Supplemental Material for
Syncalize - Precise Automatic Timecodes for Video and Audio Devices Using Consumer Hardware

1. Pseudocode Syncalize video decoding

Decoding starts with the ArUco marker detection which is part of the standard OpenCV toolkit. The detected marker edges are correlated to the coordinates in the normalized pattern coordinate system using the OpenCV findHomography function. This calculates a 3x3 image transformation homography matrix. The estimated homography is now used to transform the pattern part of the input image into a normalized flat view of defined dimensions. All positions of runner squares, global counter squares, and ArUco markers are defined in this normalized coordinate system. The *Decoding* Algorithm 1 can now extract the encoded bits of the counter and the state of all runner squares (set or cleared).

The global counter value is derived from the Gray-encoded bits of the gi_{set} global counter bits. The start and stop of the longest activated consecutive stretch of ri_{set} runner bits is calculated using the *Find Longest Consecutive* Algorithm 2 yielding $runner\_start$ and $runner\_stop$.

Finally, a sub-granularity refinement is calculated for the borders of the runner positions. The edge positions of the runner are typically not fully captured during the camera's exposure. The *Calculate Subgranularity* Algorithm 3 calculates the fractions of a single pattern frame during which the edges were lit during the camera's exposure. This information leads to a higher temporal resolution than the single frame duration. Exposure start and stop times can be calculated by multiplying $runner\_start - runner\_start\_frac$ with $1/target\_framerate$ (and $runner\_stop + runner\_stop\_frac$).

2. Pseudocode Syncalize audio decoding

The audio signal is converted into a spectrogram using the SciPy[1] libraries *ShortTimeFFT* function. This calculates signal strength per spectrogram bin for N (oversampling) $\times$ Bd (Baud rate) timings. For each of the two frequencies used during encoding ($f_{carrier} \pm f_{deviation} \cdot 0.5$) the *Spectrum Filtering* Algorithm 4 calculates a suitable triangular function (hat function) frequency filter for all SFFT spectrum bin frequencies.

A dot product between the spectrogram and both spectrum filters results in two filter responses per timing step: one for the frequency which encodes clear bits (0) and one correlating to set bits (1) in the bitstream. The *Compare Downsampling* Algorithm 5 calculates the decoded bitstream.

The bitstream is correlated with the seven-bit Barker code (binary 1110010) to identify potential starting points. This is done per chunks of (2*20-1)*N as these should only contain a maximum of one full 20-bit package. The validity and sub-sample-accurate start of a potential Barker code is calculated using *Evaluate Barker Code* Algorithm 6. We expect at least an equivalent of two bits of silence before a timing package ($n_{silence\_bits}$ = 2). The needed correlation filters are created by the *Create Correlation Filters* Algorithm 7. Correlation of a Barker code will result in a quality of 1.0 while misalignment or missing windows of silence will quickly lower the quality score below zero. Detections meeting a minimum quality score (e.g. 0.125) are again downsampled with Algorithm 5 (as they are now better centered by the sub-sample-accurate start information) and decoded by the Hamming decoder. If fewer than two bit errors are detected, then this package (decoded timecode and sub-sample-accurate time within the audio file) is added to the list of decoded packages, decoding continues N samples after the decoded package. Otherwise, up to N samples of phase offsets are applied to the current chunk to find a valid timecode. The final list of all decoded samples is filtered by calculating the difference between the encoded time value and the position within the audio file. The mean value from all samples within the 25%-75% Quantile of these differences is chosen as the final robust estimation for the overall synchronization offset.

