Justus Liebig University Giessen

Faculty 06

Department of Sports Science

Data Analysis Final Project Report

Deadline:

01.10.2024

Group Name:

Amazing_15p

Members of the group:

Yeganeh Mohammadi	[redacted]	yeganeh.mohammadi@sport.uni-giessen.de
Sabrina Miller	[redacted]	sabrina.miller@sport.uni-giessen.de
Farzaneh Vafaeinezhad	[redacted]	farzaneh.vafaeinezhad@sport.uni-giessen.de

1.Introduction

In designing our data analysis GUI, our primary goal was to make data exploration intuitive and insightful. This introduction explains the rationale behind our design choices.

We focused on creating a user-friendly interface that simplifies data analysis. Key features include displaying data and preparing it for analysis, providing users with a clear starting point. We also allowed users to customize pre-processing settings, ensuring flexibility for different datasets.

A notable feature is the comparison tool for the three conditions (A, B, C), which enables users to select specific points, visualize comparisons, and gain clearer insights. Our given data involved 20 subjects per group, with each subject performing 30 trials at a rate of 250 Hz, resulting in approximately 200 valid data points per trial. The GUI is built to efficiently handle datasets of this size, ensuring a smooth analysis experience.

Overall, every decision in developing the GUI was aimed at making data exploration easy, accessible, and insightful, enhancing the user experience and enabling meaningful analysis.

2.About the Data

The data presented in this project illustrates knee angle throughout a complete gait cycle, offering a detailed understanding of how the knee articulates and functions during walking. This dataset highlights the intricate dynamics of knee flexion and extension across various phases of the gait cycle, revealing key insights into the relationship between the knee joint and the locomotion process.

During walking, each part of the body moves in a specific pattern within the gait cycle. Although factors like walking surface, footwear, and incline can introduce slight variations, the overall movement pattern remains consistent under different conditions.

By analyzing the knee flexion angle during walking in Figure1, we gain insight into how this joint behaves under various circumstances. In the dataset, categorized into conditions A, B, and C, subtle differences in the range of motion and the timing of peak knee flexion are observed.

Figure 1. Knee Flexion Angle Throughout the Gait Cycle

Each phase of the gait cycle, represented in the plot, corresponds to a specific part of walking, identifiable by peaks or changes in slope. The phases for condition A are detailed in Figure 2.

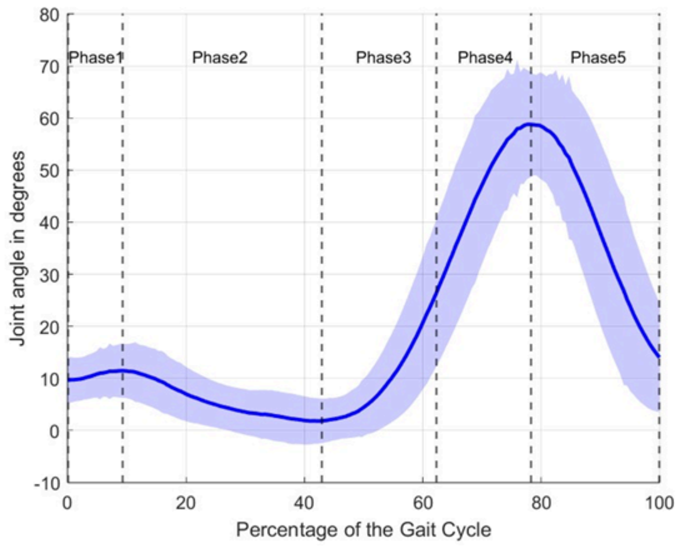


Figure 2. Knee Movement Throughout the Gait Cycle

Heel Strike (Initial Contact): At the moment of heel strike, the knee is ideally in a slightly flexed position. However, individuals may exhibit a range of knee postures at this point. In condition A, the knee flexion is approximately 10 degrees.

- Phase 1: After initial contact, the knee flexes further to about 12 degrees, corresponding to the moment of maximum weight-bearing load on the joint.
- Phase 2: Following the first peak in knee flexion, the knee extends to nearly full extension as the body moves smoothly over the stance limb.
- Phase 3: As the heel rises, the knee enters a secondary phase of flexion, coinciding with the propulsion phase of the gait cycle. The knee then flexes further in preparation for the swing phase (pre-swing).
- Phase 4: During toe-off, the knee flexion reaches approximately 27 degrees. This, combined with ankle dorsiflexion, allows the toe to clear the ground. The knee continues to flex to a maximum of 58 degrees.
- Final Phase: The knee then extends, preparing for the next heel strike.

The values mentioned above vary slightly across conditions A, B, and C, but the overall range remains similar. Figure 3 provides a general visualization of knee movement during the gait cycle.

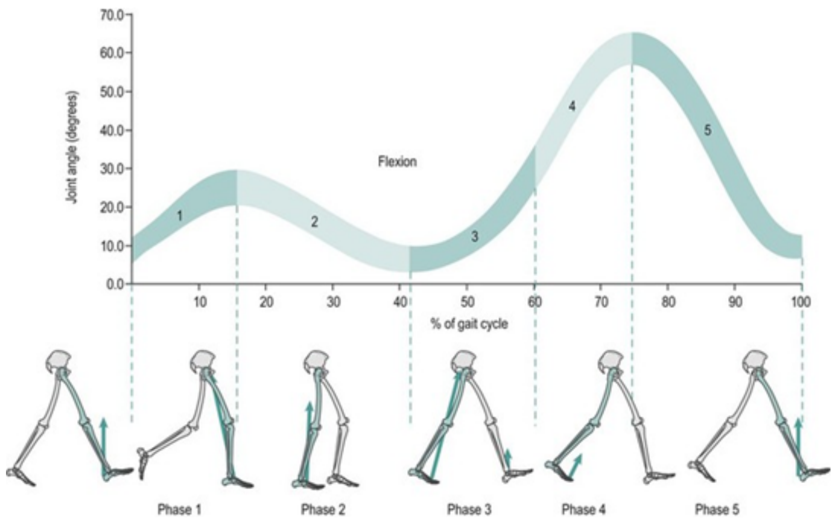


Figure 3. Leg Movement According to Each Phase of the Gait Cycle(1)

3.Features in the Graphical User Interface (GUI)

3.1 Setting Tab

3.1.1 Outliers Detection

Nearly every technique has been examined to detect outliers in our dataset. From these methods, we have selected the most convenient methods for users to choose from. In terms of outlier parameters, we have extensively tested them and established optimal default values. Additionally, a comprehensive explanation of these methods will be provide:

Slope method

The "*slopeOutlier*" function is designed to detect outliers based on the rate of change between consecutive data points within a sliding window. This method identifies outliers by focusing on regions of the data where the slope—the rate of change—is unusually steep.

The function begins by initializing the input data "*matrix_input*" as "*outlierfree_data_slope*". It processes the data row by row, applying a sliding window approach. For each row, the function iterates over a specified range defined by "*start_end_points*", sliding a window of size *"window_size"* across the data.
Within each window, the function calculates the slope by using the *diff* function to compute the differences between consecutive data points. The maximum slope within the window is then determined using the max function, which identifies the steepest rate of change. This maximum slope is compared to a predefined threshold to assess whether the window contains outliers.
If the maximum slope within the window exceeds the specified threshold, the entire window is marked as an outlier. Consequently, all data points within this window are replaced with NaN. This step ensures that the identified outliers are effectively excluded from the dataset.
After processing each window, the modified data (with outliers replaced by NaN) is written back into the original data matrix, preserving the structure of the dataset. Once all rows have been processed, the function handles the missing values by filling them using the filling_method specified by the user. This step ensures that the dataset remains complete and ready for further analysis.
The "*slopeOutlier*" method is particularly effective for detecting sudden jumps or drops in the data, which are indicative of anomalies. This approach is useful when outliers are characterized by sharp changes in a short period, a scenario that standard deviation or median-based methods might not capture.
The default threshold for detecting outliers based on the slope is set to 7.2, and a sliding window of size 3 is used. The choice of these parameters was based on the need to identify significant changes in the data while minimizing false positives. This method is our default for outlier detection because it effectively identifies

outliers based on visual patterns, while also keeping the number of detected outliers lower compared to other methods.

The Interquartile Range (IQR) method

The Interquartile Range (IQR) method was employed to detect and handle outliers within the dataset. The IQR method is a robust statistical technique that identifies outliers by focusing on the middle 50% of the data, which is less sensitive to extreme values or skewed distributions compared to other methods, such as the Z-score, which assumes a normal distribution. This approach provides an effective way to identify data points that deviate significantly from the general trend without making any assumptions about the data's underlying distribution. The IQR-based outlier detection method works by calculating the range between the first quartile (Q1), which is the 25th percentile of the data, and the third quartile (Q3), the 75th percentile. The difference between these quartiles is the Interquartile Range (IQR), calculated as:

$$IQR = Q_3 - Q_1$$

Using the IQR, the method defines outliers as data points that fall below Q1 - threshold * IQR or above Q3 + threshold * IQR which in this case, the threshold was set to 2.3, meaning that any point more than 2.3 times the IQR away from the central range was considered an outlier.

To make the detection process more robust, a sliding window technique was applied. This technique involves scanning through the dataset in segments, ensuring that outliers are detected locally within smaller windows of data rather than across the entire dataset. The sliding window ensures that the method can detect contextually appropriate outliers, adapting to local fluctuations in the dataset.

For each row of the input data matrix, a window of a specified size (set to 8 in this implementation) moves across the dataset. For every window, the IQR is calculated, and any data point that lies outside the range defined by the threshold is considered an outlier. These outliers are then replaced with NaN values, ensuring that they are effectively removed from the data without affecting subsequent analyses. After the outliers are identified and replaced with NaN, the method includes a process to fill in these missing values. Depending on the nature of the dataset and the requirements of the analysis, different filing techniques can be applied. In this case, the filling method is specified as an input parameter, allowing the user to choose the most appropriate strategy for their data. Common methods include linear interpolation or filling with a constant value. By filling the missing values, the data remains complete and suitable for further analysis, while still having addressed the issue of outliers.

The parameters for the sliding window size and threshold were chosen through an iterative process. By visually inspecting the data and testing different parameter values, the window size was set to 8 and the threshold to 2.3. These parameters provided a balance between detecting a meaningful number of outliers and minimizing their impact on the data. The window size and threshold were varied across ranges (window size from 2 to 20, threshold from 0.1 to 10) until optimal values were identified.

Code Description:

The MATLAB code implements an Interquartile Range (IQR)-based method to detect and handle outliers in a dataset using a sliding window approach. The function "*IQROutlier*" operates by processing the data row by row, applying the IQR method to each segment of data defined by a sliding window.

