Indexing Videos Containing Human Motion in the Form of Dance

Master Thesis
Nicoletta Farabullini

Institut für Informatik
der
Universität Zürich

Database Technology Group
Prof. Dr. Michael Böhlen

Supervisor
Sven Helmer

Zürich
February 21, 2022

Contents

1 Introduction

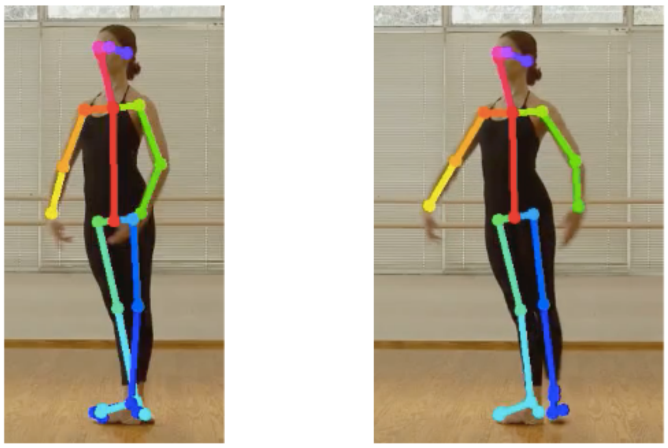
Many information retrieval systems make use of textual interfaces to extract information required by users. This is an efficient method for analyses of text-based collections, however textualizing a query or content in other application domains does not work as well. For instance, describing a wide range of human poses with words is very difficult. A first approach to solving this issue was implemented in a previous MSc project ("Retrieval and Visual Analysis of Dance Moves") at UZH, where Vasiliki Arpatzoglou and Artemis Kardara extracted data in form of time series containing stick figure models of a dancer in each frame and performed some preliminary similarity measures. Subsequently, a User Interface was built to visualize and compare at most two videos at a time.

The goal of this Master Thesis is to build up on this work by investigating different similarity measures and building an indexing structure to achieve a faster retrieval process time. Consequently, brute force methods for DTW (Dynamic Time Warping) with Euclidean Distance and Cosine similarity were first evaluated by comparing videos clustering groups derived from these analyses with a ground truth from the Master Project documentation. Because of the multi-dimensionality of the input data, these two methods were applied by evaluating angles vectors of individual body poses and individual body parts. The best performing method resulted to be the DTW with Euclidean Distance for angles vectors of individual body parts. Next, the lower bounding technique proposed by Keough (LB_Keough) was implemented; this method makes use of different Sakoe-Chiba lengths and Minimum Bounding Recatangles (MBRs) to obtain more accurate results with the best performing combination. This technique was applied both as a similarity measure as well as to create an indexing structure to obtain a candidate set of videos with respect to a given query. For the former application, utilizing this technique to create clustering groups of videos and evaluating the accuracy outputted a higher performance than the best performing brute force method with lower run time. In regards to the indexing structure, the LB_Keough method was used to create a candidate set of videos. This set was then run in combination with the DTW with Euclidean Distance for angles vectors of individual body parts to obtain a final set. Different combinations of values for the MBRs and Sakoe-Chiba lengths were evaluated to tune the index performance. It was observed that certain parameters combination of the LB_Keough already produced a satisfactory candidate set without the need for DTW. Subsequent analysis of run time and accuracy with respect to a set value for the MBRs and changing values for the Sakoe-Chiba length was performed to establish the trade-off. Results showed that lower Sakoe-Chiba lengths output an overall higher accuracy with lower run time. Therefore, it was concluded that using the LB_Keough method is sufficient to obtain accurate results in a small amount of time for clustering and querying of a video.

This Master thesis provides a solid foundation for fast analysis and comparison of motion videos. Further work will involve applying this work to different data sets. Additionally, implementation into a retrieval system to avoid the usage of computer main memory would be helpful.

2 Related work

This thesis develops on a previously completed Master project "DanceMoves: A Visual Analytics Tool for Dance Movement Analysis"[Vas21]. In this previous work, dance videos were divided into frames (25 per second), which were inputted in the "OpenPose" library in Python. This library overlaps a stylized figure on top of the dancer and extracts positions of body joints in the form of JSON files, which are then used to calculate vectors of 29 angles among different joints. For example, in one frame the dancer will assume a certain pose and a vector of 29 angles will be created from said position. Later in time, in a second frame the dancer will assume another pose and a new vector of angles will be created. A sample illustration is shown in Figure 1:



	P1	P2
Body part 1	95	94
Body part 2	80	82
Body part 3	176	177
Body part 4	103	105
Body part 5	137	141
Body part 6	113	116
Body part 7	174	174
Body part 8	126	123
Body part 9	111	117
Body part 10	178	165
Body part 11	70	77
Body part 12	125	136
Body part 13	82	86
Body part 14	177	178
Body part 15	88	86
Body part 16	173	177
Body part 17	142	144
Body part 18	153	151
Body part 19	46	68
Body part 20	95	98
Body part 21	51	51
Body part 22	107	105
Body part 23	56	56
Body part 24	45	44
Body part 25	87	88
Body part 26	95	94
Body part 27	168	170
Body part 28	94	98
Body part 29	86	86

Figure 1: Stylized figure for two dance poses

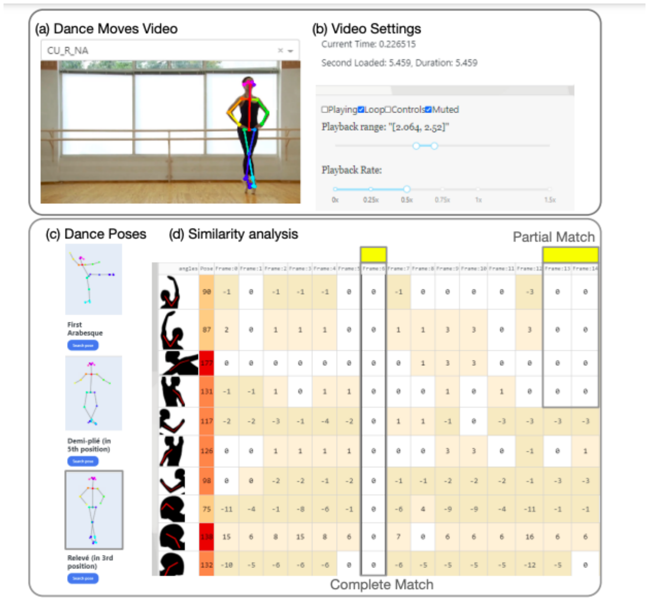
Where each body part entry corresponds to the following angles:

1. angle from nose to neck to left shoulder,
2. angle from nose to neck to right shoulder,
3. angle from left shoulder to right shoulder,
4. angle from left shoulder to left upper arm,
5. angle from left lower arm to left upper arm,
6. angle from right upper arm to right shoulder,
7. angle from right upper arm to right lower arm,
8. angle from left eye to nose to left ear to eye,
9. angle from left eye to nose to neck,
10. angle from nose to neck to right eye to nose,
11. angle from left eye to nose to right eye to nose,
12. angle from right eye to nose to right ear to eye,
13. angle from right hip to right upper leg,
14. angle from right upper leg to right lower leg,
15. angle from left hip to left upper leg,
16. angle from left upper leg to left lower leg,
17. angle from left lower leg left ankle to heel,
18. angle from right lower leg to right ankle to heel,
19. angle from right foot to right toes,
20. angle from right foot to right lower leg,
21. angle from right foot to right ankle to heel,
22. angle from left foot to left lower leg,
23. angle from left foot to left ankle to heel,
24. angle from left foot to left toes,
25. angle from torso to right shoulder,
26. angle from torso to left shoulder,
27. angle from torso to nose to neck,

28. angle from torso to right hip,

29. angle from torso to left hip

Each vector of angles comprises of 29 entries. By combining all vectors, a data frame is obtained where each row is a vector of angles for a specific body part. A general overview of the process is illustrated in Figure 2 [Vas21]:



Current Time: 0.226515

Second Loaded: 5.459, Duration: 5.459

☐Playing ☑Loop ☐Controls ☑Muted

Playback range: "[2.064, 2.52]"

angles	Pose	Frame:0	Frame:1	Frame:2	Frame:3	Frame:4	Frame:5	Frame:6	Frame:7	Frame:8	Frame:9	Frame:10	Frame:11	Frame:12	Frame:13	Frame:14
	90	-1	0	-1	-1	-1	0	0	-1	0	0	0	0	-3	0	0
	87	2	0	1	1	1	0	0	1	1	3	3	0	3	0	0
	177	0	0	0	0	0	0	0	0	1	3	3	0	0	0	0
	131	-1	-1	1	0	1	1	0	0	0	1	0	1	0	0	0
	117	-2	-2	-3	-1	-4	-2	0	1	1	-1	0	-3	-3	-3	-3
	126	0	0	1	1	1	1	0	0	0	3	3	0	-1	0	1
	98	0	0	-2	-2	-1	-2	0	-1	-1	-2	-2	-2	-1	-3	-3
	75	-11	-4	-1	-8	-6	-1	0	-6	4	-9	-9	-4	-11	-1	-1
	138	15	6	8	15	8	6	0	7	0	6	6	6	16	6	6
	132	-10	-5	-6	-6	-6	0	0	-6	-5	-5	-5	-5	-12	-5	0

Figure 2: Dance poses matrix

Two data frames were compared against each other by first creating a DTW path with respect to body parts vectors of angles. Then, looping over the path, corresponding values in two data frames of two videos were used to compute cosine similarity and the results were appended to a list. The final similarity output was given by the mean of values in said list. Finally, a UI was created to view and compare videos.

Even though this work provided a solid beginning to the analysis of body movements in videos, further investigations need to be performed in regards to similarity measures and comparison of a query video with the dataset based on the best performing similarity technique. To evaluate the validity of each similarity method, outputs were compared against a ground truth.

3 Similarity measures

Different similarity measures can be used to compare vectors of angles. When comparing two videos, vectors of angles from each frame of one video have to be extracted and analyzed against vectors of the second, e.g. $\vec{x} = x_1$, x_2,..., x_n and $\vec{y} = y_1$, y_2,..., y_n. The two similarity techniques used in this work are:

Euclidean distance[Wikc]

$$ed(\vec{x}, \vec{y}) = \sqrt{\sum_i (x_i - y_i)^2} \tag{1}$$

Cosine similarity[Wika]

$$cosSim(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\mid \vec{x} \mid\mid \vec{y} \mid} = \frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}} \tag{2}$$

These two methods can bring similar results as they can be made proportional to each other

3.1 Proportionality of Euclidean Distance and Cosine Similarity

Cosine Similarity and Euclidean distance can be re-arranged to be proportional[Wika]. In fact, when looking at the squared Euclidean distance it is already possible to observe a mutual multiplication component:

$$ed(x, y)^2 = (\sqrt{(x-y)^2})^2 = (x-y)^2 = \sum_i (x_i - y_i)^2 = \sum_i x_i^2 + \sum_i y_i^2 - \sum_i 2 * x_i \cdot y_i \tag{3}$$

Dividing x and y by their respective sums:

$$ed(\hat{x}, \hat{y})^2 = \sum_i (\frac{x_i}{\sqrt{\sum_i x_i^2}} - \frac{y_i}{\sqrt{\sum_i y_i^2}})^2 \tag{4}$$

Folding it out:

$$ed(\hat{x}, \hat{y})^2 = \sum_i (\frac{x_i}{\sqrt{\sum_i x_i^2}})^2 + \sum_i (\frac{y_i}{\sqrt{\sum_i y_i^2}})^2 - \sum_i \frac{2 * x_i \cdot y_i}{\sqrt{\sum_i x_i^2} * \sqrt{\sum_i y_i^2}} \tag{5}$$

Re-positioning summations:

$$ed(\hat{x}, \hat{y})^2 = \frac{\sum_i x_i^2}{\sum_i x_i^2} + \frac{\sum_i y_i^2}{\sum_i y_i^2} - 2 * \frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i x_i^2} * \sqrt{\sum_i y_i^2}} \tag{6}$$

The first two components of the sum in Equation(6) can be simplified to 1:

$$ed(\hat{x}, \hat{y})^2 = 2 - 2 * \frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i x_i^2} * \sqrt{\sum_i y_i^2}} = 2 * (1 - \frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i x_i^2} * \sqrt{\sum_i y_i^2}}) \tag{7}$$

Equation(7) is directly proportional to Equation(2):

$$ed(\hat{x}, \hat{y})^2 = 2 * (1 - cosSim(x, y)) \tag{8}$$

In other words, the Euclidean distance can be expressed in terms of cosine similarity if the two vectors are normalized to unit length[Wika]. Therefore, it could be the case that they will bring similar results.

To observe the accuracy of these measures, it is not sufficient to perform a one-to-one comparison as similar movements might have not occurred at exactly the same time. The Dynamic Time Warping (DTW) method was implemented in combination with these two techniques to account for these differences in times.

4 Dynamic Time Warping (DTW)

The Dynamic Time Warping (DTW) is an efficient technique when comparing time series that are out of sync, i.e where a one-to-one comparison of data points would not output the correct similarity measure. The classical DTW makes use of the Euclidean distance to compare all entries of a time series against one another. The resulting time complexity is of O(n^2)[Wikb].

