Ein neuer Algorithmus zur Zeitsynchronisierung von Ereignis-basierten Zeitreihendaten als Alternative zur Kreuzkorrelation

Christoph Schranz[1] & Sebastian Mayr[1]

[1]Salzburg Research Forschungsgesellschaft m.b.H., Salzburg, Österreich

Kurzfassung

Mit der Verwendung von Sensordaten aus mehreren Quellen entsteht oft die Notwendigkeit einer Synchronisierung der entstandenen Messreihen. Ein Standardverfahren dazu ist die Kreuzkorrelation, die jedoch übereinstimmende Zeitstempel voraussetzt und empfindlich gegenüber Ausreißern reagiert. In diesem Paper wird daher ein alternativer Algorithmus für die Synchronisierung von Ereignis-basierten Zeitreihendaten vorgestellt.

Schlüsselwörter: Ereignis-basierte Zeitreihendaten, Synchronisierung, Kreuzkorrelation

Einleitung

In vielen praktischen Anwendungen, wie zum Beispiel beim Vergleich eines neuartigen Sensors mit der etablierten Messmethode, liegen zwei oder mehrere Messungen derselben oder einer korrelierenden Metrik mit versetztem Zeitstempel vor. Ursachen für einen solchen Zeitversatz können unter anderem Synchronisierungsprobleme der unterschiedlichen Geräte sein. In diesem Fall muss der Zeitunterschied zwischen den Messungen ermittelt und die Zeitstempel der als nicht synchron angenommen Messung korrigiert werden.

Das Standardverfahren zur Synchronisierung von Ereignis-basierten Zeitreihen ist es, die jeweiligen Datenpunkte auf eine konstante Abtastrate zu interpolieren um anschließend die Kreuzkorrelation für einzelne Zeitversätze zwischen den Messungen zu berechnen. Eine Verringerung der Laufzeit kann dabei durch die Verwendung der Fourier-Transformation (FFT) erreicht werden (Lyon, 2010). Dieses Standardverfahren besitzt jedoch mehrere Nachteile: Falls die inhärente Frequenz der Signaländerung geringer ist als jene der auftretenden Ereignisse, sinkt die Präzision des ermittelten Zeitversatzes. Zusätzlich zeigt sich auch eine geringe Robustheit der Ergebnisse bei kurzen Zeitreihen sowie bei fehlenden oder falschen Werte, wodurch oft eine aufwändige Korrektur erforderlich ist. (Pearson, 2005)

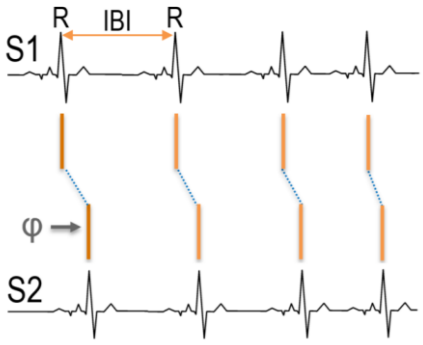
In dieser Arbeit wird ein neues Verfahren für die Synchronisierung Ereignis-basierter Zeitreihendaten vorgestellt. Der Fokus bei der Entwicklung dieses Verfahrens war eine hohe Präzision des resultierenden Zeitversatzes und Robustheit.

Methode

Die hier vorgestellte Methode zur Synchronisierung Ereignis-basierter Zeitreihendaten basiert auf einem Reißverschlussprinzip, angelehnt an den effizienten Stream-Stream-Join Algorithmus von Schranz 2020. Grundsätzlich wird für jedes Ereignis der Abstand zum jeweils Nächsten der anderen Zeitreihe berechnet. Der Mittelwert aller dieser Abstände dient als Maß für die Synchronität der Zeitreihen für einen gegebenen Zeitversatz φ. Aufgrund