The function accepts several inputs: the dataset to be processed *(matrix_input)*, the start and end points of the region to be analyzed *(start_end_points)*, the size of the sliding window *(window_size)*, a threshold value *(threshold*) that determines how far a data point must be from the IQR to be considered an outlier, and the method for handling missing values (*filling_method*).

The core of the function operates in two main stages. In the first stage, the function loops through each row of the data matrix and processes each segment of the data using a sliding window. For each window, it calculates the first quartile (Q1) and the third quartile (Q3) using MATLAB's prctile function, which represent the 25th and 75th percentiles of the data, respectively. The IQR is then computed as the difference between Q3 and Q1. These outliers are then replaced with NaN, effectively marking them for further processing. After identifying outliers within each window, the function updates the original dataset by replacing these outliers with NaN values. Subsequently, missing values are addressed using the specified filling_method, such as linear interpolation or a constant value.

Z-score

Another outlier detection method is the Z-score method, which measures how far data points deviate from the mean in terms of standard deviations. This approach is particularly useful for identifying values that differ significantly from the average behavior of the dataset. This method is particularly useful for datasets that are assumed to follow a normal distribution or for cases where deviations from the mean are of primary interest and the sliding window technique ensures that the method remains flexible and can adapt to non-uniform data distributions. The Z-score method works by calculating how many standard deviations a data point lies from the mean. It is computed as:

$$Z = (x - \mu)/\sigma$$

where x is a data point, μ is the mean of the data in the window, and σ is the standard deviation. The Z-score reflects whether the point is close to the mean (Z-score near zero), significantly higher than the mean (positive Z-score), or significantly lower than the mean (negative Z-score). Any data point whose absolute Z-score exceeds the threshold value is classified as an outlier.

Code Description:

The function "*ZScoreOutlier*" detects these outliers locally by applying a sliding window to the dataset and calculating the Z-score within each window. The function takes several inputs, including the dataset to be analyzed (*matrix_input*), the start and end points of the data region to be processed (*start_end_points*), the size of the sliding window (*window_size*), a threshold value (*threshold*) that defines the cutoff for identifying outliers, and a method to fill missing values *(filling_method*) after the outliers are removed.

The function first loops through each row of the input data matrix. Within each row, it applies a sliding window of a specified size *(window_size)*. For each window, it calculates the Z-score for each data point. The Z-score formula takes the current window's mean and standard deviation, allowing for localized outlier detection that adapts to the changing of the data across the row.

If any data point in the window has a Z-score greater than the specified threshold (in absolute value), it is flagged as an outlier. These outliers are replaced with NaN, ensuring they are marked for further processing. After the outlier detection process, the code updates the original dataset by replacing the identified outliers with NaN in the corresponding positions. Once all rows and windows have been processed, the function fills these NaN values using the specified filling_method. This could involve methods such as linear interpolation or constant filling, depending on how the user wants to handle the missing values.

In our analysis, the function is applied using the following default parameters:

- A sliding window size of 11 is used, allowing for localized outlier detection across small segments of the data.
- The Z-score threshold is set to 2.2. Any data point with a Z-score greater than ±2.2 is flagged as an outlier.

Using these parameters, the method detected and handled more outliers compared to IQR method in this dataset.The choice of default parameters was determined by trial and error, balancing between detecting a sufficient number of outliers while minimizing false positives. The sliding window approach allows the method to adapt to varying patterns in the dataset, while the Z-score threshold ensures that only data points with significant deviations from the mean are identified as outliers.

Median Absolute Deviation (MAD)

The **Median Absolute Deviation (MAD)** method is a robust statistical technique used for detecting outliers in a dataset. Unlike methods that rely on the mean and standard deviation, MAD uses the median, which is more resilient to extreme values and outliers. This makes the MAD method particularly useful in datasets that exhibit skewness or have heavy-tailed distributions, where extreme values might distort more traditional measures like the mean. The MAD is a measure of the spread or variability in a dataset, focusing on the median rather than the mean. The formula for MAD is:

$$MAD = median(|x_i - \text{median}(x)|)$$

This formula calculates the absolute deviation of each data point from the median and then takes the median of those deviations. The MAD method is highly effective at identifying outliers because it is less sensitive to extreme values, making it more robust than other methods, such as standard deviation. Outliers are identified based on how far they deviate from the median. In the MAD method, a data point is considered an outlier if its absolute deviation from the median exceeds a certain threshold multiplied by the MAD. The formula for detecting outliers is:

$$Outliers = |x_i - median(x)| > thresholds \times MAD$$

The threshold determines the sensitivity of the detection process. A higher threshold makes the method less sensitive to outliers, while a lower threshold detects more outliers.

The reason we chose this method is that MAD is more robust than methods like Z-score or standard deviation because it uses the median, which is not influenced by extreme values. Besides, it does not assume that the data follows a normal distribution, making it suitable for datasets with skewed or heavy-tailed distributions. In addition, MAD can detect outliers locally within different parts of the dataset, enhancing its effectiveness in datasets with varying trend when applied with a sliding window approach.

Code Description:

The function "*MADOutlier*" is designed to detect and handle outliers in a dataset using the MAD method, employing a sliding window approach to ensure localized outlier detection.

The function starts by initializing the input data matrix matrix_input as outlierfree_data_MAD. It then processes the data row by row, applying a sliding window approach to detect outliers. For each row, the function iterates over a specified range defined by start_end_points, applying a window of size window_size to segments of the data. Within each window, the function calculates the median of the window's values. It then computes the MAD, which is defined as the median of the absolute deviations of the data points from the median. This step is crucial because the MAD provides a robust measure of variability that is less influenced by outliers compared to standard deviation.

The outlier detection process involves comparing the absolute deviations from the median to a threshold multiplied by the MAD. Any data point where this deviation exceeds the threshold is identified as an outlier. These outliers are then replaced with NaN values within the current window. Once all rows have been processed and the outliers have been marked as NaN, the function proceeds to fill these missing values. The fillmissing function is used to handle NaN values according to the specified filling_method, ensuring that the dataset remains complete for subsequent analysis.

The default parameters used for the MAD method in this analysis are:

- Threshold: The default threshold is set to 7.7, determining the sensitivity of the outlier detection.
- Window Size: A sliding window of size 6 is used to detect outliers in small segments of the dataset.

3.1.2 Filtering/Smoothing

In this study, we provided the user with the option to choose either smoothing or filtering methods for processing the knee angle data during walking. Each method serves a specific purpose and has different strengths depending on the desired outcome of the data analysis:

low-pass Butterworth filter

Filtering is essential to remove high-frequency noise from the data, which can obscure the underlying trends and key features, especially in periodic datasets like knee angles during walking. The "**low-pass Butterworth filter**" was chosen because it is generally more appropriate for this type of dataset. Butterworth filter is commonly used for smoothing signals in the time domain. It is usually chosen because of its flat frequency response in the pass band (2). The Butterworth low-pass filter is also a common method for filtering the gait cycle data (3). Moreover, in (4), the performance of three methods for filtering the gait cycle data were investigated. The three methods were moving average, Butterworth, and Savitsky-Golay (sgolay) filters. The result of this study shows that the Butterworth low-pass filter presents the most robust performance in differentiation conditions. A high-pass filter would not be suitable because it could remove important low-frequency components that represent the actual movement trends. Besides these researches, it can be seen in Figure 4, the Butterworth filter shows a more smoothed data compared to the smoothing methods, especially, the moving average.

For our filter implementation:

- We chose a **filter order of 5**, which offers a balance between smoothing the data and maintaining its key characteristics.
- The **cutoff frequency of 0.05** was selected to effectively eliminate unwanted noise while keeping the important frequencies associated with the gait cycle.

The filtering process involves designing the Butterworth filter using MATLAB's "*butter*" function and then applying the filter using the "*filtfilt*" function. This method avoids phase shifts and ensures that the data remains aligned with the time points. The input data is segmented by start and end points to avoid issues with NaN values, ensuring clean and continuous filtering for each data segment.

Figure 4. shows the filtered data using Butterworth filter and the default filter order and cutoff frequency. The outlier detection and filling methods in the images of the filtering part are set on default.

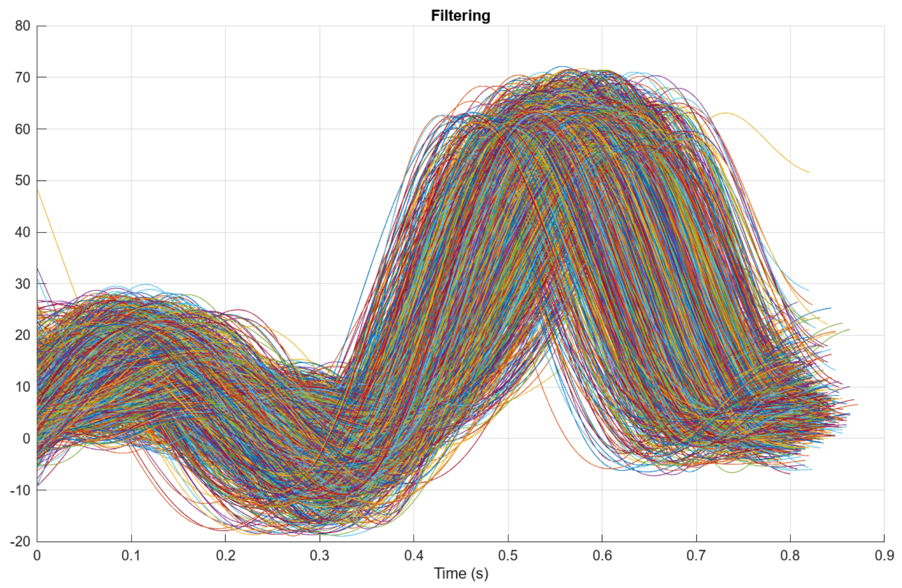


Figure 4. Butterworth filtered data

Smoothing methods

Alternatively, the user can choose to apply smoothing techniques, which are useful for reducing noise while preserving key features such as the peaks and troughs in the data. We implemented several smoothing methods, each offering different characteristics based on the nature of the data.

The default smoothing method we provided is the “moving average” method, with a window size of 20 points. This method is a simple, effective way to reduce high-frequency noise and smooth the periodic knee angle data from walking. One of the primary advantages of the moving average is its ability to perform simple noise reduction without significantly distorting the overall trend of the data. Given that walking has relatively periodic cycles, with repetitive knee flexion and extension phases, the moving average can effectively smooth out minor fluctuations while preserving the general shape of the knee angle cycle, as shown in Figure 5.