For example, two time series $x = x_1$, x_2,..., x_n and $y = y_1$, y_2,..., y_n need to compared against each other. They both have peaks of the same height but occurring at different times as for example in Figure 3:

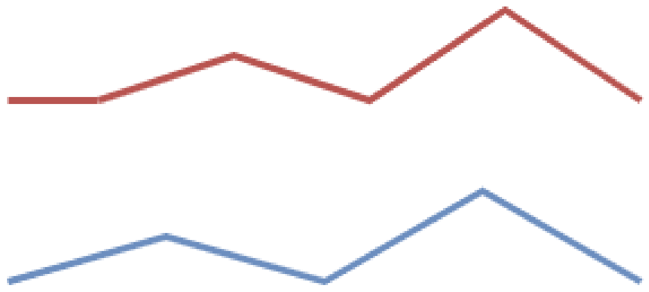


Figure 3: Time series

If only data points occurring at the same point in time were compared, low values for the Euclidean distance measures would be observed in very few spots. A visual illustration is shown in Figure 4:

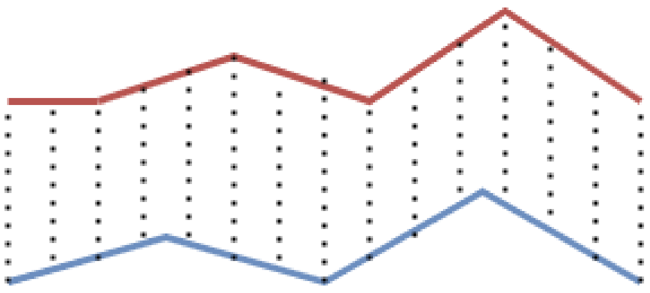


Figure 4: Time series comparison with basic Euclidean distance

Under these circumstances, the similarity between these two series would not be accurate as similar points in different times would not be considered. However, a comparison using the DTW technique highlights a higher number of similar data points as all entries of the two time series are compared against each other. The resulting optimal comparisons are illustrated in Figure 5:

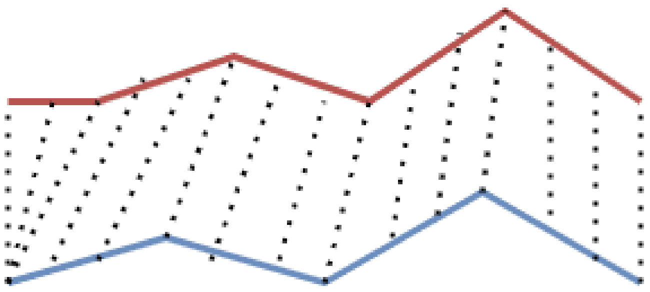
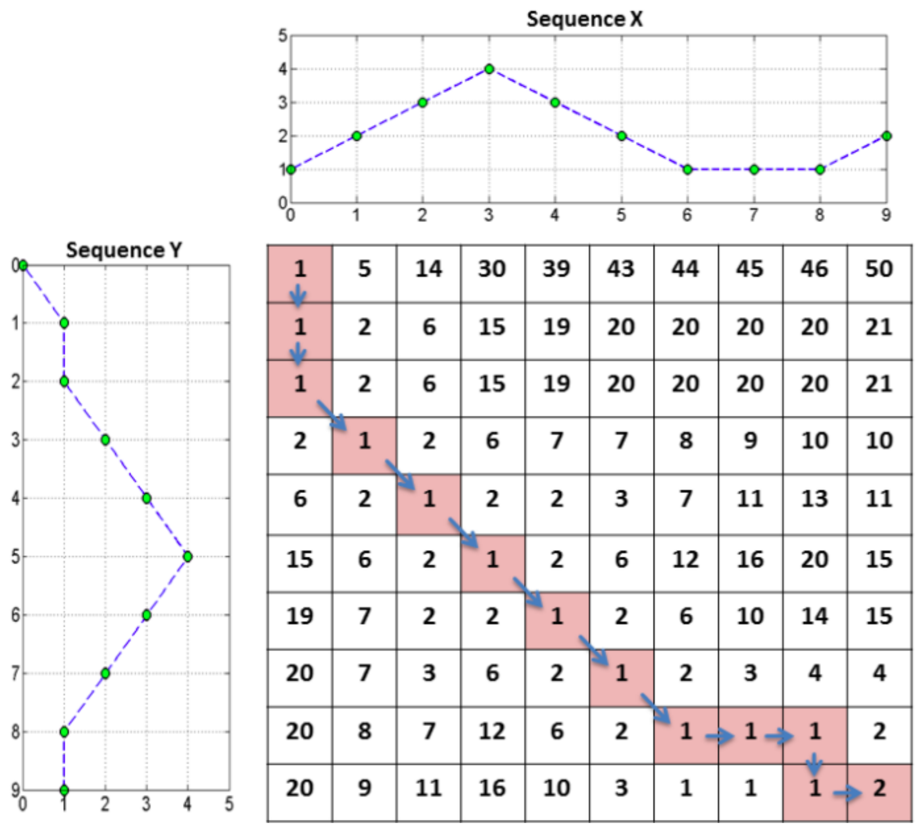


Figure 5: Time series comparison with DTW

This scenario brings a far more accurate similarity measure.

On a mathematical level, the DTW technique compares each point of the two time series, a subsequent distance matrix is constructed with all similarity outputs. By analysing values in this matrix, a path of minimum distances is highlighted and the sum of all entries in the path determines the similarity of the two time series. An example is illustrated in Figure 6[Ahm12].



1	5	14	30	39	43	44	45	46	50
1	2	6	15	19	20	20	20	20	21
1	2	6	15	19	20	20	20	20	21
2	1	2	6	7	7	8	9	10	10
6	2	1	2	2	3	7	11	13	11
15	6	2	1	2	6	12	16	20	15
19	7	2	2	1	2	6	10	14	15
20	7	3	6	2	1	2	3	4	4
20	8	7	12	6	2	1	1	1	2
20	9	11	16	10	3	1	1	1	2

Figure 6: DTW matrix example

On a computational level, the DTW algorithm uses dynamic programming to construct a matrix and extract the minimum distances among data points of the two time series (e.g. s and t). These data points are progressively summed[Wikb]:

```python
def dtw(s, t):
    n, m = len(s), len(t)
    # initialize DTW matrix
    dtw_matrix = np.zeros((n + 1, m + 1))
    for i in range(n + 1):
        for j in range(m + 1):
            dtw_matrix[i, j] = np.inf

    # set first value of matrix to 0
    dtw_matrix[0, 0] = 0

    # fill in matrix
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            cost = abs(s[i - 1] - t[j - 1])
            # take last min from a square box
            last_min = np.min([dtw_matrix[i - 1, j],
            dtw_matrix[i, j - 1],
            dtw_matrix[i - 1, j - 1]])
            dtw_matrix[i, j] = cost + last_min

    return dtw_matrix[n, m]
```

Listing 1: DTW_for_1D_time_series

The last entry of the DTW matrix is the final similarity measure between the two time series. If two time series are identical, this measure will output 0. On a more general level, the higher the measure the higher is, the more dissimilar the two series are.

This concept can be applied to this work as videos can be regarded as time series made of angles of vectors. While the classical DTW implementation makes use of the Euclidean distance, Cosine Similarity can also be used to calculate the DTW matrix. Because angles are an essential component of this project, it is paramount to investigate both similarity measures.

4.1 DTW with Cosine Similarity

Cosine similarity is a widely used technique to measure similarity between vectors of angles. Unlike Euclidean distance where values range from 0 to infinity, this method is bound to a range between 0 and 1 as multiplications of the vectors entries are divided by multiplications of the unit lengths as per Equation (2). The more similar the two vectors are, the closer to 1 the similarity output will be. Consequently, to make a more linear comparison of this method with Euclidean distance, it is paramount to subtract the final output from 1.

To implement Cosine Similarity in the DTW algorithm, it is necessary to replace the Euclidean distance portion of the code derived from Equation (1) with Equation (2):

```python
def dtwcosSim(s, t):
    n, m = len(s), len(t)
    # initialize DTW matrix
    # in the same way as for Euclidean Distance

    # fill in matrix
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            cosSim = (s[i - 1] * t[j - 1])/(n * m)
            cost = 1 - cosSim

            # fill in matrix
            # in the same way as for Euclidean Distance

    return dtw_matrix[n, m]
```

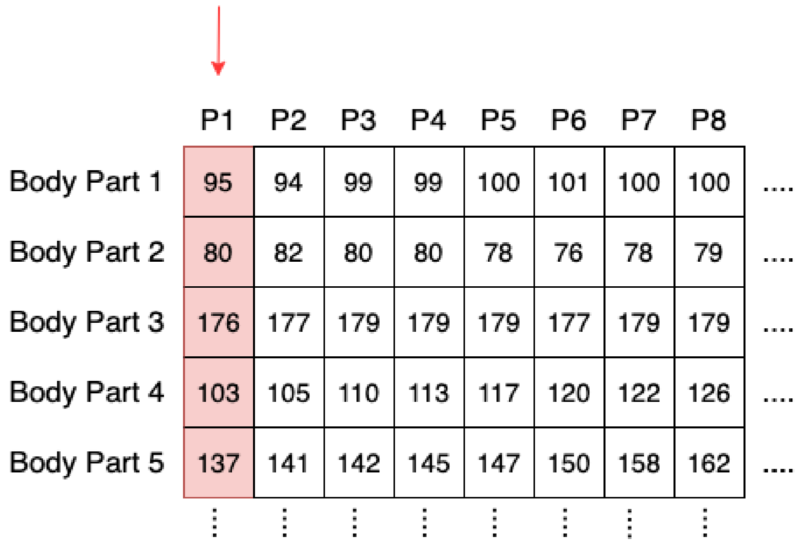
Listing 2: DTW_cosSim_for_1D_time_series

Up until now, only one-dimensional cases were considered, i.e. where only two time sequences need to be compared. However, the dance videos comprise of frames ranging between 21 and 133, with each frame including vectors of 29 angles. This new scenario requires a shift to a multi-dimensional framework.

5 Multi-dimensional DTW

The input dataset in this work contains 52 dance videos. Each video is divided into 25 frames per second, resulting in a number of frames for the dataset ranging between 21 and 133. Additionally, each frame comprises a vector of 29 body parts angles. This multi-dimensional scenario requires an adaptation of the DTW algorithm to account for changing number of multiple dimensions. Starting from the dataframe containing all angles of a video, two approaches were examined:

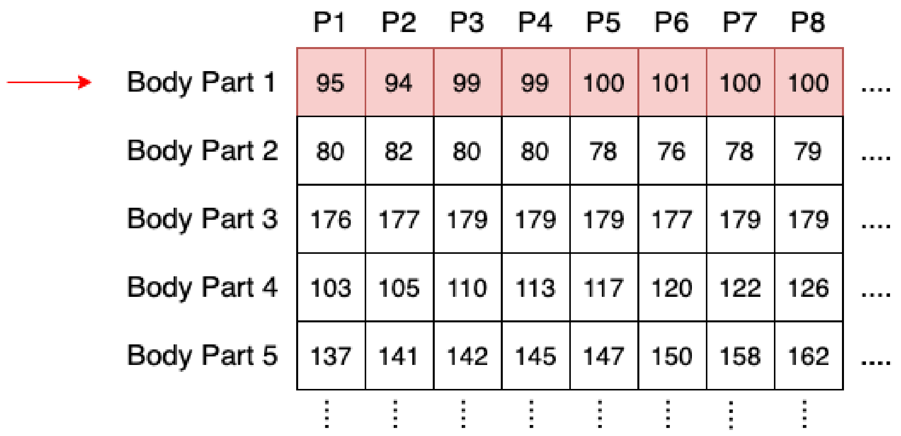
- Evaluating similarity measures for each body pose angles - each data point in the 1D time series is now a vector of 29 angles in the vertical direction of the data frame. An example can be viewed in 7, where P1 represents a body pose:



	P1	P2	P3	P4	P5	P6	P7	P8	
Body Part 1	95	94	99	99	100	101	100	100	....
Body Part 2	80	82	80	80	78	76	78	79	....
Body Part 3	176	177	179	179	179	177	179	179	....
Body Part 4	103	105	110	113	117	120	122	126	....
Body Part 5	137	141	142	145	147	150	158	162	....
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Figure 7: Video 1 - body pose angles - vertical approach

- Evaluating similarity measures for angles of individual body parts angles - each 1D time series is now a sequence of n angles (corresponding to the number of frames) in the horizontal direction of the data frame. An example can be viewed in 8, where Body Part 1 represents a vector of angles corresponding to a body part of the dancer:



	P1	P2	P3	P4	P5	P6	P7	P8	
Body Part 1	95	94	99	99	100	101	100	100	....
Body Part 2	80	82	80	80	78	76	78	79	....
Body Part 3	176	177	179	179	179	177	179	179	....
Body Part 4	103	105	110	113	117	120	122	126	....
Body Part 5	137	141	142	145	147	150	158	162	....
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Figure 8: Video 1 - body parts angles - horizontal approach

Before going into further explanation regarding these two approaches, it is important to establish an interpolation method to account for missing values in the dataframe and the evaluation technique to compute the accuracy.

5.1 Accounting for missing values

It is important to note that angles vectors might include missing values (NaNs). This can be for example due to a body part being hidden behind the dancer's body. Any mathematical operation involving NaNs in Python outputs nan as a result. For example, Table 1 shows two vectors of angles $\vec{x}$ and $\vec{y}$, where the latter has some missing values:

Table 1: Example of $\vec{x}$ and $\vec{y}$ vectors values

i	$\vec{x}$	$\vec{y}$
1	91	16
2	88	160
3	179	177
4	112	93
5	143	177
6	108	94
7	136	172
8	94	NaN
9	133	NaN
10	130	NaN
11	96	NaN
12	87	NaN
13	76	92
14	172	179
15	78	92
16	163	175
17	161	175
18	151	171
19	32	165
20	127	113
21	74	25
22	109	84
23	71	18
24	35	168
25	90	89
26	90	92
27	178	109
28	85	93
29	90	93

Here, the 8th, 9th, 10th, 11th, and 12th rows cannot be evaluated in the same way as other integer entries as the result will be NaN. Because the number of missing values is small compared to the total amount, these iteration steps can be skipped and the final sum will include computations from 24 entries instead of 29. Even though this is not an ideal scenario, an approximate similarity measure can still be extracted.

However, it can also be the case that no entry in the dataset can be evaluated as missing values are present in either row of the two vectors. An example is shown in Table 2:

Table 2: Example of $\vec{x}$ and $\vec{y}$ vectors with missing values

i	$\vec{x}$	$\vec{y}$
1	NaN	16
2	NaN	160
3	NaN	177
4	112	NaN
5	143	NaN
6	108	NaN
7	NaN	172
8	94	NaN
9	133	NaN
10	130	NaN
11	96	NaN
12	87	NaN
13	NaN	92
14	NaN	179
15	78	NaN
16	NaN	NaN
17	NaN	NaN
18	151	NaN
19	32	NaN
20	127	NaN
21	NaN	25
22	109	NaN
23	NaN	18
24	NaN	168
25	NaN	89
26	90	NaN
27	NaN	109
28	85	NaN
29	NaN	93

Under these circumstances, it is necessary to interpolate the missing value from the previous computation, i.e. the DTW matrix cell is filled with the entry corresponding to the same column but previous row. This interpolation method does not bring an exact similarity result, however it can be regarded as close enough to a precise output. The accuracy of the similarity measures with this interpolation method is evaluated by using these methods to create clustering groups of the videos and compare them to a ground truth.

