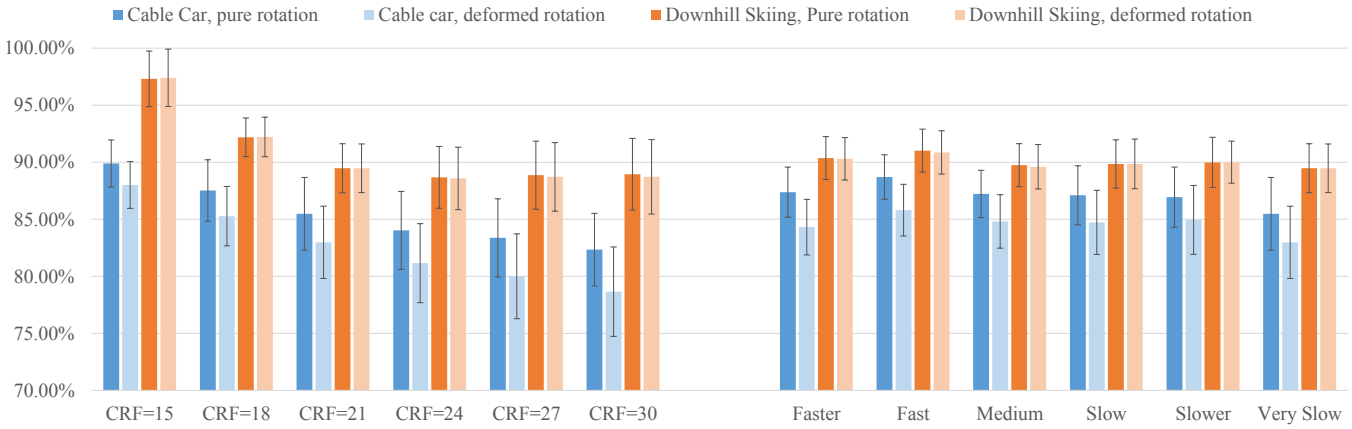


Figure 9: *Our stabilization lowers the video bit rate by 10%-20% at equal quality settings. The reductions are consistent for different settings of the encoding quality and preset parameters (left: fixed preset "very slow", varying quality; right: fixed quality CRF=21, varying preset). We plot the bit rate for pure rotation and deformed rotation stabilization relative to the unstabilized video bit rate for two sequences.*

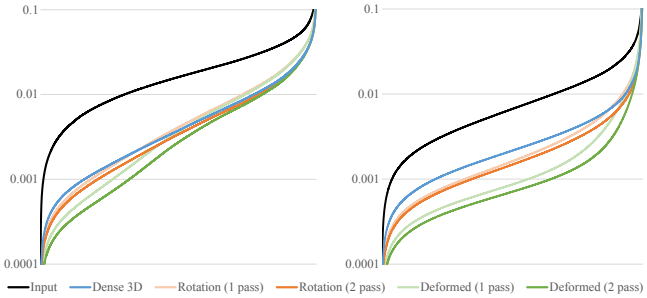


Figure 10: *Numerical evaluation of different 360° video stabilization variants. The curves show the cumulative distribution function of first-order (left) and second-order (right) smoothness costs for the stabilized feature trajectories.*

Method	Original	Dense rotation estimation	Our Method			
			Rotation only		RS comp.	
			1-pass	2-pass	1-pass	2-pass
E^1 average	21.80	8.21	9.72	7.86	9.30	**7.32**
E^1 median	16.07	3.77	4.11	3.40	3.66	**2.74**
E^2 average	11.83	4.93	5.32	4.25	4.42	**3.25**
E^2 median	6.70	2.17	1.63	1.49	0.87	**0.71**

Figure 11: *Numerical evaluation of smoothness penalties in tabular form. Smoother results are signified by lower numbers.*

curacy, robustness, smoothing ability and speed than either pure 2D or pure 3D methods. The results are sufficiently smooth to be played back at high speed-up factors. For this purpose we present a simple 360° hyperlapse algorithm that remaps the video frame time stamps to balance the apparent camera velocity.

360° video is an exciting new medium, whose underlying technology is rapidly improving. While today's 2K resolution still leaves room for improvement when shooting 360° video, 4K and larger resolutions will soon become widely available to consumers. Even if the desired end result is a regular narrow field-of-view video, it may soon "make sense" to always shoot in 360°, because no angle is missed and the video can be framed later. Our stabilization algorithm provides a fundamental building block to the 360° video processing pipeline.