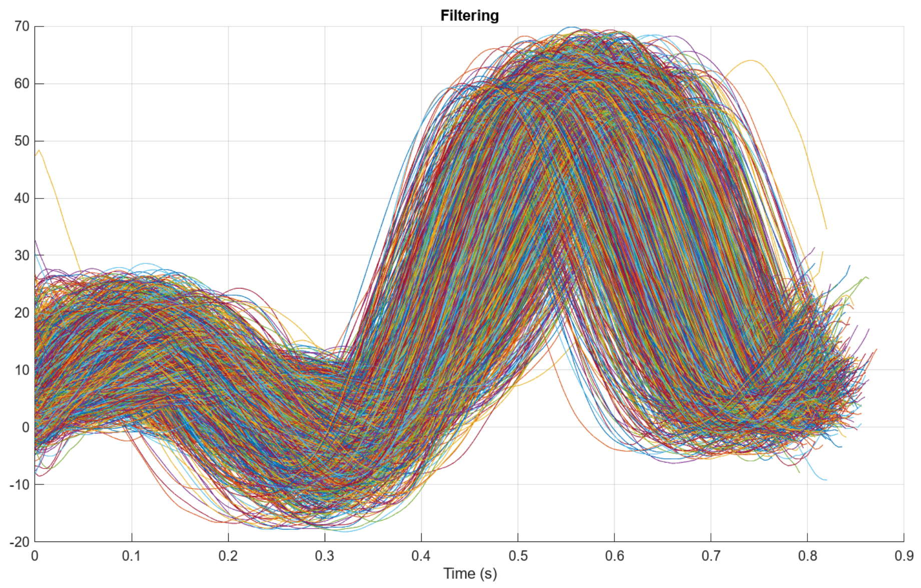


Figure 5. Smoothed data using Moving Average method

However, it's important to note the limitations of the moving average. While it is effective at smoothing data, it can sometimes over smooth and blur sharp transitions, such as the peaks in knee flexion or extension. This could result in the loss of precision regarding the location and magnitude of these key points in the knee angle cycle. Therefore, while the moving average is ideal for general noise reduction, users who require precise peak detection or more detailed local features may need to consider other smoothing methods provided in the system:

1. “Lowess” (Locally Weighted Scatterplot Smoothing): This method is more flexible than moving average, as it captures local trends and slight nonlinearity in the data. It’s particularly useful when the knee angle data exhibits subtle variations that are important for analysis. However, it's more computationally intensive and can be sensitive to outliers, as shown in Figure 6.

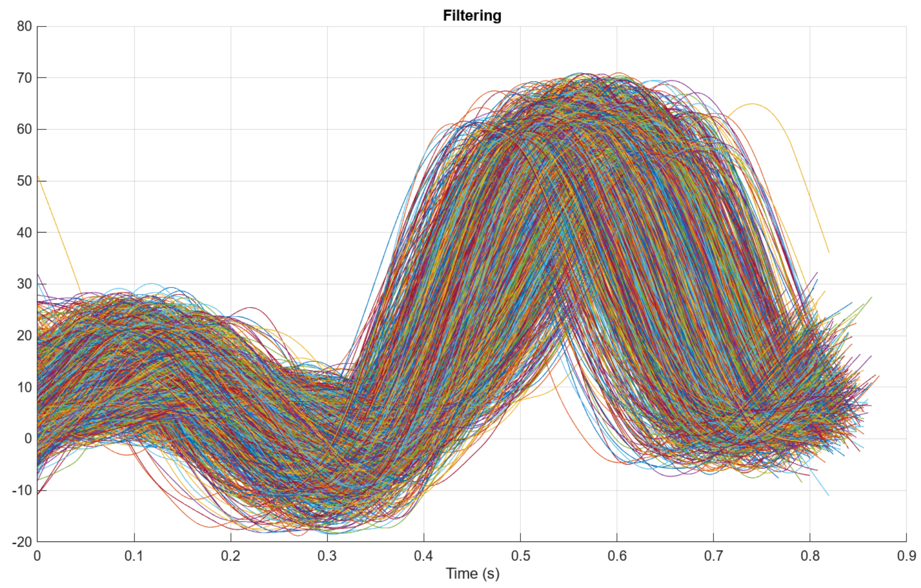


Figure 6. Smoothed data using Lowess method

2. “Loess” (Locally Weighted Polynomial Regression): Loess is similar to Lowess but fits local quadratic models, making it more suitable for data with pronounced curvature or rapid changes, such as sharp transitions in knee flexion or extension phases during walking. One limitation of using Loess is its higher computational intensity compared to both moving average and lowess methods. Additionally, similar to lowess, Loess can be influenced by outliers unless the robust version is used, as shown in Figure 7.

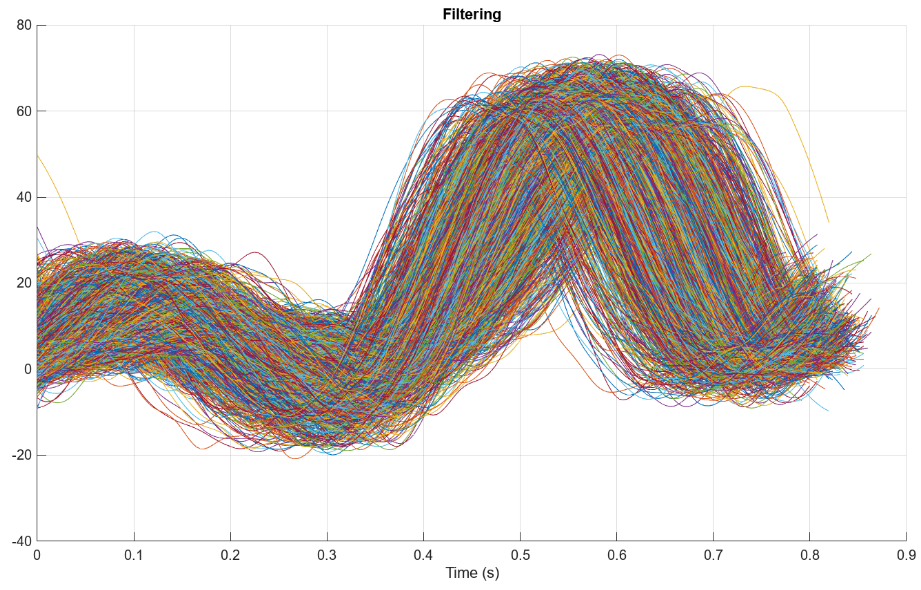


Figure 7. Smoothed data using Loess method

3. “Robust Loess” (rloess): This method adds robustness by minimizing the influence of outliers, making it ideal for noisy data or data with irregularities. It smooths the knee angle data effectively while reducing the

impact of erratic points, such as those caused by measurement errors or unexpected movements, as shown in Figure 8.

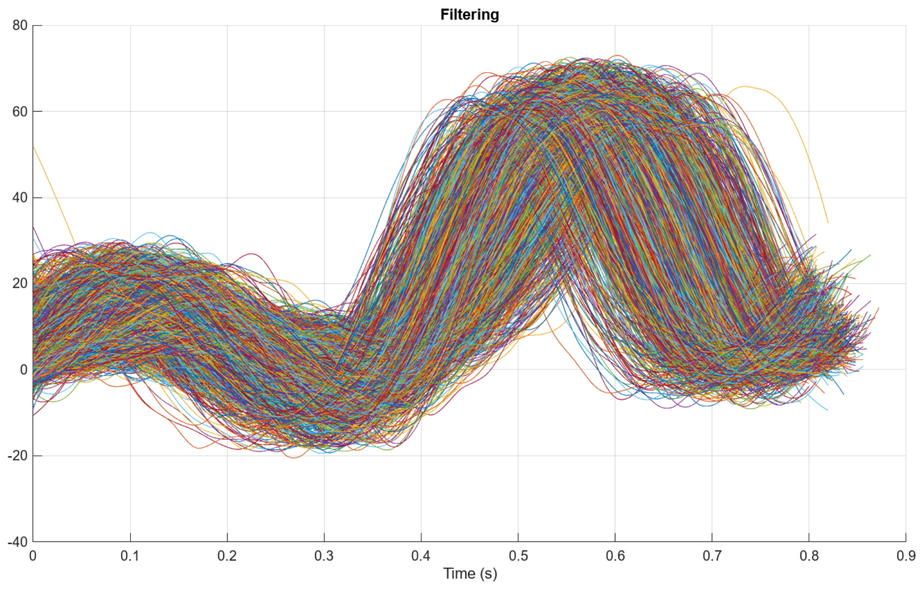


Figure 8. Smoothed data using Rloess method

4. "Savitzky-Golay Filter (sgolay)": This filter is particularly effective for preserving key features such as the peaks and troughs in the knee angle cycle. Unlike the moving average, which can over smooth, Savitzky-Golay is designed to smooth the data while retaining the critical details of the gait cycle, making it useful for detailed gait analysis where preserving the exact shape of knee movement is crucial, as shown in Figure 9.

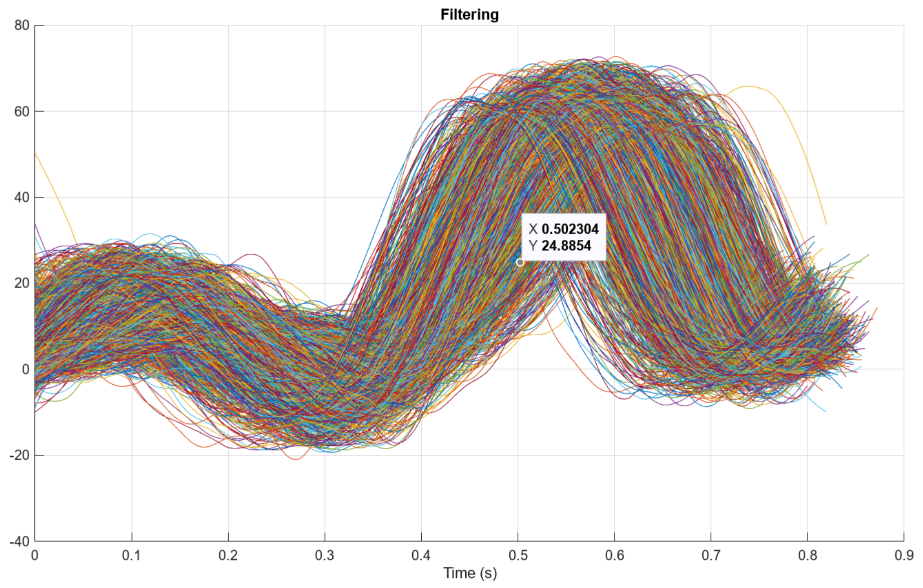


Figure 9. Smoothed data using Sgolay method

3.1.3 Temporal Normalization

Temporal normalization is another important preprocessing step that we offer users, allowing for the alignment of data across different trials or conditions. This is particularly useful when working with time-series data, like our dataset, where trials can vary in length, making direct comparisons challenging. Temporal normalization ensures that each trial has the same number of data points, thus enabling better comparison and analysis across subjects or conditions.