des iterativen Vergleichs der Zeitstempel zum jeweils nächsten Gegenüber wird der Algorithmus im Rahmen dieser Arbeit als *nearest_advocate* bezeichnet.

```
def nearest_advocate(arr_1, arr_2, time_delta=0.0, max_distance=0.5):
    arr_2 = arr_2 - time_delta       # apply time shift
    cum_distance = 0.0; counter = 0  # initialize cumulative values
    i1, i2 = forward_until_first_intersection(arr_1, arr_2)  # skip indices
    for ts_2 in arr_2[i2:]:          # iterate over all timestamps in arr2
        while arr_1[i1+1] <= ts_2:   # forward i1 until arr_1[i1+1] > ts_2
            i1 += 1
            if i1 + 1 > len(arr_1):  # array 1 is finished, return
                return cum_distance / counter
        if arr_1[i1] <= ts_2:
            # here the invariance arr_1[i1] <= ts_2 < arr_1[i1+1] holds
            cum_distance += min(ts_2-arr_1[i1], arr_1[i1+1]-ts_2,
                                max_distance)
            counter += 1
    return cum_distance / counter
```



(a) (b)

Abb. 1 (a) Pseudocode des *nearest_advocate*; (b) Zwei idente um φ versetzte EKG (schwarz) mit deren charakteristischen Schläge (R-peaks, orange) und den minimalen Abständen (blau strichliert).

In Abbildung 1.a wird der Pseudocode dargestellt: Gegeben zwei sortierte Reihen aus Zeitstempeln der Ereignisse mit überlappenden Zeitabschnitten, berechnet *nearest_advocate* den mittleren Abstand von jedem Zeitstempel in Reihe 2 zu dem jeweils nächsten Gegenüber in Reihe 1 (siehe Abb. 1.b). Der Parameter *‚time_delta‘* gibt dabei den zu evaluierenden Zeitversatz an und *‚max_distance‘* den maximal akzeptierten Abstand zwischen zwei gegenüberliegenden Ereignissen.

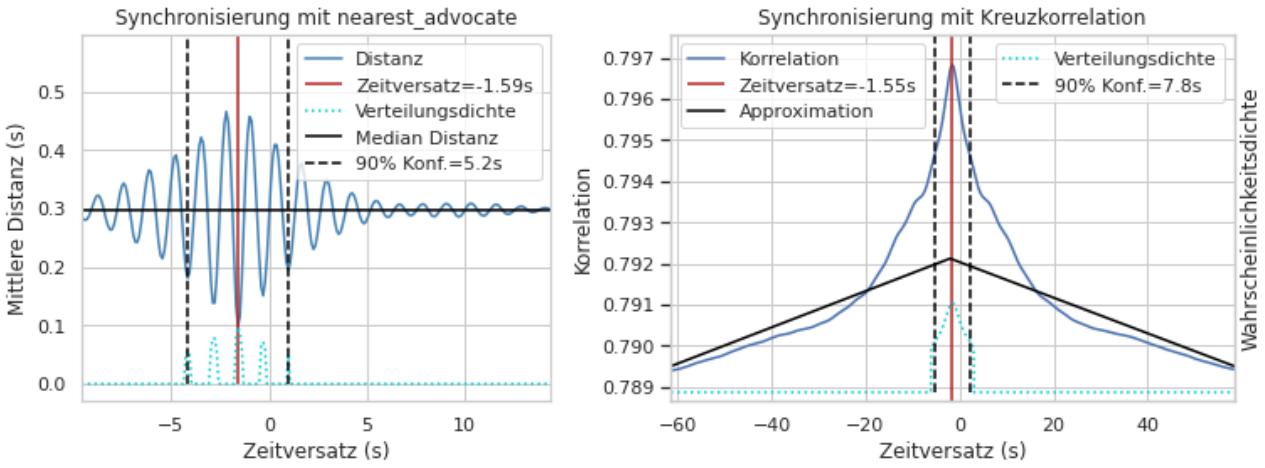
Der Algorithmus besitzt eine lineare Laufzeit- und Speicherkomplexität für einen zu prüfenden Zeitversatz. Insgesamt beträgt die Laufzeitkomplexität somit $|time\_deltas| \cdot (|arr_1| + |arr_2|)$. Für die Wahl der zu testenden Zeitversätze ist zu beachten, dass für die Erkennung der Form und somit des Optimums der Ergebniskurve etwa die 10- bis 20-fache Rate der erwarteten Nyquist- oder Signalfrequenz empfohlen wird (Wescott, 2018).

