Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Schweizer Paraplegiker Forschung
Schweizer Paraplegiker Zentrum
Dr. Diego Paez-Granados

Semester Thesis

Biosignal Synchronization Across Devices In Robotics Application

Autumn Term 2024

Supervised by:
Dr. Diego Paez-Granados

Author:
Pascal Vogel

Declaration of Originality

I hereby declare that the written work I have submitted entitled

Biosignal Synchronization Across Devices In Robotics Application

is original work which I alone have authored and which is written in my own words.[1]

Author(s)

Pascal	Vogel

Supervisor(s)

Diego	Paez-Granados

Supervising lecturer

Robert	Riener

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Stans, 16.12.2024
Place and date


Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Intellectual Property Agreement

The student acted under the supervision of Dr. Paez-Granados and contributed to research of his group. Research results of students outside the scope of an employment contract with ETH Zurich belong to the students themselves. The results of the student within the present thesis shall be exploited by ETH Zurich, possibly together with results of other contributors in the same field. To facilitate and to enable a common exploitation of all combined research results, the student hereby assigns his rights to the research results to ETH Zurich. In exchange, the student shall be treated like an employee of ETH Zurich with respect to any income generated due to the research results.
This agreement regulates the rights to the created research results.

1. Intellectual Property Rights

1. The student assigns his/her rights to the research results, including inventions and works protected by copyright, but not including his moral rights ("Urheberpersönlichkeitsrechte"), to ETH Zurich. Herewith, he cedes, in particular, all rights for commercial exploitations of research results to ETH Zurich. He is doing this voluntarily and with full awareness, in order to facilitate the commercial exploitation of the created Research Results. The student's moral rights ("Urheberpersönlichkeitsrechte") shall not be affected by this assignment.
2. In exchange, the student will be compensated by ETH Zurich in the case of income through the commercial exploitation of research results. Compensation will be made as if the student was an employee of ETH Zurich and according to the guidelines "Richtlinien für die wirtschaftliche Verwertung von Forschungsergebnissen der ETH Zürich".
3. The student agrees to keep all research results confidential. This obligation to confidentiality shall persist until he or she is informed by ETH Zurich that the intellectual property rights to the research results have been protected through patent applications or other adequate measures or that no protection is sought, but not longer than 12 months after the collaborator has signed this agreement.
4. If a patent application is filed for an invention based on the research results, the student will duly provide all necessary signatures. He/she also agrees to be available whenever his aid is necessary in the course of the patent application process, e.g. to respond to questions of patent examiners or the like.

2. Settlement of Disagreements

Should disagreements arise out between the parties, the parties will make an effort to settle them between them in good faith. In case of failure of these agreements, Swiss Law shall be applied and the Courts of Zurich shall have exclusive jurisdiction.

Stans, 16.12.2024
Place and date

P. Vogel
Signature

Contents

Preface

This project aims to address the challenge of synchronizing biosignal data across multiple devices in robotics applications. Synchronization is essential for ensuring reliable classification of biosignal data, which plays an important role in analyzing and processing collected data. By developing and evaluating a synchronization method with the potential for robustness, this work contributes to ensuring the accuracy and reliability of classification tasks.

I express my gratitude to my supervisor, Dr. Diego Paez, for his invaluable guidance, insight, and encouragement throughout this project.

Through this project, I have gained a deeper understanding of the challenges of signal synchronization across devices.

Abstract

Clock drift in biosignal sensors introduces significant challenges for data synchronization, resulting in misaligned data streams that decrease the reliability of downstream classification tasks, such as Activities of Daily Living (ADL) recognition. This semester thesis addresses the synchronization problem by developing an event-based pipeline capable of aligning multimodal signals, including accelerometer and pressure mat data.

The pipeline employs an event based signal synchronization, using Local Outlier Factor (LOF) for outlier detection, event clustering, and Dynamic Time Warping (DTW) for event matching, followed by signal alignment. Its modular design enables easy extension to additional sensors and modification of individual steps. An artificially drifted test dataset, derived from SCAI-SENSEI V2 recordings, was used to evaluate the pipeline's performance.

The results on the test dataset show that the pipeline achieves a mean absolute error (MAE) below 0.2 seconds for accelerometer-to-accelerometer synchronization and below 0.3 seconds for pressure-mat-to-accelerometer synchronization, for drifts up to 0.75 and 1 second, respectively. However, the pipeline's performance decreases at higher drift levels due to an increased likelihood of incorrect or missed synchronizations. These findings highlight both the potential and current limitations of the pipeline, emphasizing the need for further optimization and validation on larger datasets.

This work represents a step forward toward fully automated activity recognition, facilitating rehabilitation and health monitoring.

Chapter 1

Introduction

1.1 Background

Clocks in biosignal sensors are prone to drift, which means that the time they measure deviates from the actual time. This deviation, commonly referred to as clock drift, occurs due to factors such as temperature variations, which affect the quartz crystal oscillators that regulate the clock rate in many devices [1]. This clock drift accumulates over time and propagates to the logged data, leading to misaligned signals across devices. A visualization of clock drift is shown in Figure 1.1.

This project aims for the synchronization of biosignals collected from multiple accelerometers and a pressure mat data. The collected data is provided as input to an Activities of Daily Living (ADL) classifier [2]. Processing the data in this way, allows for a better monitoring of the activities of patients e. g. in rehab or in nursing homes, allowing for valuable insights and more customized treatment plans in the future. However, clock drift in sensors causes misalignment in the data streams, leading to inaccurate classification results. This occurs because the ADL classifier processes multiple input signals using a sliding window of varying size, depending on the specific activity of daily living (ADL) being detected. When the signals do not sufficiently overlap, the classifier fails to detect the activity. Resolving this synchronization issue is critical to ensuring the reliable performance of the ADL classifier.

