Real-Time Hyperlapse Creation via Optimal Frame Selection

Neel Joshi Wolf Kienzle Mike Toelle Matt Uyttendaele Michael F. Cohen
Microsoft Research

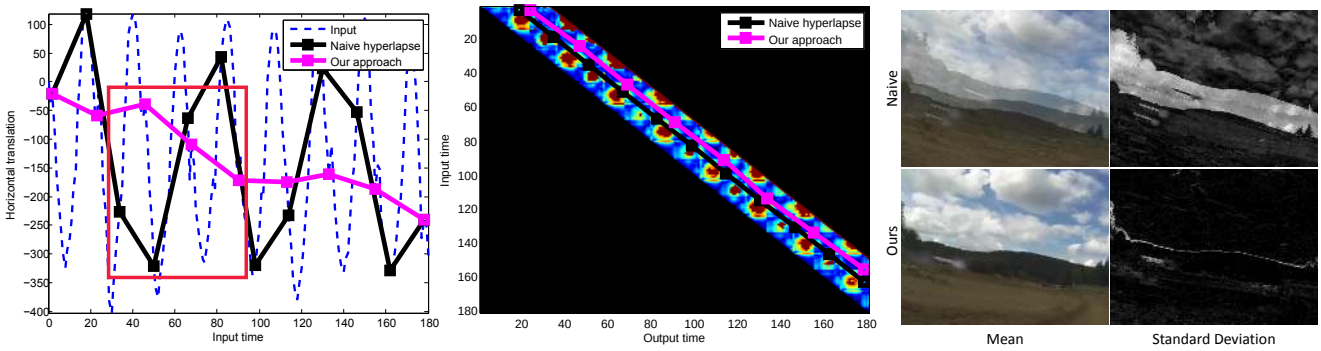

Figure 1: *Hand-held videos often exhibit significant semi-regular high-frequency camera motion due to, for example, running (dotted blue line). This example shows how a naive 8x hyperlapse (i.e., keeping 1 out every 8 frames) results in frames with little overlap that are hard to align (black lines). By allowing small violations of the target skip rate we create hyperlapse videos that are smooth even when there is significant camera motion (pink lines). Optimizing an energy function (color-coded in Middle image) that balances matching the target rate while minimizing frame-to-frame motion results in a set of frames that are then stabilized. (Right) To illustrate the alignment we show the mean and standard deviation of three successive frames (in red box on the Left plot)* after *stabilization for the naive hyperlapse (Top Right) and our result (Bottom Right) – these show that our selected frames align much better than those from naive selection.*

Abstract

Long videos can be played much faster than real-time by recording only one frame per second or by dropping all but one frame each second, i.e., by creating a *timelapse*. Unstable hand-held moving videos can be *stabilized* with a number of recently described methods. Unfortunately, creating a stabilized timelapse, or *hyperlapse*, cannot be achieved through a simple combination of these two methods. Two hyperlapse methods have been previously demonstrated: one with high computational complexity and one requiring special sensors. We present an algorithm for creating hyperlapse videos that can handle significant high-frequency camera motion and runs in real-time on HD video. Our approach does not require sensor data, thus can be run on videos captured on any camera. We optimally select frames from the input video that best match a desired target speed-up while also resulting in the smoothest possible camera motion. We evaluate our approach using several input videos from a range of cameras and compare these results to existing methods.

CR Categories: I.3.8 [Computer Graphics]: Applications; I.4.8 [Image Processing and Computer Vision]: Applications;

Keywords: time-lapse, hyperlapse, video stabilization

1 Introduction

The proliferation of inexpensive, high quality video cameras along with increasing support for video sharing has resulted in people taking videos more often. While increasingly plentiful storage has made it very easy to record long videos, it is still quite tedious to view and navigate such videos, as users typically do not have the time or patience to sift through minutes of unedited footage. One simple way to reduce the burden of watching long videos is to speed them up to create "timelapse" videos, where one can watch minutes of video in seconds.

When video is shot with a stationary camera, timelapse videos are quite effective; however, if there is camera motion, the speed-up process accentuates the apparent motion, resulting in a distracting and nauseating jumble. "Hyperlapse" videos are an emerging medium that addresses the difficulty of timelapse videos shot with moving cameras by performing camera motion smoothing (or "stabilization") in addition to the speed-up process. They have a unique appealing dynamism and presence.

The two main approaches for stabilizing camera motion are hardware-based and software-based. Hardware-based methods utilizing onboard gyros can be quite successful [Karpenko 2014], but require specialized hardware at capture time, thus cannot be applied to existing videos. As they are blind to the content of the video, they also fail to stabilize large foreground objects. Software-based computer vision methods operate on the pixels themselves. They range from 2D stabilization to full 3D reconstruction and stabilization. Existing 2D approaches can work well when camera motion is slow, but breakdown when the camera has high-frequency motion. 3D approaches work well when there is sufficient camera motion and parallax in a scene [Kopf et al. 2014], but have high computational cost and are prone to tracking and reconstruction errors when there is insufficient camera translation.

In this paper, we present an algorithm for creating hyperlapse videos that runs in real-time (30 FPS on a mobile device and even faster on a desktop) and can handle significantly more camera mo-

tion than existing real-time methods. Our approach does not require any special sensor data, thus can be run on videos captured by any camera. Similar to previous work in video stabilization, we use feature tracking techniques to recover 2D camera motion; however, unlike previous work, camera motion smoothing and speed-up are optimized *jointly*. We develop a dynamic programming algorithm, inspired by dynamic-time-warping (DTW) algorithms, that selects frames from the input video that both best match a desired target speed-up and result in the smoothest possible camera motion in the resulting hyperlapse video. Once an optimal set of frames is selected, our method performs 2D video stabilization to create a smoothed camera path from which we render the resulting hyperlapse.

We evaluate our approach using input videos from a range of camera types and compare these results to existing methods.

2 Related Work

Our work is related to previous work in 2D and 3D video stabilization and approaches directly designed for producing timelapse and hyperlapse videos.