Algorithm 1 Decoding Algorithm

Input: Image, list of coordinates for positions of runner squares, binary counter squares, and ArUco marker corner bits
Output: State (set/cleared) of runner squares and global counter squares
$n_r \leftarrow 30$ ▷ Number of runners, e.g. 6x5 = 30
$n_g \leftarrow 8$ ▷ Number of global counter bits
$n_a \leftarrow 4$ ▷ Number of ArUco markers
$n_c \leftarrow 4 * n_a$ ▷ Number of ArUco marker corners
$\lambda_{lit} \leftarrow 0.125$ ▷ Dynamic threshold, Hyperparameter
for $ri = 0$ to $n_r - 1$ **do** ▷ Runner at index ri
  $\tilde{Ari}_{inner} \leftarrow$ median brightness of inner area
  $\tilde{Ari}_{border} \leftarrow$ median of surrounding border
  $Lri_{val} \leftarrow \tilde{Ari}_{inner} - \tilde{Ari}_{border}$
end for
for $gi = 0$ to $n_g - 1$ **do** ▷ Global counter bit at index gi
  $\tilde{Agi}_{inner} \leftarrow$ median brightness of inner area
  $\tilde{Agi}_{border} \leftarrow$ median of surrounding border
  $Lgi_{val} \leftarrow \tilde{Agi}_{inner} - \tilde{Agi}_{border}$
end for
for $cj = 0$ to $n_c - 1$ **do** ▷ Corner bits cj from markers
  $\tilde{Acj}_{inner} \leftarrow$ median brightness of inner area
  $\tilde{Acj}_{border} \leftarrow$ median of surrounding border
  $Lj_{max} \leftarrow \tilde{Acj}_{inner} - \tilde{Acj}_{border}$
end for
for $ri = 0$ to $n_r - 1$ **do**
  $Lri_{max} \leftarrow \frac{\sum_{j=0}^{n_c-1} coord_distance(ri,cj) \cdot Lj_{max}}{\sum_{cj=0}^{n_c-1} coord_distance(ri,cj)}$
end for
for $gi = 0$ to $n_g - 1$ **do**
  $Lgi_{max} \leftarrow \frac{\sum_{j=0}^{n_c-1} coord_distance(gi,cj) \cdot Lj_{max}}{\sum_{cj=0}^{n_c-1} coord_distance(gi,cj)}$
end for
$L_{thr} \leftarrow \tilde{Lxi}_{max} \cdot \lambda_{lit}$
for $ri = 0$ to $n_r - 1$ **do**
  $ri_{set} \leftarrow (Lri_{val} > L_{thr})$
end for
for $gi = 0$ to $n_g - 1$ **do**
  $gi_{set} \leftarrow (Lgi_{val} > L_{thr})$
end for

3. Used Timing Pattern Statistic

Tab. 2 show statistics on the *Syncalize* video timing pattern used for all experiments. The video's audio track uses the *Syncalize* audio encoding by superimposing three individual signals: 11kHz at 24baud with one package per second, 3kHz at 48baud with two packages per second, and 7.7kHz at 72baud with four packages per second. All baud rates are multiples of 24 so the result of one SFFT for 72baud (oversampling 8) can be used for all three signals.

4. iPhone as a pattern playback device

We repeated our controlled exposure experiments using an iPhone 15 Pro with a 6.1 inch 460 ppi OLED display as our pattern playback device. Fig. 1 shows views similar to the original paper's Figure 5. A slight degradation in estimation accuracy is observed in the new results Tab. 1 compared to the original paper's Table 1 (where a Lenovo tablet was used). This can be attributed to the considerably smaller display size of the iPhone compared to the tablet. Nevertheless, the timing estimates remain within the single-digit ms range, which is sufficient for most practical applications. The iPhone's adaptive screen refresh rate was no problem for the test and stayed at 120Hz during playback of the *Syncalize* video file.

Algorithm 2 Find Longest Consecutive

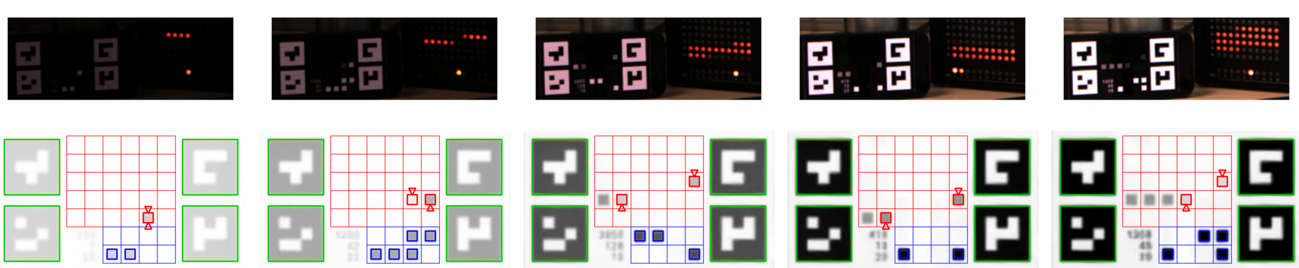
Input: List of binary values
Output: start index and stop index
$n \leftarrow$ length of $values$
$padded \leftarrow$ concatenate([0], $values$, $values[.. - 1]$, [0])
$diff \leftarrow padded[1..] - padded[.. - 1]$
$starts \leftarrow$ indices where $diff = 1$
$ends \leftarrow$ indices where $diff = -1$
$lengths \leftarrow ends - starts$
$max\_idx \leftarrow$ index of maximum value in $lengths$
$start\_index \leftarrow starts[max\_idx]$ mod n
$end\_index \leftarrow (ends[max\_idx] - 1)$ mod n