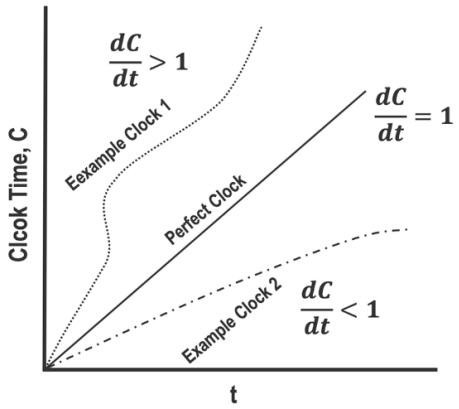

Figure 1.1: Graphical representation of clock drift, from [3]

1.2 Data

The dataset used in this project is the SCAI-SENSEI V2 dataset, which was provided by the Spinal Cord Injury & Artificial Intelligence Lab (SCAI-Lab). It comprises recordings from 20 participants, each equipped with multiple sensors to monitor various biosignals during controlled activities. While the dataset includes a variety of sensors, this project focuses on the following four:

- A wrist-worn biosignal sensor (Corsano),
- An in-ear sensor (Cosinuss),
- A breast patch (Vivalink), and
- A pressure mat (Sensomative) placed on a wheelchair.

The recordings span approximately 45 minutes and capture participants performing ADL both inside and outside a wheelchair. Additionally, at the start and end of each recording, participants performed a standardized sequence involving:

1. Sitting down on a wheelchair,
2. Waiting for 5 seconds,
3. Standing up,
4. Waiting for another 5 seconds,
5. Repeating this cycle a total of three times.

1.3 Objective

The objective of this project is to develop a synchronization method capable of aligning data from multimodal sensors. Specifically, the focus is on synchronizing accelerometer data across multiple devices, as well as integrating data from the pressure mat, which measures a different physical modality.
The proposed algorithm should be easy to use, easily adjustable, and extendable to accommodate additional sensors in the future. The targeted synchronization error is 200 milliseconds or less. This threshold is derived from the requirements of the ADL classifier, which uses a sliding window with a minimum size of 1 second and requires an overlap of at least 80% for reliable operation.

Chapter 2

Methodology

In this project, an event-based synchronization approach was selected due to the limitations of alternative methods. A hardware-based synchronization approach, which involves sending a synchronization signal directly to the sensor, is not supported by the sensors used in this study. Further, using a correlation-based synchronization methods was dropped due to concerns of suitability because of the significant morphological differences between the multimodal signals.
In an event-based approach three main steps must be done:

1. Identify events in both signals
2. Find matching events from both signals
3. Use matched events to synchronize signals

2.1 Assumptions

Some assumptions were made to guide the implementation of the algorithm. Namely:

- Observable Events Across Sensors:

  It is assumed that events can be observed across multiple sensors. Each sensor must share events, at least transcendentally, with all other sensors.

- Linear Drift:

  The drift between the clocks of different sensors is assumed to be well approximated by a linear model between, in time ranges of up to a day. This assumption is important because if the drift were non-linear, relying on only occasional synchronization points would not be enough to maintain accurate alignment.

2.2 Code

The complete implementation of the synchronization pipeline, along with the dataset preprocessing and evaluation scripts, can be found in the associated Git repository: `https://github.com/SCAI-Lab/sensor_synchronization`.
This chapter provides an overview of the code developed for this project, including its structure and key functionalities. The explanation is supplemented with pseudocode to illustrate the logic of important functions. The code is implemented in Python 3.12 [4] and is designed to be modular, making it easy to extend and adapt for future applications. The flow of the program is described by the flow diagram depicted in Figure 2.1.

