# A Non-Linear Filter for Gyroscope-Based Video Stabilization

Steven Bell[1], Alejandro Troccoli[2], and Kari Pulli[2]

[1] Stanford University, Stanford, CA, USA
sebell@stanford.edu
[2] NVIDIA Research, Santa Clara, CA, USA
{atroccoli,karip}@nvidia.com

**Abstract.** We present a method for video stabilization and rolling-shutter correction for videos captured on mobile devices. The method uses the data from an on-board gyroscope to track the camera's angular velocity, and can run in real time within the camera capture pipeline. We remove small motions and rolling-shutter distortions due to hand shake, creating the impression of a video shot on a tripod. For larger motions, we filter the camera's angular velocity to produce a smooth output. To meet the latency constraints of a real-time camera capture pipeline, our filter operates on a small temporal window of three to five frames. Our algorithm performs better than the previous work that uses a gyroscope to stabilize a video stream, and at a similar level with respect to current feature-based methods.

**Keywords:** video stabilization, rolling-shutter, gyroscopes.

## 1 Introduction

Cell phones and other mobile devices have rapidly become the most popular means of recording casual video. Unfortunately, because cell phones are hand-held and light-weight devices operated by amateurs in the spur of the moment, most videos are plagued by camera shake. Such shake is at best mildly distracting, and at worst completely unbearable to watch. Additionally, most mobile cameras use a rolling shutter sensor, where each horizontal scanline of pixels is sequentially exposed and read out. When the camera moves during the exposure, each image row captures a slightly different viewpoint, resulting in a distorted image. Vertical motions cause the image to be squeezed or stretched vertically, and horizontal motions shear the image so that vertical lines tilt to left or right.

At the same time, cell phones have gained the processing resources and features that make real-time, on-device video processing possible. The majority of mid-to-high range mobile devices contain a multi-core CPU complex, a graphics processing unit (GPU) and an inertial measurement unit (IMU) with a 3-axis gyroscope. In this paper we address the challenge of performing video stabilization on such devices, using the gyroscope for motion tracking. Unlike most proposed stabilization methods, which operate as a post-processing step on a captured

D. Fleet et al. (Eds.): ECCV 2014, Part IV, LNCS 8692, pp. 294–308, 2014.
© Springer International Publishing Switzerland 2014

video, our method can run in real-time as part of the camera capture pipeline. In addition, our motion filter does a better job at removing camera shake than previous methods that stabilize the video stream using gyroscope data [1].

Correcting a video frame before it is sent to the hardware video encoder is beneficial in several ways. First, our algorithm has access to uncompressed data, which is an improvement over off-line methods that need to decode and re-encode and degrade the video quality when doing so. Moreover, because encoding methods such as H.264 rely on finding patches of image data which match between frames, removing frame-to-frame shake and increasing the temporal consistency of a video may improve the encoding quality and reduce the final storage size. Finally, it is important to consider that many (perhaps most) videos shot with cell phones are watched on the same device instead of being uploaded to a sharing site. Likewise, video chatting, because of its real-time peer-to-peer nature, requires that any stabilization be done on the device without inducing any lag.

Motion tracking is greatly simplified when using a phone's on-board 3-axis gyroscope. The camera orientation can be computed from the gyroscope measurements using a handful of multiplications and additions, while image-based methods must analyze thousands or even millions of pixels. As a result, motion estimation using the gyroscope can dramatically reduce CPU utilization, memory bandwidth, and battery usage compared to image-based methods. A typical MEMS gyroscope consumes about 4 milliwatts [2], while the power consumed by the CPU and memory traffic can easily be tens or hundreds of milliwatts.

Additionally, image-based methods can fail when features are sparse or when large objects move in the foreground. A gyroscope, by contrast, always reports the device motion regardless of how much and in which way the objects in the scene move. Furthermore, the gyroscope measurements allow us to estimate intra-frame camera orientations which we can use to accurately correct rolling shutter on a per-frame basis.

Compared to state-of-the-art stabilizers [3], our method provides a similar level of stabilization quality at a fraction of the processing cost, with no degradation due to foreground object motion.

## 2 Background and Prior Work

Video stabilization removes jitter from videos based on the assumption that high-frequency motions are unintended and are the consequence of hand tremor. It is essentially a three-stage process, consisting of a motion estimation stage, a filtering stage that smooths the measured motion, and a re-synthesis stage that generates a new video sequence as observed by a virtual camera moving under the filtered motion.

Two-dimensional stabilization involves tracking image keypoints to find the camera motion between frames, usually modeled as an affine or projective image warp [3–7]. The video is re-synthesized by defining a virtual crop window that is transformed according to the smoothed camera path. Matsushita *et al.* [7] smooth the camera motion by applying a Gaussian kernel to a local window

Insta360
Exhibit 1004 Page 2

of 2D transforms. Gleicher *et al.* [4] take a different approach by segmenting the camera path into shorter paths that follow a particular motion model, as defined by cinematic conventions. Grundmann *et al.* [5] integrate this kind of motion segmentation with saliency, blur, and crop window constraints in a unified optimization framework.

