Wireless Software Synchronization of Multiple Distributed Cameras

Sameer Ansari, Neal Wadhwa, Rahul Garg, and Jiawen Chen
Google Research, Mountain View, CA

We present a method for precisely time-synchronizing the capture of image sequences from a collection of smartphone cameras connected over WiFi. Our method is entirely software-based, has only modest hardware requirements, and achieves an accuracy of less than 250 µs on unmodified commodity hardware. It does not use image content and synchronizes cameras prior to capture. The algorithm operates in two stages. In the first stage, we designate one device as the leader and synchronize each client device's clock to it by estimating network delay. Once clocks are synchronized, the second stage initiates continuous image streaming, estimates the relative phase of image timestamps between each client and the leader, and shifts the streams into alignment. We quantitatively validate our results on a multi-camera rig imaging a high-precision LED array and qualitatively demonstrate significant improvements to multi-view stereo depth estimation and stitching of dynamic scenes. We plan to open-source an Android implementation of our system `libsoftwaresync`, potentially inspiring new types of collective capture applications.

***Index Terms*—computational photography**

I. Introduction

Many computer vision algorithms require multiple images from different viewpoints as input. These algorithms can fail spectacularly when applied to dynamic scenes if the images are not captured at the same time. Depth from stereo, view interpolation and view supervision losses are just a few examples of such algorithms. We provide a solution to temporally synchronize the cameras of multiple inexpensive smartphones, so that they can capture images simultaneously (Fig. 1).

Our solution is software-based, does not require any additional hardware, and scales up to multiple devices. Furthermore, the devices are synchronized prior to capture and can be located in any physical arrangement as long as they can communicate over a wireless network.

Traditionally, synchronizing cameras in advance of capture is solved by using specialized hardware such as analog video gen-lock or the IEEE 1394 isochronous interface. While effective and reliable, hardware solutions require cumbersome wires, limiting their portability, and more importantly, are unavailable on the vast majority of cameras. For non-specialized cameras, this limits usage to nearly static scenes where several frames of timing error are acceptable.

Another option is to capture image sequences and align the frames after capture. A clapperboard or similar "oracle" device that is visible to all cameras can assist in aligning the frames. However, accuracy is limited to half a frame duration. Post-processing methods to obtain sub-frame alignment exist [1]–[5], but can break down for low-texture or noisy scenes where finding correspondences is hard. Even if these algorithms yield perfect sub-frame alignment, it is necessary to temporally interpolate the frames to the aligned sub-frame time [3], which is another challenging problem. In addition, such methods may require capturing much more data than necessary to ensure that the same instant in time is captured by all cameras.

In contrast, our system is capable of achieving a synchronization accuracy of less than 250 µs before capture, allowing for the multi-view simultaneous capture of highly dynamic phenomena such as sports action shots, birds in flight, and splashing liquids (Fig. 2). To acquire synchronized images, users are only required to install our software, position the phones, and press the shutter button on one of the phones. Our system does not look at image content; instead, it uses only image timestamps and wireless messages for synchronization. This means the cameras can have non-overlapping fields of view and can be placed in any configuration. For example, a hand-held, inexpensive light field capture rig can be created by placing the cameras in a square array. Or the cameras can be placed in a linear array, so that a view interpolation algorithm can create a "bullet time" effect where the camera moves around an apparently frozen dynamic subject [6].

Fig. 1. Five smartphone cameras simultaneously capture a highly dynamic scene. The cameras are synchronized using our software-based solution, which requires no wires or additional hardware. The images are shown in Fig. 2.

Our system is especially useful for collecting data for deep learning algorithms that require synchronized images for training [7]–[9] and also need to be deployed on a smartphone. Were such data collected with hardware-synchronized non-smartphone cameras, biases introduced by differences in the sensor or lens could ruin the algorithm's performance. A portable rig containing smartphones synchronized using our system provides an inexpensive method to collect such datasets in the wild.

Our system operates in two stages. First, the clocks of all

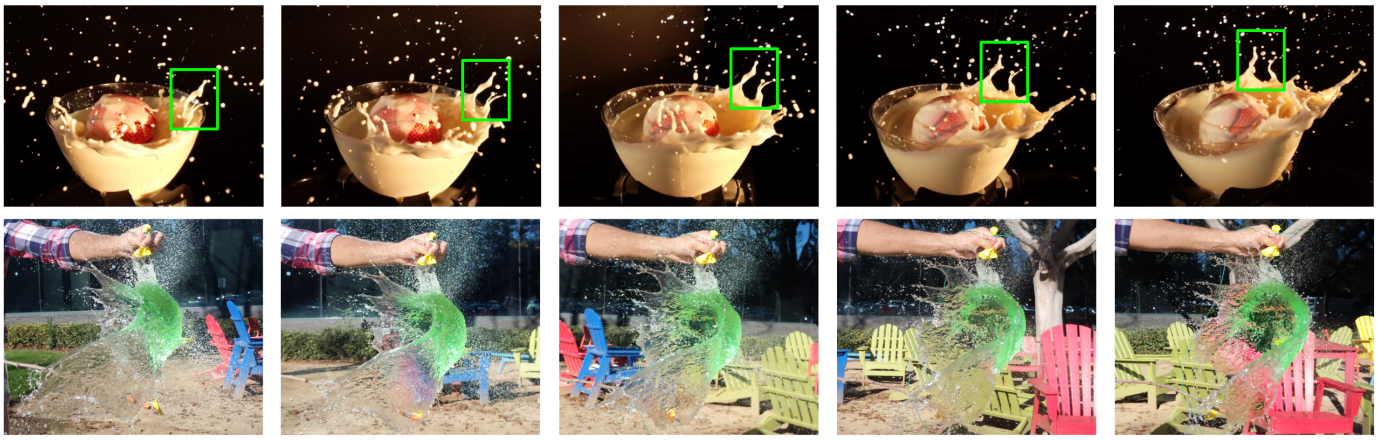


Fig. 2. Top: Synchronized images of a ball splashing in milk captured by the cameras in Fig. 1. The same splash of milk is highlighted in green in the photos. Bottom: Synchronized outdoor images of a bursting water balloon a wider baseline setup. Note the consistent shape of the water concavity.

the phones are synchronized to the clock of a leader device using a variant of the Network Time Protocol (NTP). This synchronization is only enough to get an accuracy of half a frame duration (e.g., 16 ms). In the second stage, the cameras are instructed to capture a continuous stream of images. We then shift the phase of all client streams to that of the leader and achieve an accuracy better than 250 µs.

In summary, we contribute the following:

- Two algorithms for efficiently phase aligning camera image streams to arbitrary precision.
- A system combining phase alignment with a variant of NTP that achieves an accuracy better than 250 µs on unmodified consumer smartphones over WiFi.
- A theoretical analysis and empirical confirmation of the accuracy and limits of our system.

We plan to release `libsoftwaresync`, an Android implementation of our system as open source.

