

Data Mining Report

Bremen Big Data Challenge 2020
Signal Segmentation and Activity Recognition

by
Harshal Bendale
&
Kartik Dube
&
Sakshi Sharma

A report presented for the degree of
Masters of Science in Data Engineering



Prof. Dr. Adalbert F. X. Wilhelm
Jacobs University Bremen

Contents

1	Executive Summary	1
1.1	Task Outline and the Dataset	1
1.2	Method of Approach	1
1.3	Result	1
2	Introduction	2
3	Dataset Description	3
4	Data Preprocessing:	5
4.1	Describing the datasets	5
4.2	Solving for Null values	7
5	Challenges	8
5.1	Challenges Faced	8
5.2	Classification:	8
5.3	Segmentation:	9
6	Solution	12
6.1	Creating Activity Batches	13
6.2	Solving Batches Using Standard Deviation	18
6.3	Scoring and Result	18
6.4	Resulting CSV	21
7	Improvements and Further Study	22
7.1	In Preprocessing	22
7.2	In Test Data	22
7.3	In Theory	22
8	References	23
A	Code	24

1 Executive Summary

1.1 Task Outline and the Dataset

The Bremen Big Data Challenge 2020 was to classify various hand movements which included everyday movements like picking up an object, switching hands or carrying an object. The Organisers provided all competing groups with sensor data recorded from hand movements and sensors placed on 51 bones in the body. The training data set on which each team had to train their classifier contained sensor data collected from 5 subjects. Data collected from 6th subject was used by the organisers as a test dataset to evaluate the solutions provided by each participating team. For each of the subjects in both the training and the test dataset, participants were provided with 2 data files containing the sensor reading data for muscle activities and motion capture from 51 bones. The data file from muscle activities are sensor readings of subjects doing different activities in the left and right hand. Each sensor (EMG) value is sampled at 600 Hz which leads to each column of the data file to represent a time series recording. The single data file from motion capture includes timestamp (ts, in seconds), followed by angles (as quaternion, four values) and position (three values - position in Direction X, Y, and Z) for 51 bones. The sampling rate of the motion capture data is sampled at 150Hz.

1.2 Method of Approach

The objective of the Bremen Big Data Challenge was to find an optimal method to segment sensor readings and classify activities. This optimal method must be able to generalize its learning in the test data set with the smallest error in predicting activity and the activity duration. With this goal set, our initial method of approach included implementation of a full processing pipeline starting with signal segmentation followed by implementation of tree-based method for classification with subsequent steps attempting to improve the performance of the classifier. Unfortunately, we were not able to follow this approach and segment the sensor readings in time to submit the predictions to the organizers. Therefore, we have focused this report on the data mining approach of segmenting sensor readings to create batches of signals for different activities. Furthermore, as addition to the segmenting approach, we have explored if we can use the average standard deviation value of activity batches in the training data set, to identify activity label of activity batches in the test data set.

1.3 Result

The organizers were suppose to evaluate the performance of each submission by using the recognition rate represented mathematically by,

$$Accuracy = \frac{TotalDurationwithcorrectlyspecifiedActivity}{DurationofAdmission} \quad (1)$$

Since, we were not able to submit the predictions to the organizers in time for the challenge we do not have the accuracy as evaluated by the organizers.

2 Introduction

Human activity monitoring has a broad range of applications which include old-age home care, prisoner monitoring, physical therapy/rehabilitation for athletes and public security[1]. In recent years the advancements in computational power and development of new statistical methods have enabled diverse applications of data-driven decision making which has increased the popularity and scope of research in human activity recognition using machine learning. The Bremen Big Data Challenge (BBDC) 2020, aimed to challenge students to find the optimal machine learning model that accurately identifies activities and the duration of each activity using the data provided.

In general, data used in recognizing human activity are typically sensor readings that are attached to the subject being monitored.[2] The data set provided by the BBDC 2020 organisers was sensor data recorded from sensors placed on both left and right forearms and on 51 different bones of 6 different subjects performing various activities. The training data set contained sensor readings of 5 subjects while the test data set contained recordings of the 6th subject. For each particular activity that was monitored, emg sensors placed on both the arms of each subject were sampled at 600Hz and the motion capture sensor reading were sampled at the rate of 150Hz.

In this project we implemented a data processing pipeline to build a model which can be broken down into the following parts:

- Finding Null values from MOCAP readings
- Replacing Null values with mean of the column in MOCAP
- Dividing the Test MOCAP data into a 2 seconds interval of 300 rows in total.
- Segmenting batched EMG data for the interval of 2 seconds.
- Recording Average Standard deviation for EMG over a segmented batch
- Recording Average Standard Deviation for MOCAP over a segmented batch

The Ideas behind the implementation and, detailed description about these processes are discussed in the sections that follow. Section 3 of this project report is focused in describing the dataset provided by the organizers of the challenge. Followed by section 4 of the report where we have discussed the preprocessing steps before we start our segmenting approach. Section 5 of this report discusses the challenges we had to face in our segmenting approach. Finally section 6 discusses the solution to the challenges discussed in section 5.

3 Dataset Description

The dataset provided by the BBDC 2020 organisers was sensor data for human subjects performing different activities. There are two types of data files, for each subject's activities depending on the sensor readings : one file for muscle activities and one file for motion capture. The muscle activity is measured with electromyography, EMG, bipolar. These EMG sensor records the movement of our muscles. It is based on the simple fact that whenever a muscle contracts, a burst of electric activity is generated which propagates through adjacent tissue and bone and can be recorded from neighboring skin areas[3]. These EMG electrodes are attached on the ring of both left and right forearm and records the movement in the muscles with respect to time. The first activity starts with 0 second and the recording is continuous which means that there are no times without an activity. The sampling rate of the EMG data is 600Hz. The figure below shows the emg reading for subject 1:

	ts	fa-o-t-r	fa-i-t-r	fa-i-b-r	fa-o-b-r	fa-o-t-l	fa-i-t-l	fa-i-b-l	fa-o-b-l
0	0.000000	32780.0	32745.0	32713.0	33655.0	33030.0	32859.0	32875.0	32615.0
1	0.001667	32714.0	32854.0	32641.0	33215.0	33318.0	32851.0	33315.0	32771.0
2	0.003333	32745.0	32834.0	32972.0	32436.0	33037.0	32677.0	33083.0	33799.0
3	0.005000	32845.0	32995.0	33117.0	32861.0	32971.0	32866.0	32978.0	32305.0
4	0.006667	32729.0	32783.0	33033.0	31935.0	33178.0	32886.0	32656.0	32441.0

Figure 1: EMG reading for subject 1

The first column is a timestamp (ts, in seconds), the other columns show the positions of the EMG electrodes:

- fa-o-t-r - Outside top of the right forearm
- fa-i-t-r - Inner top of the right forearm
- fa-i-b-r - Inner bottom of the right forearm
- fa-o-b-r - Outside bottom of the right forearm
- fa-o-b-r - Outside bottom of the right forearm
- fa-o-t-l - Outside top of the left forearm
- fa-i-t-l - Inside of the top left arm
- fa-i-b-l - Inside of the left forearm
- fa-o-b-l - Outside at the bottom of the left forearm

The motion capturing data represents positions and angles for a virtual skeleton calculated from the sensor data. The skeleton may have bones for which no markers were attached to the test subjects, so not all bones are necessarily useful. The figure belows shows an example of the markers for bones in a virtual skeleton:

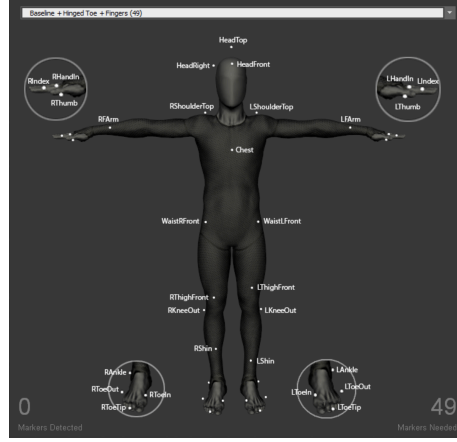


Figure 2: markers for bones in a virtual skeleton

The Figure below shows the first few rows of the motion capture reading for subject 1:

	ts	Hip_Rotation_X	Hip_Rotation_Y	Hip_Rotation_Z	Hip_Rotation_W	Hip_Position_X	Hip_Position_Y	Hip_Position_Z	Ab_Rotation_X	Ab_Rotation_Y
0	0.000000	0.011911	0.999891	-0.001309	-0.008632	-0.176948	1.019471	0.360290	0.016897	0.992654
1	0.006667	0.011864	0.999891	-0.001185	-0.008728	-0.176936	1.019454	0.360227	0.016668	0.992648
2	0.013333	0.011901	0.999890	-0.001186	-0.008759	-0.176948	1.019441	0.360193	0.016844	0.992646
3	0.020000	0.011923	0.999889	-0.001143	-0.008815	-0.176937	1.019425	0.360153	0.016761	0.992641
4	0.026667	0.012177	0.999886	-0.001557	-0.008755	-0.177073	1.019357	0.360247	0.017070	0.992656

Figure 3: Motion capture reading for subject 1

The first column is a timestamp (ts, in seconds), followed by angles (as quaternion, four values) and position (three values - position in Direction X, Y, and Z) for the individual bones. There are 51 bones, so the file has 358 columns ($51 * 7 + 1$). The names of the bones can be found on the first line. Due to the imaging technology, there may be times when there is no data for some bones. The sampling rate of the motion capture data is 150Hz.

These sensors are placed on 6 subjects and each of whom has between one and six recordings. Among those 5 of the subjects were used in the training data set and five recordings of a sixth person are used for evaluation. In the data there are the following activities of the hands, the prefix "la-" describing an activity of the left hand and the prefix "ra-" an activity of the right hand: The figure below shows list of different activities for both left and right hand:

la-object-pick	ra-object-pick	- The hand grabs an object.
la-object-carry	ra-object-carry	- The hand holds an object.
la-object-place	ra-object-place	- The hand drops an object.
la-object-switch-hands	ra-object-switch-hands	- An object is passed from one hand to the other.
la-object-orient	ra-object-orient	- The hand moves a lying object.
la-nothing	ra-nothing	- The hand does not perform any of the previous actions.

Figure 4: Different activities performed by each subject

The figure below shows four activities of the left hand ('la') of subject in s01 in recording t01. This is how training labels look like:

	s01t01.la	0.0	7.37	la-nothing
0	s01t01.la	7.37	9.42	la-object-pick
1	s01t01.la	9.42	9.43	la-nothing
2	s01t01.la	9.43	11.43	la-object-pick
3	s01t01.la	11.43	12.93	la-object-switch-hands
4	s01t01.la	12.93	14.60	la-object-carry

Figure 5: Activities performed by subject 1 with left hand

Each line corresponds to an activity of a hand. The columns have the following meanings:

- Column 1: A hand in a picture of a person. 'la' describes the left hand and 'ra' the right hand. The example shows four activities of the left hand ('la') of subject * in s01 in recording t01.
- Column 2: Start time of the activity (in seconds, inclusive)
- Column 3: End time of the activity (in seconds, exclusive)
- Column 4: Hand activity.

4 Data Preprocessing:

4.1 Describing the datasets

The EMG dataset contained no null values even after the data being recorded at 600 Hz, whereas on the other hand the MOCAP dataset contained null values in 1000s varying between multiple columns. Describing the MOCAP gave a hint from where to begin to tackle the challenge. We first found out the null values in both the types of dataset.

Index	fa-o-t-r	fa-i-t-r	fa-i-b-r	fa-o-b-r	fa-o-t-l	fa-i-t-l	fa-i-b-l	fa-o-b-l
count	78524	78524	78524	78524	78524	78524	78524	78524
mean	32763.2646	32888.771	32742.2486	32791.8568	33064.4837	32762.2866	32752.686	32768.8438
std	543.879121	363.531799	996.848111	988.881762	838.118891	466.65929	1448.28738	1815.88784
min	24171	25298	4613	16144	4498	28849	14679	925
25%	32789	32789	32631	32654	32933	32623	32454	32477
50%	32761	32820	32755	32799	33111	32765	32751	32770
75%	32814	32922	32853	32918	33268	32882	33839	33112
max	44788	37995	47484	47837	44868	49323	62374	63532

Figure 6: Statistical Description of EMG for S01t01

Index	fs	Hip_Rotation_X	Hip_Rotation_Y	Hip_Rotation_Z	Hip_Rotation_W	Hip_Position_X	Hip_Position_Y	Hip_Position_Z	Ab_Rotation_X	Ab_Rotation_Y
count	17822	17398	17398	17398	17398	17398	17398	17398	17398	17398
mean	58.736667	-0.0232433043	0.56832434	-0.0518489477	-0.36459152	-0.208794473	1.01895579	0.106387887	0.892731511	0.5596315
std	33.914517	0.122746935	0.58892672	0.8940956358	0.516714884	0.618541806	0.8252537562	0.497154185	0.8835112889	0.5151685
min	0	-0.999688	-0.786155	-0.478265	-0.999985	-1.005887	0.854416	-0.978865	-0.248674	-0.786562
25%	29.36833333	-0.06752975	0.474443	-0.182767	-0.80844775	-0.7337825	0.98878575	-0.18235525	0.88883925	0.4977255
50%	58.736667	-0.0248915	0.7652855	-0.0448665	-0.5641985	-0.9132785	1.0878895	0.1483445	0.8485935	0.773819
75%	88.108	0.00950425	0.9188125	-0.00479525	0.07888875	0.157395	1.0223125	0.4583185	0.88494475	0.989259
max	117.473333	0.551142	0.999913	0.44848	0.706457	1.643462	1.268299	1.372114	0.264285	0.998885

Figure 7: Statistical Description of for S01t01

Index	0
Hip_Rotation_Z	292
Hip_Rotation_W	292
Hip_Position_X	292
Hip_Position_Y	292
Hip_Position_Z	292
Ab_Rotation_X	466
Ab_Rotation_Y	466
Ab_Rotation_Z	466
Ab_Rotation_W	466
Ab_Position_X	466
Ab_Position_Y	466

Figure 8: count of Null values for EMG and MOCAP datasets

We mapped each left and right hand activities to MOCAP and EMG datasets to ease the coding process. We also plotted the time series graph to more understand the null values in both the datasets. We also had to understand whether the null values in MOCAP are of a particular activity or of the same activity in different columns. Given that the sum of null numbers vary in

MOCAP data, it is obvious that the null values are in random columns and in different activities. We will verify that by making time series graphs.