2.1 Software-based video stabilization

Software-based video stabilization is the removal of undesirable high-frequency motion, often caused by the instability of handheld cameras. 2D stabilization is a well-known technique that operates by manipulating a moving crop window on the video sequence to remove much of the apparent motion of the camera. Corresponding features are detected and used to recover frame-to-frame camera pose as parameterized by a rigid transform. These camera poses are smoothed to produce a new set of transforms that are applied to create a new smooth camera path [Matsushita et al. 2006; Grundmann et al. 2011]. While 2D stabilization cannot model parallax, 3D methods can. 3D methods use structure-from-motion to estimate 6D camera pose and rough scene geometry and then render the scene from a novel smoothed camera path [Liu et al. 2009; Liu et al. 2011]. Both 2D and 3D approaches have been extended to compensate for rolling shutter artifacts [Baker et al. 2010; Forssen and Ringaby 2010; Liu et al. 2013].

2.2 Hardware-based video stabilization

Hardware-based approaches replace the feature tracking methods with a sensor-based approach. The most common commercial approach for reducing camera jitter is image stabilization (IS). These methods use mechanical means to dampen camera motion by offsetting lens elements or by translating the sensor to offset camera motion as measured by inertial sensors (i.e., gyros and accelerometers) [Canon 1993]. Recent work has shown how to use these sensors to directly measure the camera motion during capture [Joshi et al. 2010] and how to use this measured motion (similar to the software-based methods) for stabilization and rolling shutter correction [Karpenko et al. 2011].

2.3 Timelapse and hyperlapse methods

There are a few recent works that directly address creating timelapse and hyperlapse videos. The simplest approach is to perform timelapse by uniformly skipping frames in a video without any stabilization, which is possible in many professional video editing packages, such as Adobe Premiere. The Apple iOS 8 Timelapse feature uses this naive frame-skipping approach while adjusting the global timelapse rate so that the resulting video is 20-40 seconds long [Provost 2014]. Work by Bennett et al. [2007] creates non-uniform timelapses, where the skipping rate varies across the video as function of scene content.

The most direct approach to creating hyperlapse videos is to perform stabilization and timelapse sequentially in either order, i.e., first stabilize and then skip frames or skip first and then stabilize. The recent Instagram Hyperlapse app uses the latter approach [Karpenko 2014] using the hardware stabilization approach of Karpenko et al. [2011]. As noted above, the Instagram approach cannot be applied to existing video. In addition, since it is blind to the video content, it can only stabilize the global inertial frame and not lock onto large moving foreground objects. Our method produces comparable results to the Instagram approach and, as we will show, performs well in the presence of large moving foreground objects.

The most sophisticated hyperlapse work is that of Kopf et al. [2014], which uses structure-from-motion (SfM) on first person videos and re-renders the scene from novel camera positions using the recovered 3D geometry. By performing a global reconstruction and rendering from a combination of input frames for each output frame, Kopf et al.'s method can handle cases where the camera is moving significantly and there is significant parallax. Kopf et al. also perform path planning in 6D to choose virtual camera locations that result in smooth, equal velocity camera motions in their final results. This approach works well when there is sufficient camera motion and parallax in a scene, but has difficulty where the camera motion is small or purely rotational, as the depth triangulation in the SfM step is not well constrained. SfM can also have difficulties when the scene is dynamic. Furthermore, this approach has high computational cost – on the order of minutes per frame [Kopf et al. 2014]. Although our approach cannot always achieve the same smoothness, it comes close and is more robust to motions other than forward motion, such as static and rotating cameras, as well as in non-rigid scenes. Kopf et al. is, however, more robust in scenes with very high parallax, for example where the foreground is very close to a moving camera, i.e., where the ambiguity between translation and rotation breaks down. That said, most importantly, our method is three orders of magnitude faster, running at faster than real-time rates on a desktop and making it amenable to real-time performance on most mobile devices.

Our approach resides in between the Instagram approach and that of Kopf et al. We use 2D methods as in Instagram, but do not require inertial sensors and do not naively skip frames. Naive frame skipping can lead to poor results as it can result in picking frames that cannot be well stabilized (see Figure 1). Instead, we allow small violations of the target skip rate if that leads to a smoother output. Our selection of frames is driven by optimizing an energy function that balances matching the target rate and minimizing frame-to-frame motion in the output video. This allows us to handle high-frequency camera motion with significantly less complexity than 3D approaches.

In concurrent work, Poleg et al. [2014], as in this work, carefully sample frames on a video, particularly in semi-regular oscillating videos such as those captured when walking, to select frames leading to more stable timelapse results. They also suggest producing two simultaneous, but offset, tracks to automatically extract stereo pairs of video. This work is the most similar to that reported here in terms of the frame sampling based method leading to a hyperlapse. However, they rely on optical flow and a shortest path optimization as opposed to our homography plus dynamic programming approach which leads to our very fast results. Poleg et al. report times of approximately 6.5 seconds per input frame on a desktop vs. 30 frames per second on a mobile device for our approach, which a difference of more than two orders of magnitude.

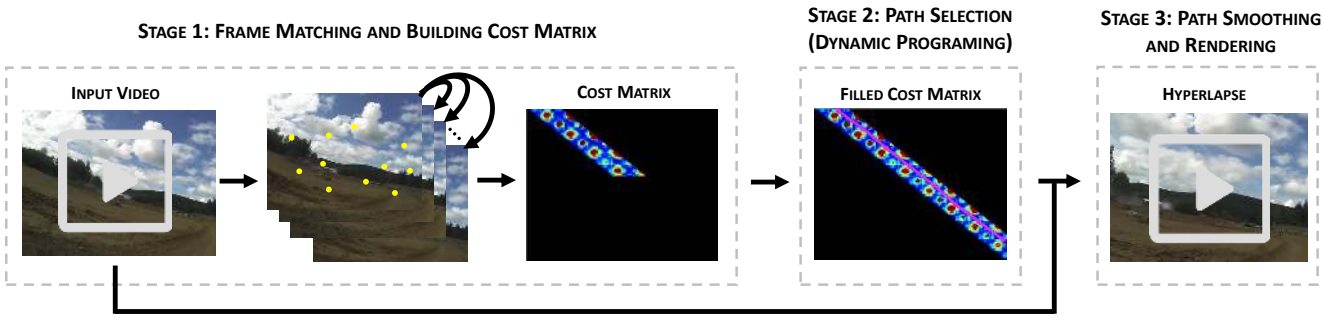

Figure 2: *Our method consist of three main stages. 1) Frame-matching: using sparse feature-based techniques we estimate how well each frame can be aligned to its temporal neighbors and store these costs as a sparse matrix, 2) Frame selection: a dynamic programming algorithm finds an optimal path of frames that balances matching a target rate and minimizes frame-to-frame motion, and 3) Path smoothing and rendering: given the selected frames, smooth the camera path and render the final hyperlapse result.*

