Asyncronous late fusion applied to gesture recognition

Philippe Saade

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Présentée et soutenue par :
Philippe SAADE

Le jeudi 11 mai 2017

Titre :

Fusion tardive asynchrone appliquée à la reconnaissance des gestes

---

ED MITT : Image, Information, Hypermedia

Unité de recherche :

IRIT, SAMOVA

Directeur(s) de Thèse :

M. Philippe JOLY - Professeur à l'Université Toulouse III
M. Ali AWADA - Professeur à l'Université Libanaise

Rapporteurs :

M. Patrick LAMBERT - Professeur des Universités à l'Université Savoie Mont-Blanc
Mme. Jenny BENOIS-PINEAU - Professeur à l'Université de Bordeaux

Autre(s) membre(s) du jury :

M. Frédéric LERASLE - Professeur des Universités à l'Université Toulouse III
Mme. Jean MARTINET - Maitre de Conférence Universitaire à l'Université Lille 1
M. Philippe JOLY - Professeur des Universités à l'Université Toulouse III
M. Ali AWADA - Professeur à l'Université Libanaise

ACKNOWLEDGEMENTS

"*No one who achieves success does so without acknowledging the help of others. The wise and confident acknowledge this help with gratitude*." Alfred North Whitehead

This thesis would have been hard to complete without the guidance of the committee members, and the support of friends and family. It is a pleasure to thank those who made this thesis possible:

To my Thesis advisors, Mr. Philippe JOLY, Professor at Paul Sabatier University and Mr. Ali AWADA, Professor at the Lebanese University, I owe my deepest gratitude for your support and guidance from the beginning of my PHD to end. For all the support and help you offered, your observations and assistance were much appreciated.

To the jury members M. Patrick LAMBERT, Professor at Polytech Annecy-Chambéry University and Mrs. Jenny BENOIS-PINEAU - Professor at Bordeaux University for agreeing to be part of the jury.

To my parents, Elias and Monique for encouraging me and making me who I am today. For all the emotional support throughout the years, thank you.

To my sister Gaêlle for being there for me and for supporting me in time of need.

I also thank the most important and precious person in my life: Maria Mansour, for supporting me throughout these five years, for better or for worse, and has been beside no matter the circumstances. I truly believe that it is a miracle that she still stands beside and takes care of me, every day, every second.

TABLE OF CONTENTS

ACRONYMS

Acronym	Description
AB	AdaBoost
ALF	Asynchronous Late Fusion
ASI	ASynchronous Index
ASIP	ASynchronous Index by Parts
ASIv	ASynchronous Index Vote (combination between the ASI and ASIP)
BoVF	(Bag of Visual Features)
CAP	Captured Dataset
cc	confidence coefficient
CR	Captured & Right-hand wave dataset
DP	Dynamic Programming
DTW	Dynamic Time Warping
ICA	Independent Component Analysis
KNN	K-Nearest Neighbors
MD-DTW	Multi-Dimensional Dynamic Time Warping
PCA	Principal Component Analysis
RF	Random Forest
RL	Right Left dataset
SS	Swimming & Soccer dataset
SVM	Support Vector Machine

GLOSSARY

Some of the terms used in the manuscript can have different interpretations in our domain of work. As a result, we define some terms as they are employed in this thesis in the table below.

Term	Definition
Action	A series of simple gestures resulting in the movement of the body and joints in the same direction within the ROM and DOF in correspondence to a reference recording. The definition of an action is explained in details in the introduction (I.C.b.8)
Asynchronous Late Fusion	The algorithm that classifies recordings containing multiple sequences where the events occur on the different sequences at different instants of time.
Class/labels	The ground truth value of a group of recordings.
Frame	A set of coordinates belonging to a single human skeleton captured at a single instant of time.
Gesture	A part of an action that represents a simple movement (the number of frames in a gesture is small in general) e.g., when waving, the first gesture is raising the hand and the second in putting it back down.
Late Classifier	The late fusion consists of training a set of classifiers separately, then fusing their decision with a late classifier. The classifier that performs the fusion will be called a late classifier.
Early Classifier	The late fusion consists of training a set of classifiers separately, then fusing their decision with a late classifier. The first set of classifiers will be called early classifiers
Real dataset Original dataset	The real/original dataset contains the recordings that have been captured by the RGB-D device. During simulation, synthetic recordings are generated from a real/original dataset.
Real recordings Original recordings	During simulation, synthetic recordings are generated from an original dataset. The recordings contained in the original dataset are called real recordings or original recordings.
Recording	A single performance of a class, or an action. In other words, it is a continuous set of captured frames that have been segmented at the start and end of the performance of this action. A recording can either be captured or artificially generated through a simulation process. Note: a recording can describe an example of something else than an action, for instance, in Chapter VI it is a single day of power consumption parameters.
Sequence	A sequence is either referred to a temporal set of multiple coordinates or a temporal set of a single coordinate (according to the context) recorded or generated for a given joint. For example, the coordinates of Left Hand in an action, or the coordinates for the Left Knee. In the simulation chapter, we consider a sequence as a single coordinate and in the remaining of the thesis for multiple coordinates, single joint.
Skeleton	A skeleton is known by the simplest representation of the human body, formed by joints and articulations. In this thesis, it will only be representation by the 20 joints captured by the Microsoft Kinect version 1.0 (Figure 4)

Skeleton estimation	The localization of the skeleton's joints at a single frame.
Series	A long recording composed of a succession of actions. The recordings might not belong to the same action.
Synthetic/Simulated recordings	Recordings that have been generated algorithmically, and that were not captured from the real world.

I. INTRODUCTION

A. Short introduction on machine learning

In the past decade, we saw many advances in machine learning with many implementations in our daily lives. People use it every day without being aware of it. Its application reached cellphone technologies nowadays (e.g., speech recognition with Google speech to text [1]), self-driven cars [2], email SPAM filtering [3]… Moreover, it helped understand patterns and trends, for example, in stock exchange, weather prediction and even in commonly used websites such as Facebook. We take interest in document classification or document indexing. It is a large field covering the classification of music, texts, images… into different categories or labeling them according to their subject, their content… The term classification can be considered as the center of machine learning. It covers the process of teaching the machine learning algorithms to analyze and comprehend the world.

Some machine learning fields include audio, sound, and speech recognition. The extracted information from audio wave signals are analyzed to characterize and categorize these signals (e.g., Shazam mobile application [4], transcription of airplane flight recorders…). Machine learning extends to other fields such as text classification (e.g., guiding the users' online research (Google) [5] and filtering of SPAM emails), handwriting recognition [6] (e.g., Optical Character Recognition (OCR) [7]…), face [8] and emotion recognition [9] and even action recognition. The latter is an old concept that witnessed considerable advances in the last few years.

Kinesiology [10] is the study of action recognition or human movement, also known as human kinetics. "Kinesiology brings together the fields of anatomy, physiology, physics, and geometry." It involves working with static (nonmoving) and dynamic (moving) systems, which leads us to cite an important difference in dynamic systems between Kinetics and Kinematics. "Kinetics are those forces causing movement, whereas kinematics is the time, space, and mass aspects of a moving system." Our thesis is applied on the kinematics of a dynamic system that moves in space and time. It involves building a statistical approach for the classification of temporal streams. More precisely, it will be implemented on the statistical study of the different decisions taken in action streams at various instants of time. This concept is often described as spatiotemporal reasoning.

B. Classification process

a. Usual classification approaches for dataset classification

A typical approach to machine learning in data classification is to capture a first dataset that is related to the classification problem, and that can be assigned with a label. This dataset can even be prepared and filtered to train the classifier properly without any errors.

Afterward, discriminant information, called features or descriptors, are computed from the captured data. It is common to try to find the most appropriate features for the problem in question.

An appropriate classification solution is established with the appropriate architecture. Some of the well-known architectures are the early fusion, late fusion, and deep architectures (II.D.h, II.D.i).

Finally, a machine learning algorithm is chosen. It is the core of the whole architecture. It is trained with previously selected features and tested on the initial dataset to perform the classification. The whole approach stated previously is usually repeated multiple times with the training and testing phases to find the most appropriate classification approach.

The performances should be obviously evaluated on a separate dataset.

The different classification solutions will be described in details in the literature review (0).

Throughout the various chapters, we adopt the method stated above to obtain the first benchmarks. The datasets, to classify in this thesis, contain temporal information, and every sample contains multiple data streams. Consequently, we adopt a Late Fusion architecture. Our purpose is to enhance the classification architecture to take into account some of the data properties and to improve the performances.

An important statement to consider throughout this thesis is that the features are trained and tested on multiple classifiers regardless of their capability to take proper decisions. Hence, we consider the classification algorithms (machine learning algorithms) as unknown “black boxes.” We neither modify their algorithm nor implement any of their extensions.

The following problems are encountered when working with the adopted classification approach:

- Considering a class of temporal events C_1 and a second class of temporal events C_2, of the same nature than C_1 but shifted in time. An event belonging to C_2 may be classified as belonging to C_1 because the classification schema might not take into account the temporal variations.
- The ability to determine the class of a temporal event is not stable along the time: some temporal sub-intervals can be more discriminant than others for such a classification process. Not all the classification tools do integrate that kind of property.

b. Asynchronous Late Fusion (ALF) (in resume)

The classification of asynchronous sequences is based on the idea that a certain label may be detected correctly in a temporal sequence (output true positive results) at different instants of

time. Nonetheless, the decision process must happen according to a predefined order to reveal a certain class in the dataset. A normal classification process considers that the provided information is enough and that each decision taken along the time is always correct. Nevertheless, this is not always the case.

Hence, a model is built according to the confidence of the decisions that were taken by the early classifier at any time. No matter the type of the output from the early classification of the streams, the model modifies it according to weights to input to the late classifier. Moreover, the decision is only taken once a certain amount of temporal information is available (a certain number of temporal decisions). As a result, the model by itself "chooses" the correct decisions and the discriminant instant in time. Moreover, in a late fusion approach, the model is able to discard early classifiers.

Action recognition, as well as some other classification problems, are governed by space and time. The decisions can be inferred by body joints' locations in space and time. An instantaneous decision is taken by fusing all the decisions made along the time. Consequently, a weight can be attributed to each decision. The idea that has been proposed in this paragraph will be developed further in Chapter V.

C. Application to Action Recognition

a. Introduction

Action recognition became very popular and is prone to be used in different fields such as in airports, hospitals and retirement houses for fall detection [11], for security and abnormal behavior detection. Another major field where action recognition can be applied is robotics. The idea of having a robot as a house help is far-fetched and might not be available for consumers before decades. Nevertheless, the technology has been implemented in some robots for research and demonstration purposes [12, 13].

Action recognition has also hit the gaming community, especially with the appearance of cheap RGB-D sensors, available for the average consumers such as the Microsoft Kinect for Xbox [14] and the PrimeSense depth camera [15]. The Kinect accessory, attached to the gaming console, was said to overtake other gaming consoles in 2015 [16]. Other game consoles also introduced their proper accessories that handle motion control, such as the PlayStation Move [17].

Thanks to these RGB-D sensors, action recognition reached the commercial and the fashion world as well as education, in addition to other fields... [18, 19, 20].

As far as we know, this is only the start of the spread. The RGB-D sensors, which are partly responsible for this technological boom, are still at an early stage of development and are still governed by lots of constraints.

b. Defining an action

In the following part, an overview of the study of the action from an anatomical, mechanical and physical point of view of Kinesiology is presented, along with the relation to its application in computer vision in this thesis. This study is vast, nevertheless, we will only state in resume some important definitions that are related to the application in computer vision. In fact, specifying the full details requires a study that is larger than the scope of this thesis. It is important to note that, so far, there is no universal definition of an action. Even the definition that we will give cannot be generalized. We visited most of the action's definitions, analyzed, summarized and combined them in our hypothesis.

1. *Pose/Gesture/Action terminology*

Before starting, it is important to define the main terms. First of all, we differentiate between pose, action and gesture recognition. Human pose is an estimation of the body configuration (position, direction,...) from a single image at a stable position in a gesture or an action, otherwise known as the recognition in a static frame. The two terms action and gesture are often confused, and a definite description is ambiguous. Hence, we give our own: a gesture is usually composed of different poses. It is segmented from an action and is usually smaller than an action. It consists of moving different body parts in a small period of time (e.g., raising a hand, lifting…). Finally, an action is the highest level. It is composed of repetitions of same gesture, and/or sequences of different gestures (e.g., Tennis Backhand Drive, running…).