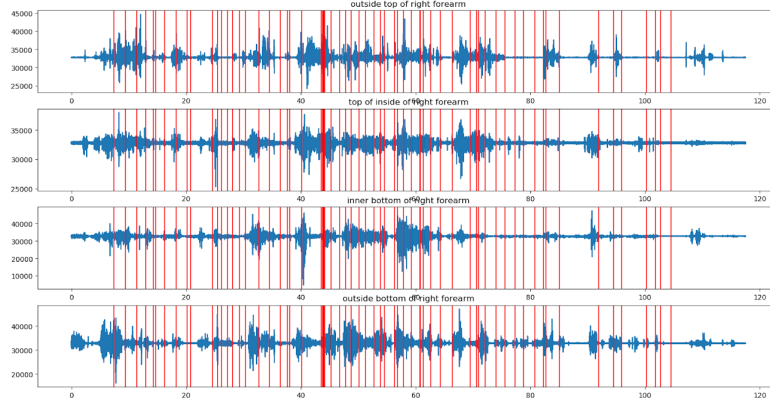


Figure 9: Time series graph for right hand in EMG, red lines separate predicted activity periods

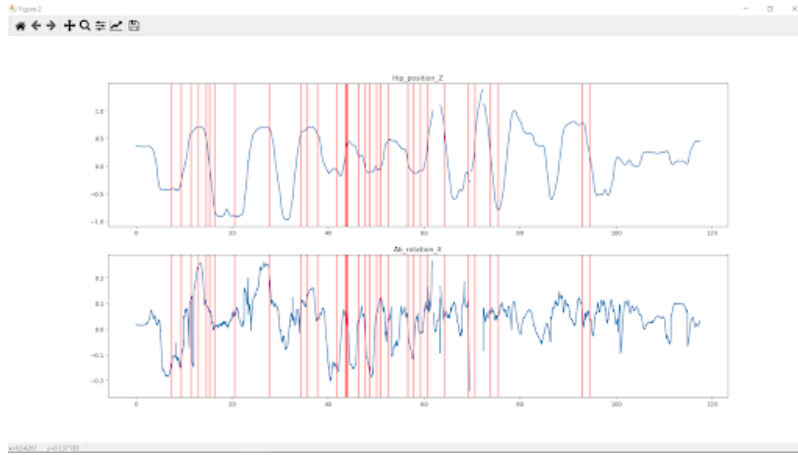


Figure 10: Time series graph for Hip_position_Z and Ab_rotation_X in MOCAP, lines separate predicted activity periods

4.2 Solving for Null values

As realised from the null description and time series of right hand EMG of s01t01 that EMG data is a continuous time series without any null values. But the MOCAP where we selected Hip_position_Z and Ab_rotation_X with only the left hand labels, the data has multiple null values at different activities. To solve this issue in MOCAP data, we'll add the mean values at the null places. This added some very small biases which can be ignored given the huge amount of data. Given more time, this process could have been optimised further (discuss in further studies).

5 Challenges

5.1 Challenges Faced

Now that we know how the data looks, let us have a look at the challenges we faced. The deliverables of the Bremen Big Data challenge 2020 was a .csv file containing four columns: first one being denoting the person, the time frame and the hand (left or right), the second column for start time, third for end time, and the fourth representing the name of the activity that that hand did in that time frame (very similar to labels.train dataset).

We identified this as a two stage problem or a two-step process. The first step being classification of the activities, and the second one being segmentation of time. Let us see in a more elaborate fashion what the aforesaid two steps are.

5.2 Classification:

In this step the expected output is to get labels of activities performed by the hands of the volunteer. To achieve this goal, we first need to label every available reading from the Mocap dataset and EMG dataset. The next step is to train a model on the data and its respective labels from Mocap dataset to predict the class (i.e. the label) of the activity of the hand. Now the question arises, of why to choose Mocap dataset over EMG to train a model for classification? After a lot of brainstorming, we were able to see what Mocap dataset is able to tell us, that EMG can't while training a model for classification. The EMG dataset covers just the forearm positions. On the other hand, Mocap dataset covers a large number of parameters which all point to almost precise body positions. Imagine a hand pointing downwards in an angle of say 45 degrees in the elbow (as shown in the Figure 11). Both the datasets, Mocap as well as EMG would be able to tell us that the hand is at 45 degrees pointing downwards. So if the person is bent over and is having the hand at 45 degrees pointing downwards from the elbow, he/she is most probably doing the activity of "object pick". But if the person is standing straight and is having the hand at 45 degrees pointing downwards from the elbow, he/she is most probably doing the activity of "object hold". We observed how the information on body position plays an essential part in determining the class of the activity the person is performing. Therefore we decided to fix on Mocap dataset for training our model for the classification purposes.

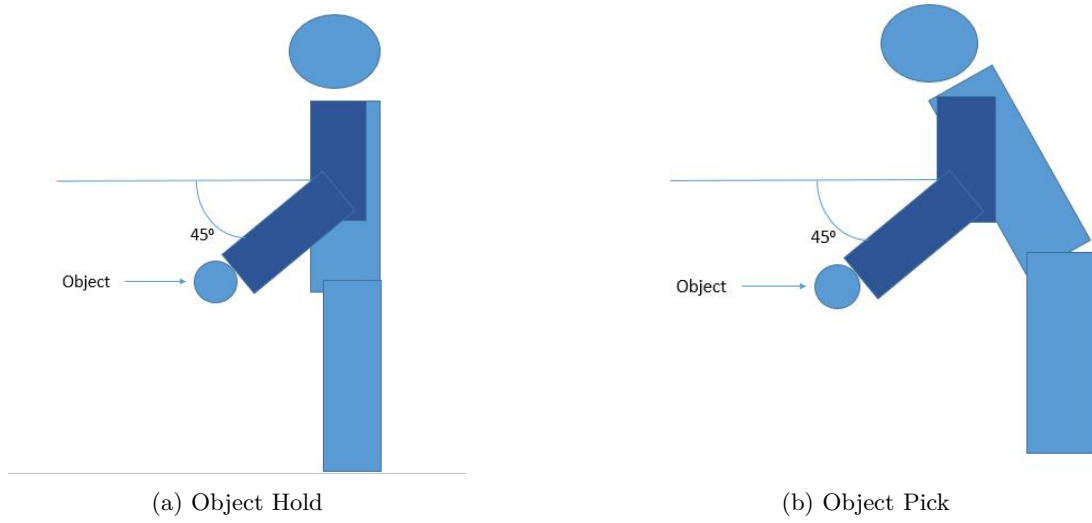


Figure 11: Pictorial representation of same angle different action

5.3 Segmentation:

This step is solely for the purpose of finding breakpoints in timeline, which is to find the start time and the end time of the activity of hand that we classified in the previous step. Before jumping to the procedure, let us consider the following example for illustration purposes.

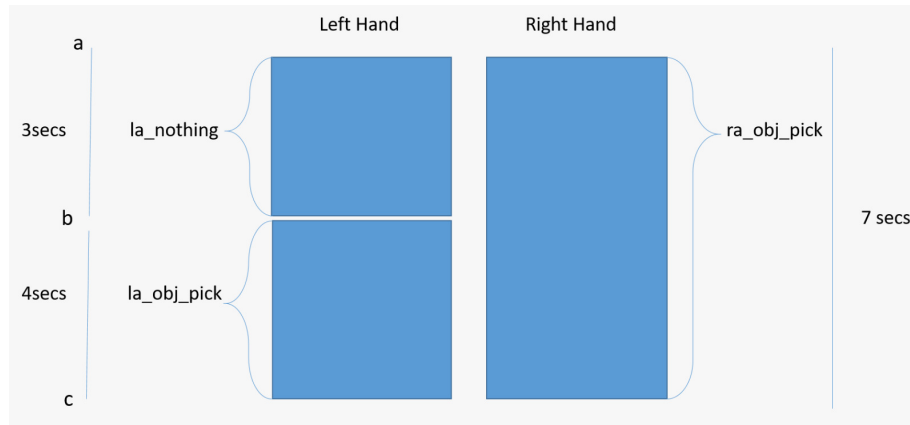


Figure 12: Diagrammatic representation of our approach

The above Figure (Figure 12) is a diagrammatic representation of the approach we used. In this approach, we considered the activities of left hand and right hand as two different things. In other words, we treated the activity of left hand as one signal and activities of right hand as another. In

the example considered for illustration, we can see that the left hand was doing nothing from point 'a' to point 'b' in the time line. And from point 'b' in time, the left hand was doing the activity of picking up the object. It continued to do so until point 'c' in time line. Whereas the right hand was doing the activity of picking up the object right from point 'a' till point 'c' in time line. Let us say the time elapsed from point 'a' to point 'b' is 3 seconds and from point 'b' to point 'c' is 4 seconds. So in conclusion the left hand performs two actions in a given time frame of 7 seconds and right hand performs just one action in the same time frame. Now looking at our labels.train dataset, we can say that for s01t01 time frame of 117.481 seconds, both the hands performs different number of activities and each activity is having a different start and stop time. Also, the time period for each activity can be different for the same activity taking place again in the given time frame. For example, in the above considered example in Figure 12, the left hand is picking up object for four seconds, does not imply that the left hand will always complete picking up object in four seconds, it may take six seconds to pick up object next time, or may be even pick it up faster than previous.

From the above discussion we can conclude that we majorly faced two challenges in the process of segmentation:

- Different number of activities taking place for different hands in a fixed given time frame
- Varying time periods for the same activity taken place by the same hand in the same given time frame.

Solutions we came up with to tackle the above mentioned challenges:

We came with three approaches for the aforesaid challenges:

1st Approach: In order to find the end point of the previous activity which is also the start point of the next activity, we asked ourselves the following questions: 'What is that one thing that distinguishes the first activity from the second?', 'What is it that will define the end of the first activity?'. As once we find just the end of the first activity, the same point will be the start point of the next activity. So we came up with the solution to consider the data of the entire left hand from EMG as one signal and the entire right hand as another signal. We plotted the signals using python and found some patterns to corresponding activities.

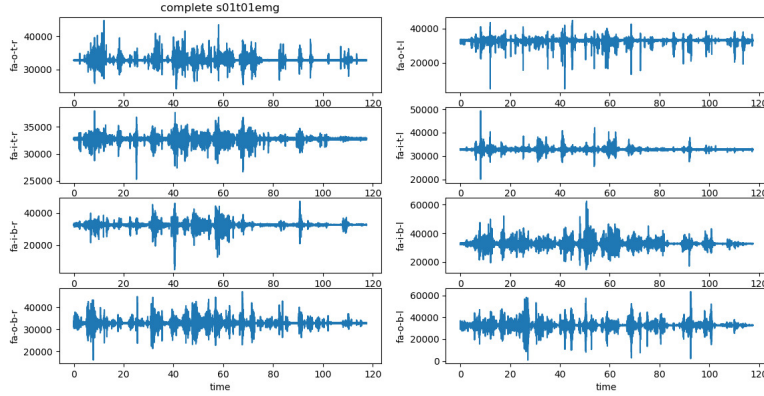


Figure 13: Plot of signals for one complete time frame of s01t01 for left and right

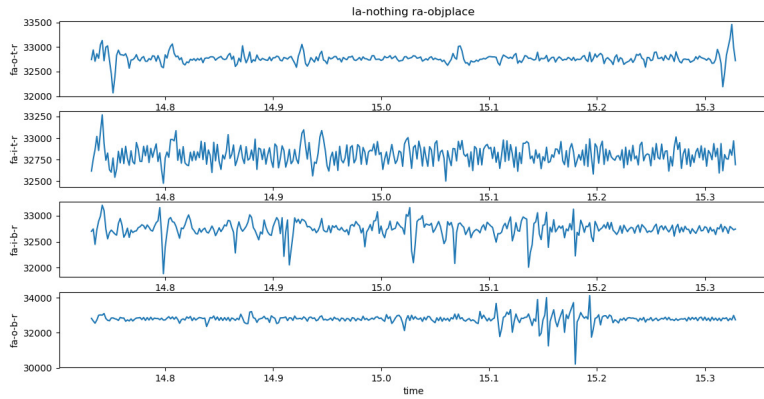


Figure 14: Signal plot of only right hand where $la_{nothing}$ and $ra_{objplace}$

The above figure is the plot of the signals we plotted in matplotlib library of python. As you can see there was quite a bit of noise present in the signal and therefore we could not process the signal in order to accurately predict the end points of the activities.

2nd Approach: To better understand this approach, let us again consider the example we considered in Figure 12. We had one hand with an activity time span for 7 seconds and other hand doing two activities of three seconds and four seconds each. The left hand does nothing for the first three seconds and then does the activity of picking up objects. In the same seven seconds, the right hand does the activity of picking up objects. In this approach we are considering the Mocap dataset. Which means, we are considering the whole body. Therefore we narrowed it down to observing the effect of activity that the left hand is performing on the signals representing the activity performed by the right hand during the same time span. For example we focus on the change in the signal of

right hand performing object pick due to the effect of left hand performing the task of object pick from point 'b' to point 'c' in the example considered in the Figure 12. Once we know the effect of left hand on right hand, we subtract that effect from the signal of right hand from point 'b' to 'c' and then what we get, we consider as the pure form of the signal representing right hand picking up objects while left hand does nothing. In order to subtract the effect, we need to resample the signal of the right hand from point 'a' to point 'b', as you can see that that signal is of length three seconds and the signal we proposed to perform subtraction is of length four seconds.