By default, the temporal normalization will be applied based on the length of the data. The system calculates the default number of time points by averaging the length of each trial (i.e., the index of the last data point for each row) and then rounding the value to the nearest tens for easier interpretation and consistency (e.g., a trial length of 204 would be rounded to 200, while 206 would be rounded to 210). This ensures that the default number of time points is appropriate for the majority of the dataset, providing a balance between maintaining enough data points and simplifying the time series. Additionally, the range for selecting the number of time points through the user interface is set to allow flexibility, starting at 50% of the default number of points and ending at 150%, giving users the ability to choose a smaller or larger number of points if needed, while still keeping the data meaningful and interpretable.

The key advantage of this approach is that it allows us to account for differences in trial duration while maintaining the integrity of the data. By resampling the data, we ensure that features like the timing of knee flexion and extension are aligned across trials, making it easier to identify patterns and draw meaningful conclusions.

Code Description:

The “*temporalNormalization”* function takes as input the original data matrix, the start and end points of each trial, and the number of time points for normalization. It outputs a temporally normalized matrix, where each trial is resampled to the specified number of points. This resampling is achieved using MATLAB' “*interp1*” function, which interpolates the data over a new, normalized time vector. Specifically, for each trial, we first create an original time vector corresponding to the length of the trial data, and then create a normalized time vector with the desired number of points. The data is then interpolated between these time vectors, resulting in the temporally normalized output.

3.1.4 Synchronization

Why not synchronize based on the minimum points?

Synchronization is a crucial step in data analysis, particularly in time-series data like the knee angle during the gait cycle, to ensure that the data from different subjects or conditions can be accurately compared. The goal of synchronization is to align key points in the dataset, making it easier to evaluate differences or patterns across multiple datasets.

One option for synchronization could be to use the minimum points in the data. However, this approach is not ideal for the dataset in question. The knee angle data fluctuates with both local and global minimum and maximum points throughout each step. While each step generally has three minimum points, these minimums are not consistent across all subjects or conditions, as shown in Figure10.

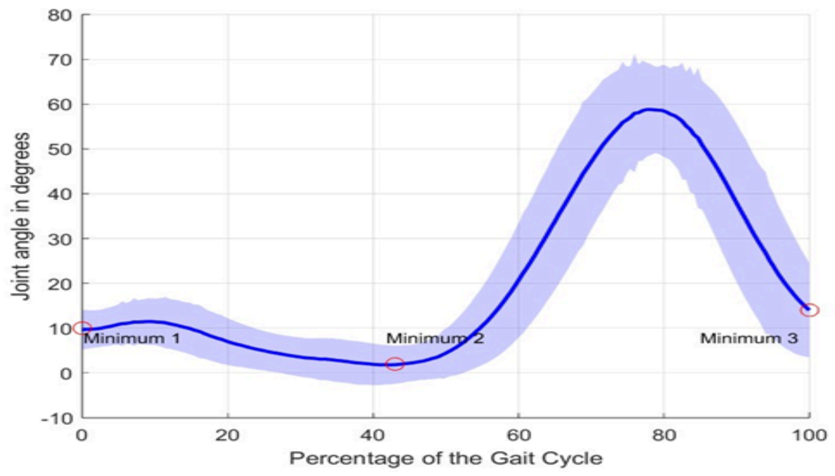


Figure 10. Minimum points showed on the condition A

The second minimum may appear as the global minimum in some cases, but this pattern does not hold universally. In other cases, the global minimum may occur at the first or third minimum points, as shown in Figure 11.

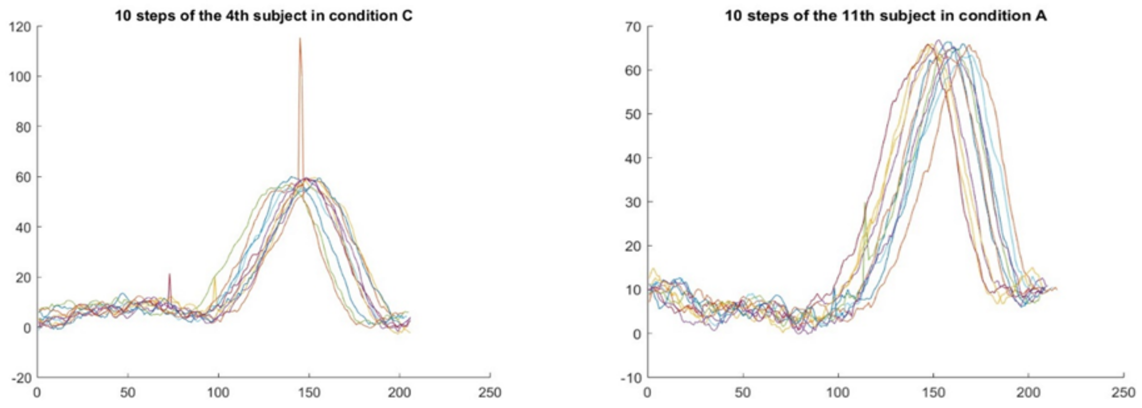


Figure 11. Data samples from different conditions

This inconsistency makes synchronization based on the minimum points unreliable. Defining a range to identify minimum points could work for certain datasets, but this method would not be flexible enough to handle all variations in different data sets.

Synchronization using maximum points

In contrast, the global maximum point is much clearer and more consistent across all datasets. It stands out visually as a distinct, well-defined peak, making it a more reliable anchor for synchronization. This is why the global maximum point was chosen as the base for synchronization, ensuring the app can be applied to all knee flexion angle datasets without the ambiguity associated with minimum points, as shown in Figure 12.

Code Description:

In the *"synchronizingMaxGroupSubject*" function, the dataset is processed row by row, where each row corresponds to a specific subject and trial. The function first identifies the global maximum within the dataset for each subject within a group. For each trial, it finds the local maximum and computes the shift required to align this local maximum with the global maximum. After calculating the shift, the trial data is moved accordingly, and the synchronized version of the data is placed in a new matrix. The function ultimately returns this matrix with the data synchronized at the subject level for each condition group.

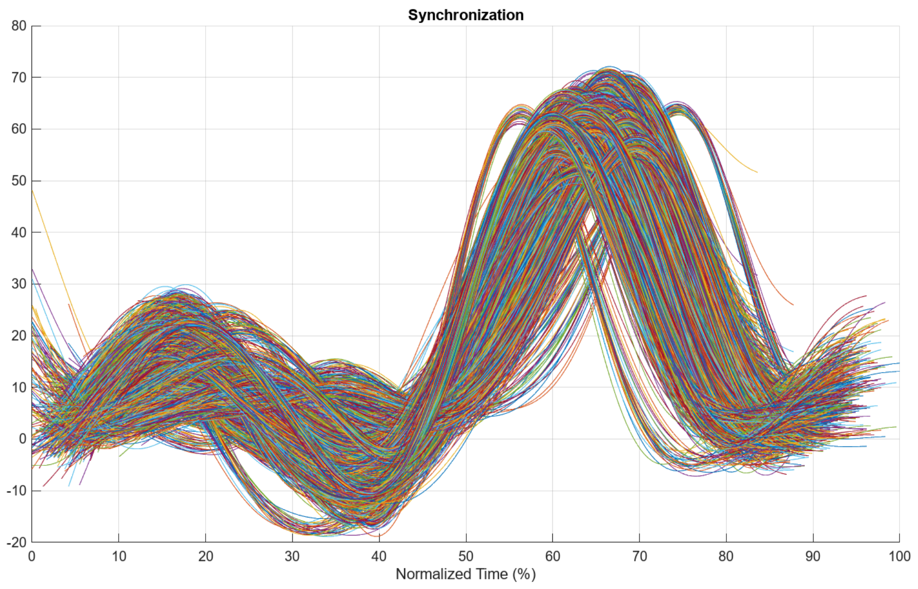


Figure 12. Synchronized data based on the global maximum points of the subjects in different conditions

In the "*synchronizingMaxGroup*" function, a similar process occurs, but at the group level. Instead of synchronizing data based on individual subjects by group, it synchronizes all trials within a condition group based on the global maximum for the group. Each trial is shifted so that its maximum point aligns with this group-level maximum. This provides a way to compare different subjects and their trials within the same group, ensuring that each dataset is aligned consistently, as shown in Figure 13.

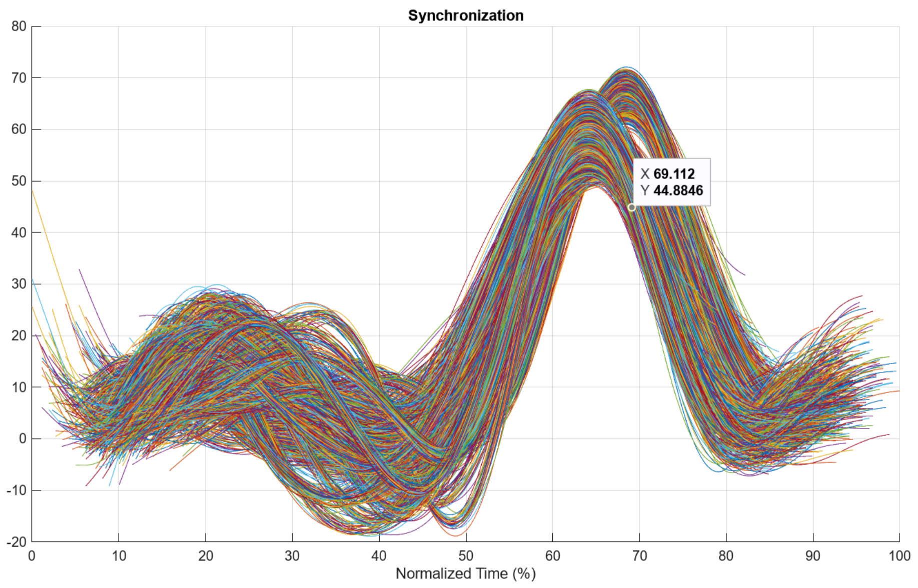


Figure 13. Synchronized data based on the global maximum points of each condition

The "*synchronizingMaxAll*" function operates on the entire dataset, identifying the global maximum across all trials and subjects. It shifts all trials so that their maximum points align with this global maximum. This allows for synchronization across the entire dataset, making it easier to compare all the data simultaneously, regardless of group or subject. As shown in Figure 14.