Image-based tracking methods suffer when depth variations induce pixel motions that, due to parallax, are not easily modeled by homographies. Furthermore, a rolling-shutter imaging sensor can introduce non-rigid frame-to-frame correspondences that cannot simply be modeled by a global frame-to-frame motion model. To address rolling-shutter, Baker *et al.* [8] estimate and remove the high-frequency jitter of the camera using temporal super-resolution of low-frequency optical flow. Following up on their earlier work, Grundmann *et al.* [3] developed a model based on a mixture of homographies that track the intra-frame motions and produces stabilized videos with corrected rolling-shutter distortions. Liu *et al.* [6] employ a mesh-based, spatially-variant motion representation coupled with an adaptive space-time path optimization that can handle parallax and correct for rolling-shutter effects.

Three-dimensional video stabilization techniques track the camera motion in the world 3D space using structure-from-motion methods [9, 10]. These 3D methods can deal with parallax distortions caused by depth variations in the scene, and synthesize the output using image warps that take into account the scene structure. Still, the motion estimation is brittle if there are not enough features or sufficient parallax; and these methods are, in general, computationally expensive.

Gyroscopes are an attractive alternative to feature-based motion estimation, since they sidestep many failure cases. Karpenko *et al.* [1] and Hanning *et al.* [11] describe video stabilization techniques for mobile devices which use the built-in gyroscope to track the camera orientation. Both of these methods apply a linear low-pass filter to the gyroscope output. Karpenko *et al.* [1] use a Gaussian kernel, while Hanning *et al.* [11] apply a variable-length Hann window to adaptively smooth the camera path. In contrast, we introduce a nonlinear filtering method which completely flattens small motions regardless of frequency, and performs low-pass smoothing when the virtual camera must move to keep the crop window inside the input frame. When the camera is nearly still, our virtual camera is fixed, removing all jitter. When moving, our method acts like a variable IIR filter, mixing the input velocity with the virtual camera velocity in a way that smooths the output, while guaranteeing that it tracks the input so that the crop window never leaves the input frame. This nonlinearity is necessary because a very low cutoff frequency is required to smooth out low-frequency motions such as those induced by walking. A large low-pass FIR or IIR filter introduces lag. Moreover, any linear filter with a low enough cutoff frequency to flatten low-frequency bouncing will also do a poor job tracking the input when the camera is intentionally moved.

None of the previous cited work has a suitable real-time implementation that eliminates camera shake. Karpenko *et al.* [1] implemented a truncated causal

Insta360
Exhibit 1004 Page 3

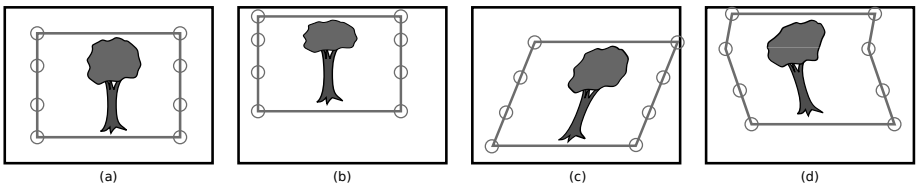


**Fig. 1.** Example crop polygons (shown in red) for a variety of scenarios: (a) no motion, (b) vertical motion causes shrinking, (c) horizontal motion causes shearing, and (d) a combination of motions causes a complex rolling-shutter distortion

low-pass filter for their real-time implementation of viewfinder stabilization on an iPhone. However, as mentioned in their paper, the truncated low-pass filter attenuates camera shake, but does not completely remove it. They suggest that for video recording it might be possible to hold back video frames for a longer period of time to achieve a smoother result, and leave this implementation for future work. But there is a limit on the number of frames that can be buffered, and as we show in Section 4, a Gaussian low-pass filter that buffers five frames still does not eliminate shake, while our method does.

Our primary contribution in this work is a novel smoothing algorithm that uses the gyroscope to track the camera motion and is suitable for real-time implementation. By using a nonlinear filter, we are able to produce static segments connected by smooth motions, while tracking the input and using little to no frame buffering.

## 3 Algorithm Description

Conceptually, video stabilization can be achieved by creating a crop rectangle that moves with the scene content from frame to frame as the camera shakes around. The position of the crop rectangle within the input frame may vary wildly, but the content within the crop rectangle remains stable, producing a smooth output. Our method is based on this idea, but instead of moving a crop rectangle, we move a crop polygon, and the region within the polygon is projectively warped to create the output video. This more flexible model allows us to model the distortions introduced by the sensor's rolling shutter, as illustrated by Figure 1.