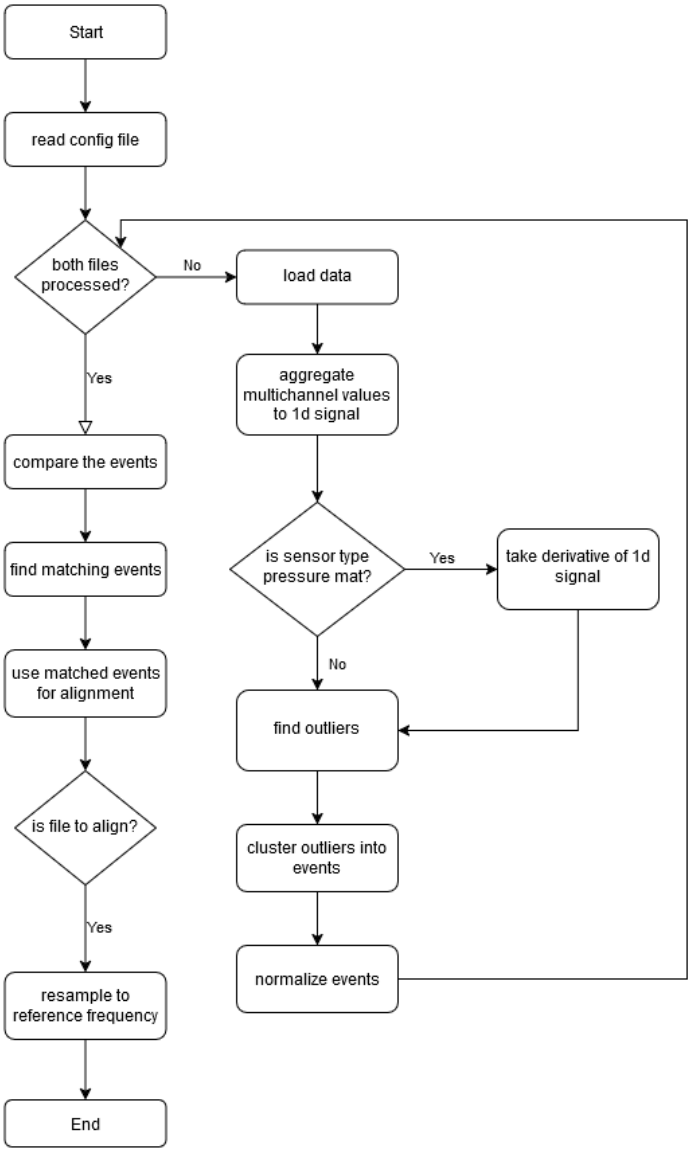

Figure 2.1: Program Flow Diagram

2.2.1 Structure

The algorithm is located in the `synchronization_pipeline` folder of the git folder. Its structure is organized as follows:

```
synchronization_pipeline/
 main.py
 config.yaml
 pipeline_steps/
    event_detection.py
    event_information.py
    outlier_detection.py
    signal_alignment.py
    event_comparison.py
 utils/
     data_utils.py
     visualization.py
```

The `main.py` script serves as the entry point of the synchronization pipeline, managing function execution and directing the overall workflow. Its overall structure is well described by the control flow diagram presented in Figure 2.1. The key steps are modularized into the respective scripts, located in the `event_detection` and `utils` folders.
The program starts by reading the configuration file (`config.yaml`, example in the Appendix A.3), which contains tunable parameters for the pipeline and file paths for the sensor data. There are two signals: a reference signal, and the signal to be aligned. For each signal, the data is loaded, and its multichannel values (e.g. x, y, z components) are aggregated into a single one-dimensional (1D) signal. This aggregation is achieved by summing the absolute values of the individual channels. This data simplification is done because the outlier detection method employed in the pipeline does not support multichannel inputs.
For pressure mat sensors, an additional preprocessing step is applied: the derivative of the 1D signal is computed. This adjustment is helpful because the raw signals from pressure mats have a different morphology compared to accelerometer signals. In our case, working with the derivative simplifies the detection of outliers and event matching. The implementation of these functions, including loading data, data aggregation, and derivative computation, is done in the `data_utils.py` module.
For outlier detection, the Local Outlier Factor (LOF) algorithm from the Scikit-learn library [5] was used. LOF identifies outliers by comparing the density of a point to the densities of its k-nearest neighbors. While the algorithm offers several tunable parameters, only the `n_neighbors` parameter is explicitly configured in this implementation. The value of `n_neighbors` is sensor-specific and must be defined in the configuration file. LOF was selected over the two alternatives, Nixtla's TimeGPT Anomaly Detection model [6] and Scikit-learn's Isolation Forest [5], as it demonstrated better performance on a small test dataset (see Table A.3 for detection method evaluation results). The corresponding code is implemented in `outlier_detection.py`.
After identifying outliers, these are clustered into events based on their timestamps. The clustering logic is implemented in the `event_detection.py` module. Events are formed by grouping outliers that meet specific temporal and density-based conditions. The pseudocode for the clustering process is described in the following:

```
Pseudocode: Cluster Events
Input: Signal with annotated events
Output: Events found in signal
```

```
Iterate over all outliers in signal
    Check if points in the next 'min_time_event seconds'
    are at least 'min_outlier_fraction_event' outliers
    Create event
    Iterate over points in event
        While 'min_outlier_fraction_event' condition holds
            Expand event
    Add event to list
    Move iteration to next outlier not being part of an event
return list of events
```

The event data is then normalized using the data of the last '`normalization_window_duration`' seconds before the event. This normalization step ensures that the data is comparable across events, as the values may vary significantly in magnitude. The relevant code for this operation is implemented in `event_information.py`. Additionally, key information such as the indices of minimum and maximum values are also extracted and stored.
After identifying and describing the events, they must be compared to determine potential matches. This comparison is performed in the `event_comparison.py` module. Events are only considered for comparison if they are temporally close. The Dynamic Time Warping (DTW) [7] distance is calculated and used as a similarity measure. For the implementation, the FastDTW [8] package was used to efficiently compute the DTW distances. The pseudocode for the comparison process is provided below:

```
Pseudocode: Compare Events
Input: 2 Arrays with Normalized Events
Output: DTW Distance of temporally close events
Iterate over events in first array
    iterate over events in second array
        if events are less than 'max_time_gap_events' seconds apart
            calculate dtw distance
    save best DTW distance match
return DTW distances of compared events
```

In the `signal_alignment.py` module, matching events are identified based on the event comparison data. Events are classified as matches, if their DTW distance is within the specified `dtw_distance_threshold`. This parameter is determined empirically and is not necessarily the same for different sensor type combinations.

```
Pseudocode: Align Signals
Input: Signals and Compared Events
Output: Aligned Signals
Iterate over found matches
    if DTW distance < 'dtw_distance_threshold'
        if signals are both accelerometer
            align lowest points from both events
        if signals are accelerometer and pressure mat
            if min value before max in accelerometer event
                align lowest point from accelerometer signal
                with highest point from pressure mat signal
            if max value before min value in accelerometer event
                align highest point from accelerometer signal
                with lowest point from pressure mat signal
```