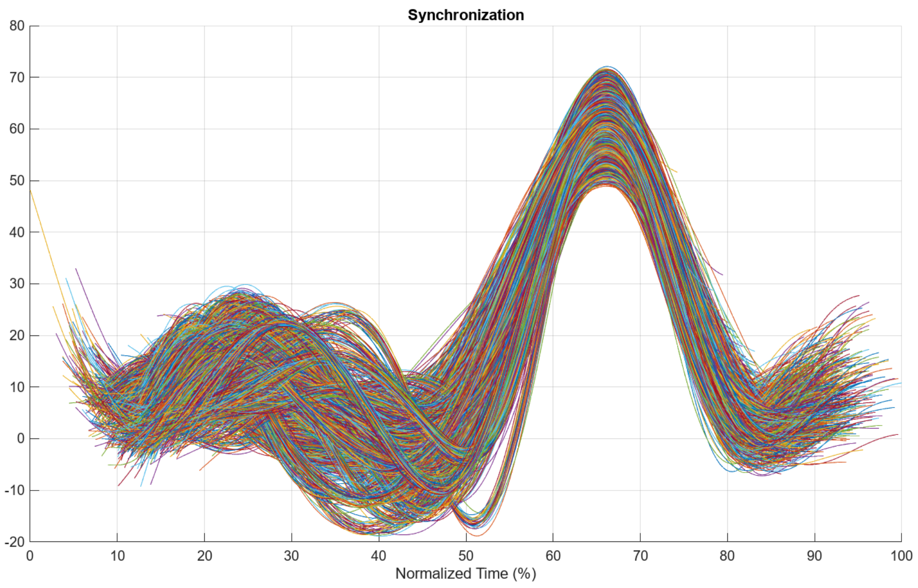


Figure 14. Synchronized data based on the global maximum points

Synchronization using cross-correlation

Another approach that was used for the synchronization was the cross correlation, "xcorr". For instance, "xcorr(x, y)" returns the cross-correlation of two sequences. Cross-correlation measures the similarity between a vector x and shifted copies of a vector y. It provides a direct measure of how well two signals match at various shifts. The peak of the cross-correlation indicates the optimal shift for alignment.

The mathematics behind the cross-correlation can be seen in the equation as follows. The x[n] and y[n] are the sequences that are required to be aligned. “xcorr” shifts y[n] over x[n] by different amounts. For each shift k the correlation is calculated using the following equation:.

$$R_{xy}(k) = \sum_{n} x\,[n]\,.\,y\,[n\,+\,k]$$

R_{xy} is the correlation value at shift k. The sum $\sum_{n}$ adds up the products of corresponding values from x and the shifted y. The output is a list of correlation values, one for each shift. The maximum value in this list indicates the best alignment between the two sequences.
After finding the proper amount of shifting, they were shifted circularly using the “circshift” function. circshift(A, K) circularly shifts the elements in array A by K positions. This means that the extra data points after shifting, for example to the right, are added to the right side of the data.
This approach is especially valid for the cyclic datasets, including knee flexion angle in the gait cycle. In the cyclic dataset, after finishing one cycle, or in this case each step, the next cycle begins, which has approximately the same trend. For this reason, the missing part of the dataset after shifting is most likely similar to the trend, which is added from the other side of the dataset. For this method, we need a reference in the for loop to align each row with it. It can be any row, to be selected as the reference. In each function, the first row is the reference.
Due to this shifting, there will be some sharp points at the points that the data were added. This happens according to the logic of this method, and a complete smooth data cannot be expected. Smoothing was already applied to the dataset to improve the results for synchronization input. After using this method, some sharp points may still appear in the data despite the overall smoothed data. As shown in Figure 15,16 and 17.

Code description

This function, “*synchronizingCrossCorrelationAll”*, aligns sequences in a matrix using circular cross-correlation. It starts by defining a reference sequence from the first row of the input matrix. An empty matrix, “*synchronized*_data”, is initialized to store the aligned sequences. The reference sequence is then stored as the first row of this matrix. For each subsequent row, the function extracts the sequence to be aligned and computes its circular cross-correlation with the reference sequence. The index of the maximum correlation value determines the shift needed to align the sequence. This shift is applied using “*circshift”*, and the aligned sequence is stored in the corresponding row of “*synchronized_data”.* As shown in Figure 15.

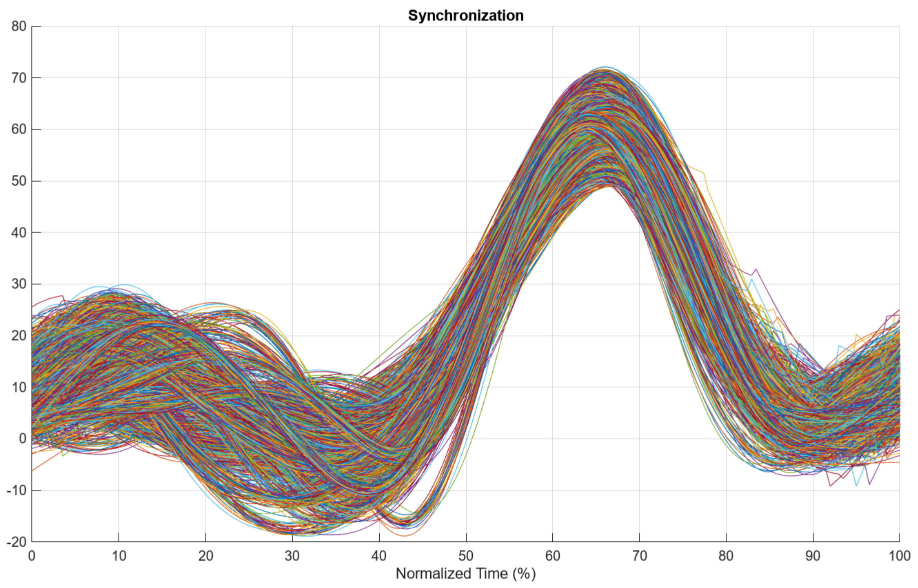


Figure 15. Synchronized data using cross correlation

The function "*synchronizingCrossCorrelationGroup*" aligned sequences within groups in a matrix using circular cross-correlation. It initializes an empty matrix, *'synchronized_data",* to store the aligned sequences and calculates the length of the data excluding certain columns. For each group, identified by unique values in the first column, it extracts the group's data and defines the first row as the reference sequence. This reference sequence is stored in the first row of "*synchronized_data*". For each subsequent sequence in the group, the function computes its circular cross-correlation with the reference sequence, identifies the shift needed to align the sequence, and applies this shift using *"circshift".* The aligned sequence is then stored in *"synchronized_data".* As shown in Figure 16.

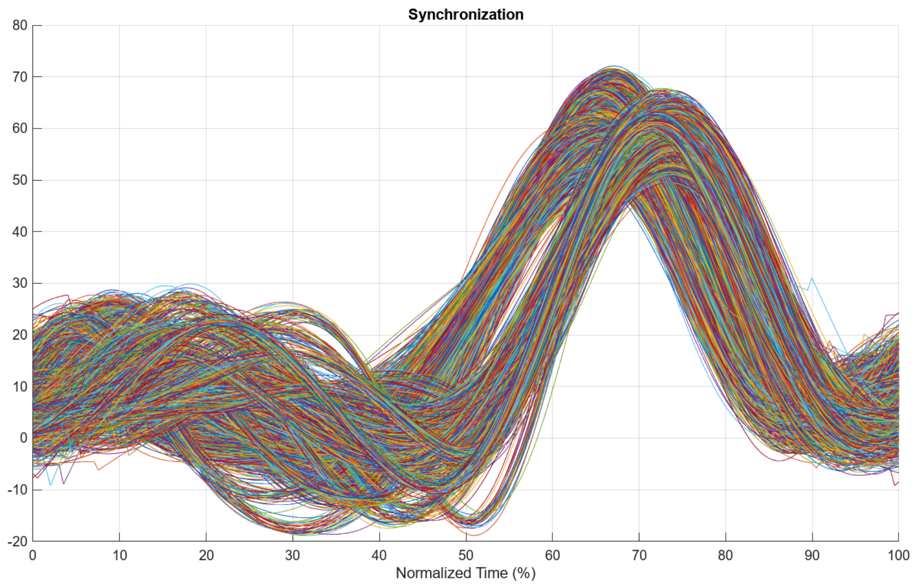


Figure 16. Synchronized data using cross correlation for each condition

The function *"synchronizingCrossCorrelationGroupSubject"* aligned sequences within groups and subjects in a matrix using circular cross-correlation. It initializes an empty matrix, "*synchronized_data", and calculates* the length of the data excluding certain columns. For each group and subject, identified by unique values in the first two columns, it extracts the relevant data and defines the first row as the reference sequence. This reference sequence is stored in the first row of *"synchronized_data"*. For each subsequent sequence, the function computes its circular cross-correlation with the reference sequence, identifies the shift needed to align the sequence, and applies this shift using *"circshift"*. The aligned sequence is then stored in *"synchronized_data"*. As shown in Figure 17.

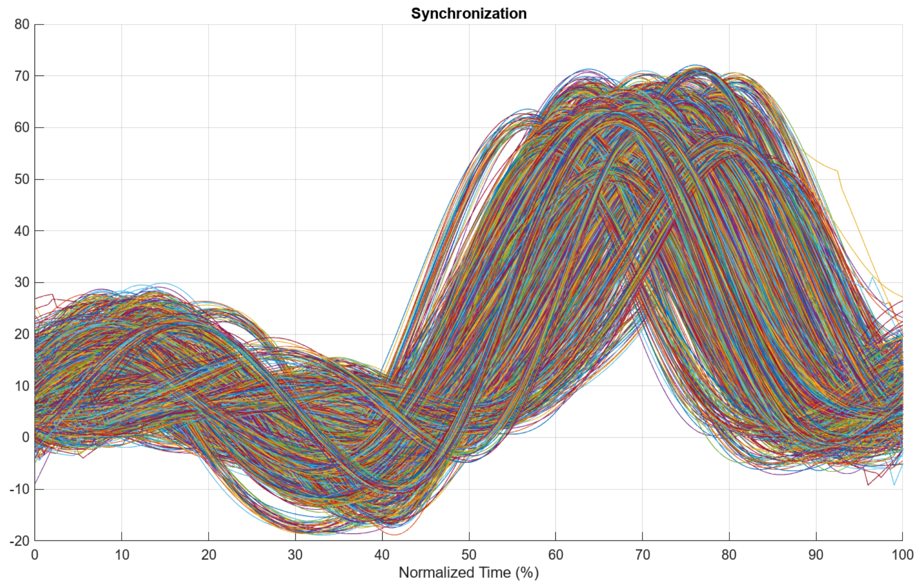


Figure 17. Synchronized data using cross correlation for different subjects in each condition

3.1.5 Normalization

Normalization is a process used to standardize time-series data, ensuring that different trials or data sets can be compared on the same temporal scale, even if they vary in length or timing. In the context of gait analysis, for example, the duration of each walking cycle might differ slightly between trials, making it challenging to compare cycles directly. Normalization solves this by rescaling the data to a consistent number of time points, preserving the overall pattern of the movement while aligning the start and end points across all trials. Two distinct approaches are offered for data normalization, depending on the characteristics of the data and the chosen synchronization method:

Sidewise Normalization

This method is used specifically for max synchronization, where the data is synchronized based on a fixed point, such as the maximum knee angle during the gait cycle. After synchronization, we normalize the data by splitting it into two parts—before and after the fixed point—and then performing interpolation separately on each side. This ensures that both the pre- and post-fixed point sections are normalized while maintaining the synchronization point.