Experiment

Für das Experiment wurden EKG-Daten (siehe Abb. 1 b) verwendet, die innerhalb des Projektes Virtual Sleep Lab während des Schlafs erhoben wurden (siehe Finanzierung). Die Zeitreihe 1 wurde mit dem laborüblichen Polysomnographen Brainvision BrainAmp ExG aufgenommen, die Zeitreihe 2 mit dem Suunto Movesense Sensor HR+. Beide Reihen liegen in Form von Arrays vor, wobei die Elemente die Zeitstempel der charakteristischen R-peaks der jeweiligen Herzschläge sind. Die Messdauer betrug etwa 30,000 Sekunden.

Folgende Algorithmen wurden im Experiment verglichen: (1) Eine Kreuzkorrelation mit FFT auf linear interpolierte Abstände der R-peaks (*interbeat intervals, IBI*). Zusätzlich (2) eine Kreuzkorrelation mit FFT auf, mittels Dreiecken der Breite 0.5s, interpolierten R-peaks, um die Präzision der Zeitversätze zu verbessern. Der hier vorgestellte Algorithmus *nearest_advocate* wird mit einer Maximaldistanz von 0.5s (3) auf alle R-peaks angewandt sowie (4) dünn besetzt (*sparse*) mit nur jedem 100sten R-peak der Zeitreihe 2. Der Suchraum für den Zeitversatz beträgt ±60 Sekunden bei einer Auflösung von 0.1 s.

In Abbildung 2 werden typische Ergebnisse für die Synchronisation dargestellt: In (a) das charakteristische Schwingen der jeweiligen Distanz (blau) um das Optimum (rot), das in den Algorithmen (2), (3) und (4) auftritt. In (b) das markante Maximum (blau) über einer dreieckigen Approximation für die Kreuzkorrelation (1). Aus dem Median der mittleren Distanzen (a) bzw. der Annäherung der Korrelationskoeffizienten mit einer Dreiecksform über zwei lineare Theil-Sen Regressoren (b) wird die Grundlinie (schwarz) bestimmt. Um das Hintergrundrauschen zu filtern, werden nur jene Kurvenanteile extrahiert, welche signifikanter als 50% des Optimums sind. Dieser Anteil wird auf eine Wahrscheinlichkeitsverteilung (türkis punktiert) normiert. Das Maß für die Präzision ist die Breite des 90%-Konfidenzintervalls dieser Wahrscheinlichkeitsverteilung (schwarz strichliert).



(a) (b)

Abb. 2 (a) Bei der Synchronisierung mit *nearest_advocate* treten um den Versatz φ (in rot) charakteristische Schwingungen auf; (b) Bei der Kreuzkorrelation eine markante Spitze um φ.

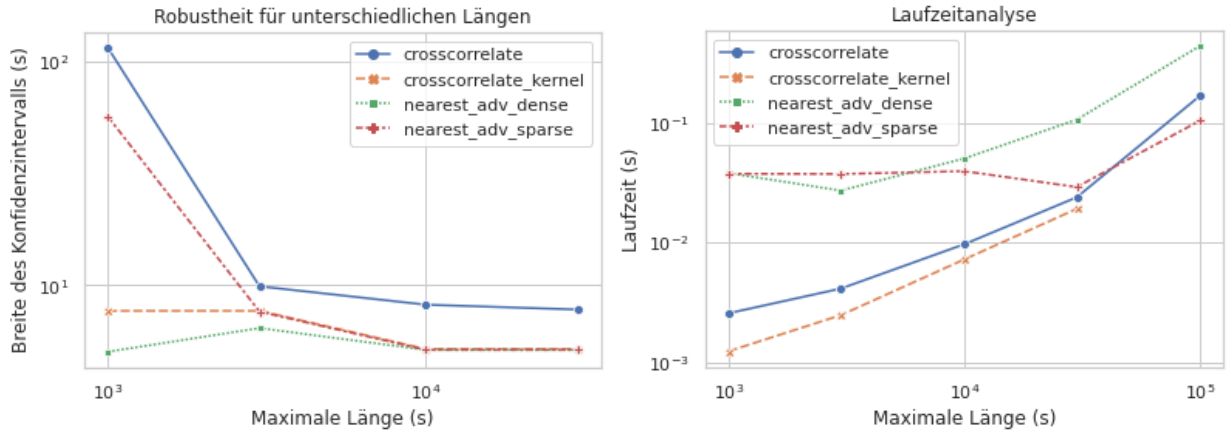
Die Experimente wurden in Python 3.9 durchgeführt. Für die Kreuzkorrelation wurde die Methode *signal.correlate* des Pakets *scipy* mit Version 1.7.3 verwendet. Um für *nearest_advocate* ebenfalls vergleichbare Laufzeiten wie in C zu erzielen, wurde dieser in der JIT-Kompilierungsumgebung *numba* 0.55.0 implementiert. (Lam, 2015)