5.2 Evaluation of similarity measure accuracy

In order to establish whether similarity measures are accurate, it is paramount to compare them with a ground truth. The ground truth for for dance videos clustering was inherited from the Master Project[Vas21]. Ideally, this list should have clearly stated which videos belonged to the different clusters, however some labeling was ambiguous. Additionally, a couple of mistakes were found and some videos listed in the ground truth were missing in the dataset. For these reasons, this list had to be filtered from 73 to 52 videos. The final list of videos is shown in Table 3:

Table 3: Final ground truth

Groups	Videos Labels
1	BS_F_BK, BS_F_LT, BS_F_RT BS_S_BK, BS_S_FT, BS_S_LT, BS_S_RT, SS_F_BK, SS_F_FT, SS_F_LT, SS_F_RT, SS_S_BK, SS_S_LT, SS_S_RT, SYN_R, SYN_S
2	LD_F_dis, LD_F_small, LD_S_dis, LD_S_small, LU_F_dis, LU_S_dis
3	BJ_FT ,SJ_FT
4	AR, TA, LU_F_big, LU_S_big
5	BJ_RT, SJ_RT, BJ_LT, SJ_LT
6	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
7	TB_F_FB, TB_S, TB_S_FB, TF_F, TF_S, TL_F, TL_S, TOS_F, TOS_S, TR_F, TR_S
8	SYN_K
9	BJ_BK, SJ_BK

The accuracy of the similarity methods in this work was evaluated based on how closely they resembled to this list.

Practically speaking, while videos are being compared against each other, a similarity matrix is filled with the final similarity measures. This similarity matrix is then used to create the clustering using the **AgglomerativeClustering** function from the Python sklearn library with the number of clusters set to 9, the affinity as 'precomputed' (since the similarity matrix was created without using any of the predefined function methods), and the linkage as 'average':

```python
from sklearn.cluster import AgglomerativeClustering

clustering = AgglomerativeClustering(
                n_clusters = n_clusters,
                affinity = 'precomputed',
                linkage = 'average'
            ).fit(similarity_matrix)
```

Listing 3: agglomerative_clustering

This function will output a numpy.ndarray of clustering labels:

```
>>>clustering.labels_
array([8, 5, 5, 5, 5, 1, 2, 2, 4, 5, 5, 6, 6, 6, 6, 6, 6, 6, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 2, 2, 4, 6, 6, 6, 6, 6, 6, 6, 7, 6, 3, 1,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Listing 4: clustering_groups

The numbers in this result represent which cluster each video belongs to. The computed clustering groups may be ordered differently from the ground truth, for this reason the following algorithm to make the comparison automated was developed:

- Merge clustering groups labels with list of video names in a data frame
- sort data frame with respect to clustering labels
- create a list with sublists, each sublist is a different clustering group (candidate list)
- extract sublist with highest number of elements from candidate list
- check what cluster from the ground truth said sublist has the highest number of matches with, i.e. the list with the highest number of mutual videos included
- Save number of matches and remove sublists from both ground truth and candidate list
- perform the same operation until the ground truth is empty

Additionally, the similarity matrix is used to create dendrograms using the **linkage**, **squareform**, and **dendrogram** functions from the scipy package by setting the method to 'average'.

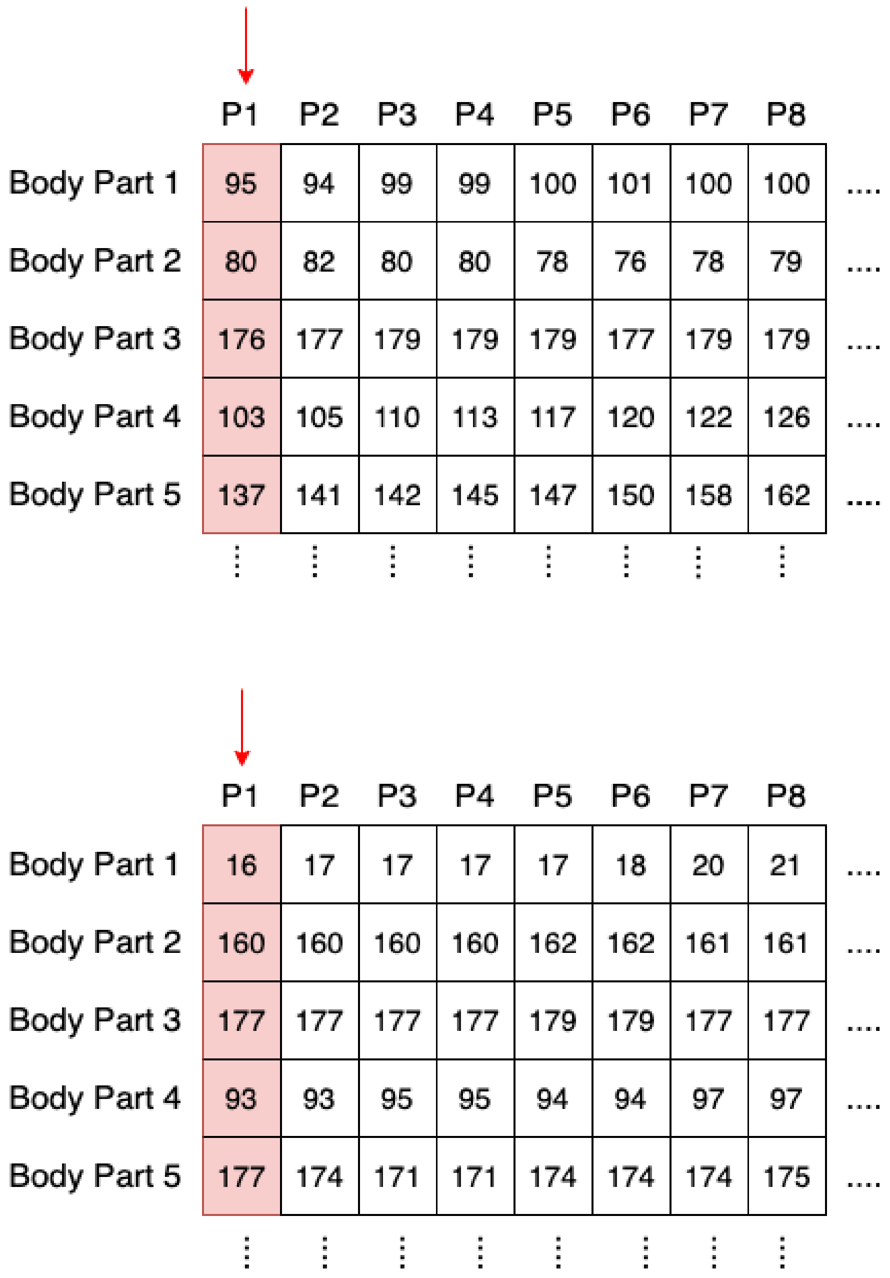
With this set-up, it is now possible to evaluate the accuracy of each similarity method.

6 DTW vertical for a multi-dimensional space

One approach to evaluate Euclidean Distance and Cosine Similarity for multiple dimensions is to extract the 29 angles for each frame (hence for each body position) of the videos and compare all frames of all videos against each other. In other words, the 1D time series from Section 4 are now replaced with angle vectors of 29 dimensions, i.e. one data point in the 1D series is now a vector of 29 angles. The algorithm proceeds as following:

- each entry of two single vectors of angles are extracted and their similarity evaluated through either simple Euclidean distance or Cosine Similarity
- the similarity values obtained from each iteration are progressively summed
- the final sum is divided by the number of computations (normally 29 by less in case of missing values)
- the summation result is used as one data point in the DTW matrix

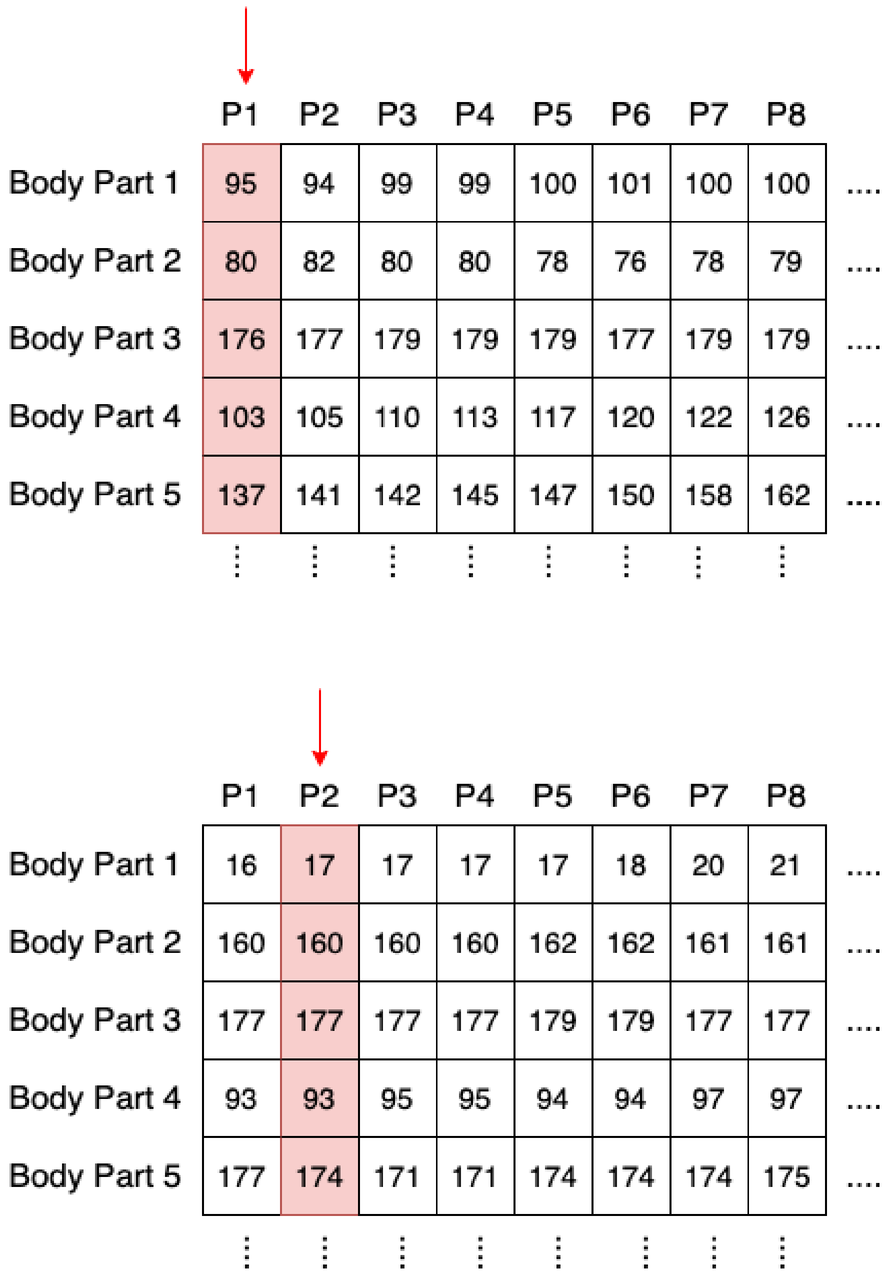
Picturing this scenario graphically, P1 angles of video 1 have to be individually compared with all positions angles of video 2 as illustrated in Figure 9 and 10:



	P1	P2	P3	P4	P5	P6	P7	P8	
Body Part 1	95	94	99	99	100	101	100	100	....
Body Part 2	80	82	80	80	78	76	78	79	....
Body Part 3	176	177	179	179	179	177	179	179	....
Body Part 4	103	105	110	113	117	120	122	126	....
Body Part 5	137	141	142	145	147	150	158	162	....
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

	P1	P2	P3	P4	P5	P6	P7	P8	
Body Part 1	16	17	17	17	17	18	20	21	....
Body Part 2	160	160	160	160	162	162	161	161	....
Body Part 3	177	177	177	177	179	179	177	177	....
Body Part 4	93	93	95	95	94	94	97	97	....
Body Part 5	177	174	171	171	174	174	174	175	....
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Figure 9: P1 of video 1 VS P1 of video 2



	P1	P2	P3	P4	P5	P6	P7	P8	
Body Part 1	95	94	99	99	100	101	100	100	....
Body Part 2	80	82	80	80	78	76	78	79	....
Body Part 3	176	177	179	179	179	177	179	179	....
Body Part 4	103	105	110	113	117	120	122	126	....
Body Part 5	137	141	142	145	147	150	158	162	....
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

	P1	P2	P3	P4	P5	P6	P7	P8	
Body Part 1	16	17	17	17	17	18	20	21	....
Body Part 2	160	160	160	160	162	162	161	161	....
Body Part 3	177	177	177	177	179	179	177	177	....
Body Part 4	93	93	95	95	94	94	97	97	....
Body Part 5	177	174	171	171	174	174	174	175	....
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Figure 10: P1 of video 1 VS P2 of video 2

For each comparison of two position angles vectors, similarity computations using Equation (1) and (2) are performed one-to-one for entries in the vectors placed at the same index position. Each similarity output is then placed in a cell of the DTW matrix.

The code implementation and results are slightly different with respect to each method.

6.1 DTW vertical with Euclidean distance for a multi-dimensional space

Adapting the original DTW algorithm to a multi-dimensional scenario requires adding a loop to evaluate the similarity measure to insert in the DTW matrix. Such loop applies Equation (1) to angles from the vectors corresponding to the same index, which are subtracted from each other and the result is squared. All results are iteratively summed. Finally, if it was possible to perform operations on the vectors, i.e. there were cases where the operation did not output NaN, the square root of the final result is taken and then divided by the number of times that the operations were computed. To account for missing values, a vector of similarity measures is updated on each iteration and its values used if all operations outputted NaN. The final DTW matrix result is divided by number of steps of the path to account for different videos having different lengths:

```python
def dtw_ed_MD(s, t):
    # number of columns in each dataframe
    # corresponding to the number of frames for each video
    n, m = len(s.columns), len(t.columns)

    # initialize DTW matrix
    # initialize vector to account for missing values

    # fill in matrix
    for i in range(1, n + 1):
        for j in range(1, m + 1):

            # set count and sum_entries to 0

            # multi-dimensional case
            for k in range(0, len(s[i - 1])):
                # check that operation does not output NA
                if (s[i - 1][k] * t[j - 1][k]) != NA:
                    sum_entries += (s[i - 1][k] - t[j - 1][k])^2
                    count += 1
            if count > 0:
                cost = sqrt(sum_entries)/count

                # fill in matrix
                # in the same way as for 1D case

                # fill in vector of values to account for missing
    data
                vec_vals[j] = cost + last_min
            else:
                dtw_matrix[i, j] = vec_vals[j]

    # divide by length of DTW path, aka the number of steps
    dtw_final = dtw_matrix[n, m]/n_steps
    return dtw_final
```

Listing 5: DTW_ED_for_MD

The resulting clustering groups and dendrogram are illustrated in Table 4 and Figure 11:

Table 4: Clustering for vertical DTW with Euclidean distance

Groups	Videos Labels
1	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
2	AR, BJ_BK, BJ_LT, BS_F_BK, BS_F_LT, BS_F_RT, BS_S_BK, BS_S_FT, BS_S_LT, BS_S_RT, LD_F_dis, LD_F_small, LD_S_dis, LD_S_small, LU_F_big, LU_F_dis, LU_S_big, LU_S_dis, SJ_BK, SJ_FT, SJ_LT, SS_F_BK, SS_F_FT, SS_F_LT, SS_F_RT, SS_S_BK, SS_S_LT, SS_S_RT, SYN_R, SYN_S, TA
3	TF_F, TF_S, TL_F, TL_S, TOS_F, TOS_S
4	BJ_RT, SJ_RT
5	BJ_FT
6	TB_F_FB, TB_S_FB
7	TR_F, TR_S
8	SYN_K
9	TB_S

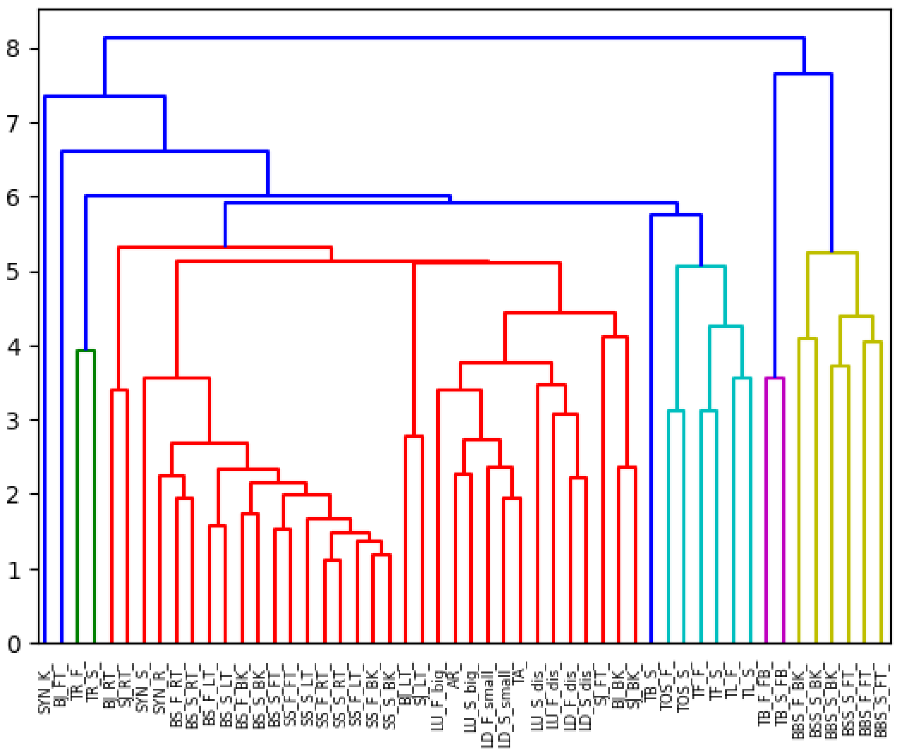


Figure 11: DTW ED dendrogram Vertical

Using this method, the total number of correctly clustered videos is 30 out of 52, i.e. the accuracy is 57.7% with a run time of approximately 5676 seconds (1.6 hours). The misclustering of the videos can be attributed to a couple of reasons:

- Some clusters from the ground truth include videos with somewhat similar poses, though they are different enough to be clustered separately. It could be that this approach is not sensitive enough to small changes in angles values as the vectors comprise values from the whole body for each frame. Consequently, similar videos that should be clustered separately are instead grouped together.
- Videos that could not be correctly evaluated due to missing values. In fact, the interpolation of the missing values might not always lead to a correct distance measure.

To evaluate whether this method is the best achievable result, it is important to compare it with Cosine Similarity.

6.2 DTW vertical with Cosine Similarity for a multi-dimensional space

Adapting the original DTW algorithm to a multi-dimensional scenario with Cosine Similarity requires adding a loop to evaluate the similarity measure to insert in the DTW matrix. Such loop applies Equation (2) to angles from the vectors corresponding to the same index, which are multiplied together. All results are iteratively summed and, if there were cases where the operation did not output NaN, divided by the multiplication of their unit lengths. To account for missing values, the same vector logic as in Section 6.1 is applied. The final DTW matrix result is divided by number of steps of the path to account for different videos having different lengths:

```python
def dtw_cosSim(s, t):
    # number of columns in each dataframe
    # corresponding to the number of frames for each video
    n, m = len(s.columns), len(t.columns)

    # initialize DTW matrix
    # initialize vector to account for missing values

    for i in range(1, n + 1):
        for j in range(1, m + 1):

            # set count, s_squared_sum, t_squared_sum, sum_entries
    to 0

            # multi-dimensional case
            for k in range(0, len(s[i - 1])):
                if (s[i - 1][k] * t[j - 1][k]) != NA:
                    sum_entries += s[i - 1][k] * t[j - 1][k]
                    s_squared_sum += s[i - 1][k]^2
                    t_squared_sum += t[j - 1][k]^2
                    count += 1

            if count > 0:
                denominator = sqrt(s_squared_sum)*sqrt(
    t_squared_sum)
                cost = sum_entries/denominator
                cost = 1 - cost

                # fill in matrix
                # in the same way as for 1D case

                # fill in vector of values to account for missing
    data
                vec_vals[j] = cost + last_min
            else:
                dtw_matrix[i, j] = vec_vals[j]