2.4 Video frame matching

A related area of techniques use dynamic programming approaches for matching and aligning image sets or videos from different times or views [Kaneva et al. 2010; Levieux et al. 2012; Arev et al. 2014; Wang et al. 2014]. Our work draws some inspiration from these papers; however, our approach is more concerned with matching a video to itself and thus shares some similarities with work for detecting periodic events in videos [Schödl et al. 2000].

3 Overview

Our goal is to create hyperlapse videos at any speed-up rate with no constraints on the video camera used, scene content, or camera motion. The key inspiration for our algorithm comes from observation of naive hyperlapse results (i.e., timelapse followed by stabilization).

When input camera motions are fairly smooth, naive hyperlapse algorithms work quite well [Karpenko 2014]; however, when there is significant high-frequency motion, the results can be quite unwatchable [Kopf et al. 2014]. When there is high frequency motion, for example rapid swaying due to running or walking, it is easy to "get unlucky" when naively picking frames and choose frames that have very little overlap. Thus, during stabilization, it is not possible to align these frames well, which is critical to creating a smooth result. The key observation is that in many videos the jittery motions are semi-periodic (e.g., due to handshake, walking, running, or head motions).

Figure 1 illustrates a simple case of this. Consider a camera motion that is a semi-periodic horizontal jitter. Here, naive skipping chooses frames that have very little overlap and potentially a lot of parallax between them. However, by allowing small violations of the target skip rate, one can choose frames where there is very little motion, resulting in better alignment and a smoother result.

The key contribution of our work is defining a cost metric and optimization method that picks a set of frames that are close to the target speed yet can be aligned well and thus stabilized in the sped-up output video. It is worth noting that the cost metric is highly correlated with the needs of the stabilizer, thus creating a unified optimal frame selection and stabilization framework.

As illustrated in Figure 2, our method consists of three main stages:

1. **Frame-matching:** using sparse feature-based techniques we estimate how well each frame can be aligned to its temporal neighbors.
2. **Frame selection:** a dynamic-time-warping (DTW) algorithm to find an optimal path of frames that trades-off matching the target rate with minimizing frame-to-frame motion.
3. **Path smoothing and rendering:** given the selected frames, smoothing the camera path to produce a stabilized result.

We will discuss these three main steps in the following section and several additional refinements in Section 5.

4 Finding an Optimal Path

Given an input video represented as sequence of frames $F = \langle 1, 2, ..., T \rangle$, we define a *timelapse* as any path p that is a strictly monotonically increasing subsequence of F. The path inherently serves as a mapping from output time to input time $p(\hat{t}) = t$, where $t \in F$.

In our framework, a *hyperlapse* is a desired path p where the time between subsequent frames is close to a target speed-up yet subsequent frames can be aligned well and the overall result has smooth camera motion. We formulate finding the best path as an optimization problem that minimizes an objective function that consists of three terms: a cost that drives towards optimal frame transitions, a term that drives towards matching the target speed-up rate, and a third that minimizes the acceleration. This cost function is used to populate a cost matrix and a path through the matrix directly corresponds to a path p. We use an approach inspired by dynamic programming and dynamic-time-warping algorithms to find the optimal path.

4.1 Frame matching cost

An optimal frame-to-frame transition is where both frames can be aligned well and have significant overlap. The first criteria is necessary for a smooth visual transition between frames, while the latter ensures that the transition can be done with minimal cropping.

Given two video frames $F_{t=i}$ and $F_{t=j}$, denote $T(i, j)$ as the homography that warps F_i to F_j (we drop the "$t=$" notation for brevity), or more specifically maps a set of features points between the frames. We compute $T(i, j)$ using a standard RANSAC (RANdom SAmple Consensus) method on sparse feature points [Fischler and Bolles 1981] – we use *Harris* corners each with a corresponding *Brief* descriptor [Calonder et al. 2010] and compute 500 features per frame. Given $T(i, j)$ we define two cost functions corresponding to our criteria for a good frame-to-frame transition.

The first term is an alignment cost:

$$C_r(i,j)=\frac{1}{n}\sum_{p=1}^{n}\left|\left|(x_p,y_p)_j^T-T(i,j)(x_p,y_p)_i^T\right|\right|_2. \quad (1)$$

This cost is equivalent to the average of the 2D geometric re-projection error for n corresponding features selected by the RANSAC process.

The second term measures motion and penalizes lack of overlap between the frames:

$$C_o(i,j)=\left|\left|(x_0,y_0)^T-T(i,j)(x_0,y_0)^T\right|\right|_2, \quad (2)$$

where $(x_0,y_0,1)$ is the center of the image. This is equivalent to the magnitude of translation of the center of the image between the two frames, which is a function of the (out-of-plane) rotation and translation of the camera. This serves as an estimate of the motion of the camera look-vector.

These costs are combined into a single cost function:

$$C_m(i,j)=\begin{cases} C_o(i,j) & C_r(i,j)<\tau_c \\ \gamma & C_r(i,j)\geqq\tau_c \end{cases}, \quad (3)$$

where $\tau_c=0.1*d$, $\gamma=0.5*d$ and d is the image diagonal in pixels. This single function simply states that if the alignment error is low (in this case less than 10% of the image diagonal), the cost is equal to the motion cost; however, if the alignment error is too large, the transformation and motion cost are not reliable, so $T(i,j)$ is set to identity and the cost is set to a large cost – half the diagonal of the image, which corresponds to zero image overlap. These cases are due to abrupt transitions in the video. The large cost causes the optimization to avoid choosing the frame; however, it can be chosen if there is no other. This ensures that the algorithm can always find a full path through the video.