### 3.1 Camera Tracking Using the Gyroscope

We model the camera motion as a rotation in a global coordinate frame. The gyroscope provides a series of discrete angular velocity measurements with timestamps, which we integrate to produce a function of time that describes the camera orientation. In theory we could be more precise by also measuring translation with the device's accelerometer, but in practice this is difficult and of limited value. If the camera is 3 meters away from a flat scene, then the image

motion induced by a 1 cm translation is equivalent to a rotation of 0.19 degrees, which is far more likely to occur [12]. Moreover, the process of estimating gravity and double-integrating acceleration to obtain translation is extremely sensitive to error; plus the use of translation information requires knowledge about the depth of objects in the scene.

In order to fix rolling shutter distortions, we need to know the orientation of the camera at the time a particular row was exposed. Given the timestamp for the first row of a frame $t_0$, the timestamp for row $r$ is

$$t_r = t_0 + \frac{r}{f_l} f_t, \tag{1}$$

where $f_t$ is the total frame time (i.e., the time elapsed between the start of two consecutive frames) and $f_l$ is the frame length in image rows. The frame length is the sum of the image height (in pixels), plus the number of blanking rows, where no data is read out. Both of these values depend on the image sensor and capture mode, but we assume that they are known and constant for the duration of the video. If these values are not provided by the sensor driver, they can also be obtained by calibration [13, 14].

We can find the device orientation corresponding to a point $x$ in an image by calculating its corresponding row timestamp and interpolating the camera orientation from known samples. Due to hardware and software latencies, there is a small offset between the frame timestamps and the gyroscope timestamps. We assume this offset $t_d$ is known and constant for the duration of the capture. In practice, we calibrate this offset as detailed in section 3.5.

We use a projective camera model with focal length $f$ and center of projection $(c_x, c_y)$; these three parameters define the entries of the camera intrinsic matrix $K$. The parameters are calibrated off-line using the OpenCV library [15]. With the $K$ matrix known, the relationship between corresponding points $\mathbf{x}_1$ and $\mathbf{x}_2$ on two different frames captured by a rolling-shutter sensor subject to rotational motion is

$$\mathbf{x}_2 = K R_c(t_2) R_c^{-1}(t_1) K^{-1} \mathbf{x}_1, \tag{2}$$

where the rotation matrix $R_c$ represents the camera orientation in the camera's coordinate system as a function of time, and $t_1$ and $t_2$ are the row timestamps for points $\mathbf{x}_1$ and $\mathbf{x}_2$.

We can re-write Equation 2 with respect to the gyroscope coordinate system and time origin as

$$\mathbf{x}_2 = K T R_g(t_2 + t_d) R_g^{-1}(t_1 + t_d) T^{-1} K^{-1} \mathbf{x}_1, \tag{3}$$

where $R_g$ is the orientation derived from the gyroscope, $T$ is the transformation between the camera and the gyroscope coordinate systems, and $t_d$ is the aforementioned time offset between the gyroscope and camera data streams. Since most mobile devices have the gyroscope and camera rigidly mounted with axes parallel to each other, $T$ is simply a permutation matrix. In our implementation, the transformation $T$ is known since the Android operating system defines a coordinate system for sensor data [16].

Insta360
Exhibit 1004 Page 5

### 3.2 Motion Model and Smoothing Algorithm

We parametrize the camera path with the camera's orientation and angular velocity at each frame. We represent the physical and virtual camera orientations at frame $k$ with the quaternions $\mathbf{p}(k)$ and $\mathbf{v}(k)$. The physical and virtual angular velocities are computed as the discrete angular changes from frame $k$ to frame $k+1$, and are represented as $\mathbf{p}_\Delta(k)$ and $\mathbf{v}_\Delta(k)$. Since the framerate is constant, time is implicit in this representation of the velocity. For each new frame $k$, our smoothing algorithm computes $\mathbf{v}(k)$ and $\mathbf{v}_\Delta(k)$ using the virtual parameters from the last frame, and the physical camera parameters from the last frame, the current frame, and optionally a small buffer of future frames (5 or less).

Our smoothing algorithm creates a new camera path that keeps the virtual camera static when the measured motion is small enough to suggest that the actual intention is to keep the camera static, and that otherwise follows the intention of the measured motion with smooth changes in angular velocity. As a first step, we hypothesize a new orientation for the virtual camera by setting

$$\hat{\mathbf{v}}(k) = \mathbf{v}(k-1) \cdot \mathbf{v}_\Delta(k-1), \tag{4}$$

where $\cdot$ denotes the quaternion product. Simply, this equation is computing a new camera orientation by rotating the camera from its last known orientation while keeping its angular velocity. Given this hypothetical camera orientation $\hat{\mathbf{v}}(k)$, we use Equation 2 to compute the coordinates of the corners of the resulting crop polygon. In virtual camera space, the crop polygon is a fixed rectangle centered at the image center, but in physical camera space, it may be be skewed or warped, and moves around within the frame, as shown in Figure 1. The crop polygon is smaller than the input size, which leaves a small amount of "padding" between the polygon borders and the input frame edges, as shown in Figure 2. We divide this padding into two concentric zones, which we will refer to as the "inner region" and "outer region". When the hypothetical crop polygon lies within the inner region of the image we assert that the hypothesis $\hat{\mathbf{v}}(k)$ is good and make it the current camera orientation. In practice, we find it advantageous to let the motion decay to zero in this case, which biases the virtual camera towards remaining still when possible. Thus, if the crop polygon remains completely within the inner region, we reduce the angular change by a decay factor $d$ and set the new virtual camera configuration to:

$$\mathbf{v}_\Delta(k) = \mathrm{slerp}(\mathbf{q}_I, \mathbf{v}_\Delta(k-1), d), \tag{5}$$

and

$$\mathbf{v}(k) = \mathbf{v}(k-1) \cdot \mathbf{v}_\Delta(k-1). \tag{6}$$

Here $\mathbf{q}_I$ represents the identity quaternion, and the slerp function is the spherical linear interpolation [17] between the two quaternions. In our implementation, we set the mixing weight to $d \approx 0.95$, so that the angular change is only slightly reduced each frame.

Insta360
Exhibit 1004 Page 6

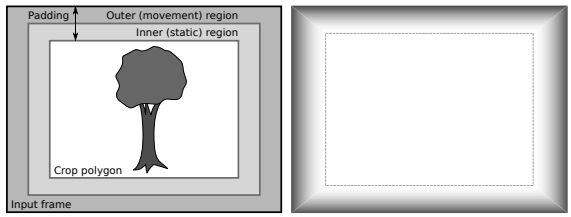


**Fig. 2.** Left: Crop polygon and the division of the padding space. Right: Velocity mixing weight. Dark blue represents a strong weight (taking the input velocity); white represents a small weight (keeping the current velocity).

If any part of the hypothetical crop polygon lies outside the inner region, we update the virtual camera's angular velocity to bring it closer to the physical camera's rate of change:

$$\mathbf{v}_{\Delta}(k) = \mathrm{slerp}(\mathbf{p}'_{\Delta}(k), \mathbf{v}(k-1), \alpha). \tag{7}$$

Here $\mathbf{p}'_{\Delta}$ is the orientation change that preserves the relative position of the crop polygon from one frame to the next, calculated as

$$\mathbf{p}'_{\Delta}(k) = \mathbf{p}(k) \cdot \mathbf{p}^*(k-1) \cdot \mathbf{v}(k-1), \tag{8}$$

where $\mathbf{p}^*$ denotes the quaternion conjugate that inverts the rotation. This equation calculates the physical camera motion from the previous to the current frame in the virtual camera reference coordinate system. The term $\alpha$ is a mixing weight that is chosen based on how much padding remains between the crop polygon and the edge of the frame, as illustrated in the right hand side of Figure 2. Intuitively, if the crop polygon is only slightly outside the inner region, $\alpha$ should be close to 1, assigning a higher weight to the current velocity. Conversely, if the hypothetical crop polygon is near the edge (or even outside), $\alpha$ should be 0, so that the input velocity is matched, and the crop polygon remains in the same position relative to the input frame. We calculate $\alpha$ with

$$\alpha = 1 - w^{\beta}, \tag{9}$$

where $w \in (0, 1]$ is the maximum protrusion of the crop polygon beyond the inner region, and $\beta$ is an exponent that determines the sharpness of the response. In the extreme case where any corner of the crop polygon would fall outside the input frame, $w$ takes a value of 1, forcing $\alpha$ to 0 and causing the virtual camera to keep up with the physical camera.

This algorithm works well, but it occasionally has to make quick changes in velocity when the crop rectangle suddenly hits the edge. If frames can be buffered within the camera pipeline for a short time before being processed, then a larger time window of gyroscope data can be examined, and sharp changes can be preemptively avoided. In the remainder of this section, we extend our algorithm to use data from a look-ahead buffer to calculate a smoother path.

Insta360
Exhibit 1004 Page 7

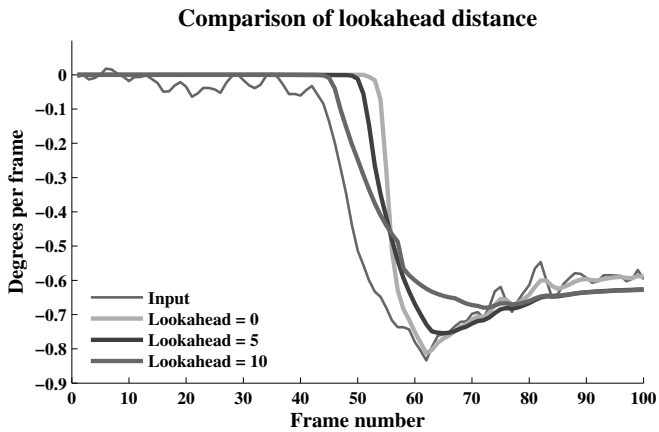


**Fig. 3.** Comparison of paths for varying lookahead distances. Larger lookahead values require more data to be buffered, but produce smoother output paths.