The alignment process involves two cases. For the initial synchronization, the entire signal is shifted to align with the reference. For subsequent synchronizations, the

signal is stretched from the previous synchronization point onwards. In real world application, one would always use stretching, as it is assumed that at the start of data collection, devices are in synch, thus giving the first synchronization point for free. Further, note that the operations apply exclusively to the timestamps of the signal points, while the corresponding 1D signal values remain unchanged. The alignments can be described as follows:

1. The **shifting** of the signal:

$$P' = P + (S_r - S_a) \tag{2.1}$$

2. The **stretching** of the signal:

$$P' = S_a + (P - S_a)\frac{S_{r_{new}} - S_{r_{old}}}{S_{a_{new}} - S_{a_{old}}} \tag{2.2}$$

   Where:

   - P': Adjusted point in the signal to align
   - P: Original point in the signal to align
   - S_r: First synchronization point in the reference signal
   - S_a: First synchronization point in the signal to align
   - $S_{r_{new}}$: New synchronization point in the reference signal
   - $S_{r_{old}}$: Previous synchronization point in the reference signal
   - $S_{a_{new}}$: New synchronization point in the signal to align
   - $S_{a_{old}}$: Previous synchronization point in the signal to align

Since the timestamps were adjusted in the aligned signal, they are now irregular. In the last step, the aligned file is sampled to the reference signal's frequency. Optionally, the results are plotted if `plotting` is set to 'true'.
A visual example depicting the separate steps is added in the Appendix A.1.

Chapter 3

Results

3.1 Test Dataset Creation

To evaluate the accuracy of the synchronization program, a test dataset was created and processed through the pipeline. This dataset is derived from the recordings described in Section 1.2, selecting data from five users. The preparation process is outlined as follows:

3.1.1 Snippet Selection

Approximately 90-second-long snippets were extracted containing the synchronization procedures at the start and end of the recordings. Not all snippets are available due to some of the sensors (*Cosinuss*, *Corsano*, *Vivalink*, *Sensomative*) not being operational during the selected time range.

3.1.2 Ground Truth Creation

A *ground truth dataset* was generated by aligning the signals to a reference sensor by hand:

1. The *reference sensor* was chosen as the one with the highest sampling frequency for each user.
2. For alignment:
   (a) Two pairs of corresponding points between the signal to be aligned and the reference signal were manually identified.
   (b) The signal to align was shifted such that the first pair of points matched.
   (c) The signal was then stretched or squeezed to ensure the second pair also aligned.

The timestamps for the ground truth alignment points can be calculated using equations (2.1) and (2.2).

3.1.3 Creation of Drifted Dataset

To simulate accumulated clock drift, an *artificially drifted dataset* was created from the ground truth dataset:

- Eight drifted files were generated by shifting the ground truth by 0.25 seconds up to 2 seconds in increments of 0.25 seconds.

- Signal stretching by a realistic factor was omitted, as the short duration of the snippets (90 seconds) would not have led to significant drift accumulation.

3.2 Evaluation Methodology

The accuracy of the program was assessed by comparing the synchronized timestamps with the ground truth at the two alignment points used during the creation of the ground truth dataset. These points were chosen because their correspondence to the reference sensor was manually determined, ensuring a precise ground truth alignment.
The synchronization program was evaluated on its ability to align the entire signal to the reference based on these two key points. The error was quantified as the *absolute difference* between the synchronized timestamps and the ground truth timestamps at these positions (Figure 3.1).
Multiple parameter configurations were evaluated to identify the optimized settings. The specific configurations used during the tests are as follows:

- `outlier_neighbors_cosinuss`: 400
- `outlier_neighbors_corsano`: 200
- `outlier_neighbors_vivalink`: 50
- `outlier_neighbors_sensomative`: 20
- `min_time_event`: 0.5
- `min_outlier_fraction_event`: 0.5
- `max_time_gap_events`: 2.0
- `dtw_distance_threshold_accelerator`: 150
- `dtw_distance_threshold_sensomative`: 150
- `normalization_window_duration`: 10

Figure 3.1: The error is calculated by taking the the absolute difference between synchronization points in the reference signal (A*, B*) and the aligned signal (A, B).

3.3 Error Quantification

For the prepared datasets, all possible combinations of sensors were tested to evaluate the synchronization accuracy. The reference sensor was always chosen to be the one with the higher frequency, so no permutations were done. For example, if data was available for three sensors X, Y, and Z (ordered by descending frequency), the following synchronizations were performed:

- Synchronizing Y to X,
- Synchronizing Z to X, and
- Synchronizing Z to Y (to represent cases where Y and Z registered an event but X did not).

Further, the three following metrics were calculated:

1. The **mean absolute error (MAE)**:

$$\frac{1}{2N}\sum_{1}^{N}(|A_i^* - A_i| + |B_i^* - B_i|) \tag{3.1}$$

2. The **standard deviation** of the errors:

$$\sqrt{\frac{1}{2N}\sum_{1}^{N}((|A_i^* - A_i| - \mu)^2 + (|B_i^* - B_i| - \mu)^2)} \tag{3.2}$$