4.2 Velocity and acceleration costs

The matching costs ensures that the hyperlapse is smooth in terms of camera motion; however, it is also important for a hyperlapse to achieve a desired speed-up. Our second cost function penalizes straying from a user-input target speed-up:

$$C_s(i,j,\nu)=min(||(j-i)-\nu||_2^2,\tau_s). \quad (4)$$

This term is a truncated L^2 on the difference between the actual jump between frames i and j and the target rate τ. We have empirically determined a setting of $\tau_s=200$; however, the results are not that sensitive to the exact value of this parameter.

The previous costs lead to a balance between choosing frames that lead to smooth motion versus violating the target speed-up rate. We have found that violating the target rate to achieve smoother camera motion can cause a perceptible visual jump as time suddenly accelerates. An acceleration penalty reduces this effect by making the speed changes more gradual:

$$C_a(h,i,j)=min(||(j-i)-(i-h)||_2^2,\tau_a). \quad (5)$$

This term is also a truncated L^2 and we also have empirically determined a setting of $\tau_a=200$; however, the results are also not that sensitive to the exact value of this parameter.

Thus, the total cost for a given speed-up rate ν for a triplet of frames is:

$$C(h,i,j,\nu)=C_m(i,j)+\lambda_s C_s(i,j,\nu)+\lambda_a C_a(h,i,j). \quad (6)$$

Algorithm: Stage 2: Path Selection

Input: ν

Initialization:

for $i=1$ to g **do**
  for $j=i+1$ to $i+w$ **do**
    $D_\nu(i,j)=C_m(i,j)+\lambda_s C_s(i,j,\nu)$
  end for
end for

First pass: populate D_ν

for $i=g$ to T **do**
  for $j=i+1$ to $i+w$ **do**
    $c=C_m(i,j)+\lambda_s C_s(i,j,\nu)$
    $D_\nu(i,j)=c+\min_{k=1}^{w}[D_\nu(i-k,i)+\lambda_a C_a(i-k,i,j)]$
    $T_\nu(i,j)=\operatorname{argmin}_{k=1}^{w}[D_\nu(i-k,i)+\lambda_a C_a(i-k,i,j)]$
  end for
end for

Second pass: trace back min cost path

$(s,d)=\operatorname{argmin}_{i=T-g,j=i+1}^{T,i+w} D_\nu(i,j)$
$\mathbf{p}=\langle d\rangle$
while $s>g$ **do**
  $\mathbf{p}=prepend(\mathbf{p},s)$
  $b=T_\nu(s,d)$
  $d=s$, $s=b$
end while

Return: p

Figure 3: *A first pass populates a dynamic cost matrix D where each entry $D_\nu(i,j)$ represents the cost of the minimal cost path that ends at frame $t=j$. A trace-back matrix T is filled to store the minimal cost predecessor in the path. The optimal minimum cost path is found by examining the final rows and columns of D and the final path $\boldsymbol{p}$ is created by walking through the trace-back matrix.*

We have empirically determined that settings of $\lambda_s=200$ and $\lambda_a=80$ work well. Again, the results are not very sensitive to the exact value of these parameters, and there is a certain amount of personal preference in these settings, i.e., depending on how important it is to a user to match the target speed or have smooth changes in velocity.

4.3 Optimal frame selection

We define the cost of a path for a particular target speed ν as:

$$\phi(p,\nu)=\sum_{\tilde{t}=1}^{\tilde{T}-1}C(p(\tilde{t}-1),p(\tilde{t}),p(\tilde{t}+1),\nu). \quad (7)$$

Thus the optimal path $\mathbf{p}$ is:

$$\mathbf{p}_\nu=\operatorname*{argmin}_p \phi(p,\nu). \quad (8)$$

We compute the optimal path using an algorithm inspired by dynamic programming (DP) and dynamic-time-warping (DTW) algorithms that are used for sequence alignment. The algorithm consists of three stages.

In Stage 1 (see Figure 2) the matching cost is computed using frame matching, as described in Section 4.1 and is stored in a sparse, static cost matrix C_m for all frames $\langle 1, 2, ..., T \rangle$. [1]

We only construct the upper triangle of C_m as it is symmetric. In principle, C_m can be fully populated, which captures the cost of a transition between any two frames, but as an optimization we compute a banded (or windowed) version of C, where band w defines the maximum allowed skip between adjacent frames in the path. This is similar to a windowed DTW algorithm. For a particular input video and value of w, C_m is static and computed once and re-used for generating any speed up $\nu <= w$. Section 6 discusses the values we use for w.

Stage 2 is the DP algorithm and it consists of two passes. A first pass populates a dynamic cost matrix D_ν, which is function of the user specified target speed-up ν. While C_m is computed once in Stage 1, the DP algorithm has to be run for each desired speed up. We use the same window size w for D_ν and C_m. The algorithm constructs D_ν by iterating over its elements, where each entry $D_\nu(i,j)$ represents the running minimal cost path that ends with the frames i and j. This cost is a sum of C_m, C_s, and C_a.[2] At each step of filling D_ν, the algorithm accumulates the cost by evaluating the C_m and C_s cost functions at i and j and finding the lowest cost preceding frame h, which depends on the previous costs and C_a. h stored in traceback matrix T for the second pass of the algorithm.

Once D is fully populated the second pass finds an optimal path by finding the minimal cost in the final rows and columns of D_ν (within some "end-gap" g) and then traces the path through the matrix. We allow "start gaps" and "end gaps" g, as the optimization performs better when it is not restricted to include the first and last frame. This is a common approach in DTW algorithms. For long hyperlapse sequences, the start and end gaps are not noticeable in the resulting video. We set $g = 4$; however, one could easily use larger gaps for long sequences. Figure 3 gives detailed pseudo-code of our frame selection algorithm.

4.4 Path smoothing and rendering

Once an optimal set of frames is selected, the next step is to compute a smooth camera motion path and warp the images to generate the result. This step is essentially the same as running standard video stabilization, which computes a sequence of frame-to-frame transformations and smooths them to create a stable result.