    # divide by length of DTW path, aka the number of steps
    dtw_final = dtw_matrix[n, m]/n_steps
    return dtw_final
```

Listing 6: DTW_CosSim_for_MD

The resulting clustering groups and dendrogram are illustrated in Table 5 and Figure 12:

Table 5: Clustering for vertical DTW with Cosine Similarity distance

Groups	Videos Labels
1	AR, BJ_BK, BJ_LT, BS_F_BK, BS_F_LT, BS_F_RT, BS_S_BK, BS_S_FT, BS_S_LT, BS_S_RT, LD_F_dis, LD_F_small, LD_S_dis, LD_S_small, LU_F_big, LU_F_dis, LU_S_big, LU_S_dis, SJ_BK, SJ_FT, SJ_LT, SS_F_BK, SS_F_FT, SS_F_LT, SS_F_RT, SS_S_BK, SS_S_LT, SS_S_RT, SYN_R, SYN_S, TA
2	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
3	TB_S, TF_F, TF_S, TL_F, TL_S
4	TB_F_FB, TB_S_FB
5	BJ_FT
6	SYN_K
7	TR_F, TR_S
8	BJ_RT, SJ_RT
9	TOS_F, TOS_S

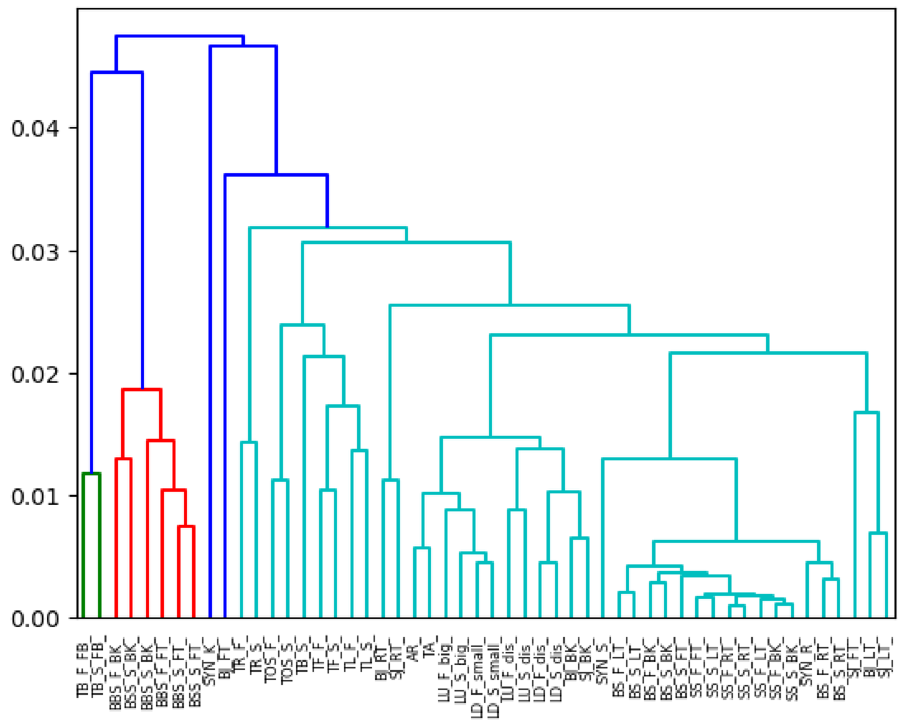


Figure 12: DTW cosSim vertical

Using this method, the total number of correctly clustered videos is 28 out of 52, i.e. the accuracy is 53.8% with a run time of approximately 8469 seconds (2.3 hours).

Comparing the clusterings of DTW with Euclidean distance with this method, it is possible to observe that they are very similar: only 3 videos are clustered differently, namely TOS_F, TOS_S and TB_S. Therefore, the same reasons for the misclustering as explicated in Section 6.1 can be attributed here to even a higher degree.

6.3 Past and future vectors combinations

One last approach to evaluate dance videos similarities in the vertical direction was to combine a set amount of vectors of angles and perform one-to-one comparison with respect to the indexes. In other words, given a position angles vector that needs to be analysed, this approach consists of combining a set amount of vectors preceding and an equivalent amount following said vector into a larger vector. The same action is performed for both videos and the entries of the resulting vectors are compared one-to-one by means of Euclidean distance.

For example two videos need to be compared, setting the number of preceding and following frames to 2, it is possible to analyse P4 by combining it with P2, P3, P5, and P6 as illustrated in Figure 13:

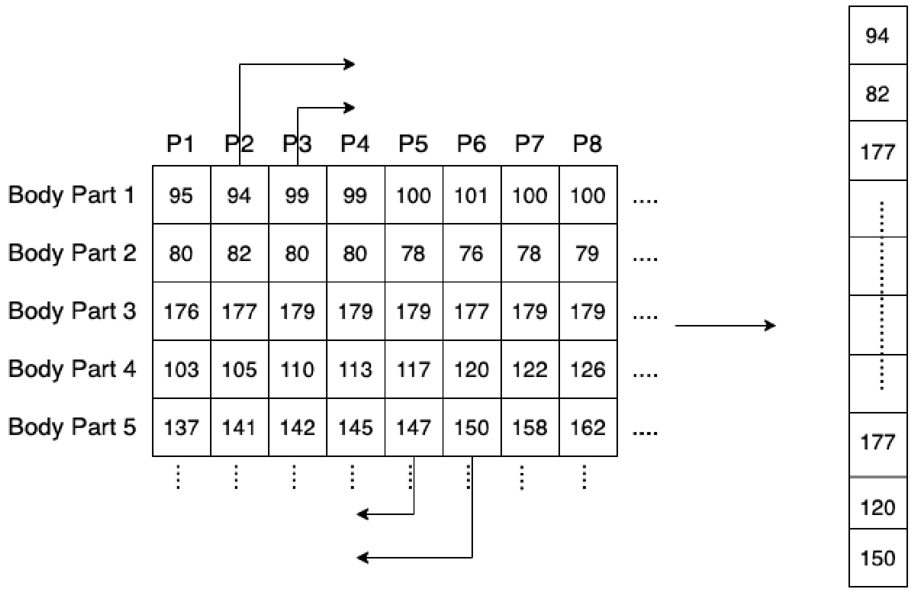


Figure 13: Past and Future vectors

The same process is performed for the second video with respect to the same pose. These two new vectors are then compared by taking the Euclidean distance of same index entries (i.e. one-to-one comparison of the angles). In this work, the number of frames was set to 6. The resulting clustering groups and dendrograms are illustrated in Table 6 and Figure 14:

Table 6: Clustering for Past and Future vectors

Groups	Videos Labels
1	AR, BJ_BK, BJ_FT, BJ_LT, BJ_RT, BS_F_BK, BS_F_LT, BS_F_RT, BS_S_BK, BS_S_FT, BS_S_LT, BS_S_RT, LD_F_dis, LD_F_small, LD_S_dis, LD_S_small, LU_F_big, LU_F_dis, LU_S_big, LU_S_dis, SJ_BK, SJ_FT, SJ_LT, SJ_RT, SS_F_BK, SS_F_FT, SS_F_LT, SS_F_RT, SS_S_BK, SS_S_LT, SS_S_RT, SYN_R, SYN_S, TA, TF_S, TL_S, TOS_S, TR_S
2	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
3	TF_F, TL_F
4	TOS_F
5	SYN_K
6	TB_F_FB
7	TR_F
8	TB_S
9	TB_S_FB

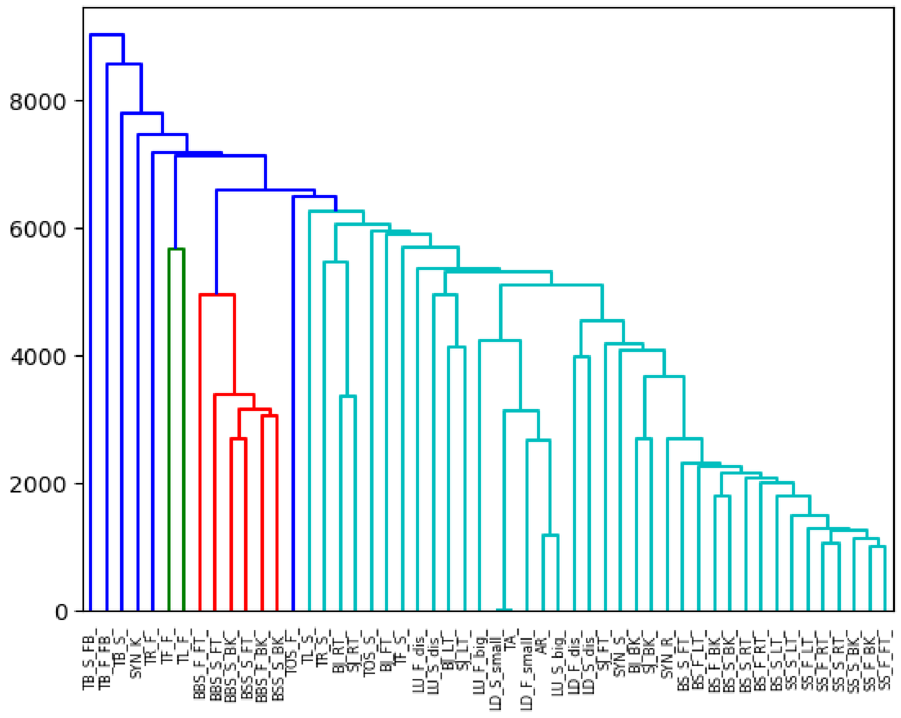


Figure 14: Past and future vectors

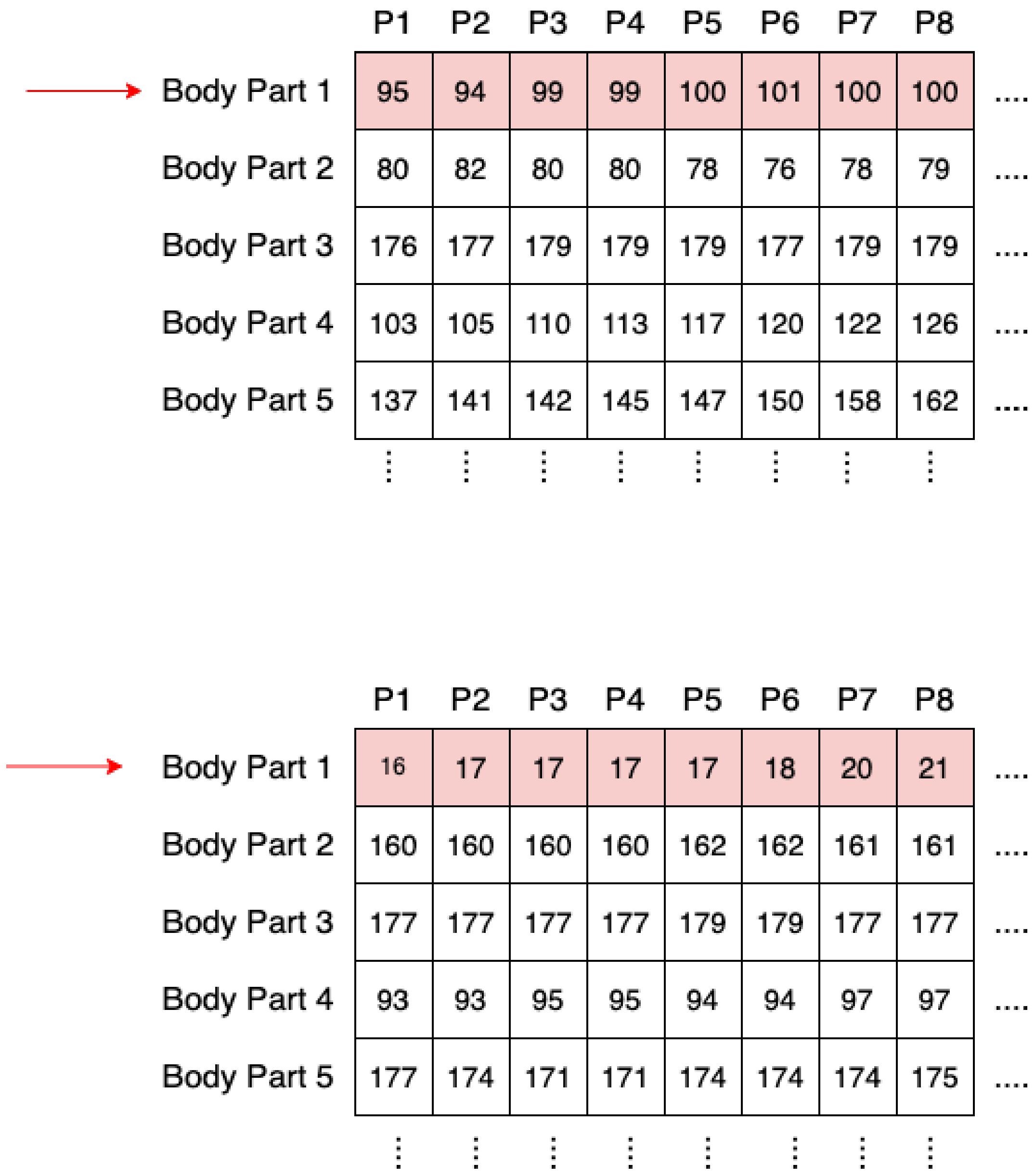
Using this method, the total number of correctly clustered videos is 24 out of 52, i.e. the accuracy is 46.1% with a run time of approximately 576 seconds (9.6 minutes). This method brings a even lower accuracy as it is a simplified approach to the one shown in Section 6.2. However, due to the absence of the DTW matrix implementation, the run time is much lower.

The methods in this section bring at best an accuracy of 57.7% for the DTW with Euclidean distance vertical approach. It is therefore necessary to perform an analogous analysis for body parts angles instead of positions angles, i.e. analysing the data frame horizontally.

7 DTW horizontal for a multi-dimensional space

Performing similarity measures of DTW with Euclidean distance and Cosine similarity vertically, i.e. taking the vector of body parts angles for each frame, outputs quite different results compared to the ground truth. A better approach involves performing similarity measures horizontally, i.e. the DTW matrix is not constructed with the vectors of angles of body positions but with the angles for each individual body part.

Picturing this scenario graphically, body part 1 angles of video 1 have to be compared with body part 1 angles of video 2. The same concept applies for all remaining body parts: body part 2 angles of video 1 have to be compared with body part 2 angles of video 2 and so on. The DTW matrix is constructed using these two angles vectors as illustrated in Figure 15.



	P1	P2	P3	P4	P5	P6	P7	P8	
Body Part 1	95	94	99	99	100	101	100	100	....
Body Part 2	80	82	80	80	78	76	78	79	....
Body Part 3	176	177	179	179	179	177	179	179	....
Body Part 4	103	105	110	113	117	120	122	126	....
Body Part 5	137	141	142	145	147	150	158	162	....
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

	P1	P2	P3	P4	P5	P6	P7	P8	
Body Part 1	16	17	17	17	17	18	20	21	....
Body Part 2	160	160	160	160	162	162	161	161	....
Body Part 3	177	177	177	177	179	179	177	177	....
Body Part 4	93	93	95	95	94	94	97	97	....
Body Part 5	177	174	171	171	174	174	174	175	....
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Figure 15: Body part 1 of video 1 VS body part 1 of video 2

It is important to note that body parts angles are compared one-to-one, i.e. it is not possible to compare body part 1 angles of video 1 with body part angles 2 of video 2 as they are different components of the body hence not relatable to one another.

7.1 DTW horizontal with Euclidean distance for a multi-dimensional space

The set up to evaluate two videos by angles of vectors of body parts instead of angles of vectors of positions is quite different. Here, a DTW matrix is created for each body part, where individual entries of each vector of angles are used directly to evaluate similarity measures with Euclidean Distance. The final entry for each DTW path is divided by the number of steps to account for different lengths in videos. Finally, all body parts angles outputs are summed together.

```python
def dtw_horizontal(s, t):
    # number of rows in each dataframe
    # corresponding to the number of body parts angles
    n, m = len(s), len(t)

    sum_dtw_values = 0
    for i in range(n):

        # initialize DTW matrix
        # initialize vector to account for missing values
        # both wrt the number of frames for each body part

        for j in range(1, len(s_frames[i]) + 1):
            for k in range(1, len(t_frames[i]) + 1):

                # set count and sum_entries to 0

                # multi-dimensional case
                if (s_frames[i][j - 1] - t_frames[i][k - 1]) != NA:
                    diff_entries = abs(s_frames[i][j - 1] -
                                       t_frames[i][k - 1])
                    count += 1
                if count > 0:
                    cost = diff_entries

                    # fill in matrix
                    # in the same way as for 1D case

                    # fill in vector of values to account for
    missing data
                else:
                    dtw_matrix[j, k] = vec_vals[k]