II. Related Work

Precision time synchronization in a distributed system is a classically hard problem. The fact that each device is physically independent with its own hardware clock makes synchronization necessary even if networking were reliable with low communications latency. When applied to the camera synchronization problem, misalignment is primarily due to two factors. First, each device has an independent notion of the current time, which may drift and is at best only loosely aligned to an external source (e.g., GPS or cell network). Second, consumer cameras typically do not feature a mechanism by which image capture can be triggered to occur at a specific time. Instead, there is a large firmware and software stack between the camera hardware and the application that requests an image, which introduces additional variable latency. Without physical wires to synchronize all clocks and cameras in the network, it is challenging to trigger all cameras at the same time.

A number of systems focus on aligning video sequences and correcting for rolling shutter after they have been captured. Some systems use active illumination to tag frames from which an offset may be inferred [1]. Others rely on the scene content itself, by detecting abrupt lighting changes [2], tracking continuous periodic motion [3], or requiring motion be simple enough that the spatiotemporal transformation can be represented by a low-dimensional model [4], [5]. As the source imagery is fundamentally unsynchronized, these post-capture techniques all ultimately rely on optical flow and interpolation to bring images into sub-frame alignment. Our goal is to synchronize cameras *before* capture, ensuring they expose the same instant in time and sidestepping the frame interpolation problem.

Gu *et al*. [10] describe a simple scheme for projector-camera synchronization by forcing the camera to wait until the projector has displayed an image before triggering capture. This technique is straightforward to implement but cannot capture moving subjects, can introduce unacceptably long shutter lag, and is impractical for video sequences. Litos *et al*. [11] describe a system that uses the Network Time Protocol (NTP) [12] to align device clocks but neglects the variable latency between when software sends a shutter command and when the camera ultimately captures a frame. Petković *et al*. [13] describe a clever system for synchronizing a projector and a camera by using the camera itself to measure the signal propagation delay. But this requires a projector and also does not account for the variable latency in triggering a camera.

The system by Ahrenberg *et al*. [14] is closest in spirit to ours. Like our system, they rely on NTP to synchronize device clocks before sending a single recording command. Their system relies on the IEEE 1394 hardware trigger to keep pace, but like the other systems above, assumes that the delay between the software trigger command and image capture to be the same among all cameras. Our system does not require a hardware trigger and explicitly models the variable latency between the software trigger command and image exposure.

III. Overview

We give a high level overview of our system. We first list our assumptions of the underlying system, which justify our algorithm and where it may be deployed.

A. Setting

Our system consists of a collection of N *devices*, each of which is equipped with a camera and a network card. In practice, these are smartphones with an integrated camera and WiFi. At startup, the devices connect over WiFi, synchronize their clocks, and puts their cameras into continuous streaming mode (Sec. IV). After a second stage where camera streams are phase aligned (Sec. V), initialization is complete. At any time after, when a capture request containing a timestamp arrives, all devices simply save the appropriate images from a ring buffer to disk. Note that this requested timestamp can be in the past, enabling zero shutter lag captures. Our goal is to do the above with reasonable latency and as such, rely on a few assumptions of the underlying hardware:

1) **Modest variance in network latency**. The first stage of our algorithm is to bring the devices' *local clocks* into alignment. Theoretically, our system can tolerate arbitrary latency variance, although it may take an unacceptable amount of time to converge. To support arbitrary devices without hardware timestamping of network packets, we include in our latency model variable delays induced by the operating system's networking stack.
2) **Hardware camera timestamps**. We require that the camera hardware tag images with timestamps *in the domain* of the device's local clock. A key component of our algorithm is accurately modeling the time between requesting an image and when it is captured. If images were not timestamped in the same clock domain, this modeling would not be possible. iPhones and numerous Android devices support this feature.
3) **Hardware image streaming** We require that the camera hardware or firmware be able to latch capture settings such that it can stream frames indefinitely with low timing variance between frames. For example, if we request images at ISO 100, exposure time of $10\,\mathrm{ms}$, and a frame duration (time between frames) of $33\,\mathrm{ms}$, we should expect a stream of images with timestamps close to $33\,\mathrm{ms}$ apart. Most commercially available cameras and smartphones support this feature.

IV. Synchronizing Device Clocks

Like previous work [11], [14], we use a variant of the Network Time Protocol (NTP) [12] to synchronize device clocks, for which we give an simplified overview. This step is unnecessary if all devices had support for the Precision Time Protocol (PTP) [15], which provides sub-microsecond hardware synchronization. However, PTP hardware is typically not available on consumer smartphones. In our system, the user designates one device as the leader and creates a WiFi hotspot. The remaining $N-1$ clients connect to the leader and estimate their clock offsets. Typically, device clock synchronization is only performed once at the beginning of a capture session. In our experiments, clocks remained stable over the course of one hour. If more accuracy is needed, the system can re-synchronize between captures.

A single message in the synchronization routine consists of the following handshake (Fig. 3) between the leader and one of the client devices, each client is handled independently:

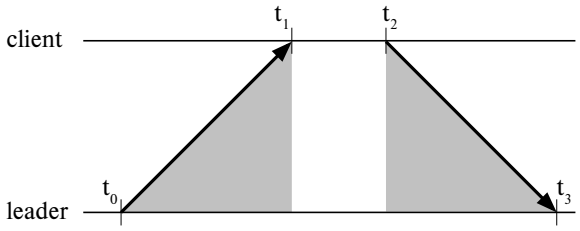


Fig. 3. NTP handshake. With one message in each direction, the leader can estimate the clock offset θ.

1) At time t_0 in the leader's clock domain the leader sends the message "t_0".
2) At time t_1 in the client's clock domain, it receives the message "t_0".
3) At time $t_2 > t_1$ in the client's clock domain, it sends the message "(t_0, t_1, t_2)".
4) At time $t_3 > t_0$ in the leader's clock domain, it receives the message "(t_0, t_1, t_2)".

We then estimate clock offset θ and round-trip delay ϕ:

$$\theta = \frac{(t_1 - t_0) + (t_2 - t_3)}{2} \qquad \phi = t_3 - t_2 + t_1 - t_0 \tag{1}$$

Note that the computation of θ assumes that communications latency is symmetric, typical in networking. If not, there will be a systematic bias which places a limit to our overall synchronization accuracy. Since individual clock readings and network latency measurements may be noisy (due to e.g., buffering), we collect multiple samples and filter them. To synchronize N devices, each of the $N-1$ client devices exchanges K messages with the leader in a round robin fashion. Following the best practices suggested by NTP, we determine K using the mean and min filters.

A. Mean Filter

The mean filter assumes that communications latency is normally distributed. This lets us compute the sample mean and variance after rejecting outliers where the round-trip delay is greater than some threshold (e.g., due to buffering or interference). Since variance decreases as $1/K$, we can determine the number of messages K required to reach a desired accuracy. However, in our experiments, we found that the mean filter converges to a systematic bias of $\approx 0.4\,\mathrm{ms}$ when compared to an external reference clock. To ameliorate this bias, we use the min filter.