We use a variant of the approach used by Joshi et al. [2012]. The alignment is computed by extracting image features (*Harris* corners with a corresponding *Brief* descriptor [Calonder et al. 2010]) for each frame and performing a windowed search between adjacent frames to find matching features. A feature is determined to be a match if the Brief descriptor distance of the best match is sufficiently different from that of the second best match (a.k.a. the ratio test [Lowe 1999]).

This process differs slightly from the process used for the frame matching described in Section 4.1, as we use tracking across multiple frames (in Section 4.1 only pairs are matched) to avoid locking onto scene motion and to distinguish foreground motion from background static features. The tracks are analyzed using a RANSAC method to find the largest set of inlier tracks such that a single temporal sequence of homographies maps all background features to their positions in an initialization frame. If too many tracks are lost after a number of frames, the tracker re-initializes with a new set of feature points. This tracking process results in a sequence of frame-to-frame homographies. These are then smoothed using the single-path version of the approach by Liu et al. [2013]. The amount of smoothing in this algorithm is controlled by the maximum allowable cropping in the result, which we set to 80%. Our tracking process also performs rolling-shutter correction using the algorithm of Baker et al. [2010]. Unlike in previous work [Kopf et al. 2014], we use a single-frame rendering approach to render the result: we warp each selected frame with the smoothed transformation and correction to produce the resulting hyperlapse sequence.

5 Refinements and optimizations

In the previous section, we defined our algorithm and general framework for creating hyperlapse videos using optimal frame selection. In this section, we describe two refinements that we incorporate into our framework.

5.1 Equal motion hyperlapse video

Equations 4 and 5 are defined in terms of *temporal* velocity, i.e., they are a function of how many frames are skipped. However, an appealing aspect of some hyperlapse videos (such as those of Kopf et al. [2014]) is that they attempt to maintain consistent *camera* velocity. In addition, large lateral swings in the camera are very disorienting when sped up too much. We can avoid these as we have camera motion estimates when we construct the cost matrix.

We approximate "equal motion" hyperlapses by modifying our speed cost to have a temporally varying target speed-up rate, $\nu(i)$:

$$C_s(i,j,\nu(i)) = min(||(j-i)-\nu(i)||_2^2, \tau_s), \tag{9}$$

$\nu(i)$ is a function of the camera velocity in the input video. We estimate this velocity in pixel space by sampling the optical flow induced by the frame-to-frame transformations in the input sequence. Given the transformations $T(i,j)$ discussed in Section 4.1, the camera velocity for frame i is the average flow of four corners of the frame:

$$v(i) = \frac{1}{4} \sum_{p=1}^{4} \left|\left| T(i,i+1)(x_p,y_p)_i^T - (x_p,y_p)_i^T \right|\right|_2, \tag{10}$$

where $(x_p, y_p) \in \{(1,1),(1,h),(w,1),(w,h)\}$ where w and h are the frame width and height.

The time varying target rate is given by:

$$\nu(i) = \alpha\nu \left(\frac{\frac{1}{T}\sum_{i=1}^{T} v(i)}{v(i)} \right) + (1-\alpha)\nu. \tag{11}$$

This function simply computes what speed-up rate is needed at frame i to result in a camera velocity that is ν times the average camera velocity. α allows one to balance between "equal motion" and "equal time" results. We set $\alpha = .8$ for our equal motion results.

5.2 Speed optimization

The most computationally expensive part of our process is computing the transformations $T(i,j)$, especially when the cost matrix window w is large, as one needs to compute w transformations per

[1] For convenience, we use the same notation for the cost function and matrix C_m, as they are essentially equivalent given that the cost function is discrete.

[2] In DTW parlance, C_m is the 'distance matrix", C_s is a "gap penalty", and C_a is a second-order penalty, and we are solving for a self-alignment, where the trivial alignment is not allowed and gaps of certain sizes are desired, rather than to be avoided.

Figure 4: *Our smartphone app and the three stages of our algorithm mapped to the user experience. Stage 1 occurs online during capture, Stage 2 is a short processing stage following capture, and Stage 3 occurs online during interactive viewing, where a user can use an on-screen slider to immediately change the speed-up for the displayed hyperlapse.*

input frame. Our optimization is to approximate the overlap cost $C_o(i,j)$ by chaining the overlap estimate between frames:

$$C_o(i,j) = C_o(i,i+1) + C_o(i+1,i+2) + ... + C_o(j-1,j). \quad (12)$$

This is similar to the approach of chaining transformations as in video stabilization algorithms [Grundmann et al. 2011]; however, we chain the cost directly instead of the underlying transformation used to compute the cost. This type of chaining works well when chaining a modest number of measurements; however, it is possible for the approximation to drift over long chains.

6 Desktop and mobile apps

We have implemented our algorithms in two applications: a smartphone app and a desktop app. While the two apps share the same code, due to differences in usage scenario and computational power, there are some subtle implementation differences.

6.1 Desktop app

In the desktop app, we assume the video has already been captured and the user provides a desired speed for the hyperlapses to create. The video is read off disk and all three stages are run in an online fashion as if the video stream was coming from a live camera stream. The hyperlapses are generated and saved to disk. We set the cost matrix window parameter w equal to two times the maximum target speed.

As a performance optimization we use a combination of chained and directly computed transformations $T(i,j)$. To decide whether to chain or directly compute the transformation, we use a simple heuristic to estimate the drift. We first compute the overlap cost $C_o(i,j)$ using the chained approximation. If $C_o(i,j) <= 0.05d$, where d is the image diagonal, we use the chain-computed cost. If $C_o(i,j) > 0.05d$, we compute $T(i,j)$ directly and recompute $C_o(i,j)$. We have found this heuristic to work well, as $C_o(i,j)$ is small only if $T(i,j)$ is closer to identity. In this case there is likely to be little accumulated drift since drift will manifest itself as a transformation that is increasingly far from the identity. There are other approaches that could be used instead, as we discuss in Section 8.

6.2 Mobile app

Our smartphone app, shown in Figure 4, typically operates on live captures, but also allows for importing of existing videos. The pipeline is the same as the desktop app; however, we currently only allow a discrete set of speed-ups of 1, 2, 4, 8, 16, and 32, and we set the cost matrix window $w = 32$. Stage 1 is run during capture, stage 2 is run immediately after capture, and stage 3 is run live as the user previews the hyperlapse and can use a slider to interactively change the speed-up rate.