        # divide by length of DTW path, aka the number of steps
        dtw_final = dtw_matrix[len(s_frames[i]), len(t_frames[i])]
    / n_steps
        sum_dtw_values += dtw_final
    return sum_dtw_values
```

Listing 7: DTW_ED_h_for_MD

The resulting clustering groups and dendrogram are illustrated in Table 7 and Figure 16:

Table 7: Clustering for horizontal DTW with Euclidean distance

Groups	Videos Labels
1	BS_F_BK, BS_F_LT, BS_F_RT, BS_S_BK, BS_S_FT, BS_S_LT, BS_S_RT, SS_F_BK, SS_F_FT, SS_F_LT, SS_F_RT, SS_S_BK, SS_S_LT, SS_S_RT, SYN_R, SYN_S
2	BJ_BK, BJ_FT, LD_F_dis, LD_F_small, LD_S_dis, LD_S_small, LU_F_big, LU_F_dis, LU_S_big, LU_S_dis, SJ_BK, SJ_FT
3	BJ_RT, SJ_RT
4	AR, TA
5	BJ_LT, SJ_LT
6	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
7	TB_S, TB_S_FB, TF_F, TF_S, TL_F, TL_S, TOS_F, TOS_S, TR_F, TR_S
8	SYN_K
9	TB_F_FB

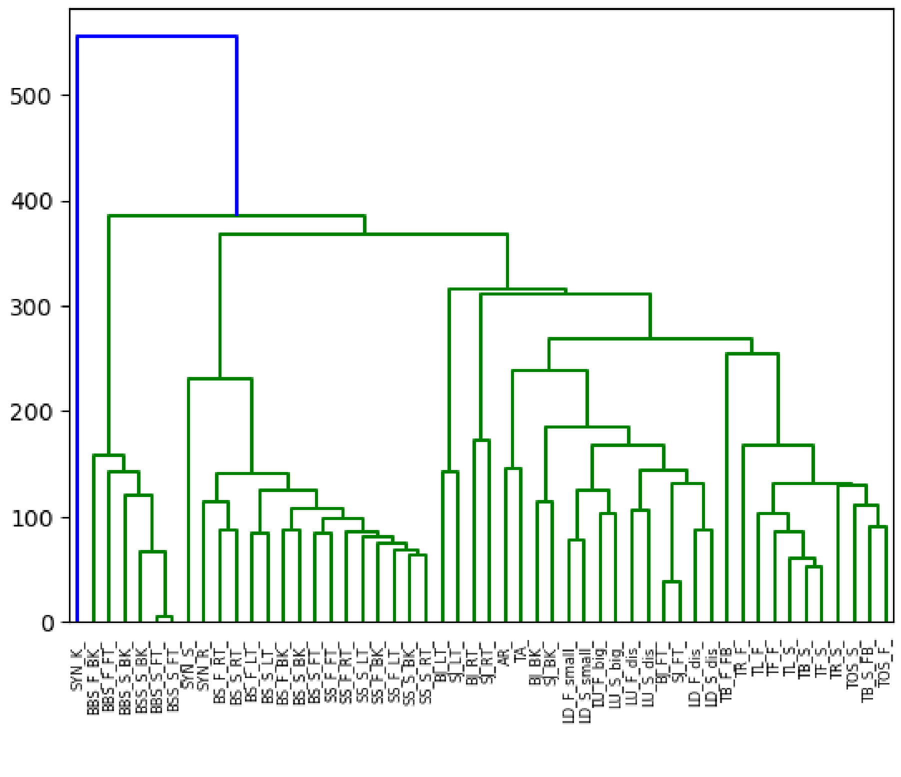


Figure 16: DTW ED horizontal

Using this method, the total number of correctly clustered videos is 42 out of 52, i.e. the accuracy is 80.7% with a run time of approximately 132218 seconds (1.5 days). Even though the run time is much longer than with the DTW vertical methods, the accuracy increases by 23 %. In this case, the misclustered videos can be attributed to a not enough accurate interpolation of the missing values.

It is important to note that this approach is only possible for DTW with Euclidean distance, not Cosine Similarity. In fact, Cosine Similarity takes as inputs two vectors of angles, however the horizontal approach compares single values of the vectors and constructs DTW matrices for every dimension individually. Under these circumstances, the Cosine Similarity equation would only include two values per cell in the DTW matrix. Looking at Equation (2), this implies that only one value for x and one for y would be used instead of a vector of angles and the output will always be 1. Thus Cosine Similarity cannot be used in this scenario. However, under this set-up it is possible to apply the lower bounding method implemented by Keough[Eam04].

8 Lower Bound Keough

The Lower Bound Keough (LB_Keough)[Eam04] is a fast data analysis technique to perform early data pruning by pre-filtering for a candidate series (C) with respect to query (Q). Q is enveloped based on a predefined Sakoe-Chiba length (r):

$$U_i = max(q_{i-r} : q_{i+r}), L_i = min(q_{i-r} : q_{i+r}) \tag{9}$$

Where U_i represents the upper envelope value for data point i and L_i the corresponding lower. A graphical illustration of the result is shown in Figure 17 [Eam04].

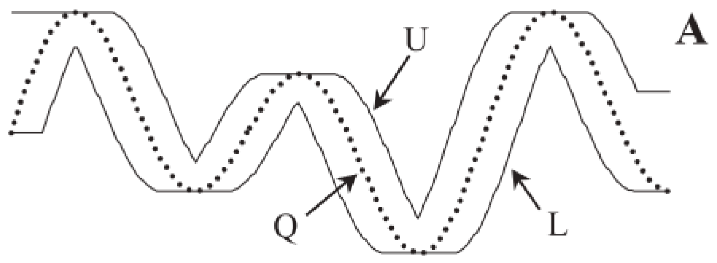


Figure 17: LB_Keough Query envelope

This envelope is then segmented into different Minimum Binding Rectangles (MBRs). MBRs are built based on the maximum value in the upper envelope and the minimum in the lower for a predefined number of data points as illustrated in Figure 18 [Eam04].

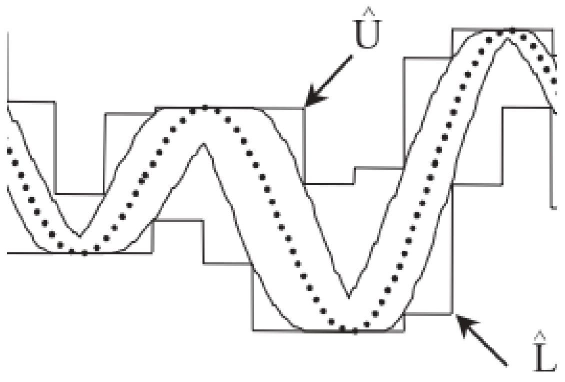
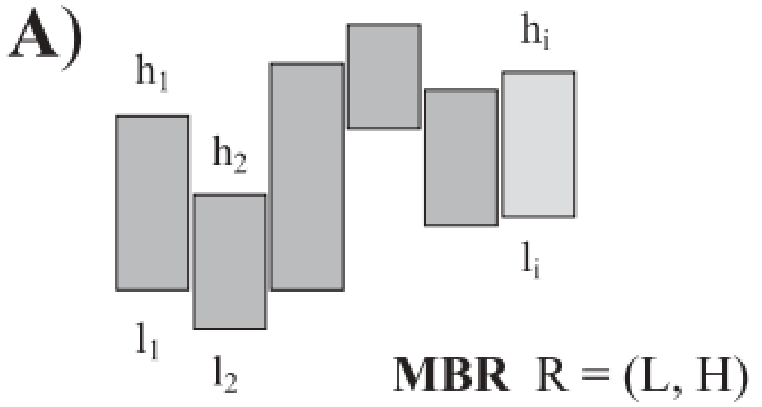


Figure 18: LB_Keough Query MBRs

The candidate series C is then also divided into the same number of MBRs as illustrated in Figure 19 [Eam04]:



MBR $R = (L, H)$

$L = \{l_1, l_2, \ldots, l_N\}$

$H = \{h_1, h_2, \ldots, h_N\}$

Figure 19: LB_Keough Candidate MBRs

Finally, the absolute distance between each Q and C MBR is calculated if the rectangles do not overlap [Eam04].

$$MINDIST(Q,R) = \sqrt{\sum_{i=1}^{N} \frac{n}{N} \begin{cases} (l_i - U_i)^2 & \text{if } l_i > U_i \\ (h_i - L_i)^2 & \text{if } l_i > L_i \\ 0 & \text{otherwise} \end{cases}} \tag{10}$$

Where n represents the length of the series and N the pre-defined number of MBRs. The paper [Eam04] assumes that the length of all series is the same, which is not the case in this work. Therefore in this analysis, when comparing the algorithm output with a predefined value, the normalization with respect to the number of MBRs is performed once all values are summed together. Equation (10) can be visually illustrated in Figure 20 [Eam04]:

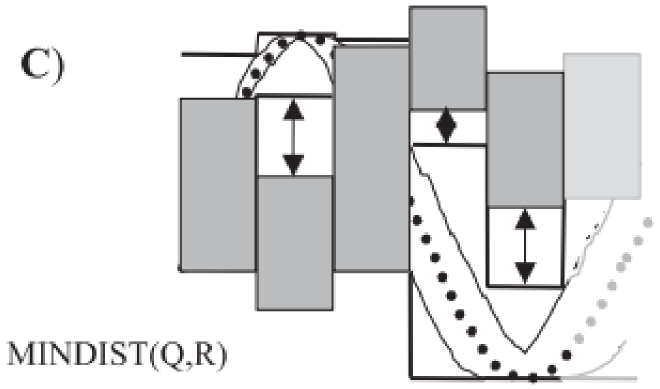


Figure 20: Distance between query and candidate MBRs

In other words, the LB_Keough calculates the minimum distance between two MBRs if the upper value of the candidate falls below the lower of the query or if the lower value of the candidate falls above the upper of the query. If values of the candidate series fall within the range for the query, the distance between the two is evaluate to 0, otherwise it is calculated. This approach leads to an approximate similarity evaluation of two time series as only values outside of a certain range will be included in the calculation of the distance. Consequently, because not all distances are calculated, the output result will be a lower value compared to the DTW approach. In this set-up, if a certain threshold were set to filter time series that fall below such value, this method will bring a superset of the actual set from the brute force approach.

The algorithm described in the paper aimed at finding the candidate series with the shortest distance from the query. For every video, if C falls below a ”best-so-far” condition, the exact DTW distance between C and the Q is calculated[Eam04]:

Algorithm 1 LB_Keough

```
$best\_so\_far = infinity$
for i in all vectors in single video do
    $LB\_dist = LB\_Keough\_dist(C[i], Q)$
    if $LB\_dist < best\_so\_far$ then
        $true\_dist = DTW(C[i], Q)$
        if $true\_dist < best\_so\_far$ then
            $best\_so\_far = true\_dist$
            $index\_of\_best_match = i$
        end if
    end if
end for
```

This best-so-far information was not used in this work as the goal of this analysis was not to merely find the most similar video to a query in a dataset. This algorithm was modified to account for a number of videos frames ranging between 21 and 133 to perform clustering groups and obtain a candidate set with respect to a query. For this reason, it is paramount to verify the accuracy of this approach for different values of MBRs and Sakoe-Chiba lengths. In fact, by setting the same number of MBRs for all videos, a varying number of vectors entries will be included in each rectangle depending on the length of the corresponding video. By looping over different values of MBRs, an ideal number that outputs the closest result to the ground truth can be found. Additionally, with different values of Sakoe-Chiba lengths, different shifts in series will be accounted such that velocity or out-of-sync moves are further taken into consideration. Finally, because of this set-up, missing values will not play such a relevant role as in the brute force methods. The expectation is that this approach will provide similar results to the ground truth as each video only comprises of a dance move, not a combination of many.

9 Lower Bound Keough for a multi-dimensional space

To verify the accuracy of the LB_Keough method and extract the ideal combination of values for the Sakoe-Chiba length and MBRs, a similar approach to the DTW horizontal method was applied.

The range of values for the MBRs and Sakoe-Chiba lengths was set between 1 and 20 as the smallest video in the dataset comprised of 21 frames. Subsequently, results for all clusterings with respect to all combinations of values were compared against the ground truth.

```

highest_n_matches = 0
sc_N_comb = []
for sakoe_chiba in range(1, sc_max):
    for N in range(1, N_max):
        # start time

        for index_query in range(0, n_videos):

            # envelope upper and lower query envelope
            u = upper_envelope(DF_query, sakoe_chiba)
            l = lower_envelope(DF_query, sakoe_chiba)

            # construct upper and lower query MBRs
            Q_u_r = construct_upper_MBRs(u, N)
            Q_l_r = construct_lower_MBRs(l, N)

            # loop over all other videos to be compared with query
            for g in range(index_query + 1, n_videos):

                # construct upper and lower candidate MBRs
                T_u_r = construct_upper_MBRs(newDF, N)
                T_l_r = construct_lower_MBRs(newDF, N)

                # calculate LB Keough distance
                lb_keough = calc_min_dist_MD(T_u_r, T_l_r, Q_u_r,
    Q_l_r, N)

        # end time
        # compare clustering groups with ground truth
        # if the number of matches is greater than
    highest_n_matches
        # than previously saved value --> save new sc_N_comb and
    new
        # highest_n_matches
print(sc_N_comb)
```

Listing 8: LB_Keough_for_MD

The combination of Sakoe-Chiba length and MBRs values that brought the highest number of matches with respect to the clustering resulted to be respectively 3 and 15. The resulting clustering groups and dendrogram for the highest number of matches with lowest time are illustrated in Table 8 and Figure 21:

Table 8: Clustering for horizontal LB_Keough

Groups	Videos Labels
1	SJ_FT, SJ_LT, BJ_LT, BJ_FT
2	TR_S, TR_F, TF_F, TB_S, TL_F, TL_S, TOS_F, TOS_S, TF_S
3	LD_S_small, TA, SJ_BK, LU_S_big, LU_F_dis, LU_F_big, LD_S_dis, AR, LD_F_dis, BJ_BK, LD_F_small, LU_S_dis
4	TB_S_FB, TB_F_FB
5	SJ_RT, BJ_RT
6	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
7	BS_S_BK, BS_S_RT, BS_S_FT, SYN_R, BS_S_LT, SS_S_LT, SS_S_BK, SS_F_RT, SS_F_LT, SS_F_FT, SS_F_BK, BS_F_BK, BS_F_LT, BS_F_RT, SS_S_RT
8	SYN_K
9	SYN_S

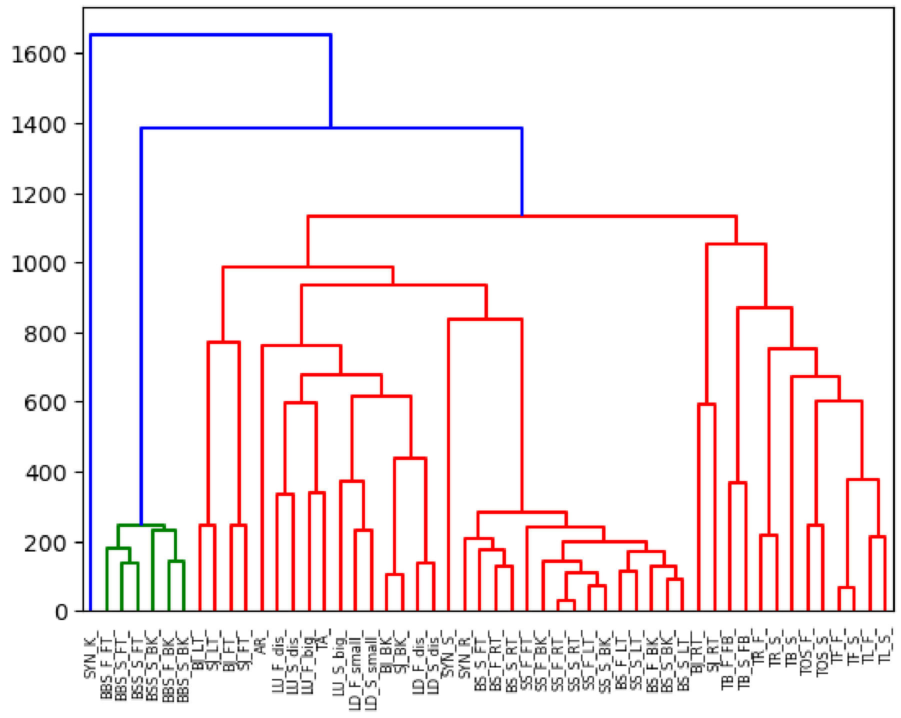


Figure 21: Horizontal LB_Keough dendrogram

Using this method, the number of correctly clustered videos is 43 out of 52, i.e. the accuracy is 82.7% with a run time was 182 seconds (3 minutes). To achieve this result, all combinations of MBRs and Sakoe-Chiba lengths had to be computed. This simulation took approximately 93696 seconds (26 hours). Thus this lower bounding technique outputted a higher accuracy compared to the best performing brute force method (DTW horizontal with Euclidean distance) in a lower time. It is interesting now to see whether lowering the number of dimensions while maintaining the same accuracy can further improve run time.

9.1 Filtering dimensions for LB_Keough

It was noticed that the same clustering accuracy could be obtained without computing some of the body angles. In fact, it is possible to remove the following dimensions:

- angle from left shoulder to right shoulder
- angle from left lower arm to left upper arm
- angle from right upper leg to right lower leg
- angle from left hip to left upper leg
- angle from left upper leg to left lower leg
- angle from left lower leg left ankle to heel
- angle from right lower leg to right ankle to heel
- angle from right foot to right toes
- angle from right foot to right lower leg
- angle from left foot to left lower leg
- angle from left foot to left toe
- angle from torso to right hip
- angle from torso to left hip

Thus, only 16 dimensions (instead of 29) are needed to obtain the highest accuracy with respect to the ground truth. By filtering these angles out, it was also possible to observe that certain joints were not needed:

- left wrist
- left hip
- left knee
- left small toe
- right small toe

The resulting clustering groups and dendrogram differ in order when compared to the analysis using all 29 dimensions. Results are shown in Table 9 and Figure 22:

Table 9: Clustering for horizontal filtered LB_Keough

Groups	Videos Labels
1	TR_S, TR_F, TB_S, TB_S_FB, TF_F, TB_F_FB, TL_F, TL_S, TOS_F, TOS_S, TF_S
2	LD_S_dis, TA, SJ_BK, LU_S_big, LU_F_dis, LU_F_big, LD_S_small, LD_F_small, LU_S_dis, BJ_BK, LD_F_dis
3	SJ_LT, BJ_FT, SJ_FT, BJ_LT
4	SYN_S
5	SJ_RT, BJ_RT
6	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
7	BS_S_RT, SYN_R, SS_S_LT, SS_S_RT, SS_S_BK, SS_F_RT, SS_F_LT, SS_F_FT, SS_F_BK, BS_F_BK, BS_F_LT, BS_F_RT, BS_S_BK, BS_S_FT, BS_S_LT
8	SYN_K
9	AR

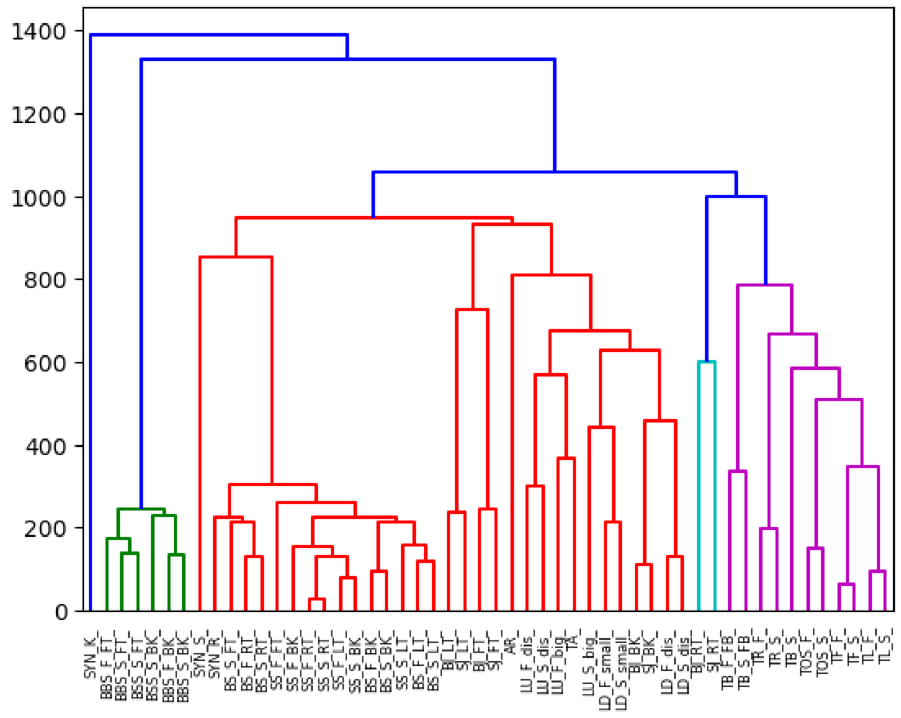


Figure 22: Horizontal filtered LB_Keough dendrogram

The run time for this simulation took approximately 182 seconds (3 minutes), which is the same as the LB_Keough with all 29 dimensions. Consequently, though the run time for the filtering of the dimensions might be slightly lower in the decimal places, it does not have an overall substantial effect. For this reason, the full list of dimensions was used for the remaining part of the work.

10 Indexing structure for query video

The next portion of this work involves building an indexing structure to query a video in the dataset and create a list of similar videos. This analysis was performed by querying an already existing video in the dataset and creating a set of videos whose similarity to the query does not exceed a predefined threshold. In other words, all videos are compared against a query, if the similarity outputs fall below a certain value, the videos are placed into a set. The indexing structure involves first running the LB_Keough method to obtain a candidate set and then compute exact similarity measure with the DTW horizontal method to extrapolate the final set. To tune performance in accuracy and run time of the LB_Keough, it is first paramount to extrapolate the final set by using only the DTW horizontal approach.

10.1 Videos set with DTW horizontal only

The set of videos found with only the DTW horizontal method will be the baseline to observe how closely the LB_Keough performs when creating the candidate set. To create this first mentioned set, a video has to be set as a query and similarities with the rest of the dataset have to be calculated. For each result, if the value is below a predefined threshold, the video will included into the set.