Ergebnisse

In Abbildung 3 werden für die Synchronisierungen der jeweiligen Algorithmen (a) die Präzision als Breite der 90%-Konfidenzintervalle der Wahrscheinlichkeitsdichten und (b) Laufzeiten für unterschiedliche maximale Längen der Zeitreihen dargestellt. Für den Fall mit maximaler Länge von 100,000s wurde die Zeitreihe synthetisch vervielfacht.

Die Streubreiten nehmen mit steigender Länge tendenziell ab und die Laufzeiten zu. Der Algorithmus *nearest_advocate* lieferte die präzisesten und robustesten Ergebnisse. Die schnellere *sparse*-Variante liefert für längere Zeitreihen ebenfalls präzise Ergebnisse und besitzt sogar die günstigste asymptotische Laufzeitkomplexität, womit diese für die Länge von 100,000s sogar schneller als die Kreuzkorrelation mit FFT (mit Laufzeitkomplexität von $n \cdot log(n)$ (Lewis, 1995)) ist. Die Kreuzkorrelation mit Kernelapproximation liefert ebenfalls

sehr präzise Lösungen, allerdings wird die rechenintensive Interpolation nicht für die hier als sehr niedrig erscheinende Laufzeit berücksichtigt.



(a) (b)

Abb. 3 (a) Breite der 90%-Konfidenzintervalle der Wahrscheinlichkeitsdichten und (b) die Laufzeiten der Algorithmen für unterschiedliche maximale Längen der Zeitreihen.

Diskussion

In diesem Paper wurde exemplarisch für R-peaks als Ereignis-basierte Zeitreihen gezeigt, dass das hier vorgestellte Synchronisierungsverfahren *nearest_advocate* für alle getesteten Längen eine höhere Präzision als das Standardverfahren Kreuzkorrelation liefert. Für kurze Zeitreihen demonstriert dieser eine sehr hohe Robustheit. Die *sparse*-Variante verspricht trotz geringerer Präzision für kurze Reihen, eine asymptotisch sehr gute Laufzeit.

In zukünftigen Analysen soll die Robustheit insbesondere gegenüber fehlender und falscher Ereigniswerte genauer untersucht werden. Außerdem wäre es sinnvoll, zusätzliche Algorithmen wie z.B. die Kreuzkorrelation mit beschränktem Suchraum zu betrachten.

Interessenskonflikt Ich bzw. wir erklären keine Interessenskonflikte.

Finanzierung Wir bedanken uns für die finanzielle Unterstützung durch das Land Salzburg innerhalb des WISS 2025 Projekt Virtual Sleep Lab (VSL-Lab) (20102-F2002176-FÜR).

Literatur

Pearson, R. (2005). Mining Imperfect Data. In *SIAM*, 250, https://doi.org/10.1137/1.9780898717884.

Lewis, J. P. (1995). "Fast Normalized Cross-Correlation." In *Industrial Light & Magic,* http://scribblethink.org/Work/nvisionInterface/nip.pdf.

Lyon, D. (2010). The Discrete Fourier Transform, Part 6: Cross-Correlation. In *The Journal of Object Technology*, 9(2), 17. https://doi.org/10.5381/jot.2010.9.2.c2

Wescott, Tim. (2018). Sampling: What Nyquist Didn't Say, and What to Do About It. In *Wescott Design Services*, https://www.wescottdesign.com/articles/Sampling/sampling.pdf.

Lam, S. K. (2015). Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC,* (pp. 1–6).

Schranz, C. (2020). Deterministic Time-Series Joins for Asynchronous High-Throughput Data Streams, In *ETFA*, pp. 1031-1034, https://doi.org/10.1109/ETFA46521.2020.9211958.