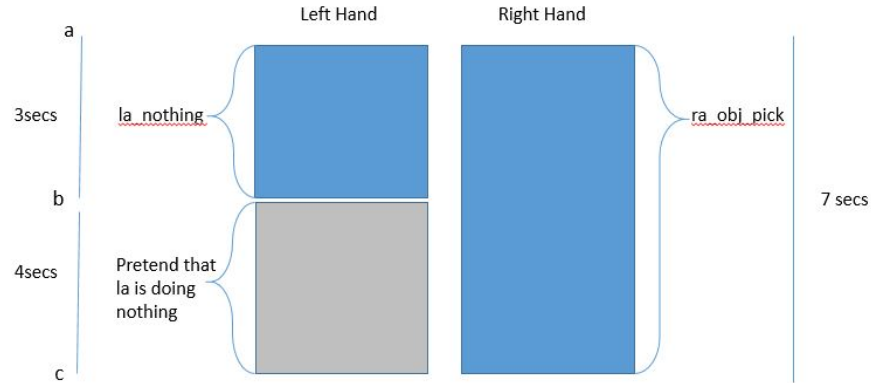


Figure 15: Approach two illustration

The above figure (Figure 15) shows a pictorial representation of our 2nd approach.

3rd Approach: In this approach, we went ahead with more of a statistical perspective. As before we considered the data from right and left arms as two different signals, and we considered that every activity would have its own unique signal. In total there would be therefore thirty six signals of thirty six activities. But the question of how to distinguish one signal from the other still remains. So for that we used one of the descriptive statistics tools, which is standard deviation. According to quantitative methods of 'Research Rundowns', the standard deviation is a very powerful statistical tool as it is a single number telling the variability, spread as well as the distribution. So in this approach we differentiate the signals based on their standard deviation values with a fixed step in time frame.

6 Solution

Given the complexity of the datasets and the challenges faced, we decided to approach the problem which was feasible to us in theory and in coding. We will further discuss the optimizations and theories that could have been implemented.

6.1 Creating Activity Batches

As discussed earlier, the duration of right hand activities varies from the left hand activities both in EMG and MOCAP datasets. We made batches of all 36 activities in the EMG dataset. We extract data such as one single left hand activity relates to only one right hand activity.

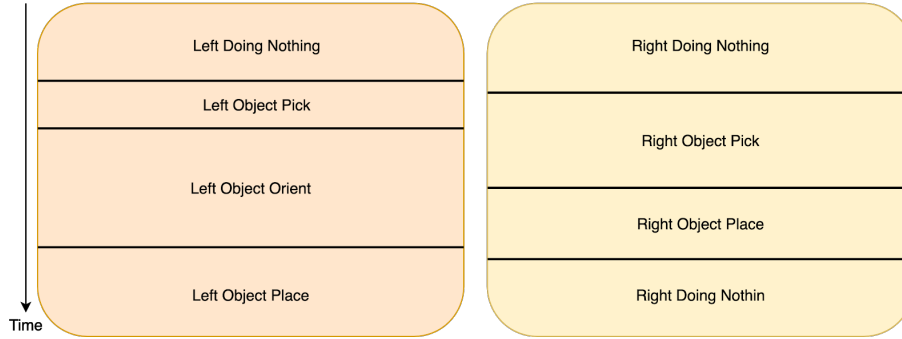


Figure 16: Batch Problem in EMG

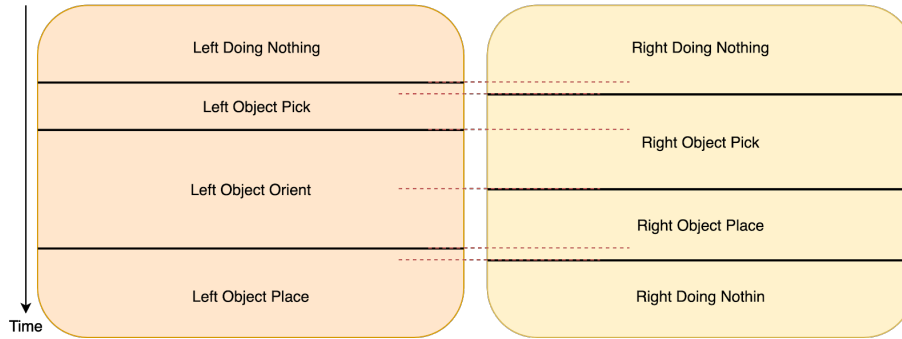


Figure 17: Batch Problem in EMG

From the figure we see that the activities are randomly arranged and we cannot combine one activity for each hand having the same number of rows. We ignored the number of rows which are not related to one hand activity and only focused on combining the activities together depending on how largely one activity of one hand resides in another.

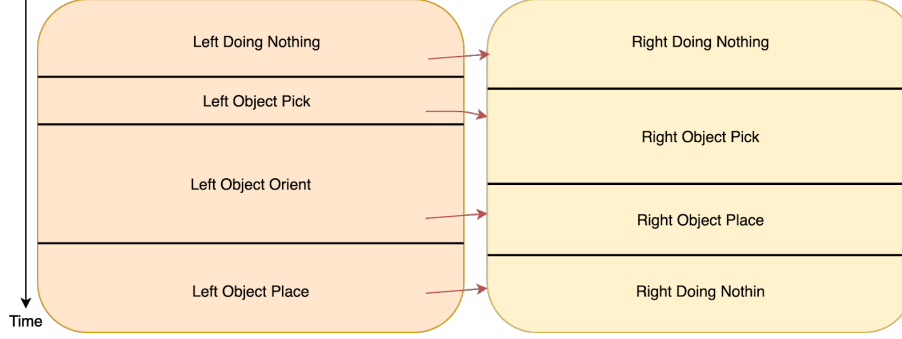


Figure 18: Batch solution in EMG

This gave us only one single activity done by both the hands, that is, when left doing nothing and right doing nothing is considered as one activity and one batch. Similarly, left object orient and right object place is one single activity. This way we made batches of all the activities from the EMG dataset of all the recordings.



Figure 19: Making Batches in EMG

We tried to use Pandas extension to upscale or downscale the rows in each activity to make them of the same number of elements, but doing so resulted in huge biases due to how much left and right hand activities vary in proportions. Plus as seen from the figure upscaling brings a lot of noise which is not required, this upscaling brings a lot of noise when a particular activity in one hand is very small in duration when compared to the other hand's activity. For example, an activity which is only 0.7 sec will generate a lot of noise if upscaled to an activity which is of 2

seconds. On the other hand, downscaling will make the time series lose its spikes and degrade the signal. For example, a 5 second signal will lose its integrity if downscaled to 2 seconds.

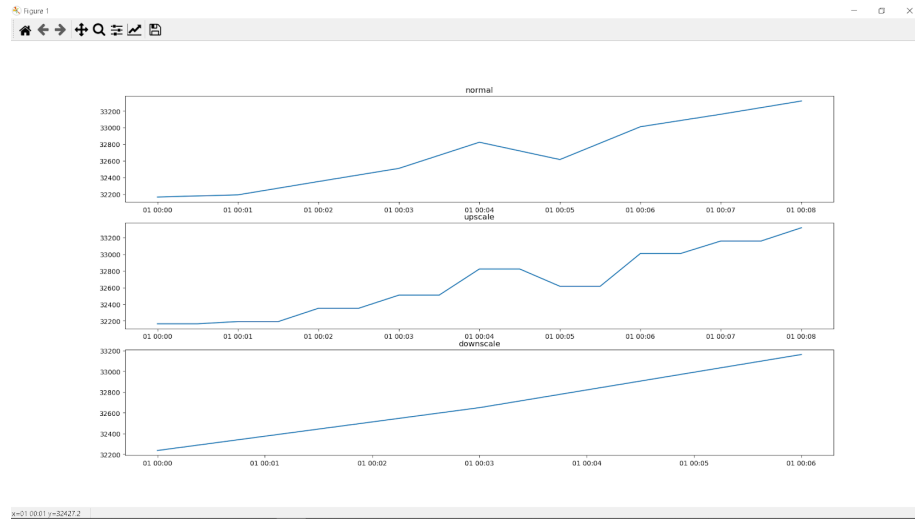


Figure 20: Scaling Problem

To find out end points of activities in MOCAP data, we decided to use Python's Ruptures library. Rupture uses change point detection algorithms to detect changes in a signal or time series data. Given that we also had time series data, Ruptures algorithms were not efficient enough to formulate the changes. We tested Rupture's Window Based Search Method [4][5] on S01t01 EMG outside top of right forearm.

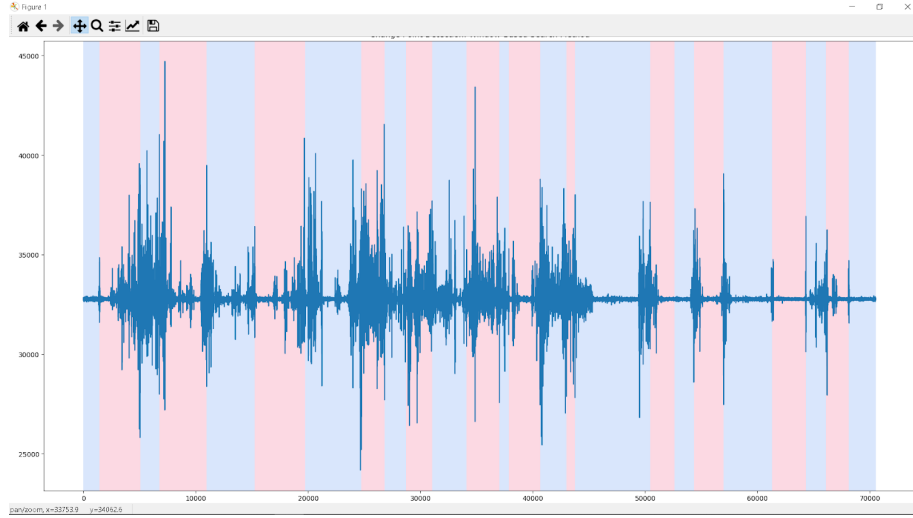


Figure 21: Signal Processing by Ruptures on S0t01 EMG of outside top of right arm, each segment denotes one activity

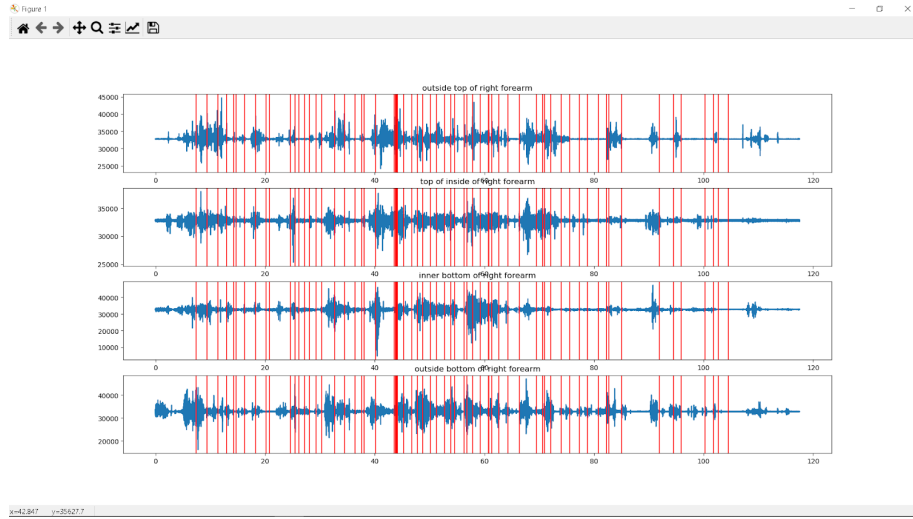


Figure 22: Original Signal on S0t01 EMG of outside top of right arm, red lines denotes end of activity

Due to this reason we were unable to find endpoints in the MOCAP datasets. The MOCAP dataset contains information about position and rotations of 57 points, having three axes. After manually analysing the average duration of one activity of each hand from the labels data, we decided to make batches of 2 seconds which is 300 rows given the MOCAP data is of 150 Hz. Batching MOCAP data using 2 seconds gave more advantage than disadvantage, for

example if an activity is of 6 seconds, the MOCAP will provide with 3 batches of two seconds having one activity. This does generate some error when one activity is of an odd number of seconds or varies by 0.5 seconds, we decided to ignore the error.

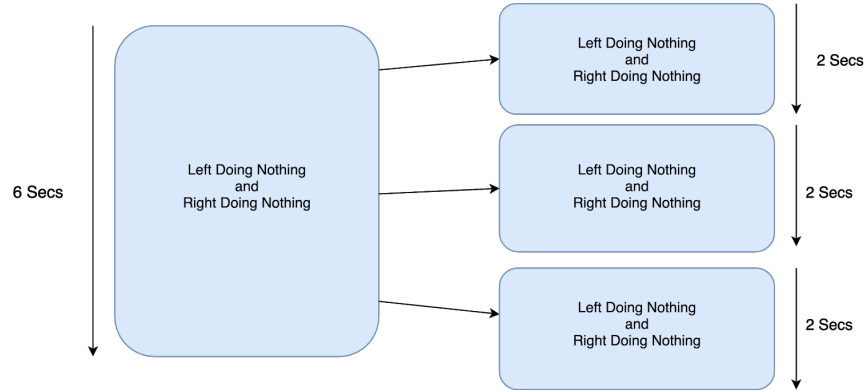


Figure 23: MOCAP 6 seconds segment example

From the figure (), If there are other activities interfering in a batch of MOCAP data then we omitted the activity having the least amount of rows in that particular batch. This brought minimal error in the whole batch and similarly easier to work on the Test MOCAP dataset.