We can span a larger window of frames by projecting the virtual camera orientation forward in time and comparing it to the actual orientation at the "future" time. Let $a$ be the number of frames to look ahead, and hypothesize

$$\mathbf{v}(k+a) = \mathbf{v}(k-1) \cdot \mathbf{v}_{\Delta}(k)^{a+1}. \tag{10}$$

We can then compute $\mathbf{v}_{\Delta}(k+a)$ and $\mathbf{v}(k+a)$ as we described for the no-lookahead case. If the projection of the crop polygon $a$ frames into the future is outside the inner region, we can update $\mathbf{v}_{\Delta}(k)$ to

$$\mathbf{v}_{\Delta}(k) = \mathrm{slerp}(\mathbf{v}_{\Delta}(k+a), \mathbf{v}_{\Delta}(k), \gamma), \tag{11}$$

where $\gamma$ is a mixing factor that defines how much of the lookahead angular change we should mix with the current one. Using values of $\gamma$ close to 1 provides a preemptive nudge in the right direction, without being a hard constraint. Note that we do not update the virtual camera position that we computed without lookahead, we only update the virtual camera velocity that we will be using for the next frame.

Figure 3 shows a comparison of paths for a range of lookahead distances (measured in frames). Larger lookahead values produce smoother paths, since they effectively "predict" large motions and gently cause the output to start moving. But it is important to note that our algorithm can work without lookahead and still produce good results.

### 3.3 Output Synthesis and Rolling-Shutter Correction

Once we have computed the new orientation of the virtual camera, we can synthesize the output by projectively warping the crop polygon from the video input to the virtual camera. Our crop polygon is essentially a sliced rectangle with multiple

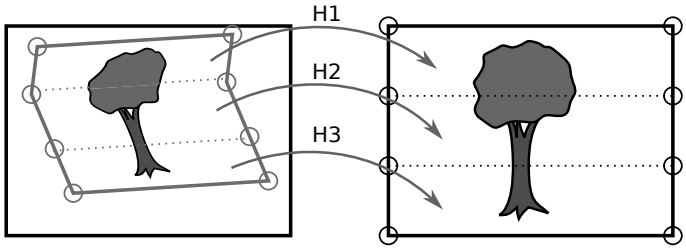


**Fig. 4.** Rolling-shutter correction is done by dividing the crop polygon in slices, each of which is subject to a different projective warp

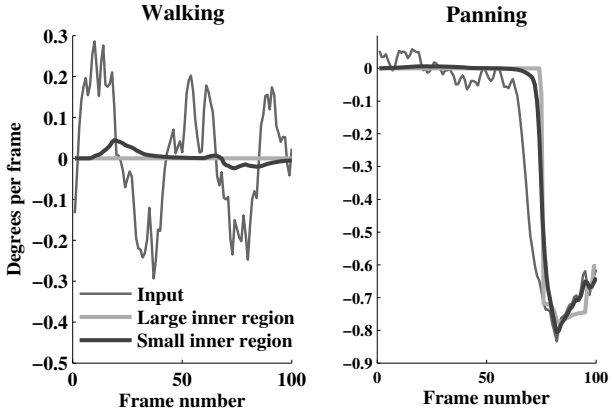


**Fig. 5.** Single-axis comparison of the effects of the inner region size on the smoothing result. A larger inner region can filter small motions more aggressively (left), but often produces sharper motions when the camera moves abruptly (right).

knee-points on the vertical edges, as shown in Figure 4. The knee-points allow us to use a different transform for every slice of the polygon and fix rolling-shutter distortions. For every slice we compute a homography matrix according to Equation 2. We fix the rotation matrix $R_c(t_2)$ to the orientation of the virtual output camera, and compute $R_c(t_k)$, the orientation of the input camera at each knee-point, from the gyroscope data. We set the coordinates of the crop polygon as texture coordinates of an OpenGL shader program that projectively maps the crop polygon from the input frame to the virtual camera. Note that in order to effectively correct for rolling-shutter effects, the gyroscope sampling rate should be higher than the frame read-out time. In our implementation we sample the gyroscope at 200 Hz and use a total of 10 slices, or 9 knee-points per vertical edge.

### 3.4 Parameter Selection

The most important parameters are the size of the output crop polygon and the amount of padding allocated to the inner and outer regions. The crop size is a trade-off between smoothing and image quality: larger crop polygons preserve

more of the input image, but leave less padding for smoothing out motions. The padding allocation is a trade-off between completely removing motion and the smoothness of the remaining motion. As illustrated in Figure 5, a large inner region (green) is able to flatten out larger motions such as walking, but must move more abruptly when the crop window approaches the edge of the frame.

### 3.5 Gyroscope and Camera Calibration

We solve for the time offset $t_d$ using a calibration procedure that we developed for this purpose. We place a calibration pattern, which consists of an asymmetric grid of circles, in front of the camera. Then we record a video and the gyroscope readings while rotating the camera vigorously. The circles are easily tracked across frames, even in the presence of motion blur and rolling shutter effects. We use the centroid of each circle as a feature point, and solve for $t_d$ iteratively by minimizing the sum of re-projection errors according to Equation 3. By repeating the calibration on multiple data sets we determined that the offset $t_d$ is nearly constant. We also considered the possibility of doing an on-line calibration, by tracking key-points as each frame is captured [18], but the off-line method proved sufficient for our purposes.