The “*normalizationSides*” function performs this sidewise normalization, accepting as input the original data matrix, the fixed point for each trial, and the desired number of time points for normalization. The function interpolates the data on each side of the fixed point, ensuring that the left side (before the fixed point) and the right side (after the fixed point) are normalized independently. This approach is particularly effective when the data has significant features or transitions around the fixed point, as it allows for more precise alignment and smoothing of both the pre- and post-event data.

Key steps include:

- Left side normalization: Data from the first non-NaN value up to the fixed point is interpolated to match the desired number of time points.
- Right side normalization: Similarly, data from the fixed point to the last non-NaN value is interpolated.
- The final output combines the interpolated data from both sides into a single, normalized time series while preserving the fixed synchronization point.

This method is especially useful when analyzing data where a specific event (e.g., peak knee flexion) is the focal point, as it ensures that data on both sides of this event is normalized separately, maintaining the integrity of the event while smoothing the surrounding data.

Cyclic Normalization

For cyclic data such as gait cycles, we assume that the movement is repetitive and will eventually return to a similar starting point. This method is ideal for continuous or repetitive activities like walking, where the end of one cycle smoothly transitions into the start of the next. The normalizationCyclic function implements this approach, effectively shifting the data to align the start and end points, ensuring that the cycle appears continuous and uninterrupted.

The function begins by calculating the average start point based on the first and last non-NaN data points for each row. If the first non-NaN value appears later than the calculated average start point, the missing data points are filled by cyclically shifting the data, ensuring that the early part of the cycle aligns with the later part. The same approach is used if there are missing data points after the last valid value, extending the data cyclically from the start of the next cycle.

This method works particularly well when the data represents repetitive motion, such as walking, where the knee angle follows a cyclical pattern. After shifting the data cyclically, we apply a filtering or smoothing process to ensure that the transition points are smooth and free of noise, enhancing the continuity of the cyclic motion.

3.2 Single Data Analysis Tab

3.2.1 Loading Data

When users attempt to load the dataset, they are provided with an option to load data that might have a different structure than expected. This ensures that the application remains flexible in handling different types of data, which may arise from changes in experimental setups or measurement formats. If users choose to load a dataset with a different structure, a separate application called "generateFilenames" is launched, designed specifically for processing these alternative formats. This enhances user experience by allowing them to seamlessly handle different dataset structures without reconfiguring the main application.

The "*generateExamplaryFilename*" function in this application checks the values from two dropdown menus that represent data categories like "Condition" or "Subject." Based on the selected options, it creates a sample filename by combining the text labels and numerical or letter-based identifiers for conditions and subjects. This customized filename structure helps the system adapt to varying data formats when users choose to upload data in a different structure, simplifying file management and ensuring accurate dataset labeling.

However, if the user doesn't want to load a new dataset, the default behavior is to load the project's knee angle dataset, maintaining continuity with the current structure. This ensures that the user can continue working with the usual dataset without any interruptions, providing a simple and easy way to manage the data.

Code description

The function "loadData" is designed to efficiently load and organize data from multiple .mat files. It takes four input parameters: app, which contains the experiment-specific variables like condition names; folder_path, specifying the directory where the data files are located; num_of_cond, the number of experimental conditions; and num_of_subj, the number of subjects.
The function outputs two components: output_matrix, which contains all the loaded data, and num_of_trials, which stores the number of trials for each condition and subject. It starts by initializing two matrices: output_matrix, which will store the data, and num_of_trials, initialized with NaN values and sized to match the number of conditions and subjects.
The function uses a nested loop to load the data. The outer loop iterates over the conditions, and the inner loop over the subjects. For each combination, it constructs the file path, loads the data into a temporary variable, and appends it to output_matrix. The first two columns of this matrix are populated with condition and subject numbers using the repmat() function, while the remaining columns store the actual data.
Once all files are loaded, the function checks if the number of trials is consistent across all files by comparing each dataset’s trial count with the first dataset. If a discrepancy is found, it identifies the condition and subject where the inconsistency occurs and displays an error message to the user.

3.2.1 Plotting Data

The “plotData” function plots data for a specific condition and subject from an experimental dataset. It accepts the inputs matrix_input, cond_input, subj_input, and axes_plot, and allows flexible selection based on user input.

First, the function converts cond_input to its corresponding index if it's a condition name, and converts subj_input to a numeric value if it's specified. It then creates a logical index to filter the data based on four possibilities:

- Only subjects are given.
- Only conditions are given.
- Both condition and subject are given.
- Neither is specified, so all data is selected.

The filtered data is stored in plotting_data, with any columns containing only NaN values removed to clean up the plot.

Next, the function clears the specified plot axes to ensure no previous data remains. It then plots the filtered data, iterating through each row, and uses the plot() function to display all data for the chosen condition and subject in one plot.

3.2.3 Saving Data

The save button callback function manages the saving of user interface (UI) data and graphical output into specified file formats. It ensures a robust saving process with error handling and user guidance.

Initially, the function retrieves the filename entered by the user. It validates the input in multiple ways to prevent errors. Firstly, it checks for an empty filename field, prompting the user to input a valid name if empty.

Next, it validates the filename to exclude invalid characters not permitted in filenames by most operating systems. If any invalid characters are detected, the user is prompted to correct the filename.

The function then constructs full file paths for .mat (*data matrix*) and .jpg (*graphical plot*) formats using fullfile to save it into the SavedFiles folder.

To avoid overwriting existing files, the function checks for files with the same name in the folder. If duplicates are found, the user is asked to choose a different filename.

If all checks pass, the function proceeds with the saving process. The data matrix is saved in .mat format using save, while the graphical plot from UIAxes is saved as a .jpg image using export graphics. A success message is displayed upon successful saving. In case of errors, an error message is shown.

3.3 Compare Settings Tab

In the "Comparing Settings" tab of our interface, users are provided with a tool to visually compare different conditions and subjects in real-time, allowing for an intuitive examination of how various steps in the data

processing affect the results. This section features two side-by-side plots where users can simultaneously visualize data from two different subjects or conditions, enabling direct visual comparison.
Each plot has its own settings menu, which allows users to independently configure the display settings for each subject or condition they wish to compare. This flexibility means that users can select different processing steps or analysis parameters for each plot, making it easy to observe how different conditions, filters, smoothing techniques, or normalization steps influence the data. For example, users can compare the raw, unprocessed data of one subject on the left plot with the filtered and smoothed data of another subject on the right plot. Alternatively, users could compare the same subject under different conditions, such as pre- and post-synchronization, or after different types of normalization.

3.4 All Data Tab

In the "All Data" tab, users have the ability to visualize and compare the results of various processing steps across multiple datasets. There are four axes available for plotting, which allow for a detailed comparison of the data at each stage of processing. Users can choose to either plot the entire dataset or focus on specific subjects within each group. This functionality enables users to compare processing steps for individual subjects, offering flexibility in data exploration and facilitating side-by-side evaluation of the results for the selected subjects.

3.5 Result Tab

3.5.1 Test parameters

T-test:

The t-test is a statistical method used to determine whether there is a significant difference between the means of two groups. It is particularly useful when the sample sizes are small, and the population standard deviations are unknown. The test evaluates the null hypothesis, which posits that there is no significant difference between the means of the two groups. When conducting a t-test, two primary statistics are calculated:

- T-Statistic: This value represents the ratio of the difference between the two group means to the variability of the groups. It quantifies how far the observed mean difference is from the null hypothesis of no difference, scaled by the standard error of the difference. A larger t-statistic indicates a greater difference between group means relative to the variability within the groups.
- P-Value: The p-value indicates the probability of observing the data if the null hypothesis is true. In general, a p-value less than 0.05 is considered statistically significant, implying that the observed difference is unlikely to have occurred by random chance. However, this threshold may vary depending on the previously defined alpha/significance level. For this reason, no result is given in the app regarding the rejection or acceptance of the null hypothesis.

Effect size for t-test:

Effect size is a critical concept in statistical analysis, as it provides insight into the magnitude and practical significance of observed differences beyond what p-values can convey. While statistical tests like t-tests indicate whether results are statistically significant, effect size measures allow researchers to understand the size of those differences in a meaningful context.

Cohen’s d

Cohen’s d is a widely used measure of effect size for comparing two group means. It quantifies the difference between the means of two groups in terms of standard deviations, providing a standardized metric that is easily interpretable. Cohen’s d is calculated by taking the difference between the two means and dividing it by the pooled standard deviation of the two groups. The resulting values are interpreted as follows:

- Cohen’s d of 0.2 indicates a small effect,
- Cohen’s d of 0.5 signifies a medium effect,
- Cohen’s d of 0.8 or greater is considered a large effect.

This measure even if a p-value indicates statistical significance, a small Cohen’s d might suggest that the difference is not substantial in real-world terms.

Code description:

The provided code implements a function to perform a two-sample t-test, comparing the means of two conditions extracted from a given input matrix. The function *“perform_ttest”* accepts parameters that include the app context, the input data matrix, and the two conditions to be compared.

Initially, the function extracts the data for the specified conditions from the input matrix, focusing on the relevant columns that start from data_start. It then flattens the data for each condition into one-dimensional arrays for easier comparison. Following this, the t-test is conducted using the MATLAB *“ttest2*” function, which returns the p-value and a statistics structure containing the t-statistic.

The t-statistic, which measures the size of the difference relative to the variation in the sample data, is extracted from the statistics structure. Additionally, the function computes Cohen's d to quantify the effect size of the difference between the two conditions. This is achieved by calculating the means and standard deviations of the two conditions and determining the pooled standard deviation.