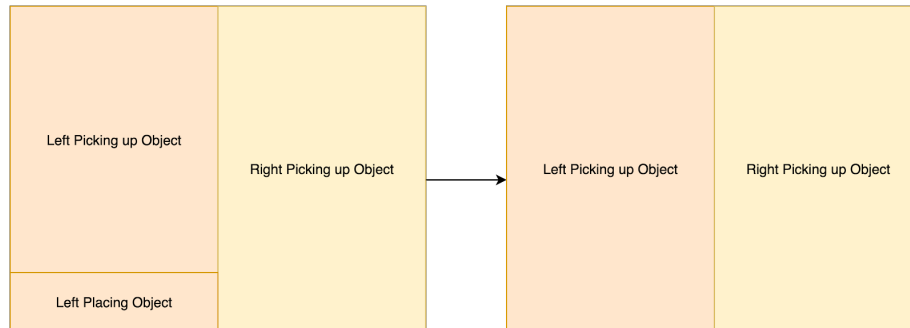
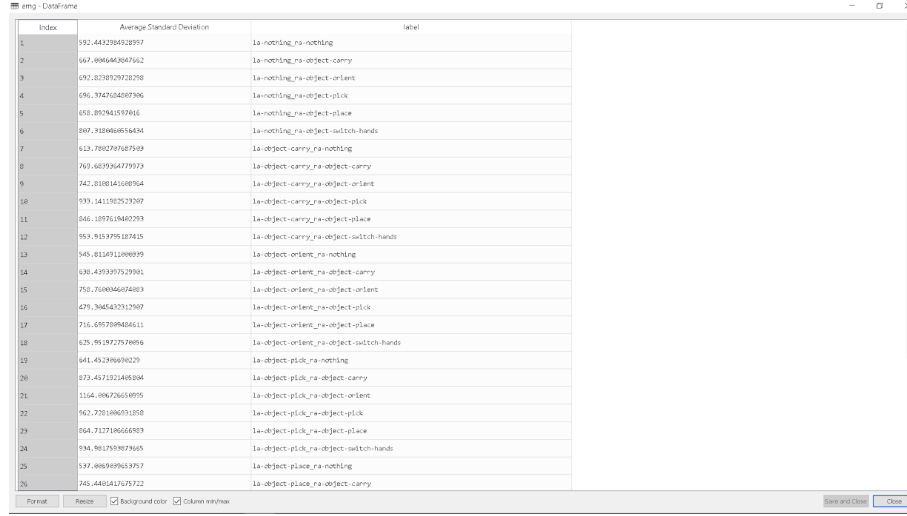


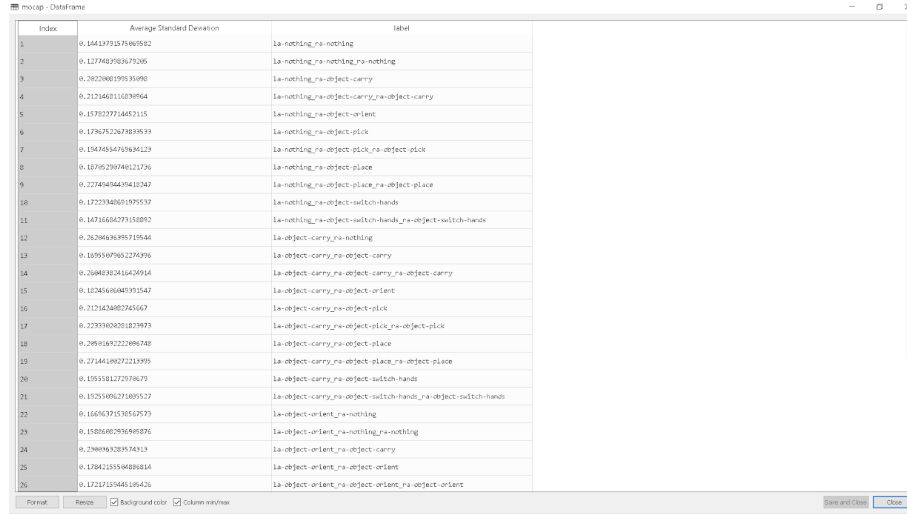
Figure 24: MOCAP batch processing

6.2 Solving Batches Using Standard Deviation



Index	Average Standard Deviation	label
1	592.443298528997	la-nothing_ra-nothing
2	667.896463667662	la-nothing_ra-object-carry
3	692.423972972870	la-nothing_ra-object-orient
4	596.394250489786	la-nothing_ra-object-pick
5	658.89241597861	la-nothing_ra-object-place
6	689.3186686934434	la-nothing_ra-object-switch-hands
7	613.780278787939	la-object-carry_ra-nothing
8	769.683964779373	la-object-carry_ra-object-carry
9	742.818141688964	la-object-carry_ra-object-orient
10	939.1411982523287	la-object-carry_ra-object-pick
11	846.1897619483293	la-object-carry_ra-object-place
12	853.819395187415	la-object-carry_ra-object-switch-hands
13	545.8114911108899	la-object-orient_ra-nothing
14	638.4393979797961	la-object-orient_ra-object-carry
15	758.760046674883	la-object-orient_ra-object-orient
16	479.3045432912987	la-object-orient_ra-object-pick
17	716.695789848611	la-object-orient_ra-object-place
18	625.951972797886	la-object-orient_ra-object-switch-hands
19	641.452386960229	la-object-pick_ra-nothing
20	873.437132148584	la-object-pick_ra-object-carry
21	1164.886736568955	la-object-pick_ra-object-orient
22	962.720188691858	la-object-pick_ra-object-pick
23	864.712718664879	la-object-pick_ra-object-place
24	934.981291897865	la-object-pick_ra-object-switch-hands
25	537.486989963873	la-object-place_ra-nothing
26	745.448417373722	la-object-place_ra-object-carry

Figure 25: Averaged Standard Deviation for EMG for every activity



Index	Average Standard Deviation	label
1	0.1641379157546582	la-nothing_ra-nothing
2	0.1277481953617526	la-nothing_ra-nothing_ra-nothing
3	0.202088199535980	la-nothing_ra-object-carry
4	0.212148111078964	la-nothing_ra-object-carry_ra-object-carry
5	0.1578227714457115	la-nothing_ra-object-orient
6	0.1786752367883933	la-nothing_ra-object-pick
7	0.194783479364123	la-nothing_ra-object-pick_ra-object-pick
8	0.18785288748612736	la-nothing_ra-object-place
9	0.2274944494112347	la-nothing_ra-object-place_ra-object-place
10	0.1723348691593937	la-nothing_ra-object-switch-hands
11	0.1671684879158892	la-nothing_ra-object-switch-hands_ra-object-switch-hands
12	0.26284636885719544	la-object-carry_ra-nothing
13	0.1895897861224596	la-object-carry_ra-object-carry
14	0.2084832416424954	la-object-carry_ra-object-carry_ra-object-carry
15	0.1824616049191547	la-object-carry_ra-object-orient
16	0.2121434882745667	la-object-carry_ra-object-pick
17	0.2233360301823873	la-object-carry_ra-object-pick_ra-object-pick
18	0.2498182222496748	la-object-carry_ra-object-place
19	0.27144188272213995	la-object-carry_ra-object-place_ra-object-place
20	0.1955181272978679	la-object-carry_ra-object-switch-hands
21	0.15251686271889532	la-object-carry_ra-object-switch-hands_ra-object-switch-hands
22	0.16676391578562973	la-object-orient_ra-nothing
23	0.1288682858888876	la-object-orient_ra-nothing_ra-nothing
24	0.298963283574319	la-object-orient_ra-object-carry
25	0.17841255588888884	la-object-orient_ra-object-orient
26	0.17217135445185426	la-object-orient_ra-object-orient_ra-object-orient

Figure 26: Averaged Standard Deviation for MOCAP for every activity

6.3 Scoring and Result

These preprocessing steps were followed for the test data:

- Finding Null values in MOCAP
- Replacing Null values with mean of the column in MOCAP
- Dividing the Test MOCAP data into a 2 seconds interval of 300 rows in total.
- Since we do not know what are the endpoints of an activity, we batched
- EMG data for the interval of 2 seconds that is 1200 rows in total.
- Recording Average Standard deviation for EMG over a batch
- Recording Average Standard Deviation for MOCAP over a batch



Figure 27: Preprocessing for Test

After finding MOCAP and EMG scores of Average Standard Deviation, we used Python's `abs()` function to calculate the closest three values from the list of previously calculated standard deviations for all 36 activities. So, now we have three possibilities from EMG and MOCAP. To score these test batches we simply provide these three possibilities with ranks, the closest possible activity gets rank 1 and the least out of 3 gets rank 3. We multiply the ranks to find out the most probable activity. The activity with the minimum score is selected[6].

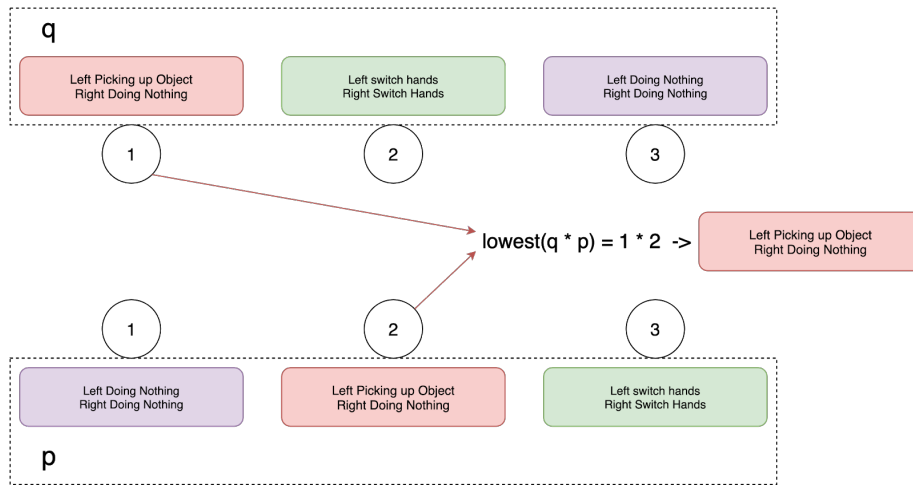


Figure 28: Scoring and selecting a Activity

6.4 Resulting CSV

	rec_name	start (in seconds)	end (in seconds)	label
1	s06t01_df	0	2	la-nothing_ra-nothing
2	s06t01_df	2	4	la-nothing_ra-nothing
3	s06t01_df	4	6	la-nothing_ra-nothing
4	s06t01_df	6	8	la-nothing_ra-nothing
5	s06t01_df	8	10	la-object-place_ra-nothing
6	s06t01_df	10	12	la-nothing_ra-nothing
7	s06t01_df	12	14	la-object-carry_ra-object-carry
8	s06t01_df	14	16	la-nothing_ra-nothing
9	s06t01_df	16	18	la-nothing_ra-nothing
10	s06t01_df	18	20	la-object-carry_ra-object-carry
11	s06t01_df	20	22	la-object-orient_ra-nothing
12	s06t01_df	22	24	la-nothing_ra-object-place
13	s06t01_df	24	26	la-object-orient_ra-object-pick
14	s06t01_df	26	28	la-object-switch-hands_ra-object-orient
15	s06t01_df	28	30	la-object-carry_ra-object-carry
16	s06t01_df	30	32	la-object-switch-hands_ra-object-orient
17	s06t01_df	32	34	la-object-orient_ra-object-switch-hands
18	s06t01_df	34	36	la-object-orient_ra-object-pick
19	s06t01_df	36	38	la-nothing_ra-nothing
20	s06t01_df	38	40	la-object-switch-hands_ra-nothing
21	s06t01_df	40	42	la-object-orient_ra-object-pick
22	s06t01_df	42	44	la-nothing_ra-object-place
23	s06t01_df	44	46	la-object-orient_ra-object-pick
24	s06t01_df	46	48	la-object-orient_ra-nothing
25	s06t01_df	48	50	la-object-place_ra-nothing
26	s06t01_df	50	52	la-object-switch-hands_ra-object-orient
27	s06t01_df	52	54	la-object-place_ra-nothing
28	s06t01_df	54	56	la-nothing_ra-nothing
29	s06t01_df	56	58	la-object-orient_ra-object-pick
30	s06t01_df	58	60	la-object-orient_ra-object-place
31	s06t01_df	60	62	la-object-switch-hands_ra-object-orient
32	s06t01_df	62	64	la-nothing_ra-nothing
33	s06t01_df	64	66	la-object-switch-hands_ra-nothing
34	s06t01_df	66	68	la-nothing_ra-nothing
35	s06t01_df	68	70	la-nothing_ra-nothing

Figure 29

7 Improvements and Further Study

7.1 In Preprocessing

The quality and reliability of MOCAP preprocessed data can be increased by uniformly increasing or decreasing values according to the initial and final values of the missing data, rather than adding mean values. We were unable to implement it due to the complexity of the overall challenge and the constraint of time.

A proper scaling algorithm could have been implemented for upscaling and downscaling the signal which would have protected the integrity of the signal no matter the scaling.

7.2 In Test Data

Like Rupture’s Window Based Search Method for finding changes in a signal, an improved algorithm could have been used that would have given not accurate but close intervals for an activity. The Time Series signal processing and Scaling algorithm would have enabled us to implement the theory of Neural Networks, which is the most suitable methodology for time series implementation.