3. The **root mean squared error (RMSE)**:

$$\sqrt{\frac{1}{2N}\sum_{1}^{N}((A_i^* - A_i)^2 + (B_i^* - B_i)^2)} \tag{3.3}$$

3.4 Discussion

The results are divided into two categories: accelerometer signals synchronized to other accelerometer signals, and pressure mat signals synchronized to accelerometer signals. A total of ten signals were evaluated in the first category, and 20 signals in the second.
The results are summarized in Tables 3.1 and 3.2, and corresponding plots are presented in Appendix A.2. For the purposes of this analysis, 'drift' refers to the temporal shift of the entire input signal relative to the ground truth.
A linear relationship between the error and the drift is observed up to a drift of 1.5 seconds, where the higher the drift, the higher the mean absolute error (MAE). Synchronization outcomes can be categorized into three types: correct synchronization, incorrect synchronization, and missed synchronization. The observed linear relationship between drift and error can be explained as follows: as drift increases, errors associated with missed synchronizations grow larger, while errors from correct and incorrect synchronizations remain constant. Further, with higher drift, the likelihood of (worse) incorrect matches increases, leading to a noticeable drop in accuracy beyond 1.5 seconds. At 2 seconds, this effect becomes even more clear, where new wrong matches results in a visible jump in the error quantifications. The jump is also observed in the RMSE value, suggesting an increase in outliers.
For the pressure mat signals, the same pattern is observed. The error for small drift is already higher compared to the accelerometer case's one. This is due to more incorrect and missed synchronizations at small drifts already.
The above explanations were mainly deduced from the plot of all errors provided in the Appendix A.8.

Table 3.1: Error Quantification Accelerometer - Accelerometer

Drift (s)	**MAE (s)**	**Standard Deviation (s)**	**RMSE (s)**
0.25	0.156	0.224	0.273
0.5	0.181	0.246	0.306
0.75	0.206	0.287	0.353
1	0.231	0.339	0.410
1.25	0.256	0.399	0.474
1.5	0.397	0.607	0.725
1.75	0.403	0.649	0.764
2	1.085	1.098	1.544

Table 3.2: Error Quantification Accelerometer - Pressure Mat

Drift (s)	**MAE (s)**	**Standard Deviation (s)**	**RMSE (s)**
0.25	0.231	0.409	0.470
0.5	0.256	0.417	0.489
0.75	0.281	0.437	0.520
1	0.307	0.469	0.561
1.25	0.332	0.510	0.609
1.5	0.359	0.558	0.663
1.75	0.596	0.711	0.927
2	1.044	0.957	1.416

Chapter 4

Summary

4.1 Conclusion

This project aimed to develop and evaluate a robust method for synchronizing biosignal data collected from multimodal sensors, specifically accelerometers and pressure mats. The proposed pipeline employs an event-based synchronization approach, which was selected due to its adaptability to multimodal signals. The synchronization pipeline is designed to be extendable, enabling the integration of additional sensors, and easily modifiable, allowing for the customization or adaptation of individual computation steps.
On the test dataset, the algorithm achieved a mean absolute error (MAE) below 0.2 seconds for accelerometer-to-accelerometer synchronization with drifts up to 0.75 seconds in the input signal. For pressure-mat-to-accelerometer synchronization, the MAE remained below 0.3 seconds for drifts up to 1 second. The higher the drift, the larger the synchronization errors due to a higher likelihood of incorrect matches. Additionally, pressure mat signals demonstrated higher error rates compared to accelerometer signals, even at smaller drifts, showing the difficulties of synchronizing signals with distinct morphologies. The results indicate that the synchronization pipeline has potential but is not yet stable enough for deployment in real-world applications. The desired target error of 200ms or less was only achieved in the test cases of accelerometer-to-accelerometer synchronization with less than 0.75 seconds drift in the input signal.

4.2 Discussion of Future Work

The current pipeline offers significant potential for enhancement, and several areas for improvement are proposed:

- **Optimize Parameters:** The current parameter space has not been fully explored. Conducting additional tests with different parameter configurations could lead to improved performance and more robust results.
- **Explore Alternative Methods:** Each step in the pipeline can potentially benefit from alternative methods. For instance, different algorithms for outlier detection, event clustering or event matching may yield better accuracy. Directly detecting events could also be tried.
- **Synchronize After Multiple Matches:** Currently, the pipeline synchronizes the signals as soon as two matching events are found. Introducing a mechanism that requires multiple pairs of matching events to agree on the drift between the signals could reduce the likelihood of incorrect matches.

- **Evaluate on Larger Datasets:** The pipeline has been evaluated on a relatively small dataset. Testing on larger datasets with greater diversity could help uncover issues in the pipeline and provide insight for further refinement.

Bibliography

[1] R. S. Strout, "The temperature coefficient of quartz crystal oscillators," *Phys. Rev.*, vol. 32, pp. 829–831, Nov 1928.

[2] M. Ejtehadi, S. Amrein, I. E. Hoogland, R. Riener, and D. Paez-Granados, "Learning activities of daily living from unobtrusive multimodal wearables: Towards monitoring outpatient rehabilitation," in *2023 International Conference on Rehabilitation Robotics (ICORR)*, 2023, pp. 1–6.

[3] N. Schütz, A. Botros, M. Single, A. C. Naef, P. Buluschek, and T. Nef, "Deep canonical correlation alignment for sensor signals," 2021.

[4] Python Software Foundation, *Python 3.12*, 2023, version 3.12.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] M. Menchero, *nixtlar: A Software Development Kit for 'Nixtla"s 'TimeGPT'*, 2024, r package version 0.6.2, https://docs.nixtla.io/, https://github.com/Nixtla/nixtlar.

[7] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.

[8] S. Salvador and P. K. Chan, "Fastdtw: Toward accurate dynamic time warping in linear time and space," 2004.

Appendix A

Supplementary Material

A.1 Visualization of the Pipeline

The following five figures depict the visualization of the pipeline, illustrating the different stages of the process.