B. Min Filter

The min filter assumes that the sample with the *minimum* latency is the most reliable. This heuristic, also used by the NTP standard, is based on the hypothesis that the message with the shortest delay experienced the least amount of processing and therefore is the most symmetric. We confirm this hypothesis quantitatively and compare the mean and min filters in Sec. VI. In our implementation, we can either exchange a fixed number of messages and take the minimum, or iterate until we observe a sample with latency below some threshold.

To guarantee a predictable startup time of less than 10 seconds on our rig of 5 smartphones with a mean latency of $4\,\mathrm{ms}$, we used a fixed $K = 300$ samples.

V. Phase-based image stream alignment

Once all device clocks are synchronized, we can assume that all images are timestamped in the leader's clock domain (by adding the appropriate offset). However, to actually capture images simultaneously on all devices, we need to account for the variable latency between when a device requests a capture and when the sensor is actually exposed. This latency can be highly variable and is hard to measure accurately. Instead, we exploit hardware image streaming: the ability to indefinitely capture a sequence of images at regular intervals. We seek to shift the phases of each stream, so that all cameras are exposing at the same time. Then, all that remains is to select from the image ring buffer which frames to save to disk.

Suppose that the leader camera and a client camera start streaming at times u_{goal} and u respectively and repeatedly capture images with period T. We seek to reduce the difference in their phases $\delta = u - u_{goal} \pmod T$ to a small value ϵ by shifting the phase of the client camera (Fig. 4a). As opposed to trying to align u and u_{goal} directly, phase shifting a frame stream can be done efficiently and accurately.

We describe two methods to phase shift a frame stream, *reset sampling*, a slower to converge method that is applicable to many devices, and *frame injection*, a faster to converge method that may not work on every device.

A. Reset Sampling

In reset sampling, we restart the camera until its phase falls within a tolerance ϵ of the goal phase (the gray region in Fig. 4b). There is significant variance in how long it takes a camera to reset ($600-800\,\mathrm{ms}$ in our experiments). Furthermore, this distribution can vary from camera to camera and in the worst case can result in phases always falling outside the tolerance. Therefore, we sleep for a random amount of time $U(0, N)$ after stopping the camera and before restarting it to make the overall distribution of resets more uniform. If the latency of starting the camera is described by the random variable S, then, the total phase offset can be modeled as:

$$U(0,N) + S \pmod T \approx U(0,N) \pmod T \approx U(0,T), \quad (2)$$

where the approximation holds as long as N is large relative to T and the range of S. We found a value of $N = 1$ second to be reasonable. Since we are able to shift the phase by a random amount in each iteration, we effectively sample phase offsets from a uniform distribution and reject those that are outside the gray region in Fig. 4b. There is an ϵ/T chance a sample will be accepted on any one iteration. This means to have a 95% chance of converging within a tolerance of ϵ, we need to reset the camera R times.

$$R = \frac{\log 0.05}{\log\left(1 - \epsilon/T\right)}. \quad (3)$$

For $\epsilon = 1\,\mathrm{ms}$ and $T = 33\,\mathrm{ms}$, $R = 98$ iterations are required.

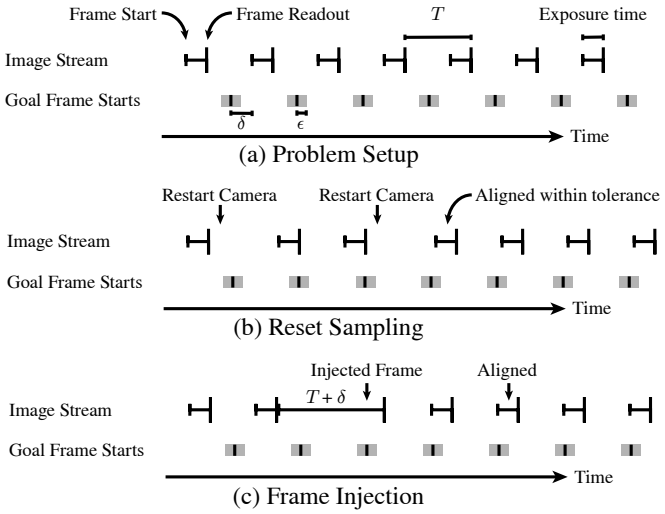


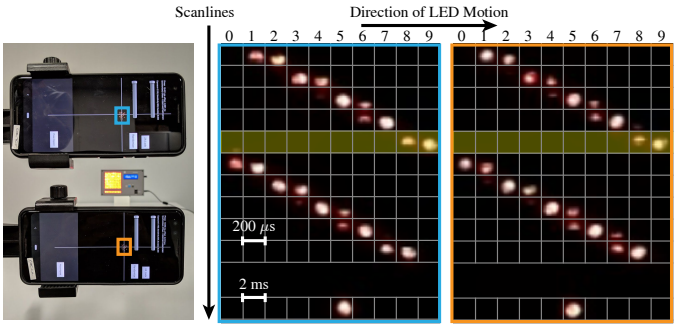
Fig. 4. Different strategies to phase align a hardware image stream to a specified goal. (a) Frames are read out every T ms and the start of each frame is offset from the goal by a phase of δ ms. We seek to align frame start times to within ϵ ms of the goal phase. (b) Reset sampling restarts the camera until the frame start times falls within the gray tolerance. (c) For certain cameras, we can achieve much quicker alignment by injecting a frame with exposure $T + \delta$ to directly phase shift the image stream.

B. Frame Injection

Some camera APIs (e.g., Android's Camera2 API [16]) allow a high priority capture request to be injected into a normal priority continuous image stream. Typically, the continuous stream is used for the viewfinder, while the high priority request is used to capture a still image with different exposure or gain settings. By injecting a frame with exposure longer than the duration T, we can shift the phase of a stream in *one* iteration. In an ideal camera, if the phase offset between two image streams is δ, we can align the streams by injecting a frame with exposure time $T + \delta$. Real cameras can behave differently than this ideal, but as long as the camera is deterministic, we can fit a function mapping a desired phase offset δ to an exposure that will shift the camera's stream by $kT + \delta$ for some integer k. For example, on one specific phone model, we empirically found that exposures of length $T + \delta/2$ would shift the phase by $2T + \delta$. Some cameras require exposure times to be an integer multiple of a scanline's readout time. While frame injection is deterministic, we model this quantization as Gaussian noise with standard deviation σ and measured σ to be on the order of tens of microseconds. Under this assumption, this method will converge in $4\sigma^2/\epsilon^2$ iterations with 95% probability. If $\epsilon = 1\,\mathrm{ms}$, frame injection converges in 1 iteration.

VI. Results

We quantitatively verify the accuracy of our method and show that it outperforms several common naive approaches by three orders of magnitude. We also demonstrate how our method qualitatively improves the quality of multi-view stereo and panorama stitching. Finally, we show simultaneous multi-view captures of several highly dynamic scenes.