References

AGARWAL, S., MIERLE, K., AND OTHERS, 2015. Ceres Solver. http://ceres-solver.org.

AYACHE, N. 2007. Vision Stéréoscopique et Perception Multisensorielle.

BAKER, S., BENNETT, E. P., KANG, S. B., AND SZELISKI, R. 2010. Removing rolling shutter wobble. *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. *Proceedings of the 18th Eurographics Conference on Rendering Techniques (EGSR'07)*, 327–338.

BOUGUET, J. 2000. Pyramidal implementation of the Lucas Kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*.

BUEHLER, C., BOSSE, M., AND MCMILLAN, L. 2001. Non-Metric Image-Based Rendering for Video Stabilization. *2014 IEEE Conference on Computer Vision and Pattern Recognition 2*, 609.

FISCHLER, M. A., AND BOLLES, R. C. 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM 24*, 6, 381–395.

GOLDSTEIN, A., AND FATTAL, R. 2012. Video Stabilization Using Epipolar Geometry. *ACM Trans. Graph. 31*, 5, article no. 126.

GRUNDMANN, M., KWATRA, V., AND ESSA, I. 2011. Auto-Directed Video Stabilization with Robust L1 Optimal Camera Paths. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

KAMALI, M., BANNO, A., BAZIN, J.-C., KWEON, I., AND IKEUCHI, K. 2011. Stabilizing Omnidirectional Videos Using 3D Structure and Spherical Image Warping. *Proceedings of the 12th IAPR Conference on Machine Vision Applications (MVA)*.

KASAHARA, S., NAGAI, S., AND REKIMOTO, J. 2015. First Person Omnidirectional Video: System Design and Implications for Immersive Experience. *Proceedings of the ACM International*

Conference on Interactive Experiences for TV and Online Video, 33–42.

KENNEDY, R. S., DREXLER, J., AND KENNEDY, R. C. 2010. Research in visually induced motion sickness. *Applied Ergonomics 41*, 4, 494–503.

KNEIP, L., AND FURGALE, P. T. 2014. OpenGV: A Unified and Generalized Approach to Real-time Calibrated Geometric Vision. *IEEE International Conference on Robotics and Automation (ICRA 2014)*, 1–8.

KNEIP, L., ROLAND, S., AND MARC, P. 2012. Finding the exact rotation between two images independently of the translation. *Proceedings of the European Conference on Computer Vision (ECCV)*.

KOPF, J., COHEN, M. F., AND SZELISKI, R. 2014. First-person Hyper-lapse Videos. *ACM Transactions on Graphics 33*, 4, article no. 78.

LANGER, T., BELYAEV, A., AND SEIDEL, H.-P. 2006. Spherical Barycentric Coordinates. *Proceedings of the Fourth Eurographics Symposium on Geometry Processing (SGP '06)*, 81–88.

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.

LIU, F., GLEICHER, M., WANG, J., JIN, H., AND AGARWALA, A. 2011. Subspace Video Stabilization. *ACM Trans. Graph. 30*, 1, article no. 4.

LIU, S., YUAN, L., TAN, P., AND SUN, J. 2013. Bundled camera Paths for Video Stabilization. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2013) 32*, 4.

LIU, S., YUAN, L., TAN, P., AND SUN, J. 2014. SteadyFlow: Spatially Smooth Optical Flow for Video Stabilization. *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, 4209–4216.

MORIMOTO, C., AND CHELLAPPA, R. 1998. Evaluation of image stabilization algorithms. *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'98)*, 2789–2792.

NISTÉR, D. 2004. An Efficient Solution to the Five-Point Relative Pose Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 6, 756–777.

SHI, J., AND TOMASI, C. 1994. Good Features to Track. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, 593–600.

SYMPY DEVELOPMENT TEAM. 2016. *SymPy: Python library for symbolic mathematics*.

YANG, L., TSE, Y.-C., SANDER, P. V., LAWRENCE, J., NEHAB, D., HOPPE, H., AND WILKINS, C. L. 2011. Image-based Bidirectional Scene Reprojection. *ACM Trans. Graph. 30*, 6, article no. 150.