Algorithm 3 Calculate Subgranularity

Input: List of ri_{set}, Lri_{val}, Lri_{max}, runner index $start$ and $stop$
Output: Fraction for runner $start$ and $stop$
$rnext \leftarrow (start + 1)$ mod n_r
$rprev \leftarrow (stop + n_r - 1)$ mod n_r
if $rnext == rprev$ or $start == stop$ **then**
  $Lrnext_{full} \leftarrow Lrnext_{max}$
  $Lrprev_{full} \leftarrow Lrprev_{max}$
else
  $Lrnext_{full} \leftarrow Lrnext_{val}$
  $Lrprev_{full} \leftarrow Lrprev_{val}$
end if
$start\_frac \leftarrow Lrstart_{val}/Lrnext_{full}$
$stop\_frac \leftarrow Lrstop_{val}/Lrprev_{full}$

Algorithm 4 Spectrum Filtering

Input: List of frequencies, frequency f, and $f_{deviation}$
Output: List of Weights ([0..1] per frequency)
$n \leftarrow$ number of $frequencies$
$weights \leftarrow$ []
for $i = 0$ to $n - 1$ **do**
  $wi_f \leftarrow$ min(max($|frequencies[i] - f|/f_{deviation}$,0),1)
  $weights \leftarrow$ concatenate(weights, $[wi_f]$)
end for



4 ms | 8 ms | 16 ms | 24 ms | 30 ms

Figure 1. Examples of recorded images (top) and decoded pattern (bottom) for different exposure times from experiments in Tab. 1. The reference LED-Panel is visible in the top right of each image. Each LED is active for 1 ms.

Algorithm 5 Compare Downsampling

Input: Filter response r_0, r_1 for bit 0 and bit 1, oversampling factor N
Output: Resulting downsampled bit stream
$nr_0 \leftarrow$ number of samples in r_0
$n \leftarrow nr_0/N$ ▷ Number of result bits
$bitresults \leftarrow []$
for $i = 0$ to $n - 1$ **do**
    $cnt\_as\_1 \leftarrow 0$ ▷ Count where bit 1 response is greater than bit 0 response
    for $j = 0$ to $N - 1$ **do**
        if $r_1[i \cdot N + j] > r_0[i \cdot N + j]$ **then**
            $cnt\_as\_1 \leftarrow cnt\_as\_1 + 1$
        end if
    end for
    if $cnt\_as\_1 >= N/2$ **then**
        $bi \leftarrow 1$
    else
        $bi \leftarrow 0$
    end if
    $bitresults \leftarrow$ concatenate($bitresults$, [bi])
end for

Algorithm 6 Evaluate Barker Code

Input: Filter response r_0, r_1 for bit 0 and bit 1, oversampling factor N, potential start index si for barker code, bits of silence expected before barker code $n_{silence\_bits}$, correlation filters $corr_0$ and $corr_1$
Output: Quality of correlation with Barker code and subsample accurate start index
$\lambda_{thr_high} \leftarrow 0.125$ ▷ Dynamic activation threshold is calculated from 12.5% quantile
$n_{silence\_samples} \leftarrow n_{silence\_bits} \cdot N$
$i_{min} \leftarrow si - N - n_{silence\_samples}$ ▷ Search optimum in $\pm N$ neighborhood
$i_{max} \leftarrow si + N - n_{silence\_samples}$
$lvl\_high_0 \leftarrow Quantile(r_0[i_{min}..i_{max}], \lambda_{thr\_high})$
$lvl\_high_1 \leftarrow Quantile(r_0[i_{min}..i_{max}], \lambda_{thr\_high})$
$n \leftarrow$ length of correlation filter $corr_0$
$quality\_value \leftarrow []$
$quality\_index \leftarrow []$
for $i = i_{min}$ to i_{max} **do**
    $q_0 \leftarrow \sum_{j=0}^{n-1} coor_0[j] \cdot r_0[i..i + n - 1]/lvl\_high_0$
    $q_1 \leftarrow \sum_{j=0}^{n-1} coor_1[j] \cdot r_1[i..i + n - 1]/lvl\_high_1$
    $qi \leftarrow \sqrt{q_0 * q_0 + q_1 * q_1}$
    $quality\_value \leftarrow$ concatenate($quality_value$, [q])
    $quality\_index \leftarrow$ concatenate($quality_index$, [qi])
end for
$max\_idx \leftarrow$ argmax($quality\_value$)
$sub\_acc\_part \leftarrow$ Find Parabola Maximum (Algorithm 8) for $quality_value[max\_idx - 1]..max\_idx + 1]$
$sub\_frame\_accurate\_start \leftarrow quality_index[max\_idx] + sub\_acc\_part$
$max_quality \leftarrow quality\_value[max\_idx]$