(a) Experimental setup (b) Top phone view (c) Bottom phone view

Fig. 5. Ground truth measurement of synchronization accuracy. (a) Two smartphones capture images of a 10×10 LED panel with an exposure of 100 µs. LEDs sweep horizontally across the grid at a rate of one column per 200 µs. The LEDs in the separate far bottom row sweep at a rate of one column per 2 ms. The central scanlines of both phones are set to capture the fifth row (highlighted in yellow). The skewed pattern is due to rolling shutter, where it takes 3.3 ms to capture the fraction of the scene shown. (b) Top phone view of the LED panel. (c) Bottom phone view of the LED panel.

A. Measuring Overall Synchronization Error

To quantitatively validate our method, we use an LED panel as an external high-precision frequency reference (Fig. 5a). The panel contains a fast-changing 10×10 array and a slow-changing 10×1 array. In the fast-changing array, a column is lit for a fixed time period τ, then the next column is lit for τ, and so forth until the last column is reached and the pattern repeats. In the slow-changing array, each LED stays on for 10τ. The two arrays allow us to measure synchronization error with a resolution on the order of τ. Since the entire panel repeats with a period of 100τ, it is possible though unlikely for the measured synchronization to be off by exactly $100k\tau$ where k is an integer. We can eliminate this possibility by using multiple values of τ.

In our first experiment, we placed two Google Pixel 3 smartphones on a tripod and captured 100 µs exposures of the LED panel with τ set to 200 µs (Fig. 5a). The frame duration T was 33 ms. To avoid confounding synchronization error with misalignment caused by mismatched rolling shutters, we oriented the phones so that their center scanlines both capture the fifth row of the fast moving array of the LED panel.

We measured the synchronization error of our min filter and frame injection methods by conducting 239 trials. In each trial, we first reset the synchronization process to undo clock sync and added a random length long exposure frame to undo the phase alignment. We then asked our system to synchronize to a tolerance of $\epsilon = 20$ µs. Example captures from the two phones are shown in Fig. 5(b-c). The skewed pattern of LEDs is due to rolling shutter.

In all pairs of images, there is no more than 1 LED difference between the pairs, showing that the total synchronization error is less than 200 µs. We also ran experiments with τ set to 10 ms to verify that our results were not off by an integer multiple of 20 ms. To get more fine-grained error measurements, we exploit the fact that short exposures and rolling shutter means some LEDs will only be partially captured. This allows us to estimate the error between the two phones with temporal resolution better than τ. We use the mean pixel intensity in each grid cell in the fifth row of Fig. 5(b-c) to estimate the sub-grid position of the LED lights.

For two devices, we found the maximum error between them to be 121 µs (Fig. 6c and Table. I). With more than two devices, any pair can have this maximum error but in opposite directions implying a worst case error of 242 µs.

We also compared to three naive synchronization methods: wired, Bluetooth, and WiFi. In the *naive wired* method, we used a selfie stick connected via an audio splitter to two Pixel 3 smartphones, such that a single button press would trigger both phones. We used the same setup as in Fig. 5(a), except we set τ to 20 ms. In our *naive Bluetooth* method, we connected wireless Bluetooth photo capture triggers to each device and used a mechanical button to press both triggers at the same time. In our *naive WiFi* method, we connected one phone to the other's WiFi hotspot. On shutter press, the phone with the hotspot then instructed itself and the client to take photos immediately. We conducted 20 trials for each of these methods. In the wired and Bluetooth methods, 2 and 1 shot failed to capture respectively due to issues with the extra hardware. These naive methods have errors over a 1000 times larger than our method and occasionally fail to work (Table I). In addition, the wired and Bluetooth solutions require extra hardware making them cumbersome to use and difficult to scale to many phones.

TABLE I
SYNCHRONIZATION ACCURACY OF NAIVE VS OUR METHOD

Method	Max error	Mean Abs Error	Stdev
Naive Wired	180 ms	103 ms	50 ms
Naive Bluetooth	200 ms	69 ms	65 ms
Naive WiFi	677 ms	123 ms	84 ms
Our Method	**0.121 ms**	**0.032 ms**	**0.025 ms**

B. Measuring Clock and Phase Accuracy

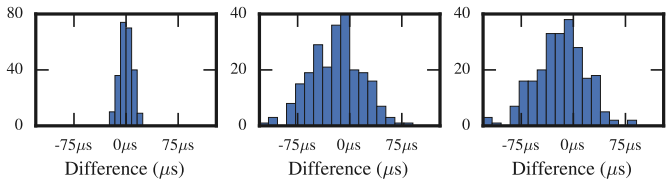
The overall accuracy of our system, ϵ_{total}, depends on many factors. The two main ones are how well estimated the clock offsets are (ϵ_{clock}) and how well aligned the image stream phases are (ϵ_{phase}). There are also other sources which we do not account for, such as imperfect hardware timestamps and variances in the readout period T. Ignoring error from other sources, we model the total error as:

$$\epsilon_{total} = \epsilon_{clock} + \epsilon_{phase}. \tag{4}$$

In the previous section, we directly measured ϵ_{total} using an external reference. We can also directly measure ϵ_{phase} by comparing the an image's actual hardware timestamps to the requested ones. ϵ_{clock} can then be estimated as $\epsilon_{total} - \epsilon_{phase}$.

We first measure phase error over the same 239 trials as the previous section. We found that the phase error is always within our requested tolerance of 20 µs and that the distribution of differences has zero mean (Fig. 6a). The clock error is not zero mean and has a slight negative bias (Fig. 6b). We analyze this further in the next section.

Note that both the maximum and mean absolute error are comfortably less than 1 ms. Table II also shows that the dominant component of the error in our system is due to clock



(a) Phase Error (b) Clock Error (c) Total Error

Fig. 6. Histograms of the phase alignment, network clock synchronization, and total synchronization errors over 239 trials.

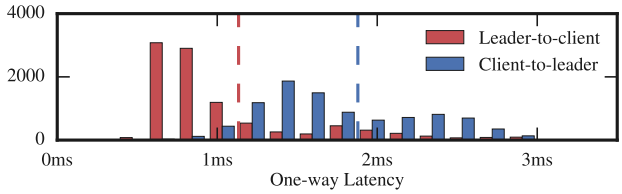


Fig. 7. Histogram of one-way network latency for 10,000 messages between leader and client. The dashed lines show the means of the distributions.

error, which, as noted earlier, can be reduced to less than 1 microsecond by using PTP hardware.