We also state multiple tasks: gesture recognition, action segmentation, action interaction recognition [21] and SL (Sign Language). Datasets related with those specific kinds of gesture will be described in the literature review (II.A.a).

2. *The skeleton*

When performing action recognition, we find a body in every frame. When tracking a person, the Microsoft Kinect represents it by a skeleton. Biologically, the skeleton is formed of axial bones and appendicular bones. They are around 80 axial bones. They constitute the center of the body (head, thorax, and trunk), as for the appendicular bones (126 bones), these are extremities, which are connected to the center. Some persons may have additional bones implanted within a tendon or muscle, called sesamoid bones, such as in the wrist, foot, and neck. In computer vision, works dealing with human posture analysis are always simplifying this complicated structure. As a matter of fact, the body is symbolized, generally, by 20 to 30 joints connected by bones (it is important to note that this simplified representation has been used for a long time now, as for example in [22]). The tracking of these joints will be described properly in section II.F when explaining the functionality of the Microsoft Kinect. Figure 1 to Figure 3 illustrate the translation that was done between complex representations of the human body to the simple one proposed by the Kinect development Kit [23].

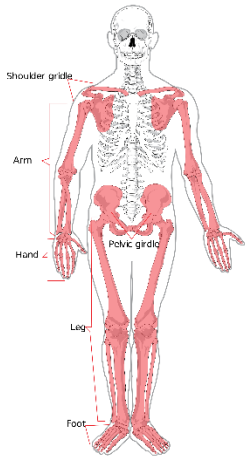


Figure 1. Appendicular Bones [24]

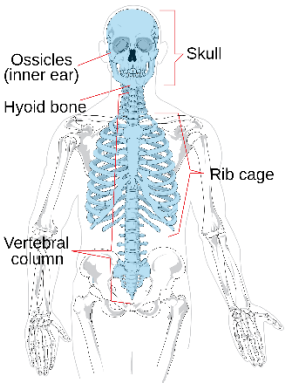


Figure 2. Axial bones [25]

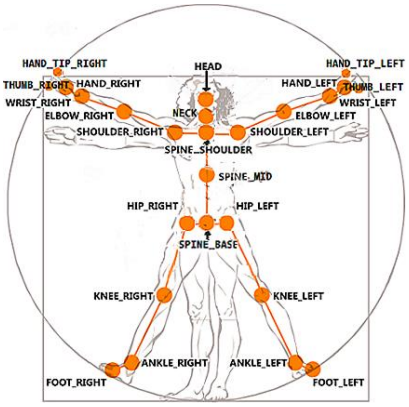


Figure 3. Microsoft Kinect Skeleton [14]

3. *Movement of the body*

The body as a whole entity moves in the world. The actions are defined by the nature of the movement. The translatory motion, in physics, is known as a linear motion, which occurs in a rather straight line going from one point to another. The whole object moves as a single entity, at the same time, in the same direction and the same distance. If the movement follows a straight line, it is also called rectilinear motion. For example, the motion of a bicycle rode along a straight road. However, if movement has a curved path, it is called curvilinear motion. For example, throwing paper airplanes, shooting a basketball. If the movement of an object occurs around a fixed point, it is called angular motion, otherwise known as rotary motion, for example, an ice-skater spinning or a spinning top.

The different types of body movements are related to the body's axes and planes. In fact, the body is divided along three planes. The sagittal plane divides the body into right and left. It passes through the body from front to back vertically. On the other hand, the frontal plane, also called the coronal plane, divides the body into front and back. It passes through the body from side to side. Finally, the transverse plane divides the body into top and bottom. It passes through the body horizontally.

4. *Center Of Gravity (COG)*

When a cardinal body plane (crossing in the sagittal, frontal, or transverse plane) crosses a part in its midline, it divides it into two equal parts. The COG is the point of intersection of the three cardinal planes. When the COG is in a position allowing the body to remain stable according to its Base of Support (BOS. e.g., contact with the floor), which is perpendicular to a vertical line of gravity (LOG), the body is called in a stable position, also known as a state of equilibrium (if no external force has been applied to the body). Any perturbation in the parameters stated before will result in a change of the physical, hence anatomical position and will lead to a movement. Consequently, any movement happening in the body is closely connected to the COG.

There are two ways to consider when representing a movement with features. Some studies analyze the changes in the COG to find its movement in space and time. Other studies perform a change of the coordinates system to match the COG, hence, becoming independent of the movement of the entire body in space. Consequently, the computation of the descriptors is only based on the movement of the joints. Moreover, it becomes easier to compare the same gesture that is performed by two persons.

5. *Joint Movement & Degrees OF Freedom (DOF)*

Osteokinematics motion involves the movement of the bones around a joint axis. Joint movements occur in different directions, around joint axes and through joint planes (sagittal, frontal, and transversal). Osteokinematics fundamental motions are the following: flexion, extension, hyperextension, abduction, adduction… The joints move through what is known as Range Of Motion (ROM), which are the maximum reach a person can move his joint.

Axes pass through the joints. The sagittal axis passes from front to back of the joint, the frontal passes from left to right and the vertical axis from top to bottom. Every joint in the body moves around an axis. The simplest example is the head joint or hip joint moving around the main body axis when the body is in a stable position.

Since the joints move around their axes and on a plane, the movement of each can be quantified by angles, called the Degrees Of Freedom. The terminology is based on the fact that the rotation always has a maximum value (Related to ROM). According to the number of DOF the joints are described as uniaxial (1 DOF), biaxial (2 DOF) and triaxial (3 DOF), which are the maximum DOF, or angles, that a joint can have.

Muscles play an important role when performing a gesture or action. They contract and extend depending on each performed action, its type, its direction, in addition to the resistance when performing it. Every muscle can either contribute to a motion or has no role whatsoever. The muscles' movement is clearly related to moving the joints through the DOF. When the

muscle does not change, the DOF does not change, and when the DOF changes, the muscles can either shorten or lengthen. Nevertheless, the muscles can contract even without motion; this change only happens at the muscles' level. For example, when holding a heavy weight in the arm without moving, the muscle is contracting, yet, there is not action. This type of movement can be detected by a slight change of the joint's position or a change in the RGB-D map.

The muscles reach a certain point where they cannot be stretched or contracted further without causing tissue damage. In some cases, they define the maximum and minimum value of the Degrees Of Freedom and the Range Of Motion of the different joints. In addition to the angles, the ROM and DOF can have three translatory values (also bounded between a minimum and maximum) caused by the stretching and the contraction of the muscles that can be found when moving a bone along a cardinal body plane. [26] [27]

The axes of the body are translated into x,y and z axes in computer vision. As seen above, moving a joint in one direction is a process that requires multiple body parts to become active, starting from the muscles to the bone itself. As a result, it is obvious that the DOF alone are not a discriminant source of information concerning a single gesture.

As an example, in [28] [29] both the DOF and the x,y,z axes are used to build a representation of the body parts' shapes, with the z-axis (considered as the vertical axis) passing through the center of the body and the x,y plane (transverse plane) perpendicular to it.

6. *Other points of view*

From a physician's perspective, the body is bound by the law of inertia (1st Newton's law), which occurs whenever the person is not performing an action but his body is moving. For example, when a plane takes off, the body is pushed backward, the neck muscles tend to extend backward (hyperextension in this case, from a biological point of view, since the neck muscles are in a stress position). The concept of the moving plane has been described previously as the translatory motion.

The 2nd law of Newton is that the amount of acceleration depends on the force applied to the object and its mass, hence the law of acceleration and the need to study the acceleration and velocity of the joints. The acceleration sometimes comes with a change of direction where a force is required to move an object in motion in another direction.

Finally, the law of reaction, "for every action, comes a reaction", explains the reason behind the "pull and push" forces in dancing moves and the contraction and stretching of the muscles.

Most of the old studies were conceived on dance choreographies. In fact, dancing or any other sport are amplified forms of everyday gestures. Every movement of the human body, bound by gravity, contracts and extends the muscles constantly. The dancer relies on this and amplifies the movements to achieve a cause-consequence relationship ("push and pull") to move from a position to another, especially while interacting with a dance partner (to understand this concept, the muscles can be considered as elastics). This is mostly observed and applied in what is called international or sports dance where the dancer, for example, extends his arms to a maximum position, therefore, extending his muscles, which causes an elastic movement and consequently the contraction of the muscles. Researchers noticed this amplification in the movement and found it easier to study what can be observed clearly. One of the precursors of the study of the

movement and the action is Laban who represented dance moves with an encoding system called Labanotation [30].

Laban represents the dancing choreographies by forms. His study is said to be one of the most complete on dance representation. This representation is composed of geometrical forms along a temporal axis and organized into different successive steps, reminding us of the works with Hidden Markov Models [31].

7. *Analysis of an example*

Each one of us has a unique way of performing an action. Actions might vary with mood changes: if cheerful, a person's walk or dance will be lighter than when being sad. A gait is the process of walking, while a gait cycle is moving a leg after the other (as well as moving the arms) and then returning to the original position. At times, a person can be recognized from a distance because of the manner he walks. Unrelatedly to the many styles that can be observed, the mechanisms of a normal gait are the same. An ankle injury or leg fracture might cause a variation in the dynamicity of the cycle and therefore, the appearance of an abnormal gait. In IV, we will be simulating actions by adding variations to the action; hence, we tend to simulate the uniqueness that has been described in the gait example.

To analyze gait in kinesiology terms, it is custom to determine first which joint motions occurs. Then, decide which muscles or muscle groups are acting (the gait is often used as an introduction to physiotherapy and kinesiology. For example, it has been described in details in [32]). Compared to a normal classification procedure in machine learning, the most discriminant joint can be first selected, then the joints' movement is described by the joint angles, then by the features at every joint.

The features are picked according to the movement. These can include the values that define an action from a physician's point of view (I.C.b.6), for example, the acceleration of the movement, the change of direction…, as well the DOF and the movement along the axes (I.C.b.5).

8. *Our definition of an action*

As a final resume, a gesture is a combination of different types of movement. It is common to have two types of body movement and joint movement happening at the same time. To illustrate this statement, we take the example of a person moving entirely in a linear motion while its individual parts moving in an angular motion. The action consists of moving the body in space, around its axes and on its planes, while moving the joints around their axes and on their planes. The joint can perform a certain movement during a period of time, then change the type of this movement, its speed, its direction… The previous statement also applies to the whole body. Of course, all the above is done within the ROM and the DOF of the joints. The whole process is a single action.

We define an action as a predefined sequence of concatenated simple gestures that cannot be clearly determined, segmented and universally defined. The same actions are composed of the same simple gestures. Every performance of an action (every recording) is unique (refer to the Gait example at I.C.b.7). Hence, the body and the joints will perform the same movements as the reference recording, with changes of dynamicity of the sequence and amplitude in the DOF.

We note that the variations in the amplitude and dynamicity must not exceed certain boundaries which could lead to entirely different actions (e.g., walking and running are composed of almost the same gestures. Nevertheless, increasing the amplitude and dynamicity of the walking action will result in running.)

Most of the actions that we study contain at least two types of motion happening simultaneously, and most of the time the angular motion is included.

c. Application of the Asynchronous Late Fusion

When working on human body analysis, a classifier can be set to learn each part of the human body singularly. For instance, it is possible to analyze the sequence of data from the head and the other joints separately. Afterward, the action can be inferred from each joint to finally fuse the decisions into a final one (for example, by choosing the most discriminant joints).

Nevertheless, a problem arises: the decision that is taken at a joint and the decision taken at a certain instant in time for a specific joint might not be discriminant (more information in V.B.a). Therefore, we aim to resolve the mentioned problem in the remaining of this thesis.

We build early classifiers to process only sub-recordings from the full recording and implement a mid-level asynchronous architecture at the output of the early classifiers. The ALF solution is applied to all types of actions as well as other temporal multi-dimensional datasets.

D. Thesis outline

A general overview of data classification starting from the audio-visual context led to the Asynchronous Late Fusion idea. In fact, most of the approaches in the domain classify sound and video flux separately with different tools. Every temporal sequence from a recording is analyzed separately, as in audiovisual stream analysis, where the classification outputs decisions at various time instants. Therefore, to infer the final decision, it is important to fuse the decisions that were taken separately, hence the idea of the asynchronous fusion. As a result, we found it interesting to implement what we call "the Asynchronous Late Fusion" in temporal sequences.