The integration of any static offset in the gyroscope measurements will result in an estimated orientation that slowly drifts away from the ground truth. However, our stabilization algorithm is not affected by such drift because it smooths the relative change of orientation. We measure orientation changes in a window of one to five frames, and in such a short time span, the integration drift is negligible.

## 4 Results

We experimented with a prototype Android tablet, in which we installed a modified version of the Android OS that reads and saves the gyroscope measurements while recording video. Using this tablet we recorded a series of videos representing typical use cases of casual video captured by a cell phone. For comparison here, we discuss three scenes: a video recorded while walking, a video focusing on a fountain, and panning video tracking a walking person. For the stabilization algorithm, we set the width and height of the crop rectangle to be 80% of the original video size, allocate the remaining 20% in width and height equally to the inner and outer regions, and set the lookahead to 5 frames. The results show that our method eliminates the high-frequency jitter while keeping the camera as still as possible. To compare against previous work we also implemented video stabilization using a Gaussian low-pass filter of the derived gyroscope orientations as done by Karpenko *et al.* [1], using a local window of eleven frames, giving a forward lookahead of 5 frames as in our method and the same 80% crop ratio. All original videos and results are included in the supplementary material that accompanies this paper.

Figure 6 shows a quantitative comparison of the camera's angular velocity rate of change for the original video, a stabilized video produced with the Gaussian

Insta360
Exhibit 1004 Page 10

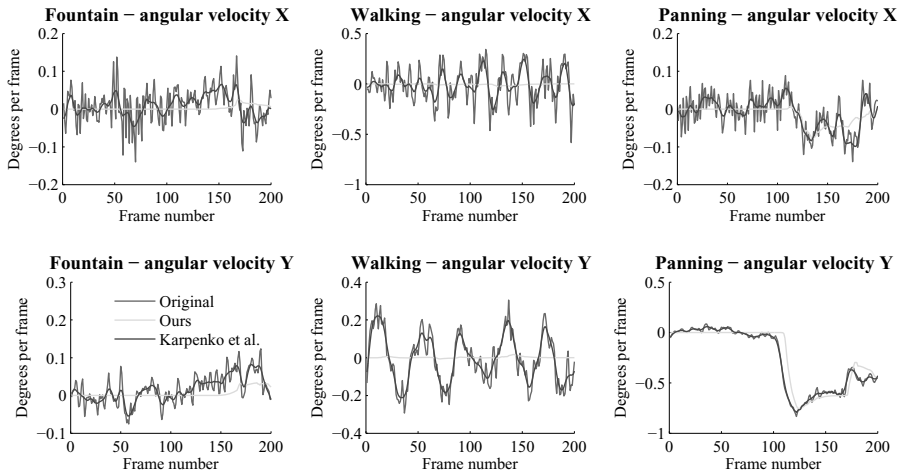


**Fig. 6.** Angular velocities of the camera in the X and Y axis. We show three different scenes: a handheld camera pointed at a fountain as steadily as possible (left), a camera held by a person walking (middle) and a camera panning while tracking a moving object (right). Our method (green) can eliminate most of the small camera motions. On the other hand, the Gaussian filter with a small window of support proposed in [1] can remove high-frequency motions, but fails to completely remove camera shake.

filtering of [1], and a stabilized video produced with our method. Our method dramatically reduces the jitter in angular velocity, and even removes it when possible, thus producing smoother results. The fountain video shows that we can effectively simulate a static camera. Our method keeps the virtual camera fixed for the first hundred and fifty frames and, when moving, follows the actual camera smoothly. In the walking video, our method removes most of the angular acceleration, producing an output video which is pleasing to view. The smoothed camera path is nearly free of rotational motions, but still contains small vertical periodic motions due to translations of the camera while walking. This is a limitation of our method, which cannot track translational camera motion; we discuss how to address this in the future work section. Finally, the panning video highlights that our method can produce a smooth virtual panning camera; the graph of angular velocity in the Y axis shows that our result follows the velocity of the original camera. The green line follows the original velocity (red line) during frames in the range [100, 150] at the same rate, though slightly shifted in time. This is by design, since our method tries to keep the static camera for as long as possible and then follows the original camera velocity. "Catching up" with the motion to center the crop window would require introducing additional accelerations.