TABLE II
DISTRIBUTION OF ERRORS BY COMPONENT

	Max error	Mean Abs Error	Stdev
Clock	$\leq 121\,\mu s$	$\leq 32\,\mu s$	$\leq 25\,\mu s$
Phase	$21\,\mu s$	$7\,\mu s$	$5\,\mu s$
Total	$\mathbf{121\,\mu s}$	$\mathbf{32\,\mu s}$	$\mathbf{25\,\mu s}$

C. Analysis of Clock Filters

We analyze the behavior and effectiveness of using the mean and min clock filters. With the same setup, the two phones exchange 10,000 NTP messages, and capture one image pair of our LED panel. From this image pair, we can estimate the true clock offset, from which we can derive the one-way latency of each message. Like our other experiments, images are hardware timestamped and messages are software timestamped, both in each device's local clock domain. Each message takes roughly $3\,ms$ round-trip and the entire procedure takes 34.3 seconds. Fig. 7 shows the difference in one-way latency distributions over 10,000 messages after removing 64 outliers where the latency exceeds $10\,ms$.

Mean filter - NTP's clock offset accuracy relies on network symmetry. Fig. 7 and Table III shows that this is not the case in our setup and that computing the mean offset over many messages converges to a bias that is approximately half the absolute latency difference. The exact values will vary based on network configuration and hardware used. The supplement contains additional details on how we model the relationship between round-trip latency and bias.

TABLE III
NETWORK LATENCY SYMMETRY AND BIAS: MEAN VS. MIN

	Mean	Min
Leader to client	$1{,}133\,\mu s$	$517\,\mu s$
Client to leader	$1{,}878\,\mu s$	$479\,\mu s$
Abs. latency difference	$746\,\mu s$	$38\,\mu s$
Round-trip latency	$3{,}012\,\mu s$	$996\,\mu s$
NTP bias	$372\,\mu s$	$19\,\mu s$

Min filter - The Min filter approach uses the message with the smallest round-trip latency. In Table III, NTP bias is approximately half the difference in one-way latencies. The min filter, by choosing the sample with lowest latency, is subject to significantly less latency asymmetry and therefore achieves a much lower bias in its clock offset. We conducted 3 quantitative experiments with sample sizes $K = 300, 1000, 10000$ and found that on average, the shortest round-trip latency to be $1.87\,ms$, $1.02\,ms$, and $0.996\,ms$, respectively with low variance. In practice, we use 300 samples which takes under 1 second per client and still provides $< 1\,ms$ total synchronization accuracy.

Errors under $200\,\mu s$ are likely to be confounded by measurement noise such as misaligned scanlines between phones, rolling shutter skew, and our method of using an LED panel running at $200\,\mu s$. Errors in the phase are under $21\,\mu s$, which is minor compared to compared to the total error (Table II).

D. Phase Alignment Convergence Time

We measure the number of iterations and total wall time required for phase alignment using both the frame injection and reset sampling methods. Note that once a client has synchronized its clock with the leader, phase alignment can proceed independent of other clients and therefore does not scale with the number of clients.

For the frame injection method, we used the same dataset of 239 trials as in earlier experiments. Aligning to $\epsilon = 20\,\mu s$ took between 4-7 injected frames, with a mean of 5.5 frames and $\sigma = 0.97$ frames. On Pixel 3, each injected frame takes $300\,ms$ so phase alignment converges in less than 3 seconds.

To measure reset sampling, we ran 10 additional trials with a tolerance of $\epsilon = 1\,ms$. As expected, reset sampling took significantly longer than frame injection, taking on average 28.7 iterations before convergence with a standard deviation of 25.2 iterations. Because reset sampling requires restarting the camera and sleeping for a random amount ([0, 1] seconds), each iteration took on average 1.23 seconds ($\sigma = 0.27$ seconds). According to Eq. 3, it will take 98 iterations or 120 seconds to achieve a 95% probability of alignment. Although reset sampling is significantly more expensive than frame injection, it does not require any modeling of camera request latency and is deployable on a wider variety of devices.

E. Applications

While our system has many applications, we focus on two here – stereo reconstruction and image compositing of dynamic scenes. In addition, we use our method to capture multiple views of highly dynamic scenes (Fig. 10 and the supplement). Our system has also been used for training a machine learning algorithm shipping with a commercial smartphone [**?**].

Stereo reconstruction of dynamic scenes - Stereo techniques reconstruct 3D geometry from two or more photos

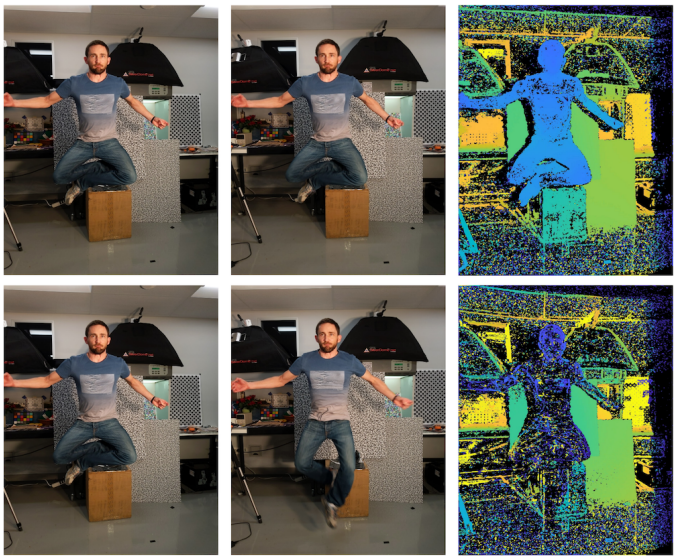


Fig. 8. Top row: A stereo pair of a jumping subject captured using two smartphone cameras synchronized using our system and the corresponding depth map computed using COLMAP [19]. Bottom row: A stereo pair that is 66 ms apart and the corresponding depth, representative of images one is likely to capture with a naive approach for synchronization.

of the same scene from different viewpoints [18]. Assuming known camera poses, these techniques typically first identifies a scene point's projections in multiple photos and then estimates its depth by triangulation. A fundamental assumption is that the scene point is stationary across photos. Therefore, we need synchronized capture to apply these techniques to dynamic scenes.

To qualitatively demonstrate the efficacy of our synchronization for depth estimation of dynamic scenes, we capture a pair of stereo images and compute depth using COLMAP [19], a state-of-art Structure-from-Motion [20] and Multi-View Stereo system [21] (Fig. 8). We compare it with naive synchronization, which we simulate by selecting frames that are 66 ms apart, a synchronization error that is comparable to the smallest average error among the different naive synchronization methods listed in Table I. Clearly, synchronization enables higher quality depth. Moreover, since our approach scales to multiple cameras, we can further increase the accuracy of depth, especially around the occluded regions, by adding more synchronized cameras (see the supplement).

Image compositing for dynamic scenes - Image composites stitch multiple images of the same scene from different viewpoints into a single image [22], [23] that may otherwise be difficult or impossible to capture using a single camera, e.g., an extremely wide angle image or a multi-perspective image [24], [25]. A typical approach consists of aligning the images, projecting them onto a suitable compositing surface, and blending them to minimize alignment errors and account for differences in exposure, color balance, etc. While a composite may be constructed from images captured by a single camera, a dynamic scene requires synchronized captures from multiple cameras to minimize stitching artifacts.