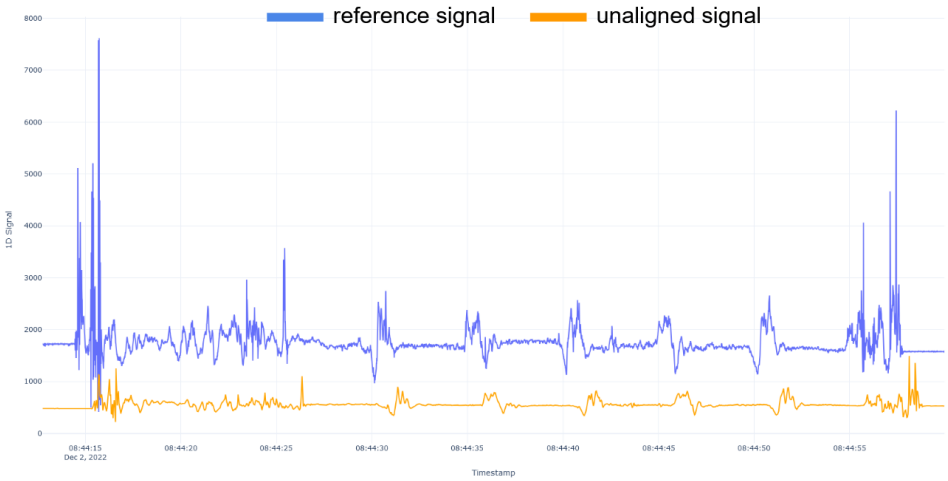

Figure A.1: Input signals.

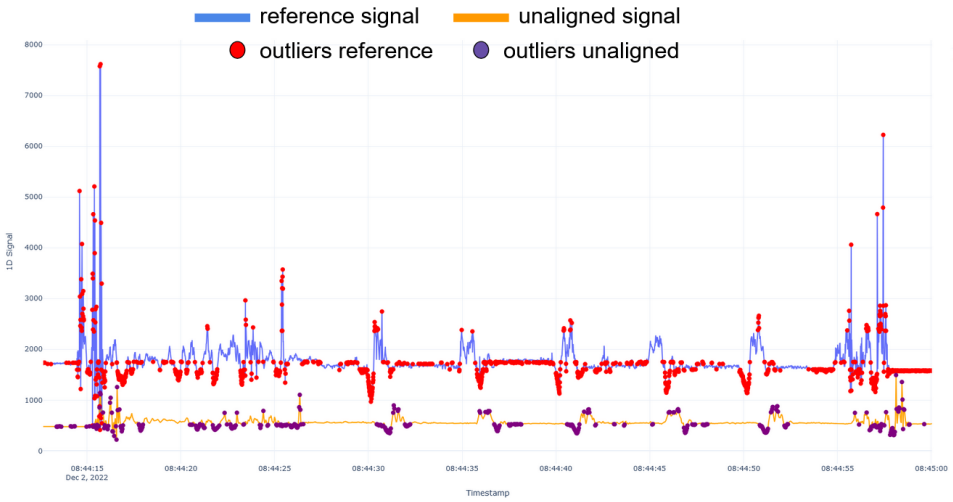

Figure A.2: Outliers detected.

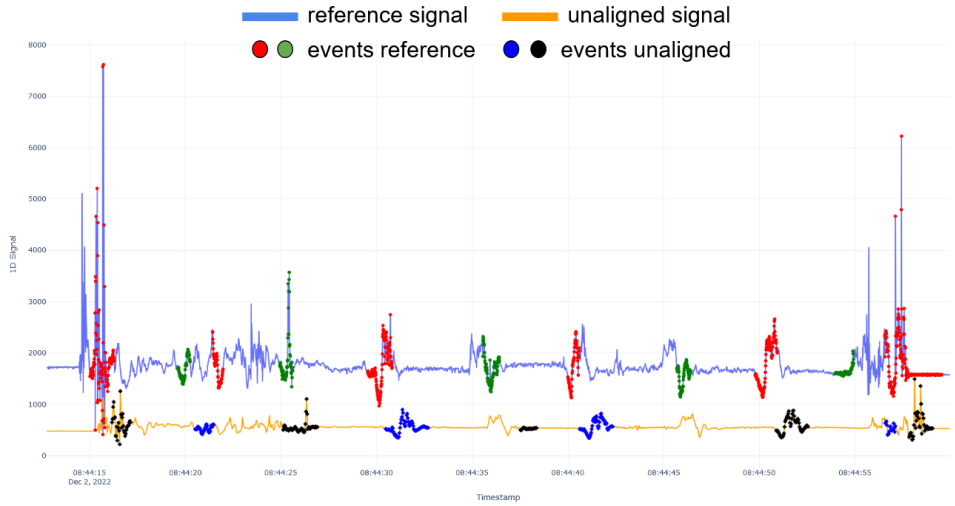

Figure A.3: Events clustered.

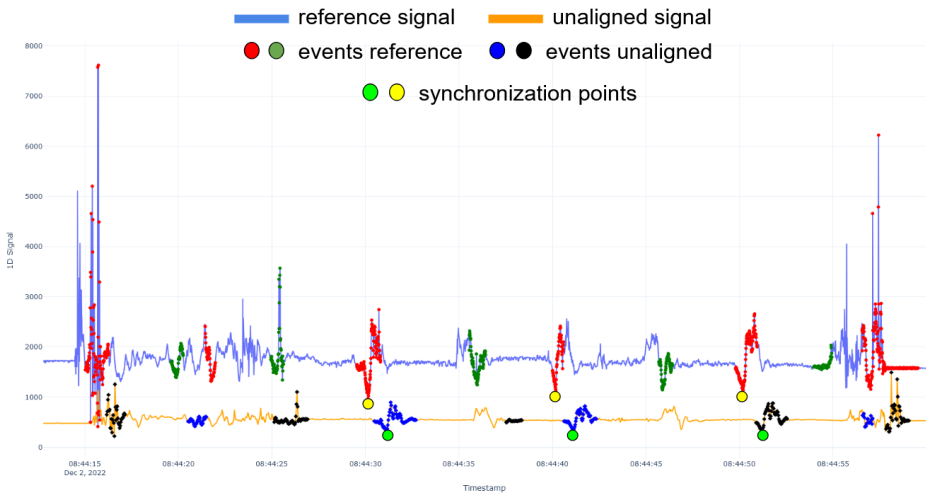

Figure A.4: Matched synchronization points.