An important question arises when performing the fusion: can the decisions in the temporal sequences be merged into a final one while taking into consideration the following:

- A sequence might not be discriminant in whole.
- The decision resulting from the analysis of a sequence might not be discriminant.
- A decision taken at a time instant might not be discriminant.

To illustrate our research in the area, we are interested in a well-known field containing datasets composed of temporal sequences: action recognition. Therefore, we develop our study on the Asynchronous Late Fusion of actions.

As we implement different classification processes while aiming to improve them with the Asynchronous Late Fusion, we present a general overview of the usual classification process in the related work.

Typically, the process starts by capturing the data, then extracting discriminant features from it and encoding it. Afterward, the encoded features are sent to classifiers, while implementing different architectures. As mentioned above and throughout this thesis, we adopt a Late Fusion classification. We will go through these points in resume while mentioning the literature on the subject. Having established the basis of a classification process, we can now implement it.

Since action recognition takes an important part of this thesis, it is important to state the joint tracking systems that are beneficial to us and briefly mention different representations of an action, as well as other related work on the subject.

At the end of the related work, we differentiate the usage of the term asynchronous from its usage in other studies and compare it to other temporal classification models such as the Hidden Markov Models.

Throughout this chapter, we will focus on the parts that we find interesting or that inspire us, i.e., the concepts and ideas that lead to skeleton normalization, the techniques for generating synthetic data, some features, classification algorithms and some segmentation methods. These will be either implemented or modified to implement in this thesis.

It is imperative to gather enough discriminant data to classify actions. Since gathering enough discriminant data is not an easy task and has some drawbacks, we propose a simulation process in the first chapter to generate synthetic actions from datasets of small size. The differences (distances) between similar actions (actions belonging to the same class) help generate actions

that are not present in the initial dataset. These will help train multiple classification algorithms while modifying the simulation parameters and test several datasets.

The second chapter is the core of this thesis. After the preparation of the datasets, the ALF is explored and studied in depth. First of all, datasets with asynchronous properties are defined, (those are the ones on which the ALF solution performs the best). Second, a proof of concept is established by analyzing the resulting confidence coefficients from the classifications of samples from a recording. Third, the ALF solution is presented in details, and its parameters are set and explained. Finally, experiments are conducted with a large set of parameters to argue the choices that were made, as well as the choice of implementing the ALF.

In resume, classifying a recording with the ALF consists of performing the following steps:

- The sequences of the recording are segmented into sub-windows where every window is classified separately; hence the asynchronous study of a sequence.
- The decisions taken at the windows are combined with a set of values (weights calculated during the training phase). The last step results in a singular value for every sequence.
- These values are then sent to a final classifier, hence the late fusion.

In the final chapter, we define an index value called Asynchronous Index; it generates a value that quantifies the compatibility of the dataset with the ALF solution. The application of the ALF was limited to segmented recordings in the previous chapter; consequently, we extend the experimentations by implementing a segmentation algorithm, testing a power consumption dataset and additional classifiers. (Additional experiments are displayed in the Appendices)

The last part of this thesis consists of a contribution to the action recognition domain; there are multiple definitions and representations of an action. Nevertheless, only a few of them that are defined clearly. The ALF model provides a mean to extract the discriminant units of an action and generates a clear visual representation from the recordings.

II. RELATED WORK

The following chapter presents an overview of the necessary background literature to understand the procedure presented in this thesis, i.e. the steps to train and test a machine learning algorithm. The application of this thesis is primarily based on action recognition. Hence, action recognition is considered as the main example in this chapter to go through the background of machine learning.

Note: the parts of the related work in which we show the most interest is either labeled with a bordered tag e.g., Key Features skeleton normalization (II.J.a), or described in a resume and explained at the end of this chapter.

A. Data collection

The first step of machine learning is data collection which involves the gathering of discriminant data containing patterns or significant information. It is one of the core requirements for machine learning. As explained in the introduction, a machine learning algorithm, also called a classifier, generates a data-driven model to categorize the data, to recognize the actions in our case. The collection of data is, therefore, one of the most important parts of machine learning, as the nature of the gathered data affects the results of the study and can lead to invalid results.

Action recognition can be studied in sequences of simple RGB images, RGB video streams or RGB-D streams. Researchers have focused on all these fields, and mostly on RGB since the studies on RGB-D are more recent and that RGB-D consumer affordable cameras are still quite novel. Nevertheless, the latter seemed to have gotten a lot more popularity than its predecessor.

Concerning RGB data, the process is simple and would only require a camera (Nowadays, most of the consumers' cameras have high-definition capability). The only requirement, to build a dataset properly, is to give the adequate instructions to the subject or extract realistic datasets from movies, web videos, TV shows… [33] [34]. As for capturing depth images, the process was complicated ten years ago, as the cameras were expensive or sensors had to be placed on the subject's body and tracked throughout the action. Even though this technique can get high precision and is still used to animate 3D characters in movies [35], the advances in RGB-D cameras (Like the Microsoft Kinect [23] or PrimeSense [15] depth camera) and their distribution at a low-cost, gave them the advantage. In other studies, researchers used multiple RGB cameras to infer the RGB-D data and even perform skeleton estimation, which corresponds to the localization of the skeleton's joints in a single frame. In fact, multi-camera view for action recognition is a popular topic. The fusion of multiple cameras allowed in [36] to track the human body skeleton. Authors of additional papers were interested in the same subject. In [37] multi-decision levels are considered: a decision is taken at the camera level and at the level of a multi-camera network. [38] performed action recognition with a score-based combination of multi-view camera streams. The interesting part of the latter is that the action recognition system is performed in high frame rate. Nevertheless, the performance can be criticized in the papers stated above because of the variations in the datasets and the environment, or due to the low resolution of the cameras that have been used.

During this thesis, the Microsoft Kinect has been used for retrieving the motion capture (MoCap) data. Like any other RGB-D sensor, it outputs a sequence of frames, or images, where every pixel is associated with a 2-D RGB information and a depth vector. The depth vector represents the distance from the camera. Hence, the RGB-D sensor outputs an RGB stream and a depth stream. Nevertheless, it implements an SDK that calculates 20 body joints position (the new Kinect One (Kinect v2) calculates the position of 25 body joints, including the hand tips and thumbs [39]). The tracked joint positions are displayed in Figure 4.

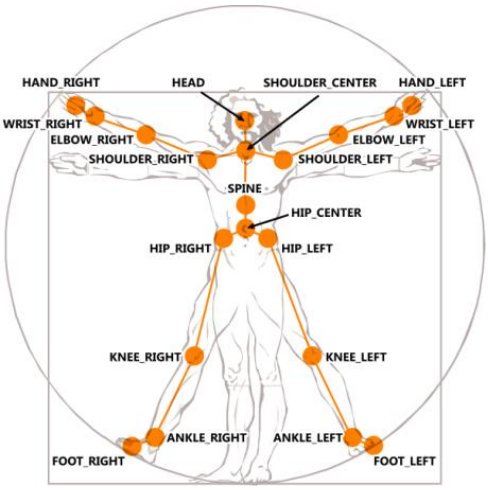


Figure 4. 20 joints, tracked by the Microsoft Kinect v1 [40]

a. Different datasets

Generally speaking, action recognition can be classified into multiple categories. The data collection, as stated above, contains video data, skeleton tracking information and/or RGB-D videos or image. The field is large; each dataset type can be related to a different category. Some of the popular categories are single-human-based actions, including human activity, gesture, and sign language. Some others reveal an interaction between multiple persons or even crowd-behavior.

We divide them into the following categories since we find that each one can be studied in a different manner than the others:

1. *Sign language dataset*

There are many resources for sign language gestures available online; the American Sign Language (ASL) is one of them. We do not take too much interest in these types of gestures because they are related to a different field and require an entirely different study. For example, [41] consists only of finding the fingers first in the images. This is related to image segmentation. We can mention a lot more studies such as [42] and [43]

2. *Human activity video datasets*

This group of datasets is also large; the datasets can even be divided into smaller groups. We state some of these: the dataset used in the popular competition Chalearn 2015 [44] where the goal was to classify cultural events in RGB videos. Others can be found on public repositories [45]. The KTH contains 600 videos. The Weizmann dataset [47] also includes examples of RGB

videos. Performances were measured on this dataset in numerous papers [48] [49] and [50] in where a study was conducted on space-time shape with the extraction of 3D point clouds. As there is a large choice today of available video datasets, we only cite here some of the popular ones: Hollywood movie dataset [51], Hollywood 2 [33], MSR [52] and larger datasets containing complex actions such as MEXAction [53] which contain 77 hours of video. Other studies have considered the complicated segmentation of RGB videos from daily living activities captured from wearable cameras [54]. Compared to most of the datasets used in research, the latter is one of the few that is considered to have imperfect data containing noise and lots of changes in lighting.

Some studies did not only rely on capturing data from one camera only. For instance, in [55], a large body of human action video called MuHAVi is available. It contains 17 action classes captured from 8 cameras.

3. *RGB-D and skeleton*

This thesis is focused on inferred skeletal data from the RGB-D information, captured from the Microsoft Kinect. There are several technologies to generate the depth coordinate. Some of these rely on the process of structured-light. The process consists of projecting an infrared random dot pattern of speckles onto a scene that is captured by an infrared camera and matches the random dot pattern with the projected pattern. Other depth cameras rely on the ToF (Time of Flight) of the infrared light from the source to the sensors to calculate the depth value [56]. Obviously, these processes generate errors. As a matter of fact, the error in the Microsoft Kinect v1, which implements the structured-light, method "increases quadratically from a few millimeters at 0.5 m distance to about 4 cm at the maximum range of the sensor" [57]**.** An extensive study will be conducted in (IV.B), on the accuracy of the Microsoft Kinect's development library when performing joint tracking. The error generated by the Kinect depends on the method implemented to infer the depth data and the hardware used in the Kinect.

Hence, we take interest in datasets that have been recorded with this RGB-D sensor specifically. Moreover, the skeletal data that is extracted from the Kinect will be studied in Chapter IV and will provide us with enough information to generate synthetic actions.

Many databases made available for public use, contain actions that have been captured by the Microsoft Kinect, like the MSRC-12 database [58]. The MSRC-12 database is composed of 12 gestures performed by 30 people. The actions are simple: start (flap both hands in the air), crouch, push an object to the right, put goggles on, gym exercises (lift both hands), shoot from a gun, bow, throw an object, surrender, change weapon, beat both hands randomly, and kick. The MSRC-12 database has been analyzed in [59] using a Hidden Markov Model and it has been stated as being the largest one that can be found online. Another available Microsoft database is the MSR Daily Activity 3D [52], which contains a large set of actions in which some are performed numerous times. In chapter IV, we test one of our algorithms on the MSR Action3D dataset [60]. The latter is larger than the other datasets and more diverse. All the databases with names starting with MSR have been recorded by the Microsoft Research team and are also available online [61].

There are other databases available like the MoCap BVH [62], which contains captured actions from 40 infrared sensors that follow white spots located on a person wearing black, hence recording flawless joint angle data. The recorded data is converted into BVH files. We also state the UMD-Telluride Kinect Dataset [63] , the G3D gaming action dataset [64] and the Cornell Activity Dataset 60 [65]; however, the actions are captured by at most four subjects.

All the citations above provide actions with simple gestures or ones that have been performed by a single person. In addition, [66] provides a source for skeletal information from human interactions.

The Kinect is not only a source for capturing actions. With its RGB-D data and the Active Appearance Model [67] that is implemented in its SDK, it also permits automatic face tracking. It becomes easier to capture and annotate faces. Hence, face tracking datasets have been distributed publicly [68]

4. *Competition datasets*

In this field, the datasets used in competitions become popular quickly since the results are set providing a base for researchers to compare their studies for a long time after the competition. Consequently, our algorithms are applied on the Chalearn 2014 dataset [21], "Track 3: Gesture Recognition». We will show that our proposed frameworks will improve any classification problem. We will develop this point in details in chapters IV and V. Note that in this same competition, during the same year (2014), an "Action/Interaction recognition" was also launched.