ANOVA (Analysis of Variance):

ANOVA is used when comparing the means of three or more groups. Unlike multiple t-tests, which increase the risk of Type I errors (false positives), ANOVA evaluates whether there is a significant overall difference among group means. It does this by analyzing the ratio of the variance between the group means to the variance within the groups. When conducting a ANOVA test, two primary statistics are calculated:

- F-Statistic: The F-statistic is derived from the ratio of the variance between group means to the variance within the groups. A high F-statistic suggests that the differences between the group means are larger than the differences observed within each group, indicating a significant effect. It essentially measures the degree to which group means vary relative to the variability in the data.

- P-Value: Similar to the t-test, ANOVA provides a p-value to indicate whether any observed differences among the group means are statistically significant. A p-value below 0.05 (if alpha level is 5%) suggests that at least one group mean is significantly different from the others.

Effect size for ANOVA:

As mentioned before, Statistical tests like t-tests and ANOVA reveal whether results are statistically significant, but effect size measures provide researchers with a meaningful context to understand the magnitude of those differences.

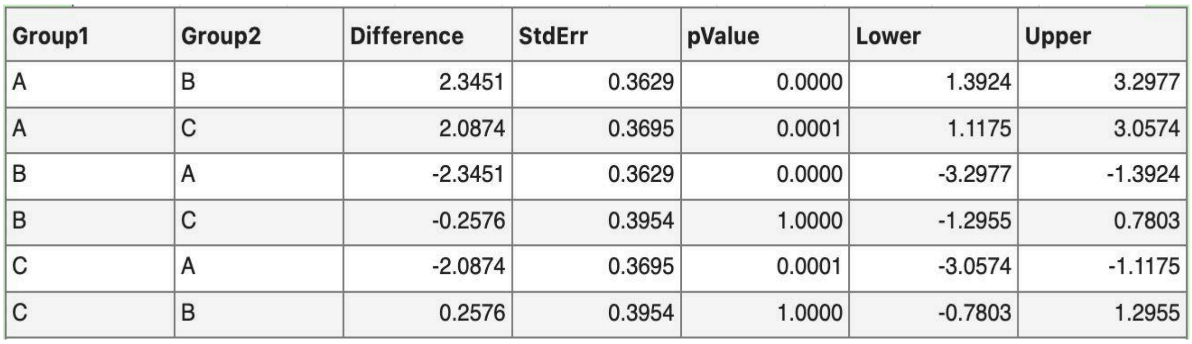
Eta Squared (η^2)

Eta squared is another effect size measure, primarily used in the context of ANOVA. It represents the proportion of total variance in the dependent variable that can be attributed to the independent variable (or grouping factor). This metric offers insight into how much of the variability in the observed data is explained by the group differences. Values for eta squared are typically interpreted as follows:

- η^2 of 0.01 indicates a small effect
- η^2 of 0.06 indicates a medium effect
- η^2 of 0.14 indicates a large effect

Post-Hoc Analysis:

Post-hoc analysis is crucial in ANOVA because it allows us to determine which specific groups are significantly different from each other. This is done after finding a significant result in ANOVA and involves running pairwise comparisons between the groups, similar to t-tests but with adjustments (like Bonferroni) to control for the increased risk of Type I errors (false positives). By performing post-hoc tests, we avoid the problem of inflating the error rate that arises when conducting multiple individual t-tests on the same dataset. Here's a detailed breakdown of the post-hoc analysis output:



Group1	Group2	Difference	StdErr	pValue	Lower	Upper
A	B	2.3451	0.3629	0.0000	1.3924	3.2977
A	C	2.0874	0.3695	0.0001	1.1175	3.0574
B	A	-2.3451	0.3629	0.0000	-3.2977	-1.3924
B	C	-0.2576	0.3954	1.0000	-1.2955	0.7803
C	A	-2.0874	0.3695	0.0001	-3.0574	-1.1175
C	B	0.2576	0.3954	1.0000	-0.7803	1.2955

Figure18. Result of post-hoc analysis

As shown in Figure 18, the initial two columns in the post-hoc results table indicate the groups being compared, with each group representing a specific condition of the experiment. This comparison is essential for determining whether there are statistically significant differences between the groups. The "Difference" column captures the mean difference in the dependent variable between the two groups being analyzed (e.g., for groups A and B). A positive difference suggests that the mean of Group 1 is greater than that of Group 2, whereas a negative value indicates that Group 2 has a higher mean.

The "Standard Error" column reflects the variability of the mean difference estimate. A lower standard error signifies a more reliable estimate. For example, in the row comparing groups A and B, the standard error is shown as 0.3629, which indicates a reasonably precise estimate of the mean difference.

The "p-value" reveals the likelihood that the observed mean difference occurred by random chance, given that the null hypothesis (which posits no difference between the groups) is accurate. A low p-value (typically below 0.05) indicates a statistically significant difference between the two groups. In the case of groups A and B, the p-value is 0.0000, confirming a significant difference.

"Confidence Interval", These values define the range within which the true mean difference between the groups is likely to fall, with a certain degree of confidence (usually 95%). If the confidence interval does not contain zero, this further supports the idea that the difference between the groups is statistically significant. In the first row (Group A vs. Group B), the lower bound is 1.3924 and the upper bound is 3.2977, meaning that the true difference is likely somewhere between these values. Since zero is not within this range, it suggests that there is a significant difference between the groups.

Final result for post-hoc Table as it is shown in Fig18:

- Group A differs significantly from both Group B and Group C (as indicated by the low p-values and confidence intervals that do not include zero).
- Group B and Group C, however, do not differ significantly (p-value of 1, indicating no significant difference between them).

Code description:

The "*ANOVA_test*" function performs a repeated measures ANOVA analysis on a given dataset within the context of a user interface application. Initially, it initializes a matrix called subject_means to store the average values for each subject across different conditions. It checks that the number of conditions does not exceed the provided condition names, ensuring data integrity.
The function then iterates through each condition and subject to calculate the mean for the respective condition. It extracts data from the matrix_input where the first column indicates the condition and the second column indicates the subject. If data for a specific subject in a condition is missing, a warning message is

generated, and a NaN value is recorded for that subject's mean. The means are aggregated into a table format, with condition names as column headers.

After computing the subject means, the function defines a repeated measures model and fits it to the data using the "*fitrm*" function. It then conducts the repeated measures ANOVA with the "*ranova*" function, extracting the F-statistic and p-value, which are displayed in designated text areas of the app.

Additionally, the function computes Partial Eta Squared as an effect size measure by calculating the sum of squares for both the effect and error. This value is also presented in the user interface. For post-hoc analysis, the function performs multiple comparisons using the "*multcompare*" function, ensuring the results are displayed in a user-friendly table format. It replaces condition numbers in the post-hoc results with the corresponding condition names, allowing users to easily interpret the outcomes of the analysis.

3.5.2 Final Plot

The plot shows the average knee angle for three groups (Conditions A, B, and C) over a normalized time frame, with shaded areas representing the variability in the data (standard deviation). The goal of this visualization is to compare the knee movement patterns across the three groups and highlight the differences in their average behavior and variability. By incorporating the shaded regions, the plot not only displays the mean trends but also offers insights into the consistency of the data for each condition. This approach allows us to observe both central tendencies and the range of fluctuations, making it easier to interpret the variations in knee angles between the groups.

Code Description:

The "*shadedPlot*" function in MATLAB visualizes average data for three groups (Conditions A, B, and C) with shaded areas representing standard deviations. It begins by clearing previous plots and holding the current axes for multiple datasets. The x-axis is normalized to range from 0 to 100 based on the input data.

For each condition, the function plots the mean data in a specific color and then creates a shaded area around the mean by calculating the upper and lower bounds (mean ± standard deviation). The shaded area, made semi-transparent using the `patch` function, visually represents variability.

After plotting, the axes are labeled, a title is added, and a legend is included to differentiate the conditions. The result is a plot that effectively shows both the average trends and the variability of the three groups over time, enabling easy comparison.

3.6 Compare parameters Tab

In the "Compare Parameters" tab, users are provided with an insightful interface to compare and analyze different parameters of their dataset, either subject-wise, group-wise, or across the entire dataset. This tab features a plot of the mean data along with key parameters, accompanied by a table that displays various statistical metrics. The goal of this tab is to allow users to comprehensively evaluate the data in terms of both overall trends and specific characteristics, facilitating deeper comparisons across subjects and conditions.

3.6.1 Plot of Mean Data and Parameters

The plot visualizes the mean data for selected subjects, conditions, or the entire dataset. The x-axis represents normalized time, while the y-axis shows the averaged values of the selected data. Key points like the minimum and maximum values, inflection points, and slope values are highlighted directly on the graph, providing visual cues about the most important features of the data. Users can select specific subjects or groups, and the mean data will automatically update to reflect the selected parameters.

Code Description:

- The function "*plotMeanDataAndParameters*" generates the plot. It takes in the mean data and calculated parameters like min/max values, slopes, and inflection points.
- The x-axis represents normalized time (from 0 to 100%), and the mean data is plotted as a solid black line.
- Min/Max values are marked on the plot with red and green circles, representing the lowest and highest points in the data respectively.
- Inflection points are highlighted with yellow circles, showing points where the curvature of the data changes, indicating transitions or turning points.
- Slopes (both min and max) are calculated by taking the first derivative of the data (using gradient) and shown as blue and magenta tangent lines, extending from the respective points of steepest change. The tangent lines help users visualize the steepest downward (min slope) and upward (max slope) transitions in the dataset.

3.6.2 Comparing Test Parameters Table

Below the plot, a table displays the key parameters of the data for deeper numerical comparison. This table includes the following metrics for the selected condition(s) and subject(s):

- Mean and Standard Deviation: For statistical evaluation of the overall trend and variability in the data.
- Minimum and Maximum Values, and Their Indices: These values provide the extremities of the dataset, helping users locate the most significant points.
- Minimum and Maximum Slopes, and Their Indices: The steepest changes in the data, which may indicate critical transitions or anomalies.
- Inflection Points and Their Indices: Providing insight into points of curvature change, which are important for understanding dynamic behaviors in the data.

Users can select data either by subject, group, or all data to see how these key parameters vary across different categories. By visualizing and tabulating this information, users can obtain a comprehensive understanding of the dataset and its key characteristics, all within a single, integrated interface.

Code Description:

The function “*getTableData*” handles the filtering and display of the parameters. Based on user input (selected condition and subject), a logical search is performed to extract the relevant rows from the parameters array cell_parameters. This search can filter data by condition, subject, or both. If no specific selection is made, the function returns the parameters for all subjects and conditions. The table columns display the condition, subject, mean, standard deviation, min/max values with their indices, min/max slopes with their indices, and inflection points with their corresponding indices.