To further benchmark our method against the state-of-the-art in video stabilization, we uploaded the videos to YouTube and ran the stabilization tool, which is based on the work of Grundmann *et al.* [3]. Qualitatively, our method

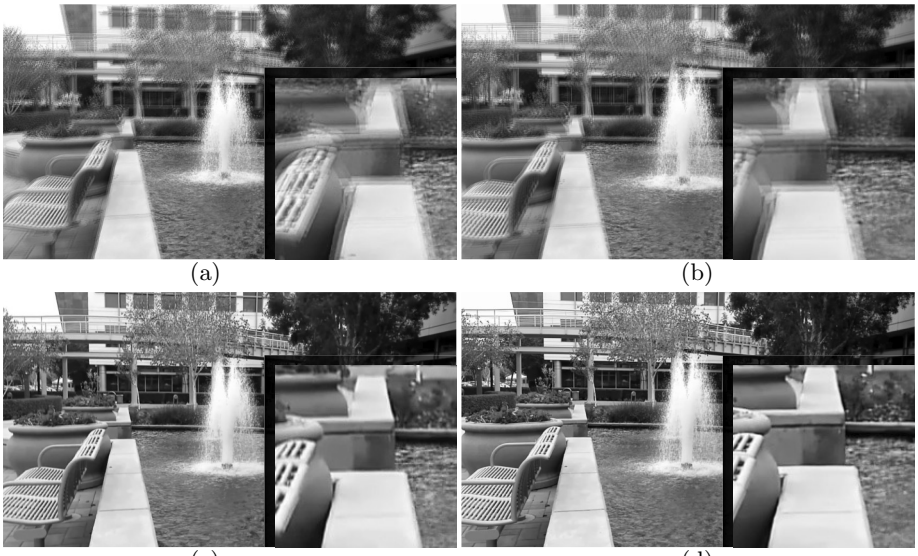

**Fig. 7.** Four different frames of the Fountain video blended together. The frames were sampled at a 10 frame interval from: (a) the original video, (b) the video stabilized using a truncated Gaussian filtering, (c) the video stabilized by YouTube, (d) the video stabilized using our method. The blended results on (a) and (b) look blurry, due to the motion of the camera. In contrast, the results from (c) and (d) look sharp, since both methods were able to eliminate the camera motion.

stabilization results are similar to those produced by the YouTube stabilization. In Figure 7 we show the results of blending together four frames taken at 10-frame intervals. The blended image from the original video is blurry, due to the motion of the camera. On the other hand, the blended images generated by sampling the stabilized videos produced from our and Grundmann's method show sharp results, showing both methods were able to remove the camera motion. Grundmann's method can dynamically adjust the size of the crop window, which we can observe is larger than ours in some cases, therefore retaining a larger area of the frame and reducing the zoom effect in the stabilized video. Our method keeps the size of the crop polygon fixed, but there is no impediment to making it a dynamic part of the virtual camera configuration, as we discuss in the next section. Additional comparisons are included with the supplementary material video.

To fix rolling-shutter effects we determined from the sensor driver that the effective frame length of our 1080p video recordings was 2214 lines. This corresponds to a read-out period of 16 ms and a blanking period of 17 ms. This is fast enough that it is difficult to perceive rolling-shutter wobble effects within a single frame. However, the effects quickly become visible in a video, even with moderate shake. Figure 8 visually shows the effect of rolling shutter correction

**Fig. 8.** Static visualization of rolling-shutter correction. The left image shows the average of four frames sampled at 10-frame intervals, without rolling shutter correction. Stabilization is applied using the top of the frame as a reference, but rolling shutter wobble causes the bottoms of the frames to be badly aligned. The right image shows the same four frames with rolling shutter correction applied. The wobble is greatly reduced, and the entire frame is much sharper.

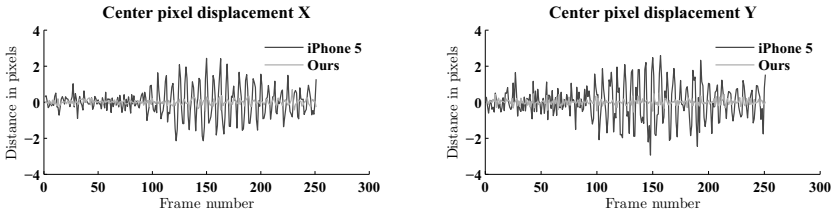


**Fig. 9.** Frame-to-frame displacement of the image center for a video sequence captured with our real-time implementation and an iPhone 5. Our algorithm is able to reduce motion for low-amplitude high-frequency shakes, as shown in the plot above.

by comparing a series of frames in a video with rolling shutter wobble. The supplementary video contains additional video which demonstrates the efficacy of our method.

While the comparisons above were done offline in order to run the same video through multiple filters, we have also implemented our algorithm within the Android video capture stack, where it runs in real time. The filter itself, running as a single thread on the CPU, operates in 160 microseconds. The image warp, implemented as an OpenGL shader on the tablet's GPU, runs in 15 ms.

Using this real-time implementation and fixing the crop ratio to 90% of the original frame size, we ran our prototype tablet side-by-side with an iPhone 5, both fixed rigidly to a supporting frame. A qualitative comparison of the video sequence shows both look similar, with some instances in which our algorithm produces better results, such as regions of high-frequency low-amplitude shakes, as shown in Figure 9.