Other studies have considered the depth data from RGB-D streams and results have been compared in the framework of the HARL international campaign [69].

The datasets mentioned above are some of the many datasets that have been made available for public use and research purposes. They provide powerful means for researchers to study gesture classifications, localization, segmentations issues (such as simple gesture segmentation with subjects writing numbers from 0 to 9 in the air [70])… since most of them contain data stream and not just simple segmented gestures.

In the end, it is essential to state there cannot be a single complete dataset with enough information for every gesture or with enough examples from the same class. In fact, all the actions stated before are not very different since they have been performed by a maximum of 12 persons for the largest database.

During the remaining of this thesis, the gestures are segmented manually to train the classifiers. As a matter of fact, when working on classification problems in chapters IV and V, the tests are performed on pre-segmented actions.

b. Synthetic Datasets

This part (Synthetic Datasets (II.J.b)) is a source of inspiration for working on the simulated actions in Chapter IV, where the use of synthetic actions to train the Microsoft Kinect, prove that artificial samples can, in fact, improve the results.

1. *Synthetic samples*

Training with synthetic data is not an uncommon procedure especially when datasets are small. For example, to train the Microsoft Kinect, synthetic actions were generated [71], yet, there was no extensive study or analysis on the matter. The main concern when building the synthetic data was it to be the closest possible to the reality (real camera images) while adding as much variation as possible: "Other randomized parameters have been added like the MoCap frame, camera pose, camera noise" (originated from the depth), rotation and translation of the character around its vertical axis, and clothing and hairstyle (Supplementary material of [71]).

Many applications of the random forest classifier, including an old paper on regression trees [72], were evaluated with simulated waveforms. The data is available in the UCI repository and described in-depth in [73] [74]. The UCI repository contains additional synthetic datasets. For example, the pseudo-periodic dataset [75] that has been generated from a function that includes noise.

2. *Synthetic features (with SMOTE)*

SMOTE (Synthetic Minority Oversampling Technique) works on the feature level. It generates new features with an algorithm that analyses the nearest neighbors and calculates them by considering the distance between a sample and one of its nearest neighbors while multiplying this distance by a random number. The random number helps populate the new values. This algorithm has been clearly explained in [76].

B. Feature Extraction

After capturing the data, the information is converted into discriminant features, which are characteristics that present, as accurately as possible, the original samples by transforming the raw data into single values or vectors. We remind that as mentioned in II.A, the data might be collected as RGB, RGB-D or even skeletal data.

[77] compares pose-based and appearance-based features when classifying actions. The study evaluates early works on action recognition involving the tracking and classification of articulated poses, as well as, low-level appearance based features such as Histogram Of Gradients, optical flow… As a result, the paper stated that pose features outperformed the appearance features. In fact, pose based features and RGB-D solved much of the problems, found in earlier work, particularly when it comes to tracking the human skeleton and proper joint extraction. Nevertheless, pose based features lack of contextual information that is found in the low-level features. [77] suggests that a combination of both types of features is an ideal solution.

a. RGB & RGB-D features

Points of interest detection methods in RGB images and videos are very popular; they group edge detection, corner detection, as well as discriminant values…

When it comes to the temporal domain with RGB images, some of the most used features to capture shape, texture, and motion information are Histogram of Gradients. It is based on the orientation of image gradients, and has been generalized to videos with the HoG3D, as in [78] where the HoG3D are calculated on 3D gradient vectors of integral videos. In addition, we cite the Histogram of optical Flow (HoF) and the Motion Boundary Histograms (MBH) [79], which represent the gradient of the optical flow. The cuboids [80], same as the HoG/HoF and HoG3D [81], are calculated from motion history volumes of points of interest. They have been computed by stacking 2D images. The points of interest, being the corners, are detected by running a 2D Gaussian smoothing kernel and a 1D Gabor filter. A cuboid contains all the points of interest locations along the time. Pieces of information about the normalized pixel values, local gradient, and motion, are then extracted from the cuboid data to generate a discriminant vector.

Additional improvements have been introduced: dense spatial and spatiotemporal descriptors have been extracted while removing camera motion. It is estimated through matching frames using SURF descriptors and dense optical flow [34].

Improvements in previous studies [82] have been proposed and addressed the computational efficiency of the HoG and HoF (Histogram of optical Flow) by analyzing frames extracted by a subsampling process at different time scales.

Silhouette-based features have been proposed in [83]. They are binary extracted Regions of Interest from images with the application of background subtraction techniques. The features are extracted from the Region of Interest using PCA or with another method called IC features (IC features are calculated with ICA, which is described as similar to PCA, but based on local image information contrary to PCA).

Other features have also been generalized from 2D images to 3D videos by adding the temporal factor, like the 3-dimensional SIFT, the extended SURF [84] and local ternary patterns instead of the LBP (Local Binary Patterns) [85].

STV are Spatio-Temporal Volumes defined by the coordinates system denoted by X, Y (spatial) and T (temporal axes), containing image planes and time information. Consequently, the STV is a 3D shape formed by stacked 2D arrays. A first introduction of the STV was done in the earlier studies of human motion (1985), in [86], where simple motion was studied through time. Afterward, it has been developed in other studies such as [87] where the 2D contours (or silhouettes) of a subject have been stacked to form a 3D temporal volume. Descriptors, like speed, direction, and shape, have been computed from the volume afterward. The STV are very similar to the cuboids, with the difference that the cuboids are usually generated from the points of interest in 2D space and the STV from all the pixels in a 2D image.

As seen previously, 2D features have been ported to the 3D domain by adding the temporal information. In the following part, the depth factor has been added, and the "old" features have also been applied to it. The IC features were implemented on depth silhouettes in [83] with their extension to depth time-segments.

Before the appearance of low-cost RGB-D sensors, multiple studies focused on the fusion of multiple sensors, some of these papers are listed below:

[88] introduces the human motion descriptors: Motion History Volumes (MVH) ("A transplantation of motion history images onto 3D STV models"). In contrast to RGB features, it gives a free-viewpoint representation for action recognition and works in "four-dimensional patterns in space and time". Consequently, it surpasses other features, because it can bypass the occlusion of the actions' parts. We found this study interesting and were inspired by it to implement our own MVH applied on skeletal joints.

The motion history volume (Calculation of the features II.J.c) is represented by the following equation (where v is a 3D volume, called a voxel):

$$v_\tau(x,y,z,t) = \begin{cases} \tau \; if D(x,y,z,t)=1 \\ \max(0, v_\tau(x,y,z,t-1)-1) otherwise \end{cases} \qquad (1)$$

Where D(x,y,z,t)=1 if (x,y,z) is occupied at time t and D(x,y,z,t)=0 otherwise, and τ is the maximum duration along which an occupatio1n of a 3D pixel is observed.

We do not focus on the algorithm behind D(x,y,z,t) because it constitutes another study and can be calculated in many ways, as per user requirements. In [89], the calculations were estimated using silhouettes and in this work, MVH is defined as being the visual hull.

The voxel must be normalized to convert the Motion History Volume into features. It is, therefore, normalized in [88], by the maximum duration of an action, hence, all the motions will have the same length. The Motion history information will become independent from the location by centering the Voxel and from the scale by normalizing it.

The independence from the rotation is solved by the Fourier magnitudes (absolute values of the Fourier transform). The template volume is expressed in a cylindrical coordinates system. It

is then scale-normalized by shifting it in the ρ- and z- directions. Finally, the 3D FFT is applied to the normalized volume to extract the features.

In [31], features were calculated from joint angles. The joint angles were inferred from multi-camera RGB data; the depth data was processed from stereo RGB images. As for kinematic data, two DOF (Degrees Of Freedom) were attributed to every joint and 14 joints were tracked in total. 6 parameters were calculated to move from the global coordinate system to the local coordinates system of the body's hips. The 3D data is fitted onto an ellipsoid human model using a co-registration algorithm [90]. (The features are the joint angles and the 6 parameters calculated previously) The procedure is complicated. Nevertheless, it is a key principle to resolve problems related to joint coordinates normalization (Calculating the angles II.J.1 and the features II.J.c). The angles in this paragraph are an inspiration for skeleton normalization, as well as for calculating the features.

With the release of depth sensors, depth descriptors [91] [92] became very common, in addition to inferring skeleton poses from depth data to calculate the features (e.g., Skeletal Quads [93])

When training the Microsoft Kinect [71] to infer the body joints, the authors relied on depth maps. The equation for calculating the features is explained below:

At a pixel x, a depth feature is computed as:

$$f_\theta(I,x) = d_I\left(x + \frac{u}{d_I(x)}\right) - d_I\left(x + \frac{v}{d_I(x)}\right) \qquad (2)$$

"Where $d_I(x)$ is the depth at pixel x in image *I*, and the parameters $\theta = (u, v)$ are the offsets u and v."

In [60], the actions have a probabilistic representation: an ActionGraph. The ActionGraph was first introduced in [94] where an action was defined by salient postures, the transitional probabilities between the salient postures, and their combination. Every salient posture is a cluster built from frames. This method has been successfully applied to 2D silhouettes as stated in [60] where the salient postures have been described by the bag-of-points. To this end, a point or a pixel, belonging to a STIP (Spatio-Temporal Interest Points) or a silhouette, is extracted from the depth map by projecting it onto the Cartesian planes and sampling a specific number of equidistant points from the contours of the projections. The 3D points are then retrieved from the 3D depth map and considered to have Gaussian distributions. Consequently, it would be possible to represent the probability of finding a posture as the joint distribution of the points. This method outperformed those that considered only 2D silhouettes and proved to be robust to occlusions.

b. Pose-based features

In [92] an interesting feature has been extracted using Local occupancy patterns (LOP). These represent cuboids that surround the joints. Whenever the LOP covers an object, its occupancy inside the cuboid, around the joint, is computed. The pairwise distance between the joints'

positions has also been extracted and combined into a vector with the LOP. The low frequencies of the FFT have been calculated from the normalized feature vector at different scales in time.

EigenJoints are said to be more accurate in modeling body joints than modeling the depth according to [95]. Without noisy background points, it is “more compact than the bag of 3D points”. The EigenJoints are encoded normalized vectors containing posture, motion and offset features calculated from 20 already existing 3D joint position differences. The posture features are pair-wise differences of joint locations in the current frame. The motion is the differences between the current and previous frames, and the offset features are the differences between the joints’ location in the current frame and the first frame of the video. The second part of the calculation of the EigenJoints is the computation of the discriminant Eigen Vectors II.C.

The studies that were referenced previously have shown the advantages of the simple pose features (geometric relation between joints) over silhouette based, low-level appearance features (color, dense optical flow, and spatiotemporal gradients). However, in [77] the combination of both methods has been recommended as being the best solution when pose-based features fail, even if this statement was not verified by the experiments!

Finally, as previously mentioned in the introduction, every action consists of different joint translations, movement of the joints around their axis and dynamicity…, making it necessary to describe the actions with a large set of discriminant features (Angular motion => joint angles, dynamicity => the velocity and acceleration…)

C. Feature encoding

Methods of feature encoding or feature representation are numerous, and since we do not focus on the machine learning process in this thesis, we will present, in this part of the related work, a quick overview of the most known. We state PCA (Principal Component Analysis), Fisher LDA (Linear Discriminant Analysis), bag-of-words… Those are some of the many representations (Dimensionality reduction as well as extraction of discriminant values) of the transformations often applied to vectors of features before being sent to the classifiers.

PCA projects data points into a space of lower dimensionality while being able to reconstruct the data with a minimum square error. The dimensions of this space are the Eigen Vectors of the covariance matrix and have the largest possible variance. Moreover, they can be normalized. In [88], further dimensionality reduction was performed by combining the PCA with Fisher LDA. PCA was also employed in [95] to "reduce redundancy and noise" in feature vectors formed by the combination of the pair-wise difference of joint locations **Error! Reference source not found.**. The discriminant parts of the resulting Eigen Vectors from the PCA are called EigenJoints. PCA is not only restrained to action recognition with joint features but was also applied to HOG descriptors in 2D images as in: [96]. This makes the tracking algorithm robust to noise, "illumination, pose and view-point changes" in [34] [97]). From the many studies on Fisher Kernels, these have been explained clearly and improved in [98] and exploited in [99], with a GMM (Gaussian Mixture Model) to be applied to gesture recognition and segmentation.