```python
for g in range(0, n_videos):
    if g != index_query:
        actual_dtw = dtw_horizontal(DF, DF_query)
        if actual_dtw <= th:
            actual_dtw_ls.append([g, actual_dtw])
```

Listing 9: DTW_videos_set

For this analysis, the threshold was set to 200 and the query video was the 6th, i.e. BJ_BK. Table 10 lists the results for this simulation:

Table 10: Set of videos below threshold for query video with DTW only

Videos labels	Similarity measure
LD_F_dis	130.601
LD_S_dis	148.298
LD_S_small	166.775
SJ_BK	113.162

The simulation took approximately 4606 seconds (1.3 hours). These results provide the ground truth to evaluate the accuracy of the candidate set produced by the LB_Keough and observe differences in run time.

10.2 Videos set with LB_Keough and DTW horizontal

Section 9 showed how the LB_Keough performs much faster than the DTW horizontal. Therefore, this method can be used to create a candidate set of videos before performing DTW calculations for the final set such that run time can be reduced. Figure 23 shows a schema of the process - first filter out videos using the LB_Keough method and then perform final filtering with DTW:

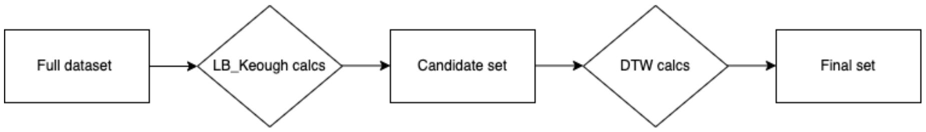


Figure 23: LB_Keough and DTW candidate set

v

In this scenario, the LB_Keough method has to output a candidate set that would include the correct videos as well as additional ones to be filtered later on with the DTW horizontal method. In order to find the best candidate set in terms of accuracy and run time, values of the MBRs and Sakoe-Chiba length need to be tuned such that the combination of LB_Keough and DTW is optimized. To extract the ideal combination of these parameters, run times for all MBRs and Sakoe-Chiba lengths for LB_Keough with DTW horizontal were measured:

```

for sakoe_chiba in range(1, sc_max):
    for N in range(1, N_max):
        # start time

        # construct query envelope
        # construct query MBRs --> Q_u_r and Q_l_r

        # loop over all other videos to be compared with query
        for g in range(0, n_videos):

            # construct candidate MBRs --> T_u_r and T_l_r

            # calculate LB Keough distance
            lb_keough = calc_min_dist_MD(T_u_r, T_l_r, Q_u_r, Q_l_r
    , N)

            # check for threshold condition
            if lb1 <= th:
                ls_lb_files.append([g, DF, lb_keough])

        # calculate accurate DTW
        for i in range(len(ls_lb_files)):
            # calculate DTW horizontal for videos in list
            # check that new distance passes threshold condition
            # append to new list