Fig. 9 demonstrates that our system can be used to avoid stitching artifacts due to motion. Our system can also synchronize exposure and white-balance across devices making it easier to blend between images in the composite.

Fig. 9. Composite of a dynamic scene stitched from two images using Microsoft Research's Image Composite Editor [26]. When the two cameras are synchronized using our system, the resulting composite does not have artifacts (Top). However, a synchronization error of 66 ms results in stitching artifacts (Bottom).

VII. Conclusion and Future Work

We presented an entirely software-based system for capturing time-synchronized image sequences on a collection of smartphones. Our method is wireless, runs on commodity hardware, and is able to achieve an accuracy of $< 250\,\mu s$, which we verified using a precision chronometer to be significantly better than naive methods that synchronize cameras prior to capture.

We empirically confirmed that the accuracy of our method largely depends on NTP clock bias, which can be minimized following the existing best practice of min filtering. We also analyzed and verified the convergence behavior of two phase alignment approaches: reset sampling, which is slow but widely applicable, and frame injection, which converges rapidly but has a minor hardware requirement.

As future work, we would like to scale up our system, i.e., to thousands of different devices distributed over a large geographical area maintaining synchronization over extended periods. Our system has only been tested on Google Pixel 1, 2, and 3 smartphones. The network is also restricted to 11 devices (a limit imposed by the Android OS) and all clients must currently communicate directly with a designated leader device. A peer-to-peer model can address both these limitations and adapt to both clock drift and motion. We will open-source `libsoftwaresync` in hopes of inspiring new possibilities in social, collective photography applications. We envision such a system being used to capture large-scale dynamic events such as concerts and sports matches.

Acknowledgment

We thank Sam Hasinoff, Jon Barron, Roman Lewkow and Ryan Geiss for their helpful comments and suggestions, as well as Nikhil Karnard, Tim Brooks, Orly Liba, and David Jacobs for their help with collecting qualitative results.

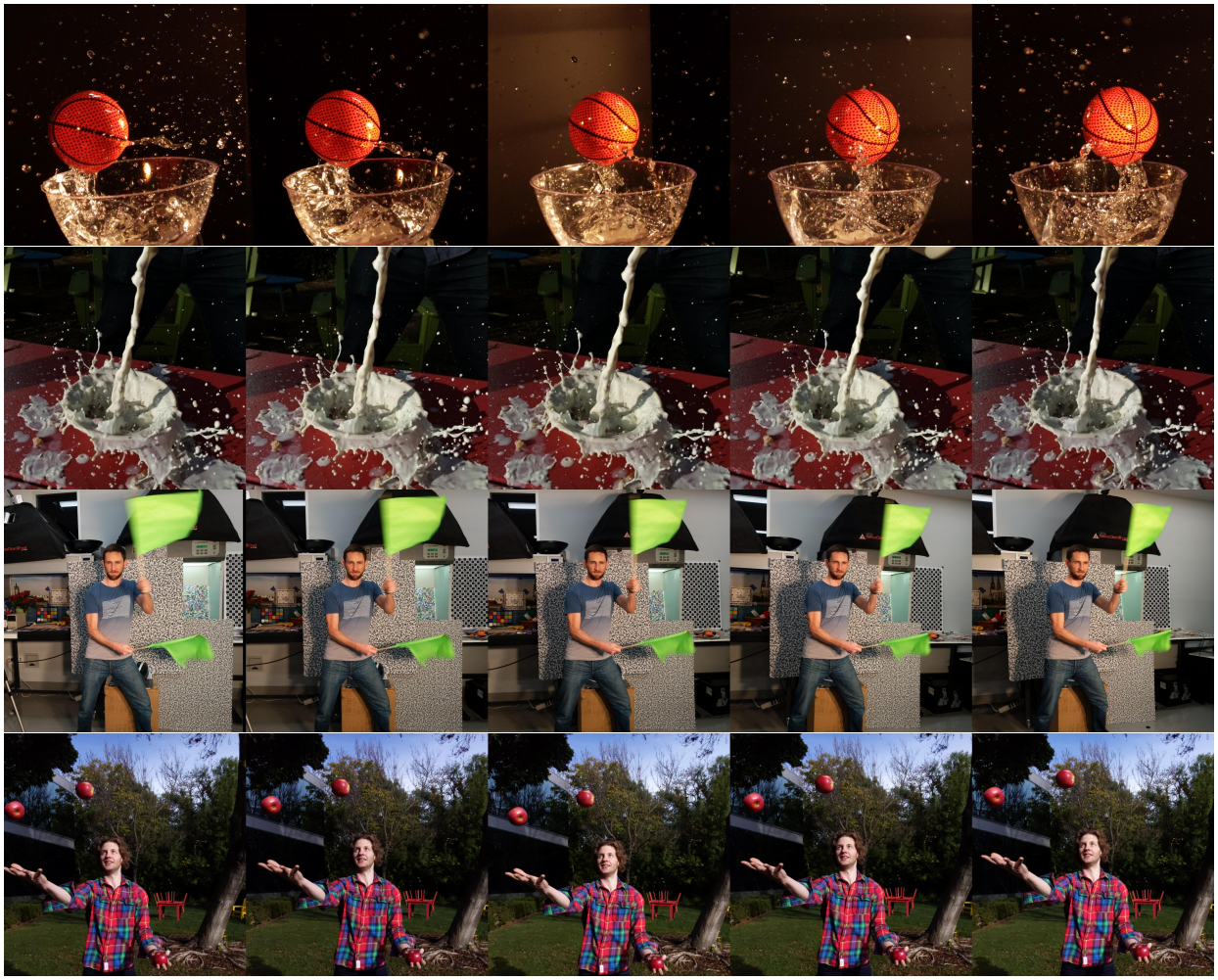


Fig. 10. Additional captures from five synchronized smartphones. Note the consistent motion blur across images.

REFERENCES

[1] D. Bradley, B. Atcheson, I. Ihrke, and W. Heidrich, "Synchronization and rolling shutter compensation for consumer video camera arrays," *CVPR Workshop*, 2009.

[2] M. Šmıd and J. Matas, "Rolling shutter camera synchronization with sub-millisecond accuracy," *Int. Conf. Comput. Vis. Theory Appl*, 2017.

[3] F. Padua, R. Carceroni, G. Santos, and K. Kutulakos, "Linear sequence-to-sequence alignment," *TPAMI*, 2010.

[4] P. A. Tresadern and I. D. Reid, "Synchronizing image sequences of non-rigid objects." *BMVC*, 2003.

[5] C. Dai, Y. Zheng, and X. Li, "Subframe video synchronization via 3D phase correlation," *ICIP*, 2006.

[6] Y. Wang, J. Wang, and S. Chang, "Camswarm: Instantaneous smartphone camera arrays for collaborative photography," *CoRR*, 2015.