One of the most popular feature representations would be the bag-of-features, or bag-of-visual-features as described in the paper [100]. This approach has been inspired by the traditional bag-of-words representation. Usually, applied in text information retrieval, where "feature vectors that represent each text documents are histograms of words occurrences in these documents". In the first place, a large set of visual features, or interest points, have been extracted from 2D frames in videos, usually with SURFs or SIFTs. Second, PCA helps with the dimensionality reduction and the extraction of discriminant values. Then, the vectors are quantified with k-means algorithm and the vocabulary is defined according to the clusters. Once the vocabulary is set (Every feature is associated with a word), "the occurrences of every word are counted" and combined into normalized histograms that form the BoVF (Bag of Visual Features). Finally, the action recognition is performed on the BoVF.

Common implementations of the bag-of-words or bag-of-features are [82], where the feature representation has been generated by 4 different algorithms after performing a dimensionality reduction with PCA: K-Means, hierarchical k-means, random forests and Fisher vectors. The simplified version of the first 3 algorithms will be detailed further in the next section (0).

There is a large number of alternatives to feature extraction, dimensionality reduction and classification techniques that we have stated or will state. Nevertheless, we summarize the most common ones and focus on the techniques that will be used in the remaining chapters.

D. Classifiers

Classifiers are decision makers. They attribute a label or a class to the tested set of data. We distinguish between three groups of classifiers: supervised, unsupervised and reinforcement. A supervised classifier is specified a "target" to recognize. Its task is composed of two parts: the training and the classification. The training consists of learning a model from a set of data that has been gathered during data collection (II.A) and encoding the samples according to one of the methods stated above (we remind that there are numerous solutions for data gathering as well as data encoding that we do not state in this document). After learning the model, researchers usually evaluate the performances of their algorithms by testing the classifier on another set of data.

An unsupervised classifier skips the training phase and directly classifies the data by using differences, similarities or even probabilities. Usually, the classification from unsupervised classifiers is performed by data clustering.

The third type, reinforcement learning, also known as semi-supervised learning, is much closer to the supervised one than to the unsupervised learning. It follows approximately the same process, consisting of learning and testing. Nevertheless, the approach implements user interaction whereas one can reward correct classification or "punish" an incorrect decision. Consequently, the classifier will adjust its model according to the newly provided data.

The above types of learning techniques include other categories:

- Bagging, also known as bootstrap aggregating, involves training multiple models on a random subset of the training samples and averaging the results from the output of the models, like Random Forest.
- Boosting chooses the most discriminant features from the set of training samples (Adaboost) and combines the features, as weak learners, to average the result of the output like bagging.
- Bayesian learning is a probabilistic approach where features are considered as governed by probability laws. The distributions and the decisions are computed according to statistical inference on the training data.
- Clustering is the process of grouping a set of data, according to their similarities, into an ensemble, called a cluster.

In the next paragraphs, we will give examples and an overview of these different categories. (Key in Training and Classification II.J.d)

a. K-Means

It is a simple unsupervised algorithm that is based on clustering the data and finding the centroid of every cluster. In action recognition, as in [100], it helped to build the BoVF (Bag of Visual Features) by clustering, hence, quantifying the feature space. Afterward, a word from the vocabulary is assigned to every cluster; every feature is assigned to a word and finally, a histogram is built from the occurrences of every word. The number of K words usually depends

on some properties of clusters (minimal purity, size, number) or according to previous knowledge on the dataset.

b. K-Nearest Neighbors (KNN)

It is the first thought of classifier when researching any machine learning technique or method since it is the easiest to implement. It classifies the samples by calculating a given distance between features of a sample and all the samples from the training dataset. The decision goes to the most representative class among the K nearest neighbors. K is usually chosen as in K-Means, upon prior assumptions on the dataset or empirically. In [101], spatiotemporal features were calculated from five main body joints (head, hands, and feet) trajectories as chaotic invariants "to model the non-linear dynamics of the actions", and classified with KNN algorithm and a cross-fold validation technique. In [102], the KNN has been used to classify actions from RGB videos and was compared to the classification method in [103] to show that the performances were not the best with this simple classifier.

c. Support Vector Machines (SVM) [104]

They were developed by V. Vapnik et al. who based their study on the structural minimization principle from statistical learning theory. This algorithm gained a lot of popularity in the last few years and had proven to be one of the best classification algorithms in the action recognition field. It works as follows:

Considering the common classification problem of separating 2 datasets, the samples (x_i,y_i) with $i=1,\dots,N$ and $y_i \in \{-1,+1\}$ are considered to be separable by a maximum-margin hyperplane. The hyperplane H can be a simple linear equation: $w.x + b = 0$. The goal is to minimize the classification error of the application of the previous equation on the training, by finding the optimal values of w and b. This is a quadratic programming optimization problem that can be found by solving a constrained minimization problem, using Lagrange multipliers α_i. As for nonlinear problems, the SVM uses the Kernel trick to map the data into the space H.

Finally, in the test phase, a given test point x is classified, by computing the distance between x and the support vectors (those are the values, or the training examples, from the dataset that lie closest to the hyperplane H, which separate the two classes, in other words, those are the values that should be taken into account to make a decision), more specifically, by calculating the sign of:

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i K(s_i, x) + b \qquad (3)$$

Where s_i are the support vectors, K the distance kernel.

There are numerous kernels to be studied depending on the dataset and the effectiveness of the functions after running multiple tests. [46] is a well-known paper in action recognition for its implementation of the SVM algorithm. Local features, containing space and temporal information, were extracted from RGB images histograms and are built by K-means clustering. Afterward, local features and feature histograms were used as an input of the SVM. This paper

compared the nearest neighbor classification to SVM and found that SVM with local features performs better.

The usage of the SVM is widespread. Human pose estimation method from RGB-D sequences was done in [105], where the pixels belonging to a body are normalized. Afterward, "superpixels" are extracted using a clustering method, and finally, these pixels are classified with an SVM into body parts.

An SVM is also used to determine, in [92], the most discriminative "actionlets," as called in the paper. They represent the concatenation of the most discriminative joints (the features of a joint are calculated from 3D joints and RGB-D data). The discriminant factor, which is calculated by an SVM, is the probability that a joint belongs to a certain class. (The "actionlets" described in this same paper, can be considered as another form of feature encoding.)

The SVM classifier bears numerous applications. Therefore, we have only mentioned the ones that are the most interesting to our study.

d. Random Forest

First introduced by Leo Brienman in 1996 [106] and then restudied in 2001 [107], it is a bagging algorithm, considered both supervised [71] and unsupervised [108]. It is based on the separation of the samples. Considering a training set (x_i,y_i) with i=1,…,N and $y_i \in \{-1, +1\}$, at each tree, the algorithm picks a random subset of the training set (the set might be composed of features or only sample values) and splits the data at a threshold value with the minimum error. The process is repeated at every node of the decision or regression tree until the lowest error is reached or the process is stopped manually. The main point of the random forest is to choose a different feature randomly at each node. The forest is comprised of trees. Hence, the decision on the test samples would be the sign of the summed results at every tree.

[109] uses a Hough transform based framework for multi-class action recognition in RGB videos. Low-level features are extracted from patches to train a Hough Forest, which relies on the Random Forest structure. The stated algorithm allowed the use of dense features and benefits from feature sharing between classes since the trees vote for multiple classes. It was extended to 3D joints and experimented on pose estimation in [110].

e. Adaboost [111]

During this thesis, the Adaboost classifier has been one of the main algorithms used to evaluate the performances of our propositions. In the last few years, it emerged to become a popular machine learning algorithm. Moreover, the algorithm came into practice with the Viola & Jones' study [112] and its application to face recognition, and it started to show its potential in action recognition [113].

The algorithm can be resumed as follows:

$x_i / i=1,...,n$ are the samples labeled $y_i=\{-1;1\}$. A weight w_i is attributed to each sample such as $w_i=1/n$

Considering *h* as the weak classifier, which labels the samples, the purpose of the Adaboost algorithm is to combine the most discriminant weak classifiers linearly into a final strong one.

Hence at each iterations *t / t=1,…,T*, the aim is to find the classifier that outputs the smallest error:

$$h_t = argmin \sum_{i=1}^{n} \left[-\frac{(h_t(x_i) \times y_i) - 1}{2} \times w_i \right] \qquad (4)$$

At each iteration, w_i is modified according to the classified examples.

After finding the weak classifier, the misclassified samples' weights are increased, and then all the weights are normalized

Finally, the strong classifier is built as follows:

$$H(x) = Sign \sum_{t=1}^{T} \left[\frac{1}{2} \ln \left(\frac{1-\varepsilon_t}{\varepsilon_t} \right) \times h_t(x) \right] \qquad (5)$$

Where epsilon is the error calculated at every iteration as being the sum of the weights of the misclassified samples at iteration t.

Note: there can be multiple types of classifiers *h* or a single one (usually one with a stumps decision) with a multitude of features. Consequently, the selected weak classifiers are associated with the most discriminant features. The usage of the large set of features is an interesting result that can be extracted from the strong classifier produced after running the training step. It inspired us to implement the same idea.

In the Chalearn 2014 [21], a method based on an Adaboost classification was ranked second out of 17 others [113]. It extracts a large set of features containing skeletal joint, angles, velocities and hand descriptors at all frames with multi-scale temporal windows. The skeletal features are calculated from normalized joint positions (according to the length of the torso), Euclidean distances between joints, as well as direct distances from joint angles in a quaternion representation and velocities. HoG descriptors are extracted from RGB-D frames, from the hand patch, and are normalized and scaled. This is done by positioning a square image around the hand and refining the segmentation of the hand by eliminating pixels exceeding a threshold (the pixels deviating "more than a threshold from the median depth of the image" are eliminated).

Adaboost was also used in [114] for building discriminant mid-level features and consequently, allowing the classification with a late fusion architecture, which will be described in details later on in II.E.b.

An extension of the Discrete Adaboost is the soft cascade classifier [115]. It consists of considering the cascade as a sequence of weak classifiers where a rejection threshold is attributed to every level of the sequence. We can state many other extensions to the Adaboost; yet, the Viola & Jones study [116] is the most known cascaded classification method for face recognition. It improves the computational efficiency of the algorithm by increasing the number

of features to train a classifier at every level of the cascade, hence, increasing the complexity only to check the rare positive cases.

f. Hidden Markov Models (HMM)

They are probabilistic methods for modeling sequences and temporal data. They are known for classifying temporal patterns such as speech, handwriting, gesture recognition… The sequences are modeled by observed states and other hidden states, which are unknown to the user, including dependent state probabilities. Considering the training sequences and an output sequence, the HMM aims to compute the set of transitions and probabilities that have generated the training sequence, which corresponds to the learning phase. Afterward, the probability of finding the output sequence and the sequence of the states that is most likely to have generated the output are calculated. This corresponds to the classification phase.

[31] represents HMM states by centroids of discriminant features' clusters, hence generating a CodeBook. The hard task of assigning all the performances of an action to an HMM is overcome. With the usage of the HMM, all the codewords depend on each other, but as seen in Figure 5, which was taken from this paper, the first codeword is discriminant by itself, this dependency is not useful for action classification. As there is a high dependency between the codewords and the actions, the Markov Model is not useful in this case.

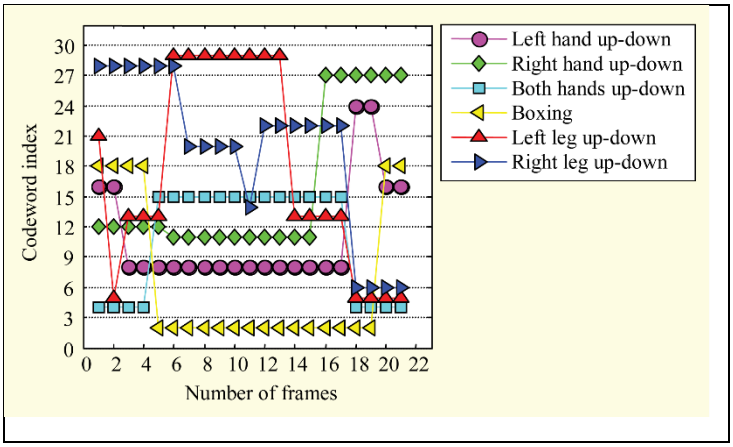


Figure 5. Pattern of codeword indices borrowed from [31]

g. Neural Networks

We state other classifiers that are also popular in the action recognition, yet we do not implement them is this thesis, like the Neural Networks (NN). Their idea has been inspired by the biological neural networks, in other words, the nervous system. This algorithm emulates the interconnection behind brains cells. A representation of a simple NN is an input layer receiving the features, an output layer, which takes the decision, and some hidden layers. Each layer consists of nodes that are interconnected by a weighing parameter. The neurons (the nodes) work in a feedforward system, where each neuron receives an input, processes it with an activation function and if the input exceeds a threshold, forwards the value to the ones it is connected to.