        # end time
        # append time and N and sakoe_chiba values to list

print(sc_N_comb, tot_time_current)
```

Listing 10: DTW_LB_query

After obtaining all of the run time values for all parameters combinations, plots were created to visualize the results. Figure 24 shows how different MBRs values behave with respect to set Sakoe-Chiba lengths and Figure 25 shows how different Sakoe-Chiba lengths values behave with respect to set MBRs.

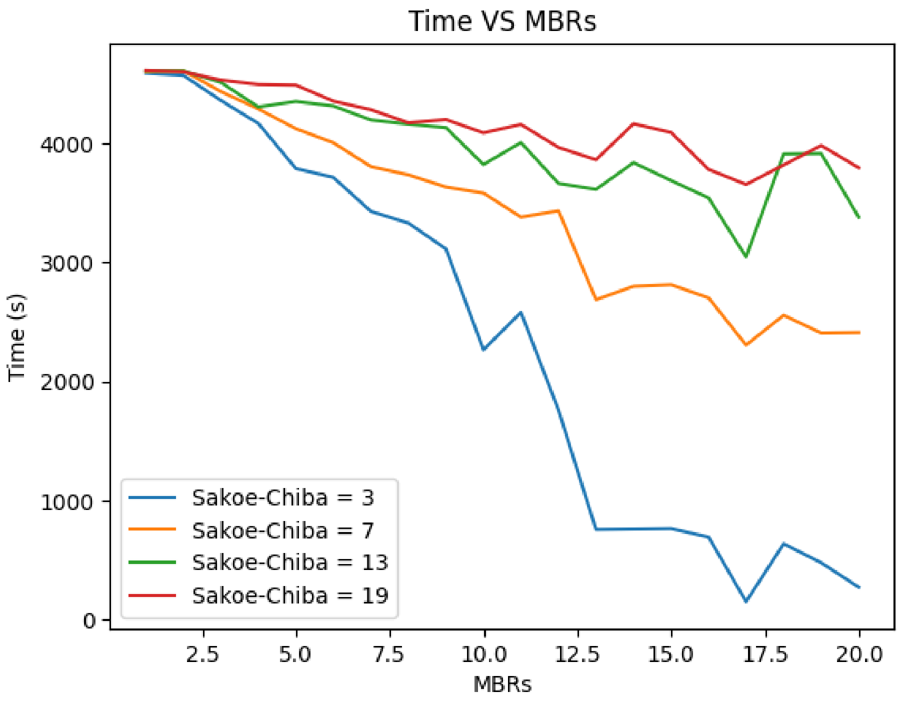


Figure 24: Timing for MBRs with set value for Sakoe-Chiba lengths

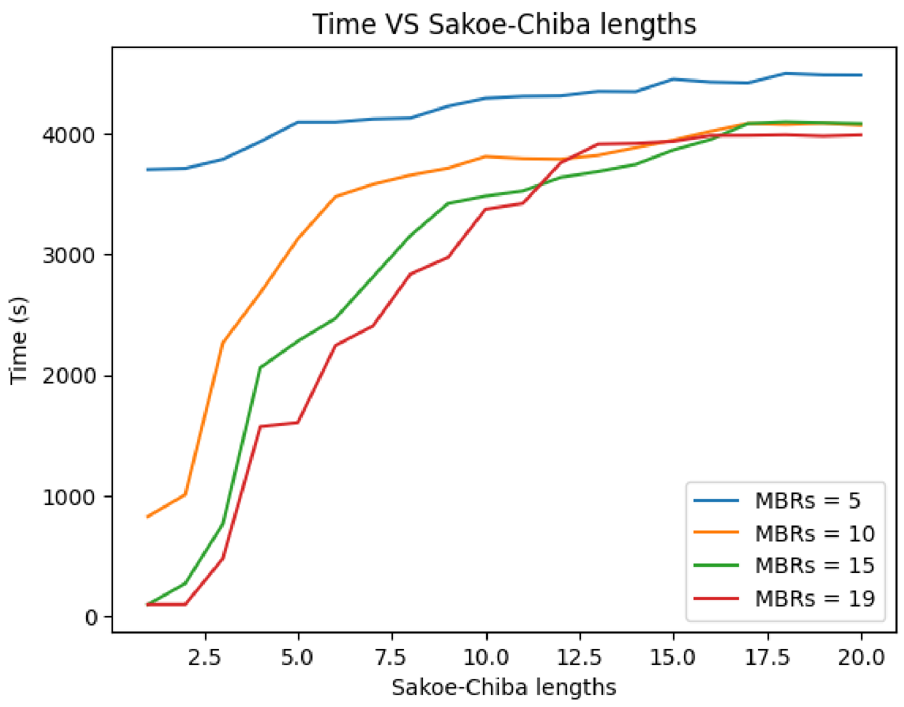


Figure 25: Timing for Sakoe-Chiba lengths with set value for MBRs

It is possible to observe from Figure 24 that MBRs values are inversely proportional to run time with the lowest measure achieved at 17 for all Sakoe-Chiba lengths. Following MBRs values output a higher run time. This behavior is attributed to the fact that small MBRs lead to more distance calculations, however large values require more time to be be constructed. On the other hand, Figure 25 shows that run times and Sakoe-Chiba lengths are directly proportional for all MBRs values and somewhat converge on a plateau at 17. This is to be expected as larger lengths will mean a larger shift in value in the time series, which in turn will require more time to be computed. It can be concluded from these results that the lowest run times are achieved with values for the rectangles in the higher range and values for the Sakoe-Chiba length in the lower. This is line with the findings from Section 9, where the optimal values for the MBRs and Sakoe-Chiba length were respectively 15 and 3.

The simulation to obtain Figure 24 and 25 took almost 17 days, whereas the simulation to obtain the optimal MBRs and Sakoe-Chiba length values in Section 9 took 26 hours with very similar results. Therefore, when creating a candidate set of videos, it might be more efficient to directly use the parameters values calculated for the best clustering accuracy. Additionally, it is of interest evaluate the accuracy of the candidate set even without applying the DTW horizontal approach.

10.3 Videos set with LB_Keough only

The LB_Keough used as a similarity measure in Section 9 outputted a higher clustering accuracy than the DTW horizontal, with optimal parameters values of 15 for the MBRs and 3 for the Sakoe-Chiba length. Looking at Figure 24, the lowest value for the timing to compute a set of videos similar to a query resulted to be 17, hence really close to the optimal value for the LB_Keough clustering. Therefore, a candidate set of videos using the optimal LB_Keough parameters values (15 for MBRs and 3 for Sakoe-Chiba length) was created as shown in Table 11:

Table 11: Candidate set with LB_Keough for optimized parameters values

Videos labels
BS_F_LT
BS_S_LT
LD_F_dis
LD_S_dis
SJ_BK
TF_S

It is possible to observe that this set does not include one video listed in Table 10 (LD_S_small). Because the candidate set did not list all videos in the final set, it is important to perform an analysis to measure the accuracy of output sets for set MBRs value at 15 and a range of all Sakoe-Chiba lengths. Three quality measures were used: Precision, Recall, and F-Score, which take into account true positives (TP), false positives (FP), and false negatives (FN). In this work, the true positives are the videos present in both the ground truth and the computed set, the false positives are the videos present in the computed set but not in the ground truth, and the false negatives are the videos present in the ground truth but not in the computed set.

Precision[Wike] - a proportion of positive results, i.e. the fraction of correctly retrieved videos:

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

Figure 26 illustrates how Precision values change with respect to Sakoe-Chiba lengths:

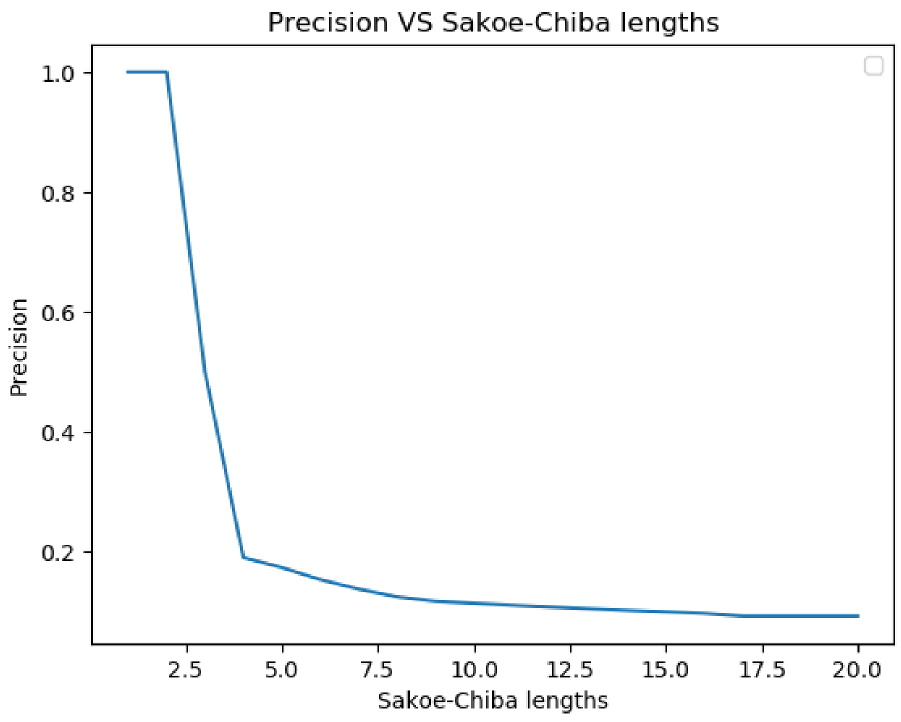


Figure 26: Precision results for different Sakoe-Chiba lengths

It is possible to observe that Precision decreases with increasing Sakoe-Chiba length values, i.e. the larger the shift is, the lower the number of relevant videos among the retrieved ones. In fact, with a higher Sakoe-Chiba length the candidate set becomes larger.

Recall[Wike] - the fraction of successfully retrieved videos:

$$Recall = \frac{TP}{TP + FN} \tag{12}$$

Figure 27 illustrates how Recall values change with respect to Sakoe-Chiba lengths:

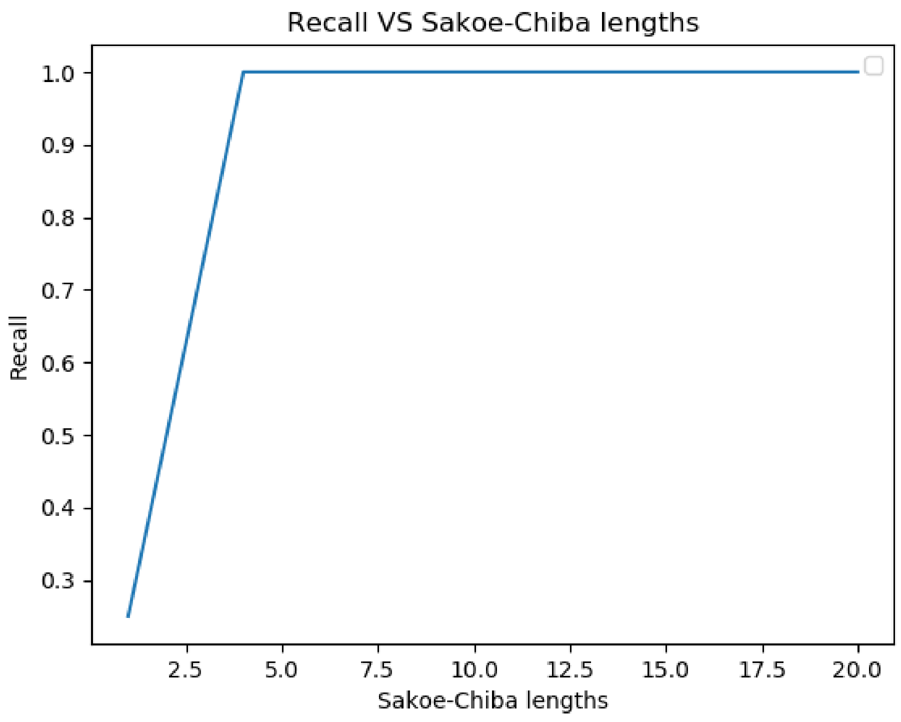


Figure 27: Recall results for different Sakoe-Chiba lengths

It is possible to observe that that Recall reaches the maximum at value 5 for the Sakoe-Chiba length and then stays steady, i.e. the larger the shift is (up until 5), the higher the number of relevant videos retrieved. This phenomenon can also be related to the fact that, with a higher Sakoe-Chiba length, the candidate set becomes larger.

F-Score[Wikd]: a harmonic mean derived from precision and recall, where values range between 0 and 1 (1 expressing perfect precision and recall):

$$F - Score = 2 * \frac{precision + recall}{precision * recall} \tag{13}$$

Figure 28 illustrates how F-Score values change with respect to Sakoe-Chiba lengths:

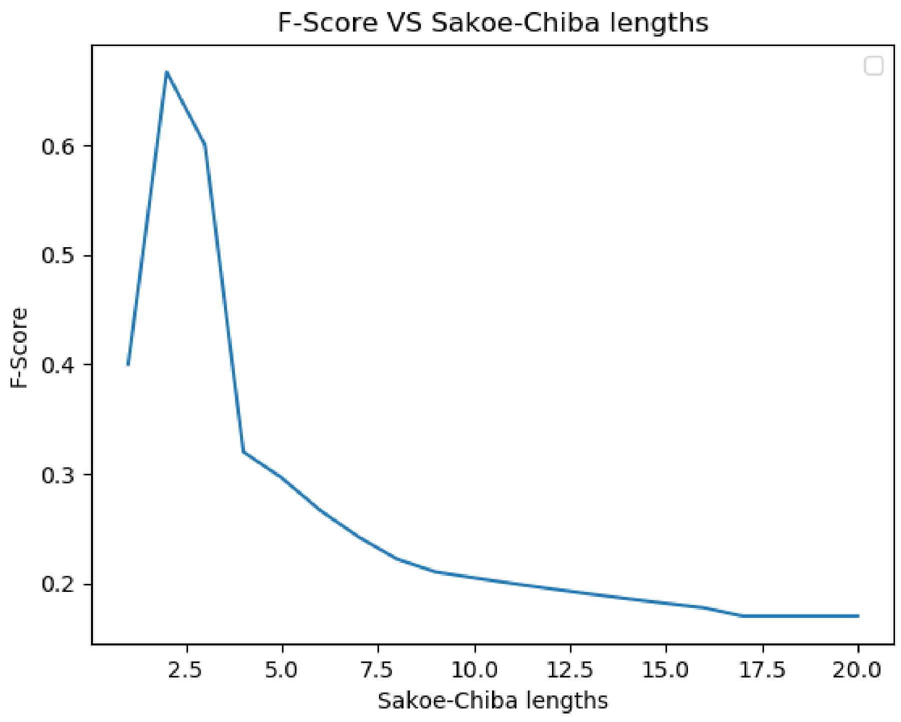


Figure 28: F-score results for different Sakoe-Chiba lengths

It is possible to observe that the F-Score reaches a peak of about 0.7 at value 2 for the Sakoe-Chiba length and then decreases. This value is in line with the previously shown results as the peak in 26 was also reached at 2 for the Sakoe-Chiba length. Therefore, it can be concluded that low Sakoe-Chiba lengths values in combination with MBRs of 15 output the most accurate results.

Finally, an additional plot was created to observe the difference in run times as illustrated in Figure 29:

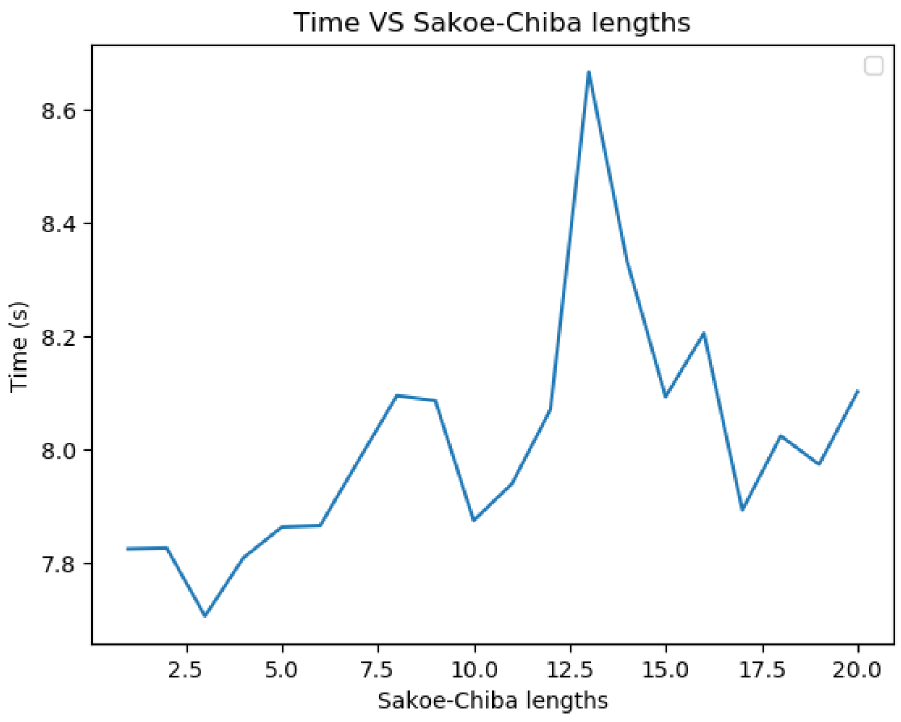


Figure 29: Run times for different Sakoe-Chiba lengths

This plot shows a peak in run time at value 3 for the Sakoe-Chiba length. However, the range in run time outputs varies of a few decimal points, which is not a large enough difference to establish a definite conclusion. It can be derived, that there is not a solid difference in time for MBRs value of 15 and ranging values for the Sakoe-Chiba length.

From all of these results, it can be concluded that the LB_Keough method can be used directly for both the clustering of the videos and the creation of a set of videos similar to a given query without the use of brute methods. In fact, with the right combination of MBRs and Sakoe-Chiba length, this lower bounding technique performs better both in terms of accuracy and run time.

11 Conclusions and Future Work

The analysis of body motion videos cannot be performed with textual interfaces. A first attempt to tackling this issue was implemented in a previous MSc project ("Retrieval and Visual Analysis of Dance Moves") at UZH, which brought preliminary but promising results. The goal of this Master Thesis was to build up on this work by investigating different similarity measures and building an indexing structure to achieve a faster retrieval process time.

To account for both similarity in body poses and velocity of the moving dancer, the Dynamic Time Warping (DTW) method was implemented in a multi-dimensional schema. Several similarity techniques were constructed and their clustering accuracy compared against a ground truth: DTW with Euclidean Distance and Cosine similarity for angles vectors of body positions (DTW vertical), Future and Past angles vectors, DTW with Euclidean Distance for angles vectors of body parts (DTW horizontal), and the Lower Bound Keough (LB_Keough). Results showed that the best brute force method resulted to be the DTW horizontal with 80.7% accuracy. However, even though the LB_Keough was built as a lower bounding technique, when used as a similarity measure it outputted a accuracy of 82.7% with values of 15 for the MBRs and 3 for the Sakoe-Chiba length. This result is slightly higher than the best performing brute force method.

The next portion of this work involved querying a video and extract a set of similar videos falling below a predefined threshold. By combining the LB_Keough to create the candidate set with the DTW horizontal to obtain the final set, the lowest run time was achieved at a value of 17 for the MBRs, whereas the Sakoe-Chiba length had a linear increase. Because these results are similar to the ones obtained in the clustering analysis, a subsequent investigation of precision, recall, and F-Score values was performed for MBRs of 15 and varying Sakoe-Chiba lengths. Results showed that lower Sakoe-Chiba lengths output an overall higher accuracy with lower run time. Therefore, it was concluded that using the LB_Keough method is sufficient to obtain accurate results in a small amount of time both for clustering and querying of a video.

This Master thesis provides a solid foundation for fast analysis and comparison of body motion videos. Further analyses will have to be performed on different datasets involving different motion types as well implementation into a retrieval system to avoid the usage of computer main memory.

References

[Eam04] Eamonn Keogh, Chotirat Ann Ratanamahatana. “Exact indexing of dynamic time warping”. In: *Springer* (2004). URL: `https://www.cs.ucr.edu/~eamonn/KAIS_2004_warping.pdf`.

[Ahm12] Ahmed Al-Jawad, Miguel Reyes Adame, Michailas Romanovas, Markus Hobert, Walter Maetzler, Martin Traechtler, Knut Moeller and Yiannos Manoli. “Using multi-dimensional dynamic time warping for TUG test instrumentation with inertial sensors”. In: *ResearchGate* (2012). URL: `https://www.researchgate.net/publication/261496694_Using_multi-dimensional_dynamic_time_warping_for_TUG_test_instrumentation_with_inertial_sensors`.

[Vas21] Vasiliki Arpatzoglou, Artemis Kardara, Alexandra Diehl, Barbara Flueckiger, Sven Helmer, Renato Pajarola. “DanceMoves: A Visual Analytics Tool for Dance Movement Analysis”. In: *EUROVIS* (2021). URL: `https://www.ifi.uzh.ch/dam/jcr:b4de7134-c558-4529-98d0-18abadc9842d/DanceMoves.pdf`.

[Wika] Wikipedia. *Cosine similarity*. URL: `https://en.wikipedia.org/wiki/Cosine_similarity`.

[Wikb] Wikipedia. *Dynamic time warping*. URL: `https://en.wikipedia.org/wiki/Dynamic_time_warping`.

[Wikc] Wikipedia. *Euclidean distance*. URL: `https://en.wikipedia.org/wiki/Euclidean_distance`.

[Wikd] Wikipedia. *F-score*. URL: `https://en.wikipedia.org/wiki/F-score`.

[Wike] Wikipedia. *Precision and recall*. URL: `https://en.wikipedia.org/wiki/Precision_and_recall`.