Supplementary Material for SocialSync: Sub-Frame Synchronization in a Smartphone Camera Network

Richard Latimer, Jason Holloway, Ashok Veeraraghavan, Ashutosh Sabharwal

Rice University, Houston, TX

1 Implementation Details

The following section provides high level framework implementation supplementing those described in Sec. 3 Camera Characterization of [1].

1.1 Design Outline

Our implementation uses client devices consisting of the HTC One (M7) and Nexus 5 running Android OS 4.3 and 4.4. The communication server is a Windows 7 laptop running a web service powered by Play! framework. The time server is a Raspberry Pi computer running an NTP daemon built into the Unix shell and synchronized to a GPS clock. A Cisco router links these four devices together.

Android App The following details are important when implementing our SocialSync Android app:

- The smartphone camera network may become unsynchronized when the auto exposure is enabled, as the frame rate may change during metering. Therefore initially, auto exposure should be allowed to correct image contrast and then it should be locked for subsequent image captures.
- The camera client object runs on a background thread separately from the main activity and UI to prevent scheduling problems, otherwise the delivery timestamps $T_D(i)$ will be delayed considerably when the preview callback is executed, resulting in estimation error when predicting $T_C(0)$. Note that both the Nexus 5 and HTC One are multi-core; measurement results may be different on single core systems.
- Computation in the `onPreviewFrame()` callback must be returned before the next callback is scheduled since the callback is executed sequentially. Should the execution time in a callback delay the scheduling of the next callback, a cascading effect may occur, thereby delaying the measurement of T_D and introducing error into the estimate of $T_C(0)$. Prolonged processing in the preview callback may result in a frame being dropped. Furthermore, the process of saving frames to disk takes considerable time, therefore the frames either must be queued and recorded after the image sequence is captured or passed to a secondary thread running on a separate core.

– Since the camera object is a client, neither calling `stopPreview()` nor releasing the client object is sufficient to stop the camera service immediately. Therefore, our stochastic synchronization protocol is achieved through a hard restart of the app, thereby forcing the camera service to stop. An alternative approach to implementing the SocialSync synchronization protocol would be to vary the frame rate by enabling the auto exposure, white balance, or changing the FPS range. While possible, this setup requires calculating the frame rate on the fly.

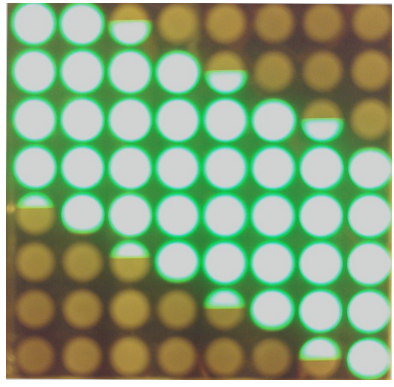

Fig. 1. The 8×8 LED clock showing rolling shutter effect. The sensor reads out rows of pixels from the top to bottom. The RPi turns on a column of LEDs, moving from left to right in 1 ms intervals. By measuring the number of pixels between two partially illuminated LEDs and dividing by the difference in time, we estimate the per pixel readout time

1.2 Rolling Shutter:

CMOS image sensors, as typically found on smartphones, implement a rolling shutter, whereby rows of pixels are exposed to light at different times, due to readout speed requirements. The rolling shutter exposes each row of the image sensor evenly, but initiates the readout of the row in time intervals ΔT_R. Thus, each row y in the image has a readout time of

$$T_R(i, N_y) = T_C(i) + (y - 1) * \Delta T_R, \tag{1}$$

where $T_C(i)$ represents the capture of the i-th frame. Knowing the rolling shutter characteristics allows us to characterize the precise timing information of every pixel in the image. Readout speed of the image sensor is measured by placing the rolling shutter rows perpendicular to the LED clock array, such that the LEDs sweep across the image plane as a column, in 1 ms intervals, as seen in Fig. 1.

By examining the difference between illuminated pixels along the diagonal temporal shear of the image and dividing by the elapsed time between pixels, we measured the rolling shutter readout speed ΔT_R as approximately 24 µs per row for a Nexus 5 and 18 µs per row for a HTC One, with 1080 sensor rows per preview image for both devices. Therefore, rolling shutter pixel readout time of pixels across the sensor may differ by 20 ms for typical 2.1 megapixel preview images.

1.3 Capture Timestamp Estimate:

By examining the captured photo of the LED array, we have created a precise timing platform for estimating $T_C(i)$ by measuring the number of illuminated LEDs at a row of pixels y, where the previous row had a set of illuminated LEDs which were shifted by 1 and represent a difference of 1 ms. If the LED clock triggers at $T_S(i)$ by an external trigger, the number of illuminated LEDs N_y for a row of pixels indicates how much time has passed since $T_S(i)$. We express this as

$$T_L(i, N_y) = T_S(i) + N_y. \tag{2}$$

We measure the number of illuminated LEDs at the boundary where a previous row of illuminated LEDs was shifted by a time duration of 1 ms and solve for $T_C(i)$ by setting $T_R(i, y)$ equal to $T_L(i, N_y)$ with $T_S(i)$, y, N_y, and ΔT_R being known constants.

References

1. Latimer, R., Holloway, J., Veeraraghavan, A., Sabharwal, A.: SocialSync: Sub-frame synchronization in a smartphone camera network (2014), Computer Vision–ECCV 2014. LF4CV submission.