To train the NN, a backpropagation procedure compares the output of the perceptron to the one that was meant to be produced, to adjust the weights of the connections while going backward across the system.

h. Deep Neural Networks

The NN can be extended into Deep Learning: the Deep Neural Networks (DNN) [117]. It is a complex architecture with multiple levels of representation of abstraction.

In some way, the late fusion architecture is already included in the deep architecture. In this work, we could only consider the DNN as a synchronous solution but, due to the complexity of the architecture modeling and the amount of process required by the training step, we did not consider this kind of classifier in our benchmarks.

i. Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a category of artificial neural networks with the addition of the dense connectivity between adjacent layers that enforces the connectivity between layers, hence adding the complexity but at the same time, improving the features' discriminance [118] . The algorithms showed their efficiency in action recognition.

A recent paper proposes an extension for the CNN that is applied on action recognition, based on pose estimation in images and video sequences [119]. The method builds a large vector containing multiple features, extracted from image patches, and from the optical flow with CNN. Again, we argue in this case the dependency on the type of the dataset.

j. Extensions

All the above algorithms have many extensions, such as K-Means forest [120], where the features are divided into clusters and by propagating the tree backward, the nearest clusters are joined into a node. We also state Fuzzy KNN and Real Adaboost that outputs a real result instead of a binary result… Nevertheless, as this thesis does not focus on the classifier itself, we will not go into the details of the alternative algorithms.

k. Sequence Alignment

Dynamic Time Warping (DTW) is a known algorithm for aligning sequences. It has been applied to gesture classification. It is based on matching two sequences by finding the lowest alignment cost. This approach relies on a kind of Levenshtein distance used to fill a 2D matrix.

A C# code was posted online as Open Source for a DTW algorithm that aligns 3D coordinates from the Kinect's Joints and labels simple gestures [121]. This method is very simple, and its learning phase does not require a lot of training data, as opposed to the algorithms stated above, which require much larger action datasets.

The Multidimensional Dynamic Time Warping (MD-DTW) has been inspired from the DTW. The DTW does not suffice for aligning sequences with multi-dimensions. Hence, with

MD-DTW this is performed by calculating the features, normalizing them, applying a smoothing filter, building a 2D matrix from the sequences, then performing an alignment with the original DTW [122].

Of course, there are many ways of aligning sequences. In [70] , it has been reported that "matching an input to all gesture models […] is too slow for gesture recognition systems with gesture vocabularies". Thus, deducing that the DTW and the MD-DTW are too slow for gesture recognition. Consequently, the gestures have been broken into smaller units, and a pruning algorithm implement with a Dynamic Programming strategy has been suggested for the alignment.

E. Architectures

Before starting the following section, it is important to note that throughout the thesis, the classifiers are considered as "black boxes." Hence, there has not been an in-depth study on the classifiers. In fact, Chapter IV focuses only on improving the classifiers' performance by refining the samples' set and enriching them. Chapter V adds a decision level by considering the output of the early level classifiers as an input of the late level classifiers. Even though we stated some extensions of classifiers in the previous paragraphs, we experiment with classifiers in their simpler format, without any extensions.

The classifiers above can be combined into early or late fusion architectures. Both will be described in details below.

a. Early Fusion

The basic idea behind early fusion classification is to output a decision from a single model or classifier directly after training or testing the feature set.

Figure 6 is an example of an early fusion action classification architecture, where the mono-dimensional features are directly combined into a vector that trains the classifier.

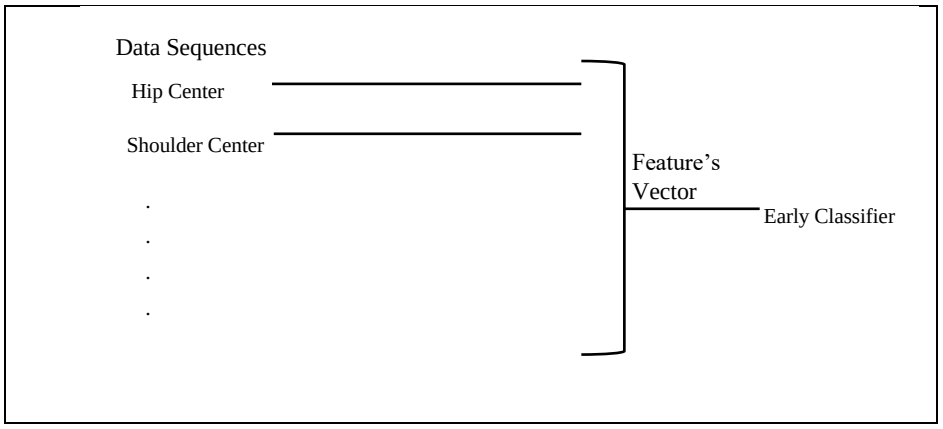


Figure 6. Early fusion architecture

The only difference between this architecture and a late fusion architecture is that in late fusion classification, a final decision is taken from the combination of multiple early classifiers. The late fusion classification will be described next.

b. Late fusion

When working with multiple data streams, at some point, the output of the classification needs to be fused. The fusion methods can be either early or late. The early fuses modalities in feature space, the late fuses modalities in semantic space. As mentioned before, no matter the type of architecture, it is required as a first step to extract unimodal features from the stream of data.

Some studies have considered hybrid fusion where the outputs of the early and late levels are fused at a final decision level [123].

A simple example of a late fusion, in one of the early studies, is its application on action recognition in RGB images [124]. The authors built mid-level motion features based on low-level optical flow features with an Adaboost classifier. Hence, the weak classifiers are the low-level features inside small cuboids in the image. Consequently, every cuboid will be described by a strong classifier. Adaboost is applied a second time on the mid-level features to find the best subset of mid-level motion features. This time, the combination of the low-level features is considered as the output of the early classifier (described as a confidence coefficient, which is a real value calculated from the weighted sum of the low-level features).

The Late Fusion architecture helped solve many problems, especially, with gesture recognition from skeleton poses, where features extracted from single joints form an input vector for a classifier. Afterward, the combination of outputs from every joint's classifier (early classifier) trains a late classifier, as in Figure 7.

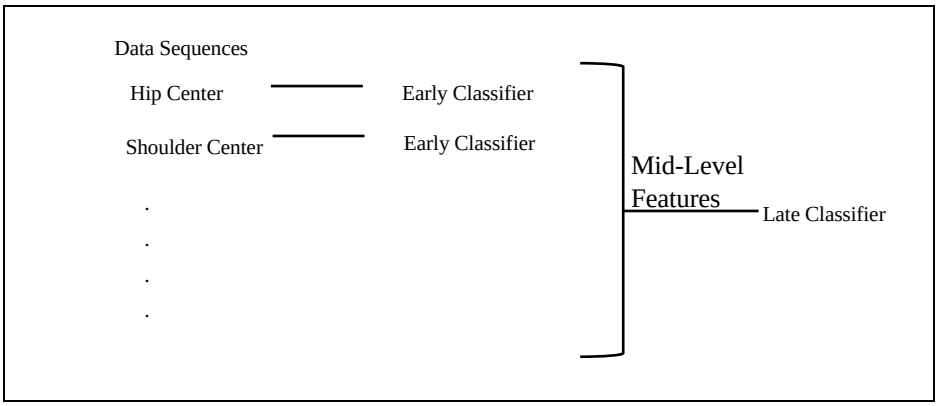


Figure 7. Late Fusion Architecture

[125] describes the feature encoding technique as a richer representation of the low-level descriptors. We state this paper to emphasize on the difference between mid-level feature representation in late fusion algorithms, in pooling, and in encoding the features' vector. In this paper, the low-level features are submitted to a set of pooling and encoding techniques with bag-of-features and other supervised algorithms and form a final dictionary that will be later used to train the classifier.

Late fusion has also been implemented in other domains not quite related to action recognition. For example, in multimedia indexing [126] where binary classifiers (any classification algorithm) are applied to the documents, the results, which are confidence coefficients, are then matched with model vectors. A rank minimization framework is proposed for image classification [127] to compute confidence scores from every model, to convert it into a comparative matrix and to infer a final relationship matrix. This method outperforms early fusion and even other late fusion methods when it comes to image classification.

F. Joint Tracking

There are some studies prior to (and after) the release of the Kinect device that tracked the human skeleton directly from the RGB-D map, implemented with different and interesting techniques.

[128] extracts extrema points (called Geodesic EXtrema) from a 3D surface mesh to form a set of points of interest. Overlapped patches of points of interest are classified with a boosting algorithm to solve the problem of detection and identification of body parts (Only 5 joints: Head, Hands, and feet) in depth images, according to the local maxima of the classifier's response. Many problems are raised in the paper such as the identification of the surface mesh, which is done by simple distance measurement, and the low number of detected joints.

The results obtained in the paper stated previously were applied and extended in [129] to perform full body recognition and reconstruction. Afterward, a lookup on a motion database with a nearest-neighbor is combined with a pose hypothesis to reconstruct the final skeleton. The limits of this approach are stated in the paper with the most important one being that a fast movement can disturb the tracking algorithm.

This thesis is based on the extraction of features from the 3D joints' positions from a depth sensor: The Microsoft Kinect. The 3D joints extraction algorithm implemented in the Microsoft Kinect is one of the few that have been made available for commercial usage and performed well in early studies, for example, when classifying dance gestures [130]. Consequently, we find it interesting to summarize the procedure of the tracking and extraction of the 3D joints' locations.

To refine the MoCap database, the process of capturing data, sampling, training the classifiers and testing joint tracking accuracy is repeated. Afterward, to populate the dataset for classification, 3D body poses are synthesized from the real training set by adding variations of camera pose, body pose, body size and shape, camera noise, clothing, and hairstyle. Depth/Scale and translation variations are handled by the chosen features (II.B.a). The body is divided into 31 parts in a texture map. More information and figures illustrating the meshes and descriptions of the variations are available in the supplementary material of [71]. Figure 8, illustrating the meshes and the addition of the variations, has been borrowed and displayed below:

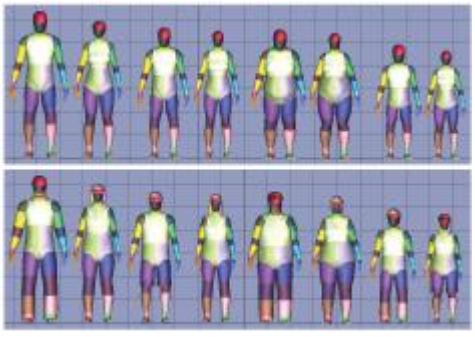


Figure 8. Illustration of the meshes. In the bottom row, the variations have been added. (Supplementary material of [71])

Individual pixel features computed from depth information are calculated as in I.B.2) and labeled using a Random Decision tree classifier on 2000 random pixels for each image.

The trees are trained as follows: at each node, a random feature is used to split the candidates, resulting in a binary decision with a left and right set. The process is repeated, and a gain is calculated by applying a Shannon entropy, representing a kind of error. The recursive step across the nodes is stopped when a depth is reached, or a sufficient gain is reached (The word sufficient is not defined clearly nor the origin of the gain in the paper). The classification is impressive as it required a large set of trained images: 1 million, on a cluster of 1000 cores for a duration of one day.

The pixel information outputted from the classification should then be "pooled across other pixels to generate reliable proposals for the positions of 3D skeletal joints". The procedure consists of finding the mode of density, based on a mean shift approach with a Gaussian kernel. Finally, the points above a probability threshold are pushed back onto the real scene.

Other studies [131] criticized the occlusion problem in joint tracking of the Microsoft Kinect [71] and aimed to improve Shotton's algorithm by combining 3D tracking and 3D pose estimation. Hence, 3D pose tracker is used to "sequentially register 3D skeletal poses". The system is built on a robustification process between the tracker and the pose detector. Moreover, the algorithm applies inverse kinematic techniques to reconstruct the pose. However, the paper in question also states that their approach needs "manual initialization and recovery from failures."