Our source videos, results and supplementary material video are available at `https://research.nvidia.com/publication/non-linear-filter-gyroscope-based-video-stabilization`.

## 5 Conclusions and Future Work

We have presented a novel solution for video stabilization with rolling-shutter correction using the gyroscope in a mobile device. Our method is fast and can run in real-time within the camera processing pipeline. The stabilization can work on each incoming frame as it is received, but can also benefit from an optional buffer window that holds up to five frames. By using the gyroscope to track the camera motion we are able to do better in some scenes than most feature-based methods, which fail when there is lack of texture, excessive blur, or large foreground moving objects. We also improve on previous techniques that use the gyroscope for camera motion by using a novel filtering approach that results in smoother motions. To achieve this we assume the intention of the person recording the video is to keep the camera as static as possible or make a smooth linear motion. These assumptions hold for a wide range of videos.

Our method may under-perform the state-of-the-art in feature tracking methods on videos where the camera is subject to large translations. Translational motion cannot be tracked by the gyroscope. While it might be possible to use the accelerometer that accompanies the gyroscope in most mobile devices, estimation of translation from the accelerometer readings is less robust due to the double integration of the accelerometer data. In addition, large translations will cause occlusions and dis-occlusions in the image due to the parallax. In this case, projectively warping the crop polygon can cause distortions near the occlusion boundaries. Unfortunately, more sophisticated methods that can handle parallax [6] cannot run in real-time.

Our method can run on scenes with no trackable features or large motion of foreground objects, which feature-based might struggle with. In addition, our method works at a fraction of the computational time and cost because we don't need to compute features at all.

We intend to improve the system in the future in several ways. Firstly, our current algorithm keeps a fixed-size crop window; better stabilization might be achieved if we can vary the crop size smoothly across frames. In addition, we would like to explore the possibility of adding the ability to handle small translations by visual tracking of a sparse set of features. This tracking will be simplified by that fact that the camera rotation is already accounted for from the gyroscope data, and could be further conditioned to be done only when a significant change in acceleration is detected by the accelerometer. Finally, we would like to explore the possibility of storing the gyroscope readings as a separate track of the output video file, to enable further off-line stabilization using the gyroscope data if desired.

**Acknowledgments.** We thank Orazio Gallo for his helpful suggestions and feedback on earlier revisions of this paper.

## References


1. Karpenko, A., Jacobs, D., Baek, J., Levoy, M.: Digital video stabilization and rolling shutter correction using gyroscopes. Technical Report CTSR 2011-03, Department of Computer Science, Stanford University (2011)
2. Invensense Corporation: MPU-6050 Product Specification, http://invensense.com/mems/gyro/documents/PS-MPU-9250A-01.pdf
3. Grundmann, M., Kwatra, V., Castro, D., Essa, I.: Calibration-free rolling shutter removal. In: IEEE ICCP (2012)
4. Gleicher, M.L., Liu, F.: Re-cinematography: improving the camera dynamics of casual video. ACM Multimedia (2007)
5. Grundmann, M., Kwatra, V., Essa, I.: Auto-directed video stabilization with robust l1 optimal camera paths. In: IEEE CVPR (2011)
6. Liu, S., Yuan, L., Tan, P., Sun, J.: Bundled camera paths for video stabilization. ACM TOG 32(4) (2013)
7. Matsushita, Y., Ofek, E., Ge, W., Tang, X., Shum, H.Y.: Full-frame video stabilization with motion inpainting. IEEE PAMI 28(7) (2006)
8. Baker, S., Bennett, E., Kang, S.B., Szeliski, R.: Removing rolling shutter wobble. In: IEEE CVPR (2010)
9. Liu, F., Gleicher, M., Jin, H., Agarwala, A.: Content-preserving warps for 3D video stabilization. ACM TOG 28(3) (2009)
10. Liu, F., Gleicher, M., Wang, J., Jin, H., Agarwala, A.: Subspace video stabilization. ACM TOG 30(1) (2011)
11. Hanning, G., Forslow, N., Forssén, P., Ringaby, E., Tornqvist, D., Callmer, J.: Stabilizing cell phone video using inertial measurement sensors. In: IEEE ICCV Workshops (2011)
12. Joshi, N., Kang, S.B., Zitnick, C.L., Szeliski, R.: Image deblurring using inertial measurement sensors. ACM TOG 29(4) (2010)
13. Forssen, P., Ringaby, E.: Rectifying rolling shutter video from hand-held devices. In: IEEE CVPR (2010)
14. Oth, L., Furgale, P., Kneip, L., Siegwart, R.: Rolling shutter camera calibration. In: IEEE CVPR (2013)
15. Various: OpenCV library, http://code.opencv.org
16. Google: Android operating system developers' API guide, http://developer.android.com/guide/topics/sensors/sensors_overview.html
17. Shoemake, K.: Animating rotation with quaternion curves. ACM TOG 19(3) (1985)
18. Li, M., Mourikis, A.: 3-D motion estimation and online temporal calibration for camera-IMU systems. In: IEEE ICRA (2013)