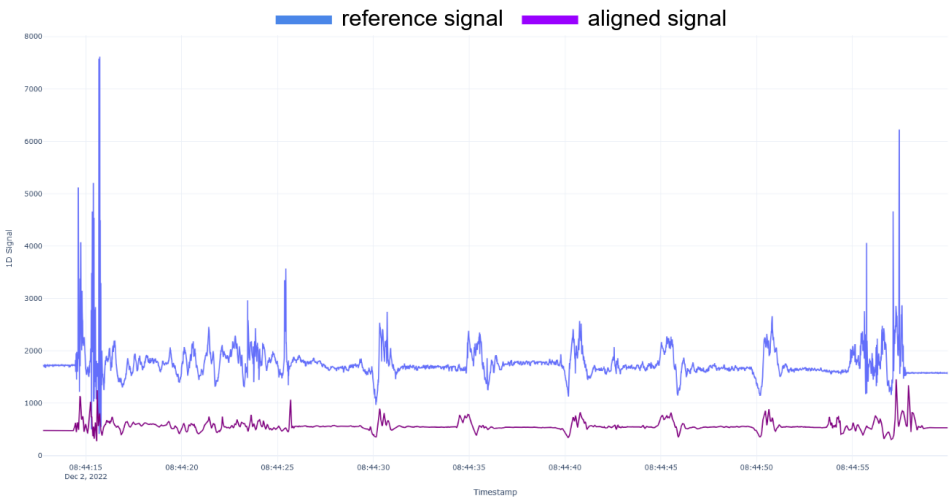

Figure A.5: Aligned signals.

A.2 Error Quantification Plots

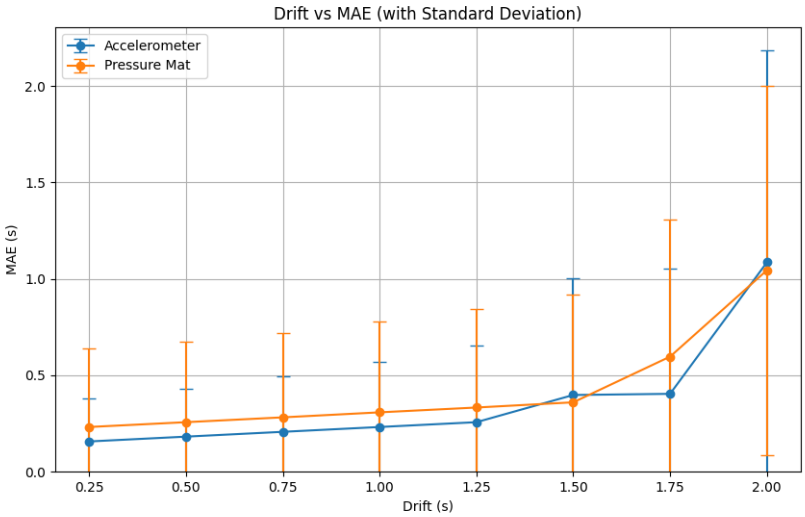

Figure A.6: MAE and Standard Deviation plotted against drift of initial file.

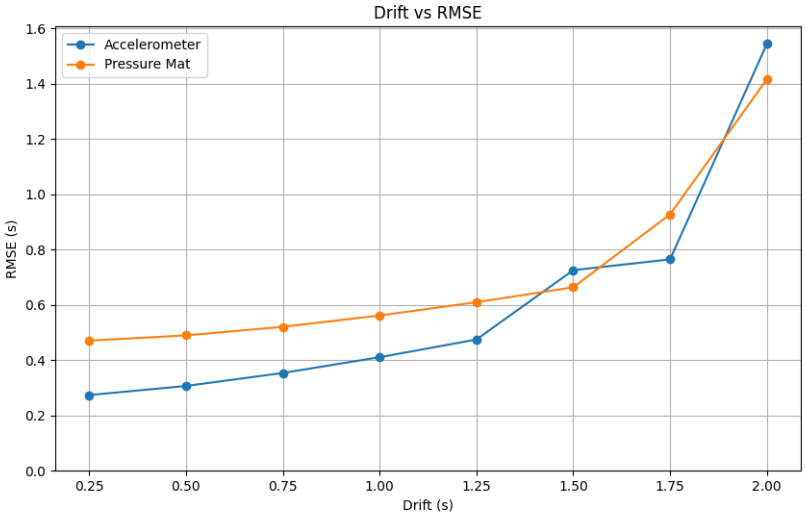

Figure A.7: RMSE plotted against drift of initial file.