Inspired by the training of the Microsoft Kinect, [132] contributions were based on improving the regression forest to recognize 3D human poses from depth images, by inferring correspondences between depth image pixels and points in a canonical articulated 3D human mesh.

The previous algorithm improved the Kinect SDK training process by fitting a skeleton to the 3-D mesh of inferred dense correspondences. Consequently, it conserved kinematics constraints (we note that kinematic constraints are defined between two rigid bodies which result in the decrease of the degrees of freedom), for example, the maximum DOF of the Elbow. The fitting was done even without the standard iterated closest point (ICP) [133], which requires good initialization and a large number of iterations to converge.

A quick resume of other steps that were taken for training the Kinect SDK can be found in [134]. Other papers such as [135] claim to have improved the performances of the Kinect even to track dogs.

In [28], by fusing the output of multiple sensors (Microsoft Kinect cameras) the authors solve the occlusion problem, increase the tracking quality and enforce kinematic constraints. Consequently, a 3D skeleton model is defined by a set of DOF for every joint and the x,y,z axes as mentioned in the introduction of action recognition. The shape is defined by cylinders and other functions to match, as much as possible, the upper legs and the torso, which are found to be more flexible than a mesh or pure cylindrical coordinates.

The pose estimation is done with particle filtering where the set of particles are updated according to previous steps.

G. Temporal Segmentation

Three tasks are required to perform proper temporal segmentation in action recognition. The first of those tasks is spatial segmentation, which consists of tracking the joints or body parts. Then comes the temporal segmentation including the localization of gestures inside the whole action. The last task is the recognition of the gesture and its labeling. In the following paragraphs, we overview a part of the background on temporal segmentation.

a. Internal action segmentation

Many publications that are related to the automatic analysis of actions are based on the implementation of a Markovian model. They postulate implicitly that an action is a succession of lingual structure, i.e. the action follows grammatical rules that assume a possible causal relation between units of language.

However, this type of approach is never applied to a large corpus, nor actions that can be considered as sophisticated. Consequently, the assumption stated above could be revised. Moreover, dividing an action into small, simple, units is a widely debated topic that has not been concluded until this day.

Inspired by the idea that the human actions are comparable to the human language, the human body will have its phonemes, morphemes, and sentences. Different approaches have been considered: dynemes, actemes, movemes, kinetemes. All resemble phonemes in human language and have been concatenated into “words” then “sentences”.

The most basic approach, the kinetemes [136] is based on the changes of velocity and acceleration to build equivalents of the “phonemes”. By clustering these small units, they are grouped into “actiongrams”. The following interesting theory has been proposed: Subject-Verb-Object where the “the subject is the body parts (noun), the action is the motion of those parts” and the object is any third component in the whole sequence (e.g., table, person…). Hence, the smaller part of the action, the kinetemes, are the verbs.

The same idea can be found in other studies such as the dynemes [137], defined as “units of full-body movement skills.” These dynemes are constituted from 3D DOF, positions, and velocities of the human body. A probabilistic HMM model is built based on a limited set of these basic units of human motion.

The movemes [138] are also analogous to the phonemes. An interesting definition has also been declared compared to the other studies: “There should be no natural way to decompose a moveme further into sub-movemes.” Nevertheless, the general definition differs; the motion primitives correspond to a word in a language.

The approach for finding the actemes [139] has been inspired by speech recognition on the extraction of subword units [140] [141]. The main idea consists in identifying segments where the signal follows some stationary properties. Then segments are aggregated using one or several hierarchical HMMs. In the following works, those signals are derived from the Fourier coefficients of cylindrical coordinates of the body [88].

All the above definitions are vague and are based on pre-existing technologies such as the HMM, or arbitrary. Consequently, they do not provide a concrete and logic definition of an action or a gesture.

As seen above, partitioning an action into smaller gestures can be effective, and indeed, in the following paragraphs and more recent papers, gestures have been partitioned into subgestures. We find interesting the simple features that have been used in building the dynemes, kinetemes, and movemes and those inspired us in the construction of our feature set (Inspiration in calculation of the Features II.J.c).

b. Gesture Spotting

The term gesture spotting has been cited in multiple papers [142]. It is defined as the study of localizing the start and end frames of a gesture of interest, in a video stream.

In [142], a study was conducted on gesture recognition in video clips; where a person is writing, in the air, digits between 0 and 9 with his hand. A solution has been proposed for solving the problem of falsely matching small gestures with other longer ones. This was done by setting the relationship between the subgestures manually. The training is performed with a dynamic programming algorithm while comparing two features vectors with a threshold. Multiple candidates are present in the same frame. The decision is taken according to a relationship between supergestures and subgestures (Supergestures always have the advantage) and a score attributed to the decisions, according to their distance from the models.

In an updated paper [70], the above method has been improved by solving the following problems: the threshold may overfit the training data, the tracking of the hand and setting the subgestures relations were done manually. The spatial segmentation is performed by locating different hand positions, with simple skin detection and motion cues at each frame, and classifying these candidates. The models defining the gestures have been learned with a dynamic programming algorithm.

One of the many gesture spotting mechanisms has been published recently in [143]. The gesture is divided into subgestures; features are extracted from every subgesture, and are smoothed afterward. After that, a median value is calculated from a large matrix containing all the subgestures features to build a final vector. The training is done with a multi-class Random Forest.

The robustification process is interesting. The initial classifier runs on the sliding windows at different temporal scales, adds the misclassifications to the original dataset and thus, is retrained. The same procedure will be approximately used in Chapter VI, where we build an additional label instead and increase the initial dataset size (Key in Segmentation II.J.e).

Finally, the testing (segmentation) is done by a sliding window, with the same procedure that has been used during robustification. The final decision is taken by picking the highest confidence coefficient. In this paper, the confidence coefficient is defined as the percentage of trees that vote for that particular class.

H. Action Modeling

Since we address the subject of modeling classes to be recognized in this thesis, it is essential to cite elements of the literature that are relevant to our topic. There are many studies related to representing actions and temporal samples as numerical or statistical sequences, but none of them is similar to the method that will be explored in depth in Chapter V. Hence, we only mention some papers that propose a digital model of actions.

As noted above, we find the modeling of samples or actions in different fields, especially when working with an HMM. For example, in [70] a model is computed with a variant of the Baum-Welch algorithm. As stated in the thesis, there are numerous parameters to assign manually, such as the number of states. The transition probabilities are also fixed to "simplify the learning task" and because there are not "sufficient training data".

A different way of modeling an action is cited in [70] with the study on subgestures. In addition, modeling techniques that have been used are old. For example, the dynemes, movemes, and actemes are, in fact, ways to describe the different states in an action.

The partitioning has been researched in the feature level too. In [92], random partitioning of the time sequences is applied at different time scales and the Fourier transform is computed from the partitions' features. Consequently, an FFT pyramid is built. The size of the partitions was not precise. Hence it could be considered random or dichotomous, according to the figures in the paper.

I. Asynchronous Fusion

The Asynchronous Late Fusion is the subject of our thesis. This term is not new in the machine learning field and has been used previously in multiple studies. In most of the work on the subject, the Asynchronous is defined as streams of information generated from different sensors.

In [144] [145], the study has been conducted on Asynchronous events with the Asynchronous HMM (AHMM), where the streams belonging to a single sample are produced by a microphone and a camera filming a speaker. To obtain an alignment between the two sequences an EM and a Viterbi algorithm were implemented by changing the dynamicity of the sequences. Applying HMM to the problems of gesture recognition raises the problem if identifying the states in an action.

Additional studies suggested the same definition of the term Asynchronous such as [146] [147], where the work has been conducted on multiple sensors, and multiple users sending data streams [148]. According to the previously cited papers, the term Asynchronous in the phrase "Asynchronous Sensor streams" is referred to when the outputted stream from the sensor has a time-of-arrival that differs from the other sensors. Hence, the sensors are not synchronized. The researches tend to synchronize these streams or solve other problems, such as source localization by taking advantage of the asynchronous property.

The Asynchronous term used in this thesis have here a different definition, and can be described in a few lines as follows: two low-level classifiers generating two different sets of decisions in time, where both sets have the same length and are synchronized, can produce a reliable label at different instants. This statement will be extended further in Chapter V.

J. Resume & inspirations

The literature above was a source of inspiration to build our hypothesis and our algorithms. As a result, the following part describes and labels the parts of the related work that we took interest in:

a. Skeleton Normalization

A movement depends on the position of the COG in space and the length of the skeleton's bones. To operate with a normalized skeleton independently from the world coordinates and the position of the COG, the DOF are calculated from the skeleton, allowing us to normalize the coordinates and remove the dependency from the position of the human body in space (III.B.a).

b. Synthetic Datasets

It is evident that there cannot be a complete action dataset. Hence, inspired by the Microsoft Kinect and the kinesiological approach to the movement (IV), we implement an algorithm that takes into consideration the variations between different actions to generate synthetic ones for training the classifiers.

c. Features

Table 1. Description of interesting features

Feature name	**Description**
DOF and joint angles	To represent a skeleton and normalize it, hence become independent of the COG and the movement in space
Voxel	The voxel holds a large amount of spatiotemporal information.
Simple features (Acceleration, velocity…)	Adaboost permits us to use simple features since the algorithm will "pick" the most discriminant ones during training. Moreover, the kinetemes, dynemes, movemes and actemes, were a source of inspiration since the acceleration and velocity proved to be a discriminant source of information for action representation. Even more simple features were calculated as in (III.B.b).

d. Training and classification

Table 2. Description of interesting classifiers

Classifier	Description
Adaboost	Shows its purpose with a large set of features with Viola Jones. Is easy to implement. Its classifiers are easy to observe and analyze.
KNN	Easy to implement and to observe
Random Forest	Is used in Kinect training
SVM	Is a common algorithm
HMM	Processes temporal sequences and can be compared to our work on asynchronous temporal classification

e. Segmentation

As seen in different trials of representation of gestures and actions such as in the kinectemes, movemes, phonemes and dynemes, the action has been segmented into small parts to study every subgestures or every sub movement. Hence, to analyze the recording, which reveals an action, it is primordial to "cut" it into sub-recordings. Consequently, during the remaining of this thesis, and especially when working on the asynchronous module, the recordings are cut into smaller parts.

In addition, a robustification process was implemented with the purpose of including an additional label to the dataset to reduce the False Positive rate.

III. CONTEXT AND FRAMEWORK

A. Datasets

The recordings were captured directly from an RGB-D sensor (in our case, the Microsoft Kinect device) and with the aid of the device's SDK, the position of the joints are inferred (As mentioned previously in the related work (II.F)). The performances of the algorithms that were studied during this thesis are measured by training and testing multiple datasets. Most of them are captured with the previously mentioned process. The choice of the datasets is made according to different properties that will be detailed in the next chapters.

We note that classification algorithms are trained with recordings that have been segmented manually, without the application of any properties or criteria. Hence, it was a subjective decision of the user who was performing the procedure.

For our experiments, we gather five datasets. Initially, a dataset, composed of actions containing basic variations, is captured. Some of those recordings are merged with other actions to form a second dataset as in Table 4, consequently inducing more confusion than the previous one during the classification.

Table 3. Custom dataset called captured dataset (CAP)

Action	Description
2 hands up	Simple action
Crouch	Simple action that produces a large difference when performed by different subjects
Raise right hand up	Action that might be confused with 2 hands up
Right Hand Wave	Right-hand wave movement with guidelines (Move hand up, then to left once and then go back to stable position, while keeping the hand below the head)
Surrender	Confused with 2 hands up
Tennis Forehand Drive	Complex tennis gestures that are hard to recognize and might be confused with one another.
Tennis Backhand Drive	

Table 4. Custom dataset with right-hand wave confusion (CR)

Action	Description
Right-hand wave A	Right-hand wave movement without following specific guidelines
Raise Right hand up	Same action as in CAP
Right-hand wave B	Right-hand wave movement with guidelines (Move hand up, then to left once and then go back to stable position, while keeping the hand below the head)
Surrender	Adds even more confusion
Tennis Forehand Drive	Complex tennis actions that are hard to recognize and might be confused with one another.
Tennis Backhand Drive	

The recordings in Table 5 are recorded to study the temporal confusion between the recordings. This matter will be discussed further in Chapter V.