7.3 In Theory

Recurrent Neural Network’s Long and Short Term Memory architecture was developed to tackle time series challenges like these. Unlike regular neural networks where neurons store information with themselves and are based on feed forward networks, LSTMS are capable of learning long term dependencies due to their feedback connections. With an adequate amount of time LSTM would have enabled us to implement a standard architecture and consistent results.

8 References

References

- [1] T. P. 'P Dohnalek', 'P Gajdos' and V. Snasel', *An Overview of Classification Techniques for Human Activity Recognition*. 2013-11-1.
- [2] V. B. "Terry Taewoong Um" and D. Kulic, *Exercise Motion Classification from Large-scale Werable Sensor Data Using Convolutional Neural Networks*. 2017-07-22.
- [3] H. Jaeger, *Machine Learning: Lecture Notes*. 2019-02-04.
- [4] O. L. V. N. "Truong, Charles ", *A review of change point detection methods*. 2018.
- [5] A. . M. "Ahmed", "Ejaz Clark", *A Novel Sliding Window Based Change Detection Algorithm for Asymmetric Traffic*. 10.1109/NPC.2008.81. 2008.
- [6] D. . "Bickel, *Basic Probability and Statistics*. 10.1201/9780429299308-1. 2019.

A Code

```
"""
file: window based search method.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt

df1 = pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t01.emg.csv')

X = np.array(df1.iloc[:,1:2])

model = "l2"
algo = rpt.Window(width=1500, model=model).fit(X)
my_bkps = algo.predict(n_bkps=50)
rpt.show.display(X, my_bkps, figsize=(18, 15))
plt.title('Change Point Detection: Window-Based Search Method')
plt.show()

"""
file: Description.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

edf = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t01.emg.csv")).iloc[:,1:]

mdf = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t01mocap_with_labels.csv")

s1t1_desc_emg = edf.describe()

s1t2_decs_mocap = mdf.describe()

s1t1_null_emg = (edf.isna()).sum()
```

```

s1t2_null_mocap = (mdf.isna()).sum()

"""
File: Resampling.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t01.emg.csv")).iloc[0:73,:]

index = pd.date_range('1/1/2000', periods=9, freq='T')

series = pd.Series([32165,32191,32351,32509,32823,32615,33009,33159,33318], index=index)

downscale = series.resample('3T').mean()

upscale = series.resample('30S').pad()

fig, ax = plt.subplots(3)

ax[0].plot(index, series)
ax[0].set_title("normal")

ax[1].plot(upscale.index, upscale)
ax[1].set_title("upscale")

ax[2].plot(downscale.index, downscale)
ax[2].set_title("downscale")

plt.show()

"""
File: MOCAP Timeseries Description.py
"""
import pandas as pd

```

```

import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

df1 = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t01mocap_with_labels.

labelencoder = preprocessing.LabelEncoder()
df1['left_hand_labels'] = labelencoder.fit_transform(df1['left_hand_labels'])

df1 = df1.fillna(df1.mean())

fig, ax = plt.subplots(2)

ax[0].plot(df1["ts"], df1["Hip_Position_Z"])
ax[0].set_title("Hip_position_Z")

ax[1].plot(df1["ts"], df1["Ab_Rotation_X"])
ax[1].set_title("Ab_rotation_X")

enc = df1["left_hand_labels"]
p_n = enc[0]
x_n = 0

for en in enc:
    if en != p_n:
        ax[0].axvline(x = df1["ts"][x_n], color = "r")
        ax[1].axvline(x = df1["ts"][x_n], color = "r")

        p_n = en
    x_n += 1
plt.show()

"""
File: right hand comparision.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

```

```

df1 = (pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\one\\sir1_right.csv')).iloc[:70489,1:]

labelencoder = preprocessing.LabelEncoder()
df1['labels'] = labelencoder.fit_transform(df1['labels'])

fig, ax = plt.subplots(4)

ax[0].plot(df1["ts"], df1["fa-o-t-r"])
ax[0].set_title("outside top of right forearm")

ax[1].plot(df1["ts"], df1["fa-i-t-r"])
ax[1].set_title("top of inside of right forearm")

ax[2].plot(df1["ts"], df1["fa-i-b-r"])
ax[2].set_title("inner bottom of right forearm")

ax[3].plot(df1["ts"], df1["fa-o-b-r"])
ax[3].set_title("outside bottom of right forearm")


enc = df1["labels"]
p_n = enc[0]
x_n = 0

for en in enc:
    if en != p_n:
        ax[0].axvline(x = df1["ts"][x_n], color = "r")
        ax[1].axvline(x = df1["ts"][x_n], color = "r")
        ax[2].axvline(x = df1["ts"][x_n], color = "r")
        ax[3].axvline(x = df1["ts"][x_n], color = "r")
        p_n = en
    x_n += 1
plt.show()


"""
File: left hand comparision.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

```

```

df1 = (pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\one\\sir1_left.csv')).iloc[:70489,1:]

labelencoder = preprocessing.LabelEncoder()
df1['left_hand_labels'] = labelencoder.fit_transform(df1['left_hand_labels'])

fig, ax = plt.subplots(4)

ax[0].plot(df1["ts"], df1["fa-o-t-l"])
ax[0].set_title("outside top of left forearm")

ax[1].plot(df1["ts"], df1["fa-i-t-l"])
ax[1].set_title("top of inside of left forearm")

ax[2].plot(df1["ts"], df1["fa-i-b-l"])
ax[2].set_title("inner bottom of left forearm")

ax[3].plot(df1["ts"], df1["fa-o-b-l"])
ax[3].set_title("outside bottom of left forearm")

enc = df1["left_hand_labels"]
p_n = enc[0]
x_n = 0

for en in enc:
    if en != p_n:
        ax[0].axvline(x = df1["ts"][x_n], color = "r")
        ax[1].axvline(x = df1["ts"][x_n], color = "r")
        ax[2].axvline(x = df1["ts"][x_n], color = "r")
        ax[3].axvline(x = df1["ts"][x_n], color = "r")
        p_n = en
    x_n += 1
plt.show()

# -*- coding: utf-8 -*-
"""
Created on Wed Apr 1 05:33:34 2020

"""

"""
File: Make time series batch array.py
"""
import pandas as pd

```

```

import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

df1 = (pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\one\\labels-train.csv', header = None))
df1 = df1.set_axis(['V', 'W', 'X', 'Y'], axis=1, inplace=False)

df_r = df1.iloc[40:113]
df_r = df_r.reset_index(drop=True)

l_df = (np.array((pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\one\\labels-train.csv',squeeze=True, h

r_df = (np.array((pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\one\\labels-train.csv',squeeze=True, h

comb_arr = np.array([0,0,"la-nothing_ra-nothing"])
emg_comb_arr = np.array([0,0])

for i in range(0,max(len(l_df),len(r_df))-1):

    if i == 0 :
        str1 = df1.Y[np.where(df1.W == 7.37)[0][0]] + '_' + df1.Y[np.where(df1.W == 7.37)[0][1]]
        stk1 = np.array([l_df[0],r_df[0],str1])
        comb_arr = np.vstack((comb_arr,stk1))

    elif i == max(len(l_df),len(r_df)) - 1:
        strk_slast = np.array([l_df[-2],r_df[-2],"la-nothing_ra-nothing"])
        comb_arr = np.vstack((comb_arr,stk1))

    else:
        closest_value = min(l_df, key= lambda l_value : abs(l_value - r_df[i]))
        str2 = df1.Y[np.where(df1.W == closest_value)[0][0]] + '_' + df_r.Y[np.where(df_r.W == r_df[i])
        stk2 = np.array([closest_value,r_df[i],str2])
        comb_arr = np.vstack((comb_arr,stk2))

print(i)

"""
File: MOCAP Processing.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt

```

```

from sklearn import preprocessing

s1t2_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t02mocap_with_lab
s1t1_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t01mocap_with_lab

s1t3_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t03mocap_with_lab
s1t4_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t04mocap_with_lab
s1t5_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t05mocap_with_lab
s1t6_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t06mocap_with_lab

s2t1_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s02t01mocap_with_lab
s2t2_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s02t02mocap_with_lab
s2t3_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s02t03mocap_with_lab
s2t4_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s02t04mocap_with_lab
s2t5_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s02t05mocap_with_lab
s2t6_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s02t06mocap_with_lab

s3t1_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s03t01mocap_with_lab
s3t2_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s03t02mocap_with_lab
s3t3_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s03t03mocap_with_lab
s3t4_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s03t04mocap_with_lab
s3t5_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s03t05mocap_with_lab

s4t1_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s04t01mocap_with_lab
s4t2_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s05t02mocap_with_lab

s5t1_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s05t06mocap_with_lab

df_list = [s1t1_df, s1t2_df, s1t3_df,s1t4_df,s1t5_df,s1t6_df,s2t1_df,s2t2_df,s2t3_df,s2t4_df,s2t5_df,
            s3t4_df,s3t4_df,s4t1_df, s4t2_df, s5t1_df]

comb_arr = np.array([0,'labels'])

for df in df_list:

df['left_hand_labels'] = df['left_hand_labels'] + '_' + df['right_hand_labels']
df = df.fillna(df.mean())
length = int(len(df) / 300) - 1

```



```

for i in range(0,length):

    new_df = df.iloc[i*375:(i+1)*375,1:]
    new_df['left_hand_labels'] = new_df['left_hand_labels'].astype(str)
    srt = max(new_df['left_hand_labels'].unique())
    sd_df = new_df.iloc[:, :-2]
    sd = sum(sd_df.std()) / 300

    stk = np.array([sd,srt])
    comb_arr = np.vstack((comb_arr,stk))

std_DF = pd.DataFrame(comb_arr, columns = ['std','label'])
encoded_sd_df = std_DF.iloc[1:,:]
le = preprocessing.LabelEncoder()
encoded_sd_df['label'] = le.fit_transform(encoded_sd_df['label'])

encoded_sd_df = encoded_sd_df.reset_index(drop=1)

ending_df = np.array([0,'label'])

for i in range(0,36):

    filter1 = encoded_sd_df.loc[encoded_sd_df['label'] == i]
    filter1.iloc[:, :1] = filter1.iloc[:, :1].astype(float)
    x_total = (filter1['std'].sum() ) / len(filter1.label)
    total_ = np.array([x_total,i])
    ending_df = np.vstack((ending_df,total_))

    ending_DF = (pd.DataFrame(ending_df,columns = ['Average Standard Deviation','label'])).iloc[1:,:]
    ending_DF['label'] = (ending_DF['label'].astype(float)).astype(int)
    ending_DF.label = le.inverse_transform(ending_DF.label)

    ending_DF.to_pickle('ending_DF_mocap.pkl')

"""
File: Preprocessing and score calculation.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt

```

```
from sklearn import preprocessing
```

```
df = (pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\one\\labels-train.csv', header = None)).set_axis([
```

```
s1t1_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t01.emg.csv", header = None, s
s1t2_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t02.emg.csv", header = None, s
s1t3_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t03.emg.csv", header = None, s
s1t4_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t04.emg.csv", header = None, s
s1t5_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t05.emg.csv", header = None, s
s1t6_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t06.emg.csv", header = None, s
```

```
s2t1_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s02t01.emg.csv", header = None, s
s2t2_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s02t02.emg.csv", header = None, s
s2t3_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s02t03.emg.csv", header = None, s
s2t4_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s02t04.emg.csv", header = None, s
s2t5_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s02t05.emg.csv", header = None, s
s2t6_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s02t06.emg.csv", header = None, s
```

```
s3t1_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s03t01.emg.csv", header = None, s
s3t2_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s03t02.emg.csv", header = None, s
s3t3_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s03t03.emg.csv", header = None, s
s3t4_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s03t04.emg.csv", header = None, s
s3t5_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s03t05.emg.csv", header = None, s
```

```
s4t1_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s04t01.emg.csv", header = None, s
s4t2_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s05t02.emg.csv", header = None, s
```

```
s5t1_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s05t06.emg.csv", header = None, s
```

```
s1t1_l , s1t1_r, s1t1_df_l, s1t1_df_r = np.array((df.iloc[:40,2:3])).flatten() , np.array((df.iloc[4
s1t2_l , s1t2_r, s1t2_df_l, s1t2_df_r = np.array((df.iloc[113:162,2:3])).flatten() , np.array((df.il
s1t3_l , s1t3_r, s1t3_df_l, s1t3_df_r = np.array((df.iloc[279:308,2:3])).flatten() , np.array((df.il
s1t4_l , s1t4_r, s1t4_df_l, s1t4_df_r = np.array((df.iloc[393:422,2:3])).flatten() , np.array((df.il
s1t5_l , s1t5_r, s1t5_df_l, s1t5_df_r = np.array((df.iloc[495:546,2:3])).flatten() , np.array((df.il
s1t6_l , s1t6_r, s1t6_df_l, s1t6_df_r = np.array((df.iloc[621:650,2:3])).flatten() , np.array((df.il
```

```
s2t1_l , s2t1_r, s2t1_df_l, s2t1_df_r = np.array((df.iloc[693:752,2:3])).flatten() , np.array((df.il
s2t2_l , s2t2_r, s2t2_df_l, s2t2_df_r = np.array((df.iloc[832:923,2:3])).flatten() , np.array((df.il
s2t3_l , s2t3_r, s2t3_df_l, s2t3_df_r = np.array((df.iloc[987:1048,2:3])).flatten() , np.array((df.i
s2t4_l , s2t4_r, s2t4_df_l, s2t4_df_r = np.array((df.iloc[1134:1163,2:3])).flatten() , np.array((df.
s2t5_l , s2t5_r, s2t5_df_l, s2t5_df_r = np.array((df.iloc[1205:1287,2:3])).flatten() , np.array((df.
s2t6_l , s2t6_r, s2t6_df_l, s2t6_df_r = np.array((df.iloc[1349:1401,2:3])).flatten() , np.array((df.i
```

```
s3t1_l , s3t1_r, s3t1_df_l, s3t1_df_r = np.array((df.iloc[1470:1515,2:3])).flatten() , np.array((df.
s3t2_l , s3t2_r, s3t2_df_l, s3t2_df_r = np.array((df.iloc[1590:1635,2:3])).flatten() , np.array((df.
```

```

s3t3_l , s3t3_r, s3t3_df_l, s3t3_df_r = np.array((df.iloc[1694:1750,2:3])).flatten() , np.array((df.
s3t4_l , s3t4_r, s3t4_df_l, s3t4_df_r = np.array((df.iloc[1843:1884,2:3])).flatten() , np.array((df.
s3t5_l , s3t5_r, s3t5_df_l, s3t5_df_r = np.array((df.iloc[1941:1993,2:3])).flatten() , np.array((df.

s4t1_l , s4t1_r, s4t1_df_l, s4t1_df_r = np.array((df.iloc[2066:2105,2:3])).flatten() , np.array((df.
s4t2_l , s4t2_r, s4t2_df_l, s4t2_df_r = np.array((df.iloc[2160:2234,2:3])).flatten() , np.array((df.

s5t1_l , s5t1_r, s5t1_df_l, s5t1_df_r = np.array((df.iloc[2380:2412,2:3])).flatten() , np.array((df.

l_list = [s1t1_l, s1t2_l, s1t3_l, s1t4_l, s1t5_l, s1t6_l, s2t1_l, s2t2_l , s2t3_l , s2t4_l, s2t5_l,
s3t4_l, s3t5_l , s4t1_l , s4t2_l , s5t1_l]
r_list = [s1t1_r , s1t2_r , s1t3_r , s1t4_r , s1t5_r , s1t6_r , s2t1_r , s2t2_r , s2t3_r , s2t4_r ,
s3t3_r , s3t4_r , s3t5_r , s4t1_r , s4t2_r , s5t1_r]
df_l_list = [s1t1_df_l, s1t2_df_l, s1t3_df_l, s1t4_df_l, s1t5_df_l, s1t6_df_l, s2t1_df_l, s2t2_df_l,
s3t4_df_l, s3t5_df_l , s4t1_df_l , s4t2_df_l , s5t1_df_l]
df_r_list = [s1t1_df_r , s1t2_df_r , s1t3_df_r , s1t4_df_r , s1t5_df_r , s1t6_df_r , s2t1_df_r , s2t2_df_r ,
s3t3_df_r , s3t4_df_r , s3t5_df_r , s4t1_df_r , s4t2_df_r , s5t1_df_r]

df_list = [s1t1_df, s1t2_df, s1t3_df,s1t4_df,s1t5_df,s1t6_df,s2t1_df,s2t2_df,s2t3_df,s2t4_df,s2t5_df,
s3t4_df,s3t4_df,s4t1_df, s4t2_df, s5t1_df]

name_list = ['s1t1_ts', 's1t2_ts', 's1t3_ts', 's1t4_ts', 's1t5_ts', 's1t6_ts', 's2t1_ts', 's2t2_ts',
's3t4_ts', 's3t5_ts' , 's4t1_ts' , 's4t2_ts' , 's5t1_ts']

comb_arr = np.array([0,0,"labels"])
master_df = np.array([[0],[0],'labels'])
std_df = np.array([0,0,"labels"])
flag = 0

for l_ , r_ ,dfl_ , dfr_ , n_l, df_ in zip(l_list, r_list, df_l_list, df_r_list,name_list, df_list):

str0 = "la-nothing_ra-nothing"
stk0 = np.array([0,0,str0])
comb_arr = np.vstack((comb_arr,stk0))

if len(l_) > len(r_):
    flag = 0
else:
    flag = 1

for i in range(0,max(len(l_),len(r_)) -1):

    if i == 0 :
        str1 = dfl_.Y[2] + '_' + dfr_.Y[2]
        stk1 = np.array([l_[0],r_[0],str1])

```

```

    comb_arr = np.vstack((comb_arr,stk1))

elif i == max(len(l_),len(r_)) - 1:
    strk_slast = np.array([l_[-2],r_[-2],"la-nothing_ra-nothing"])
    comb_arr = np.vstack((comb_arr,stk1))

else:

    if flag == 1:
        closest_value = min(l_, key= lambda l_value : abs(l_value - r_[i]))
        str2 = dfl_.Y[np.where(dfl_.X == closest_value)[0][0]] + '_' + dfr_.Y[i]
        stk2 = np.array([closest_value,r_[i],str2])
        comb_arr = np.vstack((comb_arr,stk2))

        if float(comb_arr[-2][0]) != float(comb_arr[-1][0]):
            x_ = (df_.loc[float(comb_arr[-2][0]):float(comb_arr[-1][0])]).iloc[:,4:]
            y_ = (df_.loc[float(comb_arr[-2][1]):float(comb_arr[-1][1])]).iloc[:,4:]

        else:
            x_ = x_
            y_ = (df_.loc[float(comb_arr[-2][1]):float(comb_arr[-1][1])]).iloc[:,4:]

        std_stk1 = np.array([sum(x_.std()) / 4 , sum(y_.std()) / 4, str2])
        std_df = np.vstack((std_df,std_stk1))

        z_ = np.array([x_,y_,str2])
        master_df = np.vstack((master_df,z_))

    else:
        closest_value = min(r_, key= lambda r_value : abs(r_value - l_[i]))
        str2 = dfl_.Y[i] + '_' + dfr_.Y[np.where(dfr_.X == closest_value)[0][0]]
        stk2 = np.array([l_[i],closest_value,str2])
        comb_arr = np.vstack((comb_arr,stk2))

        if float(comb_arr[-2][1]) != float(comb_arr[-1][1]):
            x_ = (df_.loc[float(comb_arr[-2][0]):float(comb_arr[-1][0])]).iloc[:,4:]
            y_ = (df_.loc[float(comb_arr[-2][1]):float(comb_arr[-1][1])]).iloc[:,4:]

        else:
            x_ = (df_.loc[float(comb_arr[-2][0]):float(comb_arr[-1][0])]).iloc[:,4:]
            y_ = y_

        z_ = np.array([x_,y_,str2])

```

```

std_stk1 = np.array([sum(x_.std()) / 4 , sum(y_.std()) / 4, str2])
std_df = np.vstack((std_df,std_stk1))

master_df = np.vstack((master_df,z_))

#combDF = pd.DataFrame(comb_arr)
#combDF.to_pickle('' + n_1 + '.pkl')
std_DF = pd.DataFrame(std_df, columns = ['left','right','label'])
encoded_sd_df = std_DF.iloc[1:,:]
le = preprocessing.LabelEncoder()
encoded_sd_df['label'] = le.fit_transform(encoded_sd_df['label'])

encoded_sd_df = encoded_sd_df.reset_index(drop=1)

#averaged
ending_df = np.array([0,'label'])

for i in range(0,36):

    filter1 = encoded_sd_df.loc[encoded_sd_df['label'] == i]
    filter1.iloc[:,2] = filter1.iloc[:,2].astype(float)
    l_total = (filter1['left'].sum() ) / len(filter1.left)
    r_total = (filter1['right'].sum() ) / len(filter1.right)
    x_total = (r_total + l_total) / 2
    total_ = np.array([x_total,i])
    ending_df = np.vstack((ending_df,total_))

    ending_DF = (pd.DataFrame(ending_df,columns = ['Average Standard Deviation','label'])).iloc[1:,:]
    ending_DF['label'] = (ending_DF['label'].astype(float)).astype(int)
    ending_DF.label = le.inverse_transform(ending_DF.label)

    ending_DF.to_pickle('ending_DF_emg.pkl')

    combDF = pd.DataFrame(comb_arr)
    combDF.to_pickle('comb_DF_TS.pkl')

    master_DF = pd.DataFrame(master_df)
    master_DF.to_pickle('master_DF.pkl')

    std_DF = pd.DataFrame(std_df)
    std_DF.to_pickle('std_DF.pkl')

```

```

"""
File: s6 test result.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

emg = pd.read_pickle("ending_DF_emg.pkl")
mocap = pd.read_pickle("ending_DF.pkl")

emg.iloc[:,0:1] = (emg.iloc[:,0:1]).astype(float)
mocap.iloc[:,0:1] = (mocap.iloc[:,0:1]).astype(float)

s06t01_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\emg\\s06t01.emg.csv")
s06t02_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\emg\\s06t02.emg.csv")
s06t03_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\emg\\s06t03.emg.csv")
s06t04_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\emg\\s06t04.emg.csv")
s06t05_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\emg\\s06t05.emg.csv")

c1 = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\mocap\\s06t01.mocap.csv")
c2 = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\mocap\\s06t02.mocap.csv")
c3 = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\mocap\\s06t03.mocap.csv")
c4 = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\mocap\\s06t04.mocap.csv")
c5 = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\mocap\\s06t05.mocap.csv")

t_list = [s06t01_df, s06t02_df, s06t03_df, s06t04_df, s06t05_df]
t_name_list = ['s06t01_df', 's06t02_df', 's06t03_df', 's06t04_df', 's06t05_df']

c_list = [c1,c2,c3,c4,c5]
n = 1200
cn = 300

result_list = np.array(['recName',0,0,'label'])

for t_, name_, c_ in zip(t_list, t_name_list, c_list) :

for i,z,ci,cz in zip(range(0,int(len(t_)/n -1)),range(0,int(len(t_)/n - 2),2), range(0,int(len(c_)/n - 1),2)) :

    compare_result = ''

    if i == 0:

```

```

        small_df = t_.iloc[0:1200,1:]
        cdf = c_.iloc[0:300,1:]
    else:
        small_df = t_.iloc[i*n:(i+1)*n,1:]
        cdf = c_.iloc[ci*cn:(ci+1)*cn,1:]

    std_sum = (sum(small_df.std()) ) / 8
    cstd = (sum(cdf.std()) ) / 375

    if std_sum < 400:
        compare_result = 'la-nothing_ra-nothing'

    else:
        closest_value_emg = min((np.array(emg.iloc[:,0:1])).flatten(), key= lambda r_value : abs(r_value))
        compare_result_emg = (np.array(emg.loc[emg['Average Standard Deviation'] == closest_value_emg])

        closest_value_mocap = min((np.array(mocap.iloc[:,0:1])).flatten(), key= lambda r_value : abs(r_value))
        compare_result_mocap = (np.array(mocap.loc[mocap['Average Standard Deviation'] == closest_value_mocap])

        score_arr = (np.array([compare_result_emg,compare_result_mocap])).flatten()
        score_df = pd.DataFrame(score_arr,columns = 'scorer')
        le = preprocessing.LabelEncoder()
        score_df['scorer'] = le.fit_transform(score_df['scorer'])
        if max(score_df['scorer'].unique()) > 3:
            compare_result = compare_result_emg[0]
        else:
            s_c = np.array(score_df)
            scorer_max = max(s_c[0]*s_c[0],s_c[0]*s_c[1],s_c[0]*s_c[2],s_c[1]*s_c[0],s_c[1]*s_c[1],s_c[1]*s_c[2],s_c[2]*s_c[0],s_c[2]*s_c[1],s_c[2]*s_c[2])
            compare_result = str(score_df.loc[scorer_max])
    print(i)

    stk = np.array([name_,z,z+2,compare_result])
    result_list = np.vstack((result_list,stk))

result_DF = (pd.DataFrame(result_list,columns = ['rec_name','start (in seconds)','end (in seconds)'],

result_DF.to_csv('Result.csv')

"""
file: window based search method.py
"""

import pandas as pd

```

```

import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt

df1 = pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t01.emg.csv')

X = np.array(df1.iloc[:,1:2])

model = "l2"
algo = rpt.Window(width=1500, model=model).fit(X)
my_bkps = algo.predict(n_bkps=50)
rpt.show.display(X, my_bkps, figsize=(18, 15))
plt.title('Change Point Detection: Window-Based Search Method')
plt.show()

"""
file: Description.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

edf = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t01.emg.csv")).iloc[:,1:]

mdf = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t01mocap_with_labels.csv")

s1t1_desc_emg = edf.describe()

s1t2_decs_mocap = mdf.describe()

s1t1_null_emg = (edf.isna()).sum()

s1t2_null_mocap = (mdf.isna()).sum()

'''
File: Labeling_MOCAP.py
'''

#This is just to labelise the data from mocap according to

```



```

#time stamps of train.label

import os
import glob
import pandas as pd
import numpy as np

#set paths for the files:
save_to_path = 'F:\\Jacobs\\SEM 2\\Data Mining\\BBDC\\Try1\\emg_with_labels'
dir_path_for_label_train = 'F:\\Jacobs\\SEM 2\\Data Mining\\BBDC\\Try1\\Labels_train_splited'
dir_path_for_emg_files = 'F:\\Jacobs\\SEM 2\\Data Mining\\BBDC\\Try1\\emg'

files_for_labels_train = glob.glob(os.path.join(dir_path_for_label_train,"*.csv"))
files_for_emg = glob.glob(os.path.join(dir_path_for_emg_files,"*.csv"))

for file in range(18,20):
    file_path_labels_train = files_for_labels_train[file]
    file_path_mocap = files_for_emg[file]
    #reading labels from labels_train_splitted:
    labels_train = pd.read_csv(file_path_labels_train)

    #split labels_train in left and right hands:
    left = pd.DataFrame(columns=labels_train.columns)
    right = pd.DataFrame(columns=labels_train.columns)
    #manually appending first row:
    left = left.append(labels_train.iloc[0])
    zero_flag = 0
    #appending the rest for left:
    for z in range(1,labels_train.count()[0]):
        if labels_train.loc[z,'Start_time'] != 0:
            if zero_flag == 1:
                right = right.append(labels_train.iloc[z],ignore_index=True)
            else:
                left = left.append(labels_train.iloc[z])
            else:
                zero_flag = 1
                right = right.append(labels_train.iloc[z],ignore_index=True)

    #reading mocap (only one file from mocap):
    mocap = pd.read_csv(file_path_mocap)

    #create label column in mocap:
    mocap['left_hand_labels']=np.nan
    mocap['right_hand_labels']=np.nan

    #labelling:

```

```

#...for left hand
#indexing variable for left labels:
i = 0
for row in range(len(mocap.iloc[:,0])):
    print('For file number: ',file,' : Now at ',i,' in left and at row ',row,' in emg.')
    if (mocap.iloc[row,0]<=left.loc[i,'End_time']) and ((left.loc[i,'Start_time'] != 0) or (left.loc[i,'Start_time'] == 0)):
        mocap.loc[row,'left_hand_labels'] = left.loc[i,'Activity']
    elif mocap.iloc[row,0]>=left.loc[i,'End_time'] and i+1==len(left.index):
        break
    else:
        i = i + 1
        mocap.loc[row,'left_hand_labels'] = left.loc[i,'Activity']

#...for right hand
#indexing variable for right labels:
i = 0
for row in range(len(mocap.iloc[:,0])):
    print('For file number: ',file,' : Now at ',i,' in right and at row ',row,' in emg.')
    if (mocap.iloc[row,0]<=right.loc[i,'End_time']) and ((right.loc[i,'Start_time'] != 0) or (right.loc[i,'Start_time'] == 0)):
        mocap.loc[row,'right_hand_labels'] = right.loc[i,'Activity']
    elif mocap.iloc[row,0]>=right.loc[i,'End_time'] and i+1==len(right.index):
        break
    else:
        i = i + 1
        mocap.loc[row,'right_hand_labels'] = right.loc[i,'Activity']

#saving labeled mocap:
file_name = file_path_mocap[-14:]
mocap.to_csv(os.path.join(save_to_path,file_name.split('.')[0])+file_name.split('.')[1]+'_with_labels.csv')
print('Done with ',file_name)

'''
File: SlicingTrainLabels.py
'''

#This is to slice the labels_train.csv into s0xt0x parts

from os import listdir
from os.path import isfile, join
import os
import glob
import pandas as pd
import numpy as np

mypath = 'F:\\Jacobs\\SEM 2\\Data Mining\\BBDC\\Try1\\mocap'
onlyfiles = [f for f in listdir(mypath) if isfile(join(mypath, f))]
mocap_csvfile_names = [s[:6] for s in onlyfiles]

```

```

print('This is Mocap csv files list:',mocap_csvfile_names)

#reading labels_train:
labels_train = pd.read_csv("F:\\\\Jacobs\\SEM 2\\Data Mining\\BBDC\\Try1\\Labels_train.csv")

#declaring counter for mocap_csv_file_names list:
m = 0
#creating temporary dataframe:
tempdf = pd.DataFrame(columns=['Category','Start_time','End_time','Activity'])
#setting file path to save csv files:
file_path = 'F:\\\\Jacobs\\SEM 2\\Data Mining\\BBDC\\Try1\\Labels_train_splited'

for i in range(labels_train.count()[0]):
    print('Now at ',mocap_csvfile_names[m],' and at ',i,' row in labels_train')
    if mocap_csvfile_names[m] in labels_train.loc[i,'Category']:
        tempdf = tempdf.append(labels_train.iloc[i])
    else:
        tempdf.to_csv(os.path.join(file_path,mocap_csvfile_names[m])+'.csv',index=False)
        m = m + 1
        tempdf = tempdf[0:0]
        tempdf = tempdf.append(labels_train.iloc[i])
tempdf.to_csv(os.path.join(file_path,mocap_csvfile_names[m])+'.csv',index=False)

"""
File: Resampling.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

df = (pd.read_csv("K:\\\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t01.emg.csv")).iloc[0:73,:]

index = pd.date_range('1/1/2000', periods=9, freq='T')

series = pd.Series([32165,32191,32351,32509,32823,32615,33009,33159,33318], index=index)

downscale = series.resample('3T').mean()

upscale = series.resample('30S').pad()

```

```

fig, ax = plt.subplots(3)

ax[0].plot(index, series)
ax[0].set_title("normal")

ax[1].plot(upscale.index, upscale)
ax[1].set_title("upscale")

ax[2].plot(downscale.index, downscale)
ax[2].set_title("downscale")

plt.show()

"""
File: MOCAP Timeseries Description.py
"""
import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

df1 = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t01mocap_with_labels.

labelencoder = preprocessing.LabelEncoder()
df1['left_hand_labels'] = labelencoder.fit_transform(df1['left_hand_labels'])

df1 = df1.fillna(df1.mean())

fig, ax = plt.subplots(2)

ax[0].plot(df1["ts"], df1["Hip_Position_Z"])
ax[0].set_title("Hip_position_Z")

ax[1].plot(df1["ts"], df1["Ab_Rotation_X"])
ax[1].set_title("Ab_rotation_X")

enc = df1["left_hand_labels"]
p_n = enc[0]
x_n = 0

for en in enc:

```

```

if en != p_n:
    ax[0].axvline(x = df1["ts"][x_n], color = "r")
    ax[1].axvline(x = df1["ts"][x_n], color = "r")

    p_n = en
x_n += 1
plt.show()

"""
File: right hand comparision.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

df1 = (pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\one\\s1r1_right.csv')).iloc[:70489,1:]

labelencoder = preprocessing.LabelEncoder()
df1['labels'] = labelencoder.fit_transform(df1['labels'])

fig, ax = plt.subplots(4)

ax[0].plot(df1["ts"], df1["fa-o-t-r"])
ax[0].set_title("outside top of right forearm")

ax[1].plot(df1["ts"], df1["fa-i-t-r"])
ax[1].set_title("top of inside of right forearm")

ax[2].plot(df1["ts"], df1["fa-i-b-r"])
ax[2].set_title("inner bottom of right forearm")

ax[3].plot(df1["ts"], df1["fa-o-b-r"])
ax[3].set_title("outside bottom of right forearm")

enc = df1["labels"]
p_n = enc[0]
x_n = 0

for en in enc:
    if en != p_n:

```

```

        ax[0].axvline(x = df1["ts"][x_n], color = "r")
        ax[1].axvline(x = df1["ts"][x_n], color = "r")
        ax[2].axvline(x = df1["ts"][x_n], color = "r")
        ax[3].axvline(x = df1["ts"][x_n], color = "r")
        p_n = en
    x_n += 1
plt.show()

"""
File: left hand comparision.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

df1 = (pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\one\\s1r1_left.csv')).iloc[:70489,1:]

labelencoder = preprocessing.LabelEncoder()
df1['left_hand_labels'] = labelencoder.fit_transform(df1['left_hand_labels'])

fig, ax = plt.subplots(4)

ax[0].plot(df1["ts"], df1["fa-o-t-l"])
ax[0].set_title("outside top of left forearm")

ax[1].plot(df1["ts"], df1["fa-i-t-l"])
ax[1].set_title("top of inside of left forearm")

ax[2].plot(df1["ts"], df1["fa-i-b-l"])
ax[2].set_title("inner bottom of left forearm")

ax[3].plot(df1["ts"], df1["fa-o-b-l"])
ax[3].set_title("outside bottom of left forearm")

enc = df1["left_hand_labels"]
p_n = enc[0]
x_n = 0

for en in enc:
    if en != p_n:

```

```

        ax[0].axvline(x = df1["ts"][x_n], color = "r")
        ax[1].axvline(x = df1["ts"][x_n], color = "r")
        ax[2].axvline(x = df1["ts"][x_n], color = "r")
        ax[3].axvline(x = df1["ts"][x_n], color = "r")
        p_n = en
    x_n += 1
plt.show()

# -*- coding: utf-8 -*-
"""
Created on Wed Apr  1 05:33:34 2020

"""

"""
File: Make time series batch array.py
"""
import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

df1 = (pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\one\\labels-train.csv', header = None))
df1 = df1.set_axis(['V', 'W', 'X', 'Y'], axis=1, inplace=False)

df_r = df1.iloc[40:113]
df_r = df_r.reset_index(drop=True)

l_df = (np.array((pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\one\\labels-train.csv',squeeze=True, h

r_df = (np.array((pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\one\\labels-train.csv',squeeze=True, h

comb_arr = np.array([0,0,"la-nothing_ra-nothing"])
emg_comb_arr = np.array([0,0])

for i in range(0,max(len(l_df),len(r_df))-1):

    if i == 0 :
        str1 = df1.Y[np.where(df1.W == 7.37)[0][0]] + '_' + df1.Y[np.where(df1.W == 7.37)[0][1]]
        stk1 = np.array([l_df[0],r_df[0],str1])
        comb_arr = np.vstack((comb_arr,stk1))

    elif i == max(len(l_df),len(r_df)) - 1:

```

```

        strk_slast = np.array([l_df[-2],r_df[-2],"la-nothing_ra-nothing"])
        comb_arr = np.vstack((comb_arr,stk1))

    else:
        closest_value = min(l_df, key= lambda l_value : abs(l_value - r_df[i]))
        str2 = df1.Y[np.where(df1.W == closest_value)[0][0]] + '_' + df_r.Y[np.where(df_r.W == r_df[i])]
        stk2 = np.array([closest_value,r_df[i],str2])
        comb_arr = np.vstack((comb_arr,stk2))

print(i)

```

```

"""
File: MOCAP Processing.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

s1t2_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t02mocap_with_labels.csv")
s1t1_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t01mocap_with_labels.csv")

s1t3_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t03mocap_with_labels.csv")
s1t4_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t04mocap_with_labels.csv")
s1t5_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t05mocap_with_labels.csv")
s1t6_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s01t06mocap_with_labels.csv")

```



```

s2t1_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s02t01mocap_with_lab
s2t2_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s02t02mocap_with_lab
s2t3_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s02t03mocap_with_lab
s2t4_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s02t04mocap_with_lab
s2t5_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s02t05mocap_with_lab
s2t6_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s02t06mocap_with_lab

s3t1_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s03t01mocap_with_lab
s3t2_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s03t02mocap_with_lab
s3t3_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s03t03mocap_with_lab
s3t4_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s03t04mocap_with_lab
s3t5_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s03t05mocap_with_lab

s4t1_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s04t01mocap_with_lab
s4t2_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s05t02mocap_with_lab

s5t1_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\mocap_with_labels\\s05t06mocap_with_lab

df_list = [s1t1_df, s1t2_df, s1t3_df,s1t4_df,s1t5_df,s1t6_df,s2t1_df,s2t2_df,s2t3_df,s2t4_df,s2t5_df,
           s3t4_df,s3t4_df,s4t1_df, s4t2_df, s5t1_df]

comb_arr = np.array([0,'labels'])

for df in df_list:

df['left_hand_labels'] = df['left_hand_labels'] + '_' + df['right_hand_labels']
df = df.fillna(df.mean())
length = int(len(df) / 300) - 1

for i in range(0,length):

    new_df = df.iloc[i*375:(i+1)*375,1:]
    new_df['left_hand_labels'] = new_df['left_hand_labels'].astype(str)
    srt = max(new_df['left_hand_labels'].unique())
    sd_df = new_df.iloc[:, :-2]
    sd = sum(sd_df.std()) / 300

    stk = np.array([sd,srt])
    comb_arr = np.vstack((comb_arr,stk))

std_DF = pd.DataFrame(comb_arr, columns = ['std','label'])

```

```

encoded_sd_df = std_DF.iloc[1:,:]
le = preprocessing.LabelEncoder()
encoded_sd_df['label'] = le.fit_transform(encoded_sd_df['label'])

encoded_sd_df = encoded_sd_df.reset_index(drop=1)

ending_df = np.array([0,'label'])

for i in range(0,36):

    filter1 = encoded_sd_df.loc[encoded_sd_df['label'] == i]
    filter1.iloc[:,1] = filter1.iloc[:,1].astype(float)
    x_total = (filter1['std'].sum() ) / len(filter1.label)
    total_ = np.array([x_total,i])
    ending_df = np.vstack((ending_df,total_))

ending_DF = (pd.DataFrame(ending_df,columns = ['Average Standard Deviation','label'])).iloc[1:,:]
ending_DF['label'] = (ending_DF['label'].astype(float)).astype(int)
ending_DF.label = le.inverse_transform(ending_DF.label)

ending_DF.to_pickle('ending_DF_mocap.pkl')

"""
File: Preprocessing and score calculation.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

df = (pd.read_csv('K:\\AA JU DE\\BBDC\\bbdc_2020\\one\\labels-train.csv', header = None)).set_axis(

s1t1_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t01.emg.csv", header = None, s
s1t2_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t02.emg.csv", header = None, s
s1t3_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t03.emg.csv", header = None, s
s1t4_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t04.emg.csv", header = None, s
s1t5_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t05.emg.csv", header = None, s
s1t6_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s01t06.emg.csv", header = None, s