Due to the reduced computational power of the smartphone, we always use chained transformations in the mobile app. Although this can lead to some drift, the errors due to this decision are usually un-noticeable. See Section 8 for more discussion on this.

7 Results

We have run our algorithm on a large, diverse set of videos from many activities and cameras: GoPros, several smartphones, and drone cameras – all videos are HD resolution (720p and higher). Some videos we acquired ourselves intending to make a hyperlapse, while others were general videos, where there was no intent to create a hyperlapse.

Figure 5 visually tabulates a selection from our test set and the color coding indicates which videos are compared with which previous methods. Most of our results are generated using our desktop app, as this allows the most flexibility for comparison. Five results in the "purple" section of Figure 5 are from our mobile app. It is not possible to create comparisons for these, as our mobile app does not save the original video due to storage concerns. Our results are presented as individual videos and several have side-by-side comparisons in our main video, all results are available at: `http://research.microsoft.com/en-us/um/redmond/projects/hyperlapserealtime/`. The reader is strongly encourage to view our these videos to fully appreciate our results; however, we also summarize a selection of results here.

For the videos indicated in "green" in Figure 5, we compared to the Instagram app in two ways. For two of the videos (Short Walk 1 and Selfie 2) we captured with the app, saved a hyperlapse, saved the raw input video using the advanced settings, and then ran our desktop app on the raw input video using the same target speed. Unfortunately, the save input video setting was recently removed in the Instagram app, so for two more captures (Selfie 1 and Dog Walk 2), we constructed a rig that allowed us to capture with two phones simultaneously, one using the Instagram app and one capturing regular HD video. We then ran our desktop app on the video from the second phone. Figure 6 shows a few consecutive frames from Instagram Hyperlapse and our approach.

To illustrate the differences in the camera motion, we show the mean and standard deviation of three consecutive frames. The sharper, less "ghosted" mean and lower standard deviation shows that our selected frames align better than those from Instagram. Where there is parallax, the images also illustrate how our results have more consistent forward motion with less turning. These dif-

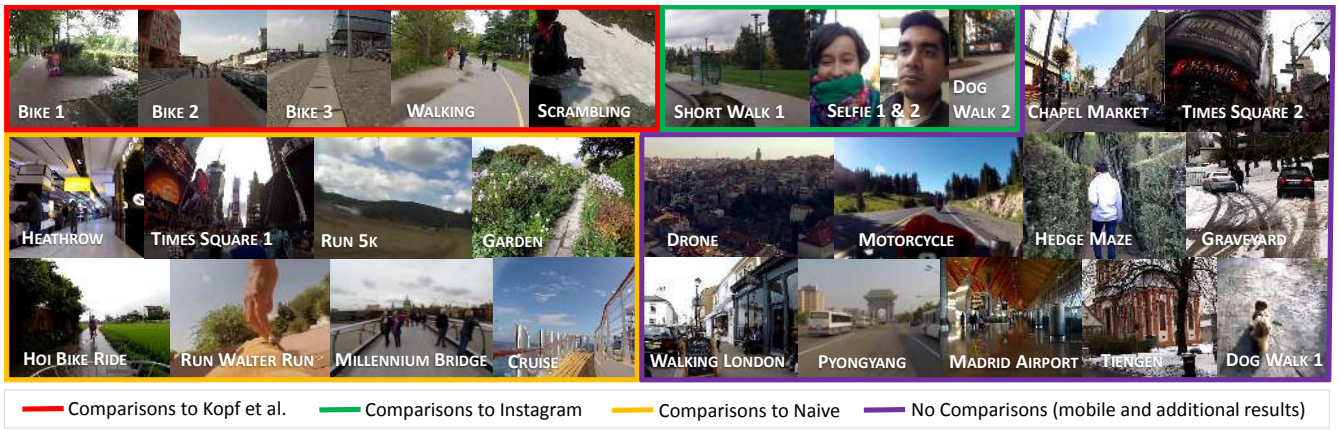

Figure 5: *Thumbnails for videos in our test set. The videos span a diverse set of activities and wide range of cameras, from GoPros and different models of smartphones to videos where the camera is unknown. We compare our results to previous work wherever possible, as indicated in the diagram.*

ferences are due to selecting a set of frames with more overlap that can be aligned more accurately and our visual tracking compensating for more than just the camera rotation that is measured by the gyros used by the Instagram app. The latter is most obvious in the "Selfie-lapses". In our main result video we shows a few side-by-side video comparisons for these results.

The five clips indicated in "red" in Figure 5 were provided to us by Kopf et al. [2014]. We ran their input video clips through our desktop app using a 10x target speed up. For Bike 1, 2, and 3 and Walking our results are similar, although the motion is less consistent. However, our computation time is two orders of magnitude faster. Just as in the Kopf et al. results our algorithm is able to skip many undesired events, such as quick turns.

The scrambling video is not as successful and illustrates some limitations, which are discussed in the following section. We would also expect the Kopf et al. approach to fail for some cases that work well in our approach, such as when there is not enough parallax to perform 3D reconstruction (e.g., if the camera is mostly rotating), and our approach has many fewer artifacts when there is scene motion, as 3D reconstruction requires the scene to be static over many

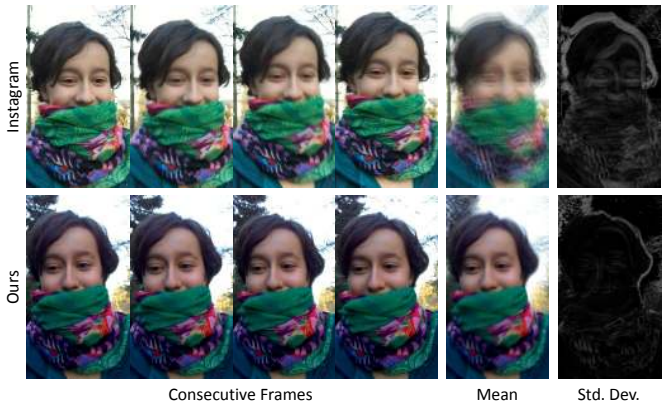

Figure 6: *Selfie-lapse: comparing Instagram Hyperlapse and our approach. Once again our approach leads to sharper, less "ghosted" mean and lower standard deviations since the Instagram approach is blind to the large foreground and thus cannot stabilize it well. The differences are more obvious in the associated videos.*

frames. Our main result video shows side-by-side comparisons for these results.