[7] J. Xie, R. Girshick, and A. Farhadi, "Deep3D: Fully automatic 2D-to-3D video conversion with deep convolutional neural networks," *ECCV*, 2016.

[8] R. Garg, C. Kumar BG, G. Carneiro, and I. Reid, "Unsupervised CNN for single view depth estimation: geometry to the rescue," *ECCV*, 2016.

[9] C. Godard, O. M. Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," *CVPR*, 2017.

[10] K. Herakleous and C. Poullis, "3DUNDERWORLD-SLS: An open-source structured-light scanning system for rapid geometry acquisition," *CoRR*, 2014. [Online]. Available: http://arxiv.org/abs/1406.6595

[11] G. Litos, X. Zabulis, and G. Triantafyllidis, "Synchronous image acquisition based on network synchronization," *CVPR Workshop*, 2006.

[12] D. L. Mills, *Computer Network Time Synchronization: The Network Time Protocol on Earth and in Space, Second Edition*, 2nd ed. Boca Raton, FL, USA: CRC Press, Inc., 2010.

[13] T. Petković, T. Pribanić, M. Djonlić, and N. Dapuzzo, "Software synchronization of projector and camera for structured light 3D body scanning," *Int. Conf. on 3D Body Scanning Technologies*, 2016.

[14] L. Ahrenberg, I. Ihrke, and M. Magnor, "A mobile system for multi-video recording," *European Conf. on Visual Media Production*, 2004.

[15] "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems," *IEEE Std. 1588-2008*, 7 2008.

[16] "Android Camera2 API," http://developer.android.com/reference/android/hardware/camera2/package-summary.html.

[17] R. Garg and N. Wadhwa, "Learning to predict depth on the pixel 3 phones," *Google AI Blog*, 2018.

[18] Y. Furukawa and C. Hernández, "Multi-view stereo: A tutorial," *Foundations and Trends in Computer Graphics and Vision*, 2015.

[19] "COLMAP," https://colmap.github.io/.

[20] J. Schönberger and J. Frahm, "Structure-from-motion revisited," *CVPR*, 2016.

[21] J. Schönberger, E. Zheng, J. Frahm, and M. Pollefeys, "Pixelwise view selection for unstructured multi-view stereo," *ECCV*, 2016.

[22] R. Szeliski, "Image mosaicing for tele-reality applications," *WACV*, 1994.

[23] R. Szeliski and H.-Y. Shum, "Creating full view panoramic image mosaics and environment maps," *SIGGRAPH*, 1997.

[24] A. Agarwala, M. Agrawala, M. Cohen, D. Salesin, and R. Szeliski, "Photographing long scenes with multi-viewpoint panoramas," *SIGGRAPH*, 2006.

[25] R. Garg and S. M. Seitz, "Dynamic mosaics," *3DIMPVT*, 2012.

[26] "Microsoft Research Image Composite Editor," https://www.microsoft.com/en-us/research/product/computational-photography-applications/image-composite-editor/.

Supplement to Wireless Software Synchronization of Multiple Distributed Cameras

Sameer Ansari, Neal Wadhwa, Rahul Garg, and Jiawen Chen
Google Research, Mountain View, CA

I. Analysis of Clock Filters

We present additional analysis of the mean and min clock filters. We collected a dataset of 10,000 NTP messages together with a pair of images of our LED panel. Since each camera is hardware timestamped in its device's local clock domain and the LED panel serves as an external clock, our setup lets us compute a reference timestamp for each message and therefore measure one-way latency.

Recall that the NTP time offset estimate is defined as:

$$\theta = \frac{(t_1 - t_0) + (t_2 - t_3)}{2}. \tag{1}$$

Fig. 1 is a *wedge scattergram* [1], plotting estimated clock offset θ vs. estimated round-trip latency ϕ. Notice that samples with larger round-trip latency also have greater variance in clock offset. Since we can measure one-way latency, we can confirm the NTP hypothesis that the "limbs" of the wedge correspond to messages where the send and receive messages experienced the greatest difference in latency. This observation suggest the use of a filter that selects for the samples with the lowest latency. In our dataset of 10,000 messages, we observed that client to leader latency was consistently higher than the reverse by $\Delta \approx 0.75\,\text{ms}$. If we compute θ for each sample in our dataset and robustly compute the mean by rejecting outliers with round-trip latency greater than $10\,\text{ms}$, we converge to a bias of $\Delta/2 \approx 0.4\,\text{ms}$.

The min filter performs well if we "get lucky" and observe a message with low round-trip latency. Over 10,000 messages, the one with the lowest round-trip latency took $479\,\mu\text{s}$ from client to leader and $517\,\mu\text{s}$ from leader to client. Such a short period of time does not permit much buffering to contaminate our clock estimate, leading to a bias of only $19\,\mu\text{s}$. In our simple implementation of min filtering, once we find a good sample, the value persists until a sample with lower latency is later observed. It is straightforward to generalize to a more sophisticated strategy that uses the best k samples, have old messages expire, etc.

The behavior of both mean and min filters accumulated over increasing sample counts is shown in Fig. 2. The mean filter converges to the bias of $\Delta/2$ while the min filter effectively latches onto the best sample found so far.

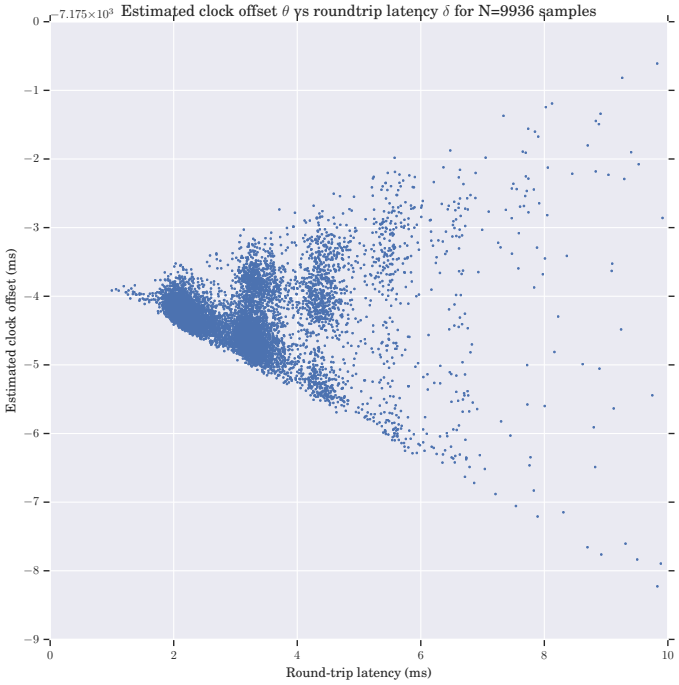


Fig. 1. *Wedge scattergram* of estimated clock offset θ on the y-axis and round-trip latency ϕ on the x-axis. Since higher latency samples have more variance when estimating θ, the min filter looks for the apex of the wedge with lowest latency and therefore clock offset variance.