```

```

s2t1_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s02t01.emg.csv", header = None, s
s2t2_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s02t02.emg.csv", header = None, s
s2t3_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s02t03.emg.csv", header = None, s
s2t4_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s02t04.emg.csv", header = None, s
s2t5_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s02t05.emg.csv", header = None, s
s2t6_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s02t06.emg.csv", header = None, s

s3t1_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s03t01.emg.csv", header = None, s
s3t2_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s03t02.emg.csv", header = None, s
s3t3_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s03t03.emg.csv", header = None, s
s3t4_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s03t04.emg.csv", header = None, s
s3t5_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s03t05.emg.csv", header = None, s

s4t1_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s04t01.emg.csv", header = None, s
s4t2_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s05t02.emg.csv", header = None, s

s5t1_df = (pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\train\\emg\\s05t06.emg.csv", header = None, s

s1t1_l , s1t1_r, s1t1_df_l, s1t1_df_r = np.array((df.iloc[:40,2:3])).flatten() , np.array((df.iloc[4
s1t2_l , s1t2_r, s1t2_df_l, s1t2_df_r = np.array((df.iloc[113:162,2:3])).flatten() , np.array((df.il
s1t3_l , s1t3_r, s1t3_df_l, s1t3_df_r = np.array((df.iloc[279:308,2:3])).flatten() , np.array((df.il
s1t4_l , s1t4_r, s1t4_df_l, s1t4_df_r = np.array((df.iloc[393:422,2:3])).flatten() , np.array((df.il
s1t5_l , s1t5_r, s1t5_df_l, s1t5_df_r = np.array((df.iloc[495:546,2:3])).flatten() , np.array((df.il
s1t6_l , s1t6_r, s1t6_df_l, s1t6_df_r = np.array((df.iloc[621:650,2:3])).flatten() , np.array((df.il

s2t1_l , s2t1_r, s2t1_df_l, s2t1_df_r = np.array((df.iloc[693:752,2:3])).flatten() , np.array((df.il
s2t2_l , s2t2_r, s2t2_df_l, s2t2_df_r = np.array((df.iloc[832:923,2:3])).flatten() , np.array((df.il
s2t3_l , s2t3_r, s2t3_df_l, s2t3_df_r = np.array((df.iloc[987:1048,2:3])).flatten() , np.array((df.i
s2t4_l , s2t4_r, s2t4_df_l, s2t4_df_r = np.array((df.iloc[1134:1163,2:3])).flatten() , np.array((df.
s2t5_l , s2t5_r, s2t5_df_l, s2t5_df_r = np.array((df.iloc[1205:1287,2:3])).flatten() , np.array((df.
s2t6_l , s2t6_r, s2t6_df_l, s2t6_df_r = np.array((df.iloc[1349:1401,2:3])).flatten() , np.array((df.i

s3t1_l , s3t1_r, s3t1_df_l, s3t1_df_r = np.array((df.iloc[1470:1515,2:3])).flatten() , np.array((df.
s3t2_l , s3t2_r, s3t2_df_l, s3t2_df_r = np.array((df.iloc[1590:1635,2:3])).flatten() , np.array((df.
s3t3_l , s3t3_r, s3t3_df_l, s3t3_df_r = np.array((df.iloc[1694:1750,2:3])).flatten() , np.array((df.
s3t4_l , s3t4_r, s3t4_df_l, s3t4_df_r = np.array((df.iloc[1843:1884,2:3])).flatten() , np.array((df.
s3t5_l , s3t5_r, s3t5_df_l, s3t5_df_r = np.array((df.iloc[1941:1993,2:3])).flatten() , np.array((df.

s4t1_l , s4t1_r, s4t1_df_l, s4t1_df_r = np.array((df.iloc[2066:2105,2:3])).flatten() , np.array((df.
s4t2_l , s4t2_r, s4t2_df_l, s4t2_df_r = np.array((df.iloc[2160:2234,2:3])).flatten() , np.array((df.

s5t1_l , s5t1_r, s5t1_df_l, s5t1_df_r = np.array((df.iloc[2380:2412,2:3])).flatten() , np.array((df.

l_list = [s1t1_l, s1t2_l, s1t3_l, s1t4_l, s1t5_l, s1t6_l, s2t1_l, s2t2_l , s2t3_l , s2t4_l, s2t5_l,
s3t4_l, s3t5_l , s4t1_l , s4t2_l , s5t1_l]

```

```

r_list = [s1t1_r , s1t2_r , s1t3_r , s1t4_r , s1t5_r , s1t6_r , s2t1_r , s2t2_r , s2t3_r , s2t4_r ,
          s3t3_r , s3t4_r , s3t5_r , s4t1_r , s4t2_r , s5t1_r]
df_l_list = [s1t1_df_l, s1t2_df_l, s1t3_df_l, s1t4_df_l, s1t5_df_l, s1t6_df_l, s2t1_df_l, s2t2_df_l,
             s3t4_df_l, s3t5_df_l , s4t1_df_l , s4t2_df_l , s5t1_df_l]
df_r_list = [s1t1_df_r , s1t2_df_r , s1t3_df_r , s1t4_df_r , s1t5_df_r , s1t6_df_r , s2t1_df_r , s2t2_df_r ,
             s3t3_df_r , s3t4_df_r , s3t5_df_r , s4t1_df_r , s4t2_df_r , s5t1_df_r]

df_list = [s1t1_df, s1t2_df, s1t3_df,s1t4_df,s1t5_df,s1t6_df,s2t1_df,s2t2_df,s2t3_df,s2t4_df,s2t5_df,
           s3t4_df,s3t4_df,s4t1_df, s4t2_df, s5t1_df]

name_list = ['s1t1_ts', 's1t2_ts', 's1t3_ts', 's1t4_ts', 's1t5_ts', 's1t6_ts', 's2t1_ts', 's2t2_ts',
            's3t4_ts', 's3t5_ts' , 's4t1_ts' , 's4t2_ts' , 's5t1_ts']

comb_arr = np.array([0,0,"labels"])
master_df = np.array([[0],[0],'labels'])
std_df = np.array([0,0,"labels"])
flag = 0

for l_ , r_ ,dfl_ , dfr_ , n_l , df_ in zip(l_list, r_list, df_l_list, df_r_list,name_list, df_list):

    str0 = "la-nothing_ra-nothing"
    stk0 = np.array([0,0,str0])
    comb_arr = np.vstack((comb_arr,stk0))

    if len(l_) > len(r_):
        flag = 0
    else:
        flag = 1

    for i in range(0,max(len(l_),len(r_)) -1):

        if i == 0 :
            str1 = dfl_.Y[2] + '_' + dfr_.Y[2]
            stk1 = np.array([l_[0],r_[0],str1])
            comb_arr = np.vstack((comb_arr,stk1))

        elif i == max(len(l_),len(r_)) - 1:
            strk_slast = np.array([l_[-2],r_[-2],"la-nothing_ra-nothing"])
            comb_arr = np.vstack((comb_arr,stk1))

        else:

            if flag == 1:
                closest_value = min(l_ , key= lambda l_value : abs(l_value - r_[i]))
                str2 = dfl_.Y[np.where(dfl_.X == closest_value)[0][0]] + '_' + dfr_.Y[i]
                stk2 = np.array([closest_value,r_[i],str2])

```

```

comb_arr = np.vstack((comb_arr,stk2))

if float(comb_arr[-2][0]) != float(comb_arr[-1][0]):
    x_ = (df_.loc[float(comb_arr[-2][0]):float(comb_arr[-1][0])]).iloc[:,4:]
    y_ = (df_.loc[float(comb_arr[-2][1]):float(comb_arr[-1][1])]).iloc[:,4:]

else:
    x_ = x_
    y_ = (df_.loc[float(comb_arr[-2][1]):float(comb_arr[-1][1])]).iloc[:,4:]

std_stk1 = np.array([sum(x_.std()) / 4 , sum(y_.std()) / 4, str2])
std_df = np.vstack((std_df,std_stk1))

z_ = np.array([x_,y_,str2])
master_df = np.vstack((master_df,z_))

else:
    closest_value = min(r_, key= lambda r_value : abs(r_value - l_[i]))
    str2 = df1_.Y[i] + '_' + dfr_.Y[np.where(dfr_.X == closest_value)[0][0]]
    stk2 = np.array([l_[i],closest_value,str2])
    comb_arr = np.vstack((comb_arr,stk2))

    if float(comb_arr[-2][1]) != float(comb_arr[-1][1]):
        x_ = (df_.loc[float(comb_arr[-2][0]):float(comb_arr[-1][0])]).iloc[:,4:]
        y_ = (df_.loc[float(comb_arr[-2][1]):float(comb_arr[-1][1])]).iloc[:,4:]

    else:
        x_ = (df_.loc[float(comb_arr[-2][0]):float(comb_arr[-1][0])]).iloc[:,4:]
        y_ = y_

    z_ = np.array([x_,y_,str2])

    std_stk1 = np.array([sum(x_.std()) / 4 , sum(y_.std()) / 4, str2])
    std_df = np.vstack((std_df,std_stk1))

    master_df = np.vstack((master_df,z_))

#combDF = pd.DataFrame(comb_arr)
#combDF.to_pickle('' + n_1 + '.pkl')
std_DF = pd.DataFrame(std_df, columns = ['left','right','label'])
encoded_sd_df = std_DF.iloc[1:,:]

```

```

le = preprocessing.LabelEncoder()
encoded_sd_df['label'] = le.fit_transform(encoded_sd_df['label'])

encoded_sd_df = encoded_sd_df.reset_index(drop=1)

#averaged
ending_df = np.array([0,'label'])

for i in range(0,36):

    filter1 = encoded_sd_df.loc[encoded_sd_df['label'] == i]
    filter1.iloc[:,2] = filter1.iloc[:,2].astype(float)
    l_total = (filter1['left'].sum() ) / len(filter1.left)
    r_total = (filter1['right'].sum() ) / len(filter1.right)
    x_total = (r_total + l_total) / 2
    total_ = np.array([x_total,i])
    ending_df = np.vstack((ending_df,total_))

ending_DF = (pd.DataFrame(ending_df,columns = ['Average Standard Deviation','label'])).iloc[1:,:]
ending_DF['label'] = (ending_DF['label'].astype(float)).astype(int)
ending_DF.label = le.inverse_transform(ending_DF.label)

ending_DF.to_pickle('ending_DF_emg.pkl')

combDF = pd.DataFrame(comb_arr)
combDF.to_pickle('comb_DF_TS.pkl')

master_DF = pd.DataFrame(master_df)
master_DF.to_pickle('master_DF.pkl')

std_DF = pd.DataFrame(std_df)
std_DF.to_pickle('std_DF.pkl')

"""
File: s6 test result.py
"""

import pandas as pd
import numpy as np
import ruptures as rpt
import matplotlib.pyplot as plt
from sklearn import preprocessing

```

```

emg = pd.read_pickle("ending_DF_emg.pkl")
mocap = pd.read_pickle("ending_DF.pkl")

emg.iloc[:,0:1] = (emg.iloc[:,0:1]).astype(float)
mocap.iloc[:,0:1] = (mocap.iloc[:,0:1]).astype(float)

s06t01_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\emg\\s06t01.emg.csv")
s06t02_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\emg\\s06t02.emg.csv")
s06t03_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\emg\\s06t03.emg.csv")
s06t04_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\emg\\s06t04.emg.csv")
s06t05_df = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\emg\\s06t05.emg.csv")

c1 = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\mocap\\s06t01.mocap.csv")
c2 = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\mocap\\s06t02.mocap.csv")
c3 = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\mocap\\s06t03.mocap.csv")
c4 = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\mocap\\s06t04.mocap.csv")
c5 = pd.read_csv("K:\\AA JU DE\\BBDC\\bbdc_2020\\test\\mocap\\s06t05.mocap.csv")

t_list = [s06t01_df, s06t02_df, s06t03_df, s06t04_df, s06t05_df]
t_name_list = ['s06t01_df', 's06t02_df', 's06t03_df', 's06t04_df', 's06t05_df']

c_list = [c1,c2,c3,c4,c5]
n = 1200
cn = 300

result_list = np.array(['recName',0,0,'label'])

for t_, name_, c_ in zip(t_list, t_name_list, c_list) :

for i,z,ci,cz in zip(range(0,int(len(t_)/n -1)),range(0,int(len(t_)/n - 2),2), range(0,int(len(c_)/n -1)),range(0,int(len(c_)/n - 2),2)) :

    compare_result = ''

    if i == 0:
        small_df = t_.iloc[0:1200,1:]
        cdf = c_.iloc[0:300,1:]
    else:
        small_df = t_.iloc[i*n:(i+1)*n,1:]
        cdf = c_.iloc[ci*cn:(ci+1)*cn,1:]

    std_sum = (sum(small_df.std()) ) / 8
    cstd = (sum(cdf.std()) ) / 375

    if std_sum < 400:

```

```

        compare_result = 'la-nothing_ra-nothing'

    else:
        closest_value_emg = min((np.array(emg.iloc[:,0:1])).flatten(), key= lambda r_value : abs(r_value - closest_value_emg))
        compare_result_emg = (np.array(emg.loc[emg['Average Standard Deviation'] == closest_value_emg])).flatten()

        closest_value_mocap = min((np.array(mocap.iloc[:,0:1])).flatten(), key= lambda r_value : abs(r_value - closest_value_mocap))
        compare_result_mocap = (np.array(mocap.loc[mocap['Average Standard Deviation'] == closest_value_mocap])).flatten()

        score_arr = (np.array([compare_result_emg,compare_result_mocap])).flatten()
        score_df = pd.DataFrame(score_arr,columns = 'scorer')
        le = preprocessing.LabelEncoder()
        score_df['scorer'] = le.fit_transform(score_df['scorer'])
        if max(score_df['scorer'].unique()) > 3:
            compare_result = compare_result_emg[0]
        else:
            s_c = np.array(score_df)
            scorer_max = max(s_c[0]*s_c[0],s_c[0]*s_c[1],s_c[0]*s_c[2],s_c[1]*s_c[0],s_c[1]*s_c[1],s_c[1]*s_c[2],s_c[2]*s_c[0],s_c[2]*s_c[1],s_c[2]*s_c[2])
            compare_result = str(score_df.loc[scorer_max])

    print(i)

    stk = np.array([name_,z,z+2,compare_result])
    result_list = np.vstack((result_list,stk))

result_DF = (pd.DataFrame(result_list,columns = ['rec_name','start (in seconds)','end (in seconds)'])

result_DF.to_csv('Result.csv')

```