The "yellow" clips in Figure 5 are other videos from GoPro, drone, dash, cell phone, and unknown cameras. For these we include comparisons to naive hyperlapse (i.e., timelapse followed by stabilization), where the only difference between the naive results and ours is our frame selection algorithm. The naive hyperlapse results are similar to the Instagram approach except that visual features are used instead of the gyros. We use naive hyperlapse as a comparison as there is no way to run the Instagram app on existing videos and Kopf et al. have not released their code. Our main result video shows side-by-side comparisons for a few results, and we include a few additional results without comparisons, as indicated in "purple".

We also show side-by-side comparisons of our equal time vs. equal motion approach for a few videos. In these comparisons the slower camera motions are sped up while the quick pans are slowed down to create a more consistent camera velocity.

While our algorithm can skip large distracting motions such as head turns, this can occasionally lead to an undesirable temporal jump. In our online results, we also show some initial results for easing large temporal transitions by inserting a cross-faded frame whenever the temporal jump is larger than twice the desired target speed up.

Table 1 summarizes running times and video properties for a selection of videos and Figure 7 shows histograms of performance for all test videos. The average running rates for our algorithm are Stage 1: 50.7 FPS, Stage 2+3: 199.8 FPS, and Total: 38.16 FPS. These timings were measured running on a single core of a 2.67 GHz Intel Xeon X5650 PC from 2011 running Windows 8.1. The algorithm has not been optimized other than what is discussed in Section 5.2 and there is no code parallelization. All stages are performed online during capture, loading, or viewing. As Stage 2 is faster than real-time, it can be applied live while watching a hyperlapse, in other words, a hyperlapse is ready for consumption after Stage 1. We have included the dynamic programming time in the Stage 2+3 plot, as it is quite insignificant, with an average performance of 4638.5 FPS, or a most a few seconds for long inputs.

Our mobile app runs at 30 FPS with 1080p captures on a mid-level Windows Phone. To evaluate the impact of the optimization in Section 5.2, we ran "Short Walk 1" with no optimization, the hybrid

Video Name	Length	Resolution	Target Speed-up	Actual Speed-up	Stage 1 FPS	Stage 2+3 FPS	DP FPS	Total FPS	Kopf et al. FPS
SHORT WALK 1	0:59	1920x1080	12x	11.30x	32.29	79.47	18242.78	22.90	n/a
BIKE 3	16:36	1280x960	10x	12.6x	8.76	101.79	2912.43	8.07	0.0016
SELFIE 2	0:39	720x1280	10x	10.17x	67.86	169.39	25500.00	47.88	n/a
TIMES SQUARE 2	17:26	1920x1080	10x	9.81x	53.68	59.57	21849.38	28.18	n/a
CHAPEL MARKET	6:17	1280x960	10x	9.88x	59.73	98.21	16957.79	37.10	n/a
HOI BIKE RIDE	17:26	1920x1080	16x	16.16x	57.83	60.92	11078.41	29.62	n/a
RUN 5K	1:05:16	1280x960	20x	19.1x	10.09	84.59	9383.01	9.014	n/a
Mean (entire test set)	n/a	n/a	n/a	n/a	50.7	199.8	4638.5	38.16	n/a

Table 1: *Detailed running times and video properties for a subset of our test set. The mean statistics reported are across the whole test set.*

approach, and the fully chained approach. The Stage 1 performance for these is 17, 38, and 89 FPS, respectively.

8 Discussion and Future Work

We have presented a method for creating hyperlapse videos that can handle significant high-frequency camera motion and also runs in real-time on HD video. It does not require any special sensor data and can be run on videos captured with any camera. We optimally select frames from the input video that best match a desired target speed-up while also optimizing for the smoothest possible camera motion. We evaluated our approach using many input videos and compared these results to those from existing methods. Our results are smoother than the existing Instagram approach and much faster than Poleg et al. and the 3D Kopf et al. approach, providing a good balance of robustness and flexibility with fast running times.

One of the interesting outcomes of our approach is its ability to automatically discover and avoid periodic motions that arise in many first person video captures. While our algorithm stays close to the average target speed, it will locally change its speed to avoid bad samplings of periodic motions.

The primary limitations of our method are when there is significant parallax in a scene, when there is not a lot of visual overlap between nearby frames, or if the scene is mostly non-rigid. In these cases, the gyro approach of Instagram and the 3D reconstruction approach of Kopf et al. can help. While we have used these methods as our primary points of comparison, it is important to note that our method is quite complementary to those approaches, and an interesting area for future work is to combine these approaches. Our visual tracking and optimal frame selection are quite helpful independently, e.g., gyros and visual tracking can be fused for robustness and to distinguish between camera rotation and translation [Joshi et al. 2010]. Similarly, our optimal frame selection algorithm could be used with any tracking method: gyro, 2D, or 3D.

Overall, our approach is quite modular, so just as the first two stages: visual matching and DP could be used with other hyperlapse methods, there are number of 3D and 2.5D stabilization methods [Liu et al. 2009; Liu et al. 2011; Liu et al. 2013] that could easily be used as stage 3 of our approach, which could further refine our results and handle misalignments due to parallax.

Another interesting direction for future work is to integrate additional semantically derived costs into our cost matrix approach. For example, one could drive the selection of frames by visual or audio saliency, measures of image quality, such as blur, or detectors for faces or interesting objects.

Lastly, while our approach is already quite fast, there a numerous opportunities for optimization. Parallelization is the most obvious. The most time consuming step of computing frame-to-frame transformations is highly parallelizable and could lead to significant improvement in running time. Similarly, there are more optimal ways to estimate transformations using chaining, e.g., directly computing transformations at fixed spacings, such as 1x, 2x, or 4x, etc. and then using chaining for short segments in between. Given faster performance, one could then increase the band window in the cost matrix, to allow for matching across more frames, which can lead to more robust skipping of undesired events, such as quick turns.