II. Additional Results

A. Multi-view Stereo Results

Multi-view stereo algorithms yield better depth maps as we add input views. Since our system scales to multiple cameras, we can capture multiple synchronized views of dynamic scenes. Fig. 3 and 4 show two such examples and compare the results to the case when the cameras are not perfectly synchronized, e.g., when using

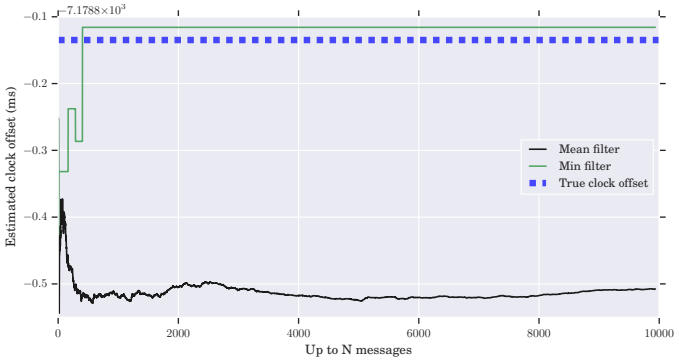


Fig. 2. Convergence of mean vs. min filters. The mean filter converges to a systematic bias of approximately half the difference in one-way network latency. In our setup, the min filter finds a low-latency sample within $K = 300$ messages.

Fig. 3. Multi-view stereo result for a jumping subject. Top: When the cameras are synchronized, the algorithm is able to use the information from all five views to generate a depth map that is better than the depth map we obtain from a stereo pair (Fig. 8 in the main paper). Bottom: When the views are not perfectly synchronized, e.g., when using a naive synchronization method, the depth map has missing information.

Fig. 4. Multi-view stereo result for a rotating pinwheel when the cameras are synchronized using our system (Top) vs simulated naive synchronization (Bottom). Pose estimation fails in the latter case due to large differences in images.

naive synchronization. To simulate naive synchronization, we choose an offset for each camera sampled uniformly from $\{-66\,\text{ms}, -33\,\text{ms}, 0\,\text{ms}, 33\,\text{ms}, 66\,\text{ms}\}$ as measured from the base camera. Depth maps are computed using COLMAP [2].

B. Synchronized Multi-view Bursts

Our system can capture high-resolution image sequences ("bursts") or lower-resolution videos from multiple cameras and is limited only by disk bandwidth. Fig. 5 shows a space-time volume of 5 frames from 5 views of a dynamic scene captured using our system.

References

[1] D. L. Mills, *Computer Network Time Synchronization: The Network Time Protocol on Earth and in Space, Second Edition*, 2nd ed. Boca Raton, FL, USA: CRC Press, Inc., 2010.

[2] "COLMAP," https://colmap.github.io/.

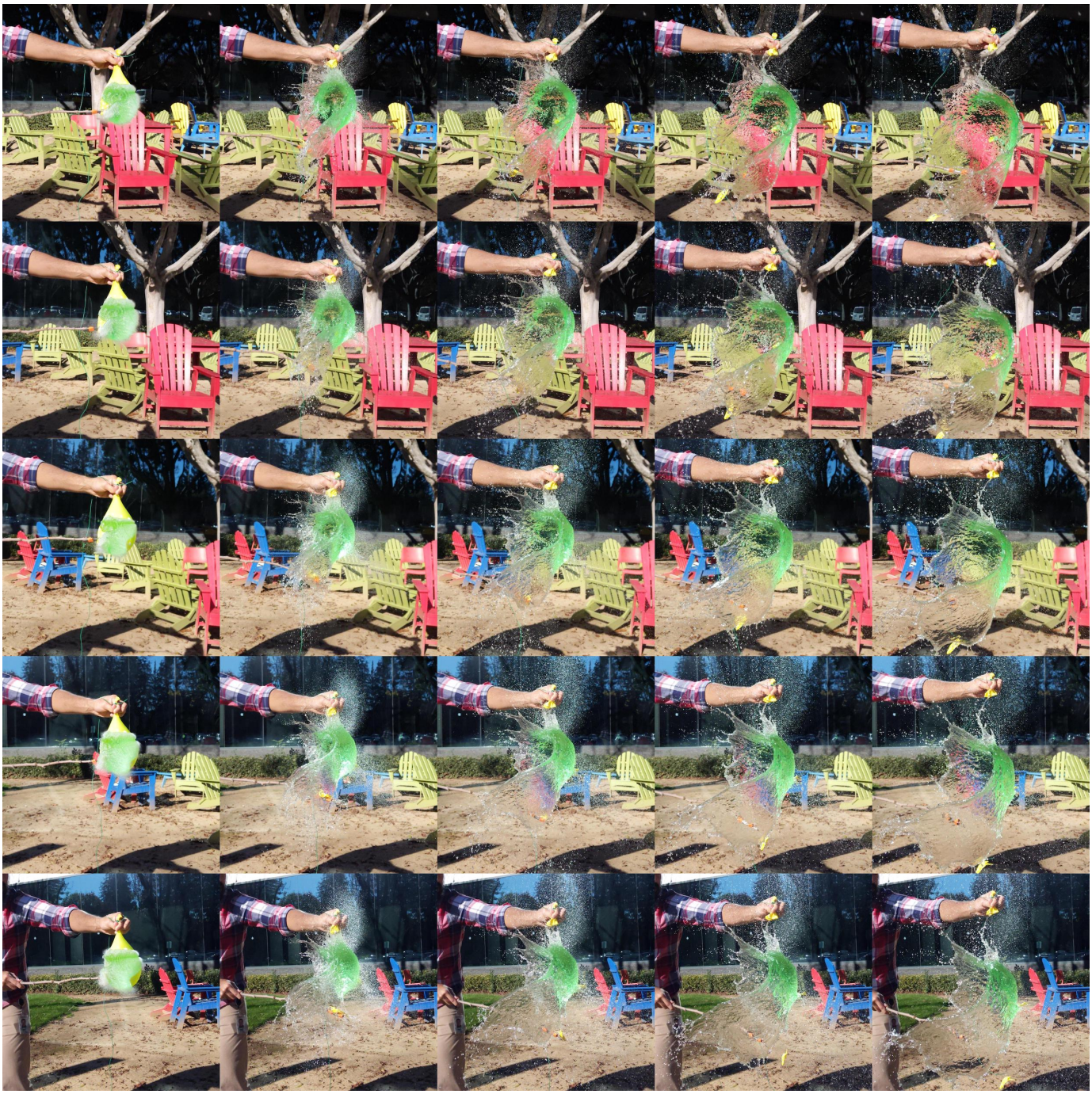


Fig. 5. Synchronized bursts of a popping water balloon from five cameras. Each row is captured by the same camera while each column is a synchronized instant in time. Since there is sufficient light outdoors, we can use a short exposure time of $43\,\mu\mathrm{s}$ while sensor readout time is $33\,\mathrm{ms}$.