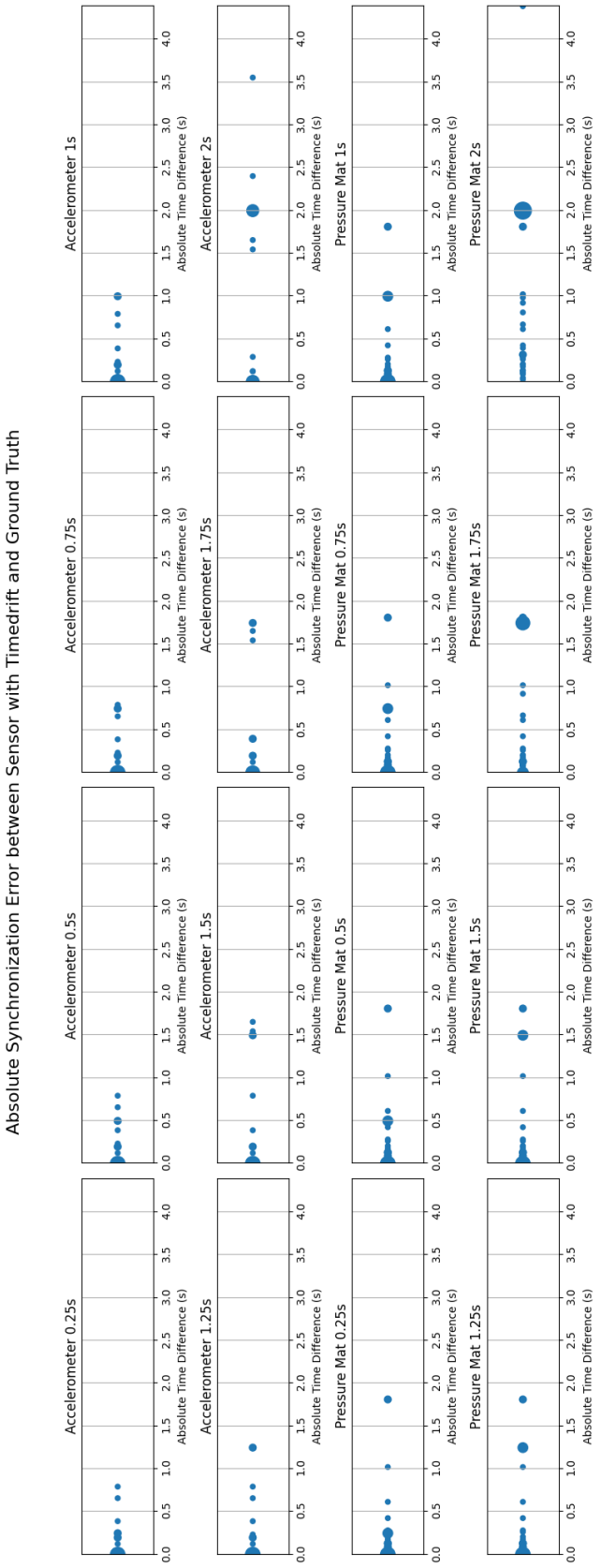

Figure A.8: Plot showing all synchronization errors from the test run, divided into two categories: accelerometer synchronized to accelerometer, and pressure mat synchronized to accelerometer. "Drift" (the time value in the plot titles) represents the time shift of the artificial input data and the ground truth. Each error is represented by a single dot, with two dots plotted per signal.

A.3 Various

Listing A.1: Example config.yaml File

```
# config.yaml

# Dynamic settings

# Path to the file containing the reference signal
reference_file: 'path_to_reference_file'

# Path to the file conatining the signal to be aligned
file_to_align: 'path_to_file_to_align'

# Supported sensors: cosinuss, corsano, vivalink, sensomative
# Name of sensor of the reference file
sensor_name_reference: 'cosinuss'

# Name of sensor of the file to align
sensor_name_align: 'corsano'

# Plotting
# Plot all the signals with the detected events annotated
plotting: True
# Scaling factor for the plotting first file
scaling_factor_reference: 500
# Scaling factor for the second file
scaling_factor_align: 1.0

# Static settings

# LOF number_neighbors parameter
outlier_neighbors_cosinuss: 400
outlier_neighbors_corsano: 200
outlier_neighbors_vivalink: 50
outlier_neighbors_sensomative: 20

# Diverse parameters
# minimum time (in s) an event must last
min_time_event: 0.5

# minimum fraction of outliers in an event
min_outlier_fraction_event: 0.5

# maximum time gap (in s) between two events to be considered
    possible matches
max_time_gap_events: 3

# Threshold for the DTW distance between two events (acc-acc)
    to be considered a match
dtw_distance_threshold_accelerator: 150

# Threshold for the DTW distance between two events
    (acc-sensomative) to be considered a match
dtw_distance_threshold_sensomative: 150

# Time over which data is normalized for DTW comparison
normalization_window_duration: 10
```

```
 # Save the aligned file
save_output_files: True

# Folder where the output files will be saved
output_folder_path: './output'
```

Sensor	Best F1 score **LOF**	Best F1 score **TimeGPT**	Best F1 score **IF**
corsano	0.838	0.825	0.837
cosinuss	0.845	0.671	0.840
vivalink	0.800	0.583	0.738
sensomative	0.563	0.323	0.391

$$F_1 = \frac{2TP}{2TP + FN + FP}$$

TP: true positives FP: false positives FN: false negatives

Figure A.9: F1 scores of outlier detection methods Scikit's [5] Local Outlier Factor (LOF) and Isolation Forest (IF), and Nixtla's [6] TimeGPT Outlier Detection on a single test data set containing the synchronization procedure described in Section 1.2

A.4 AI use Statement

I hereby declare to have used AI, namely ChatGPT and Deepl, to aid in composing this semester thesis.
ChatGPT helped structuring the text and also providing ideas for alternative sentence formulations. It also helped in writing code faster. Deepl was used for translations as English is not my native language.