Acknowledgements

We thank Rick Szeliski for his many suggestions and feedback. We thank Eric Stollnitz, Chris Sienkiewicz, Chris Buehler, Celso Gomes, and Josh Weisberg for additional work on the code and their design choices in the apps. We would also like to thank our many testers at Microsoft for their feedback and test videos, and lastly we thank the anonymous SIGGRAPH reviewers for helping us improve the paper.

References

AREV, I., PARK, H. S., SHEIKH, Y., HODGINS, J., AND SHAMIR, A. 2014. Automatic editing of footage from multiple social cameras. *ACM Trans. Graph. 33*, 4 (July), 81:1–81:11.

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

BENNETT, E. P., AND MCMILLAN, L. 2007. Computational time-lapse video. *ACM Trans. Graph. 26*, 3 (July).

CALONDER, M., LEPETIT, V., STRECHA, C., AND FUA, P. 2010. Brief: binary robust independent elementary features. In *Proceedings of the 11th European Conference on Computer vision: Part IV*, ECCV'10, 778–792.

CANON, L. G. 1993. *EF LENS WORK III, The Eyes of EOS*. Canon Inc.

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 (June), 381–395.

FORSSEN, P.-E., AND RINGABY, E. 2010. Rectifying rolling shutter video from hand-held devices. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 507–514.

GRUNDMANN, M., KWATRA, V., AND ESSA, I. 2011. Auto-directed video stabilization with robust l1 optimal camera paths. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, 225–232.

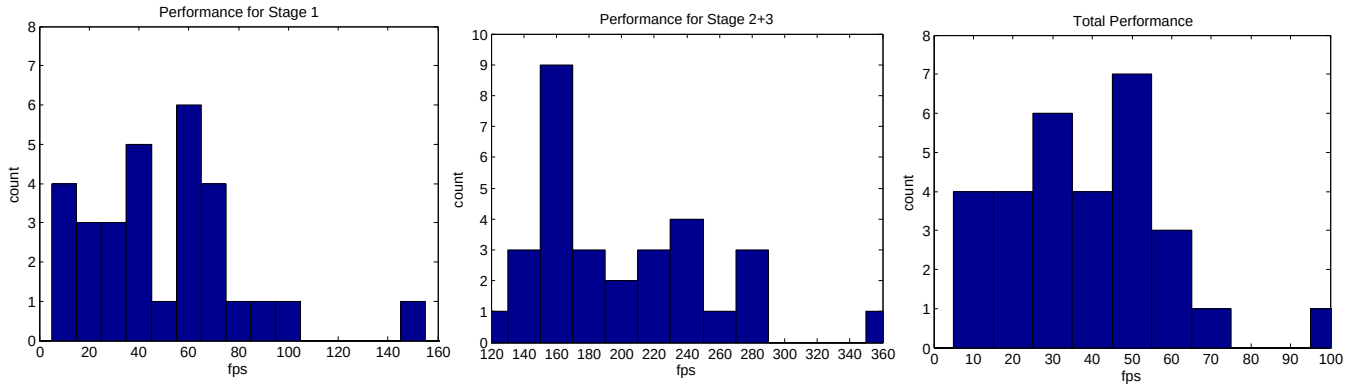

Figure 7: *Real-time performance. The running times for our test set of video in terms of frames-per-second (FPS) relative to the input video length. The average running times are Stage 1: 50.7 FPS, Stage 2+3: 199.8 FPS, and Total: 38.16 FPS. As Stage 2, is faster than real-time, it can be applied live while watching a hyperlapse, in other words, the hyperlapse is ready for consumption after Stage 1.*

JOSHI, N., KANG, S. B., ZITNICK, C. L., AND SZELISKI, R. 2010. Image deblurring using inertial measurement sensors. *ACM Trans. Graph. 29*, 4 (July), 30:1–30:9.

JOSHI, N., MEHTA, S., DRUCKER, S., STOLLNITZ, E., HOPPE, H., UYTTENDAELE, M., AND COHEN, M. 2012. Cliplets: Juxtaposing still and dynamic imagery. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, ACM, New York, NY, USA, UIST '12, 251–260.

KANEVA, B., SIVIC, J., TORRALBA, A., AVIDAN, S., AND FREEMAN, W. 2010. Infinite images: Creating and exploring a large photorealistic virtual space. *Proceedings of the IEEE 98*, 8 (Aug), 1391–1407.

KARPENKO, A., JACOBS, D., BAEK, J., AND LEVOY, M. 2011. Digital video stabilization and rolling shutter correction using gyroscopes. *Stanford University Computer Science Tech Report CSTR 2011-03*.

KARPENKO, A., 2014. The technology behind hyperlapse from instagram, Aug. http://instagram-engineering.tumblr.com/post/95922900787/hyperlapse.

KOPF, J., COHEN, M. F., AND SZELISKI, R. 2014. First-person hyper-lapse videos. *ACM Trans. Graph. 33*, 4 (July), 78:1–78:10.

LEVIEUX, P., TOMPKIN, J., AND KAUTZ, J. 2012. Interactive viewpoint video textures. In *Proceedings of the 9th European Conference on Visual Media Production*, ACM, New York, NY, USA, CVMP '12, 11–17.

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

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

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

LOWE, D. 1999. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, 1150–1157 vol.2.

MATSUSHITA, Y., OFEK, E., GE, W., TANG, X., AND SHUM, H.-Y. 2006. Full-frame video stabilization with motion inpainting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 28*, 7 (July), 1150–1163.

POLEG, Y., HALPERIN, T., ARORA, C., AND PELEG, S. 2014. Egosampling: Fast-forward and stereo for egocentric videos. *arXiv*, arXiv:1412.3596 (November).

PROVOST, D., 2014. How does the iOS 8 time-lapse feature work?, Sept. http://www.studioneat.com/blogs/main/15467765-how-does-the-ios-8-time-lapse-feature-work.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video textures. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 489–498.

WANG, O., SCHROERS, C., ZIMMER, H., GROSS, M., AND SORKINE-HORNUNG, A. 2014. Videosnapping: Interactive synchronization of multiple videos. *ACM Trans. Graph. 33*, 4 (July), 77:1–77:10.