Algorithm 7 Create Correlation Filters

Input: List representing Barker code bits, oversampling factor N, expected bits of pause before barker code $n_{silence\_bits}$, correlation filters $corr_0$ and $corr_1$
Output: Two correlation filters $corr_0$ and $corr_1$
$n \leftarrow$ number of Barker code bits
$weight_{barker} = 1.0/n$
$weight_{silence} = 1.0/n_{silence\_bits}$
$corr_0 \leftarrow []$
$corr_1 \leftarrow []$
for $i = 0$ to $n_{silence\_bits} * N - 1$ **do**
    $corr_0 \leftarrow$ concatenate($corr_0$,[$-weight_{silence}$])
    $corr_1 \leftarrow$ concatenate($corr_1$,[$-weight_{silence}$])
end for
for $i = 0$ to $n - 1$ **do**
    if Barker code bit at index i is set **then**
        $weight_0 \leftarrow -weight_{barker}$
        $weight_1 \leftarrow weight_{barker}$
    else
        $weight_0 \leftarrow weight_{barker}$
        $weight_1 \leftarrow -weight_{barker}$
    end if
    for $j = 0$ to $N - 1$ **do**
        $corr_0 \leftarrow$ concatenate($corr_0$,[$weight_0$])
        $corr_1 \leftarrow$ concatenate($corr_1$,[$weight_1$])
    end for
end for

Algorithm 8 Find Parabola Maximum

Input: List of three values [$prev$, $curr$, $next$]
Output: Position of maximum of a fitted parabola in relation to the central value
$coeff_{parabola} \leftarrow$ Order 2 Polynomial for x=[0,1,2], f(x)=[0, $curr$-$prev$, $next$-$prev$]) ▷ (e.g. np.polyfit(..,..,2))
$rel\_max\_unbound \leftarrow -coeff_{parabola}[1]/(2 \cdot coeff_{parabola}[0]) - 1$
$rel\_max \leftarrow$ min(max($rel\_max\_unbound$, -0.5), 0.5)

iPhone	$\sigma_{\Delta t}$	frame rate		exposure time	
		mean	err	mean	err
	ms	Hz	Hz	ms	ms
camera: 8.1 Hz					
4 ms	1.41	8.07 σ0.10	−0.03	8.06 σ0.66	4.06
8 ms	1.17	8.10 σ0.09	0.00	8.06 σ2.24	0.06
16 ms	1.71	8.10 σ0.04	0.00	13.88 σ3.38	−2.12
24 ms	3.51	8.10 σ0.01	0.00	23.84 σ1.18	−0.16
36 ms	0.88	8.10 σ0.04	0.00	36.00 σ1.02	0.00
camera: 20 Hz					
4 ms	0.02	20.00 σ0.00	0.00	8.32 σ0.03	4.32
8 ms	2.84	20.00 σ0.01	0.00	5.27 σ1.00	−2.73
16 ms	0.34	20.00 σ0.02	0.00	17.97 σ0.15	1.97
24 ms	3.50	20.00 σ0.02	0.00	23.79 σ0.98	−0.21
36 ms	0.30	20.00 σ0.04	0.00	37.00 σ0.12	1.00

Table 1. *Syncalize* estimation accuracy of video offset uncertainty ($\sigma_{\Delta t}$), frame rate, and exposure time at fixed frame rates and exposure times with pattern played on an iPhone 15 Pro. All values computed from 150 estimates.

playback rate	120 Hz
single frame duration	8.33 ms
number of runner position	30
runner duration	250 ms
global counter range (8-bit$\setminus\{0\}$)	255
total number of frames	7650
total runtime	63.75 s

Table 2. Timing statistics of the pattern sequence with a playback rate of 120 Hz.

References

[1] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. 1