The function “*convertTableData*” processes the table data for display. It converts the first column (condition) from numeric values to meaningful condition names (e.g., 'A', 'B', 'C'). The inflection points and their indices are rounded to 4 decimal places for readability and all numerical values are converted to strings so they can be displayed clearly in the table, ensuring that arrays or other data types are handled correctly for display purposes.

4.Discussion

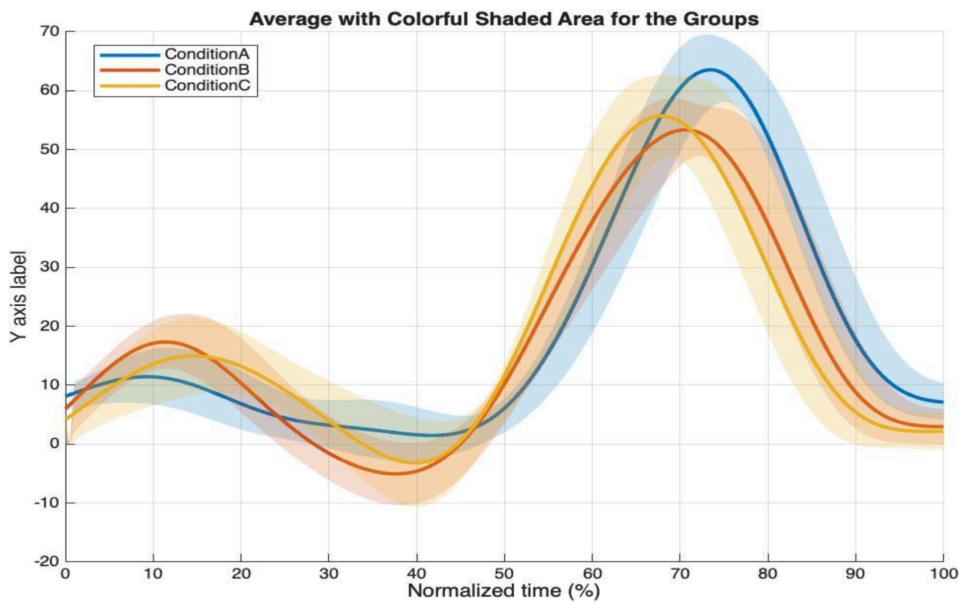


Figure 19. Average with colorful shaded area for each condition

The Final plot shown in Figure 19, illustrates a detailed comparison of knee angle data across three distinct conditions, labeled as Groups A, B, and C. Each group reflects different gait conditions, exhibiting variations in the knee's movement patterns, including the height of peak knee angles, the timing of these peaks, and the general behavior of the knee during the gait cycle. These variations arise from a combination of different factors, such as the specific gait phases and external influences like speed or surface texture.

As mentioned earlier, there are inherent variations in the datasets recorded from different parts of the body, such as the knee angle. These variations are not unique to this case but can also occur across different steps for the same subject, as the timing of events like heel strike or heel off can vary within a certain range (5). In addition to general fluctuations, more significant differences can be observed. These differences include the peaks and their timing, as well as the overall range of fluctuations, as shown in Figure 19. Generally, these variations can be summarized as follows:

- Global Max Points:
    - Condition A: The highest peak, occurring the latest.
    - Condition B: The lowest peak, occurring before Condition A.
    - Condition C: Intermediate peak, occurring after Condition B.
- Stance Phase Characteristics:
    - Condition A: Smoother stance phase with fewer noticeable peaks.
    - Conditions B and C: More pronounced peaks, with Condition B showing the lowest peak.

- Start and End Points:
    - Start points: B > A > C
    - End points: A > B > C

Several factors could account for these variations, which are explained below:

4.1 Different surface textures or slopes

The complexity of the task can affect the gait cycle, potentially requiring greater knee flexion or a quicker response. In (6), a study investigated the effects of walking surface and slope on knee joint angles. The study compared walking on even, flat surfaces with walking on grass and cobblestone. The results showed that datasets acquired from walking on grass had, on average, a higher global maximum in the gait cycle compared to walking on flat surfaces.

The study also compared the effects of different slopes—uphill and downhill—with flat surfaces. The average knee angle for uphill walking was higher than flat walking at the start of the gait cycle, and this difference remained until the end of phase 2, with a higher endpoint for the uphill surface as well. In contrast, for downhill walking, the start and end points were similar to those of flat walking, but the rest of the data showed higher average values.

4.1.1 Reasons for rejecting this categorization

Based on the findings of this study, it is unlikely that the dataset provided was recorded on different surfaces or slopes. Changes in surface texture generally only result in slight alterations to the global maximum, which does not fully explain the variations seen in the given dataset. Our dataset shows significant differences in the timing of peaks and in the stance phase, which were not observed in the slope data. While the downhill dataset exhibited differences in all peaks, the overall fluctuations remained consistent, which is not the case for our dataset.

4.2 Subject's Conditions and Intensity Levels

Various musculoskeletal disorders affecting bones, joints, and muscles can significantly alter movement patterns. Differences observed in the averages of gait cycle datasets may stem from such conditions. In (7), knee flexion during the gait cycle was examined for different severities of knee osteoarthritis. The study compared three levels of the disease: mild, moderate, and severe. As the intensity of knee osteoarthritis increased, the total range of knee angle movement decreased. However, the overall trend remained similar across all levels, with peaks occurring at approximately the same times, showing no significant shift in their timing.

Another relevant study, (8), examined the impact of achondroplasia on the gait cycle. It was found that children with achondroplasia exhibit less fluctuation in the stance phase. Despite this, the timing of peak occurrences remains largely unchanged, showing no significant difference from those without the condition.

4.2.1 Reasons for rejecting this categorization

This explanation is also unlikely to apply to the given dataset. Knee osteoarthritis, even in its severe form, only introduces slight differences in the gait cycle. Unlike the dataset we are analyzing, where peak timings vary across conditions, subjects with different levels of knee osteoarthritis show no change in peak timing. Similarly, the study on achondroplasia does not apply to our dataset, as it does not account for the timing variations observed in the given conditions.

4.3 Different Walking Speeds

Variations in walking speed can significantly influence the gait cycle. Numerous studies have shown that changes in speed not only affect peak magnitudes—making them higher or lower—but can also shift the timing of these peaks. In Figure 20, it is evident that higher walking speeds generally result in a higher global maximum peak, along with a slight shift toward the later stages of the gait cycle(9).

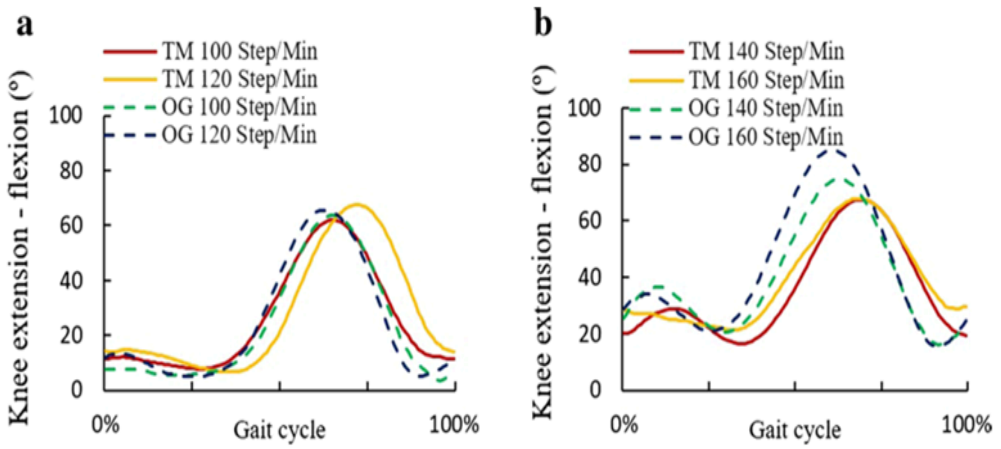


Figure 20. Knee flexion angle acquired from different speeds over-ground or on the treadmill

The interpretation for each condition can be summarized as follows:

Condition A	-Characteristics: Exhibits the highest global maximum, which occurs later in the gait cycle, with a smoother stance phase. -Inferred speed: Fast -The higher peak and its later occurrence suggest a controlled, possibly faster pace, as the subjects might be focusing on maintaining balance and stability while achieving maximum knee flexion.
Condition B	-Characteristics: Displays the lowest global maximum, with a later peak compared to Condition C. -Inferred speed: Slow -The lowest maximum peak indicates that subjects in this condition are walking at a slower speed.

Condition C	-Characteristics: Shows a global maximum lower than Condition A but higher than Condition B. -Inferred speed: Moderate -When compared to the other conditions, Condition C corresponds to a moderate walking speed, based on the global maximum peak.

The height of the global maximum was prioritized when assigning walking speeds to each condition. This decision is supported by studies showing that higher walking speeds typically result in higher global maxima (10). However, there are varying conclusions about the timing of the peaks (11). Additionally, in Condition A, the smoother stance phase—despite expectations of greater fluctuations at higher speeds (12)—suggests that the subjects may have been using a treadmill. As shown in Figure 20, data recorded at higher speeds on a treadmill can result in a smoother stance phase.

In conclusion, based on the characteristics of the global maxima and the timing of the peaks, walking speed can be inferred for each condition. Condition A likely represents a fast pace, Condition B a slow pace, and Condition C a moderate pace. The smoother stance phase observed in Condition A may be due to treadmill use, which can affect gait cycle dynamics, even at higher speeds. Prioritizing the height of the global maxima helps in understanding the relationship between speed and gait cycle changes across conditions.

5. Reference

1. https://musculoskeletalkey.com/biomechanics-2/

2. On improving gait analysis data: heel induced force plate noise removal and cut-off frequency selection for Butterworth filter

3. Marker displacement data filtering in gait analysis: A technical note

4. Filtering biomechanical signals in movement analysis

5. GaitSense: A Potential Assistance for Physical Rehabilitation by Means of Wearable Sensors

6. Effects of outdoor walking surface and slope on hip and knee joint angles in the sagittal plane

7. A machine learning-based diagnostic model associated with knee osteoarthritis severity

8. Gait in children with achondroplasia – a cross-sectional study on joint kinematics and kinetics

9. Lower limb joint motion and muscle force in treadmill and overground exercise

10. Muscle mechanical advantage of human walking and running: implications for energy cost

11. Mimicking human-like leg function in prosthetic limbs

12. Kinematic analysis of human gait cycle