Table 5. Custom dataset with Swimming and Soccer (SS)

Action	Description
Swimming Crawl	Right-hand moves up then down, then left-hand moves up then down (creates confusion with Swimming Crawl)
Swimming Butterfly	Both hands move together (up and down)
Soccer	Subject throw the ball and then shoots
Not Soccer	Subject shoots and then throws the ball (creates confusion with soccer)

The recordings in the dataset mentioned in Table 6 is composed of 4 classes, where a subject is sitting in a chair and is performing sequences of actions: alternating between raising right hand, then lowering and raising the left hand, then lowering it. In the table, we display the hand sequences that are performed.

This dataset has interesting properties that will be explored in Chapter V.

Table 6. Custom dataset with right hand up & left hand up (RL)

Action
Right, Left, Left, Left
Left, Right, Left, Left
Left, Left, Right, Left
Left, Left, Left, Right

We have mentioned the gait previously in I.C.b.7 to focus on the uniqueness of an action. We now capture another dataset that will interest us in chapter V. The dataset is focused on different applications and performances of the gait. "It is a series of rhythmic, alternating movements, of the trunk and limbs which result in the forward progression of the center of gravity and the body." [149]

The subject performs the gait in multiple situations, to build the dataset: army march, normal gait, and abnormal gait.

Many diseases and conditions might result in an abnormal gait cycle, of which we site:

Parkinson disease, for example, is characterized by a loss of brain cells producing dopamine and lead to a weakening of motor functions. Symptoms include stiffness, tremor, impaired balance and shuffling gait [150]. Arthritis of the leg or foot joints, foot problems, fracture, infection, legs that are of different lengths… even shoe problems might induce abnormal gait movement.

Gait testing is also an important part of any neurological examination. [151]

Gait is assessed by asking the patient to walk across the room while the physician observes and note any abnormalities. Next, the patient is asked to walk heel to toe across the room, then on his toes only and finally on his heels only.

During our experiments, we chose to differentiate between a normal gait, a gait during a neurological examination, an abnormal gate (Parkinsonian shuffling), another abnormal one, characterized by stiffness of one knee similar to a fractured leg or knee, and the army march. The actions are described in Table 7. The dataset is composed of the 270 actions distributed equally among the actions.

(Additional details on the Gait can be found in Chapter 22 of [10])

Table 7. Custom gait dataset

Action	Description
Army march	The subject walks by moving his left leg and raising his right arm at the same time, then moving his right leg and left arm
Incorrect army march	The subject walks by moving his right leg and raising his right arm at the same time, then moving his left leg and left arm
Parkinsonian-like shuffling	The subject walks with the following characteristics: - Stooped with the head and neck forward - Knees flexed - Slow and small steps
Neurological Experiment	The subject walks heel to toe across the room
Normal gait	The subject walks normally
Left leg fracture	The subject walks with a stiff left knee
Right leg fracture	The subject walks with a stiff right knee

In addition to the captured dataset, a late fusion classification is performed on the actions of MSRC-12 [58], MSR Action3D [60] and MoCap BVH [62] datasets.

The MSR Action3D contains 20 action types. 10 subjects perform the same action two or three times.

The captured dataset, called *CAP* [152], contains 149 recordings, divided equally into 7 classes, the CR 130, the SS 140 and RL 130. During the rest of this thesis, additional datasets will be introduced depending on various properties.

During a Late Fusion classification, which we will adopt in chapter IV, all datasets are divided equally into three groups for classification: Early, Late Fusion, and Evaluation sets. In fact, training and testing a classifier with the same set of data will give perfect results and since the decision from the early classifiers are the training input of the late classifier, it is important to use different sets of recordings to train each level.

In Chapters V & VI, a third level will be required for the training. Hence, the datasets will be divided into another group of recordings.

We display some frames for every action in the custom datasets in Appendix IV – Datasets.

B. Feature extraction

a. Calculating the angles

As explained in I.C.b.5, the 3D joint angles (also known as the Degrees Of Freedom, DOF) are related to the muscles and joints of a person performing a movement. Moreover, they are computed to become invariant from the body shape and the position of the coordinates of a subject in space. We compute the DOF to improve the feature's stability and represent the bone orientations for the connected joints. Inspired from the BVH angles, which belong to the MoCap database, the difference between the bone's rotation and the initial stable position of the body is calculated to obtain the 3D joint angles. The initial stable position is a predefined set of bones and joints standing straight in an upright position as in Figure 9, also called: skeleton in a T position.

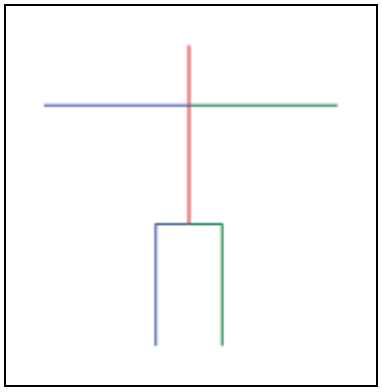


Figure 9. Stable Skeleton with the joints of the Microsoft Kinect 1.0

Preliminary work was based on the most intuitive solution that consists of calculating the Degrees Of Freedom (DOFs), from the analysis of the human kinematics. Hence, the angles that were computed for 16 joints are listed in Table 8:

Table 8. Degrees of freedom from 16 body joints

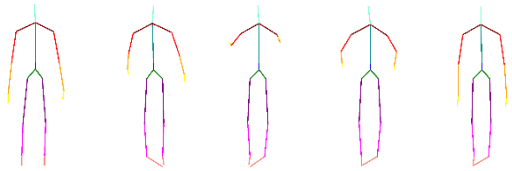
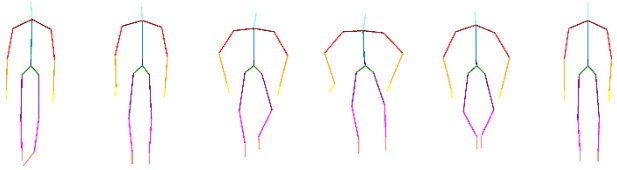
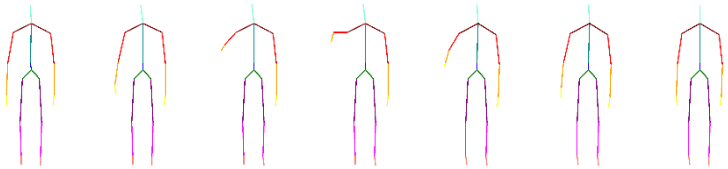
Joint name as displayed in the Microsoft Kinect SDK	**DOFs**
HipCenter	X and Y
ShoulderLeft and ShoulderRight	3
ElbowLeft and ElbowRight	3
WristLeft and WristRight	X and Y
HipLeft and HipRight	3
KneeLeft and KneeRight	1
AnkleLeft and AnkleRight	1
Spine, Head, and ShoulderCenter	3

Nevertheless, calculating only this number of angles was not enough. In fact, the purpose was to normalize the joint coordinates according to a stable skeleton using the reverse of the algorithm that was used to infer the angles. It was impossible to find the initial 3D coordinates again, having calculated only 1 DOF for some joints. We were not able to obtain a set of consistent coordinates to apply on the stable skeleton. Moreover, applying the DOF on the coordinates will result in a skeleton that is independent of the movement in space. In other words, the body will not perform a linear or curvilinear motion anymore (refer to I.C.b.3, I.C.b.4 & I.C.b.5). As a result, the solution that was proposed above was not consistent.

We also explored the conversion of the coordinates to BVH file format. Most of what we found online, did not return the expected results; when some were exporting the angles to a BVH successfully, their algorithm required the subject that was tracked to stand in an H position or required high performances or some calibration [153] [154] (Figure 10). This did not help us since we had to add a lot of preprocessing algorithms before being in a situation to use the output of this method. Therefore, we developed our software to extract those angles (Appendix IV – Datasets

Due to graphical constraints, we only display the frames that we judge as the most relevant for an action.

Table 104. Custom dataset called CAPtured dataset (CAP)

Action	Frames
2 hands up	
Crouch	
Raise right hand up	

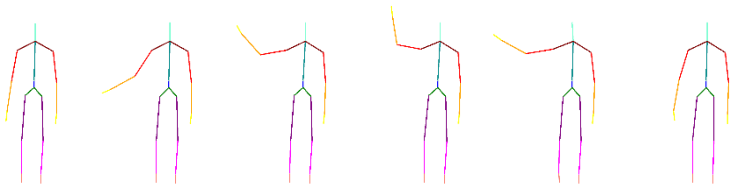
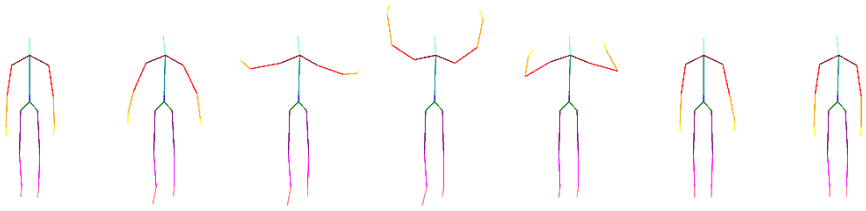
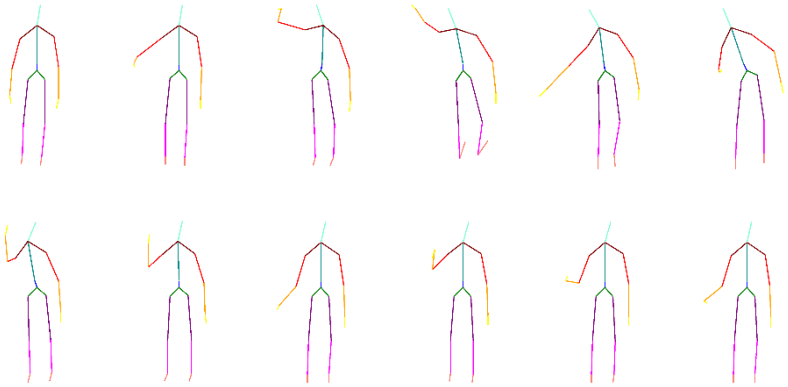
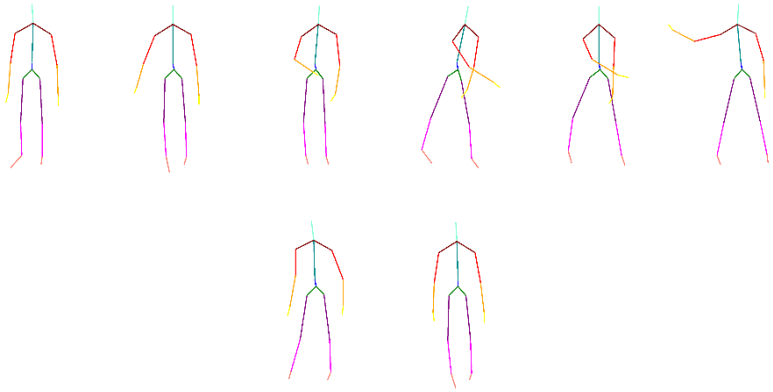
Right Hand Wave	
Surrender	
Tennis Forehand Drive	
Tennis Backhand Drive	

Table 105. Custom dataset with right-hand wave confusion (CR)

Action	Frames

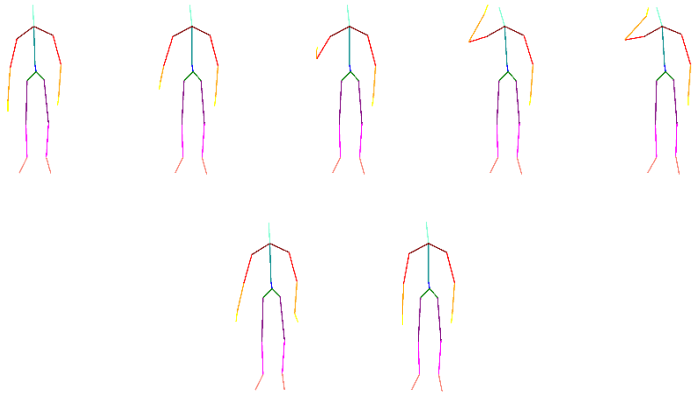
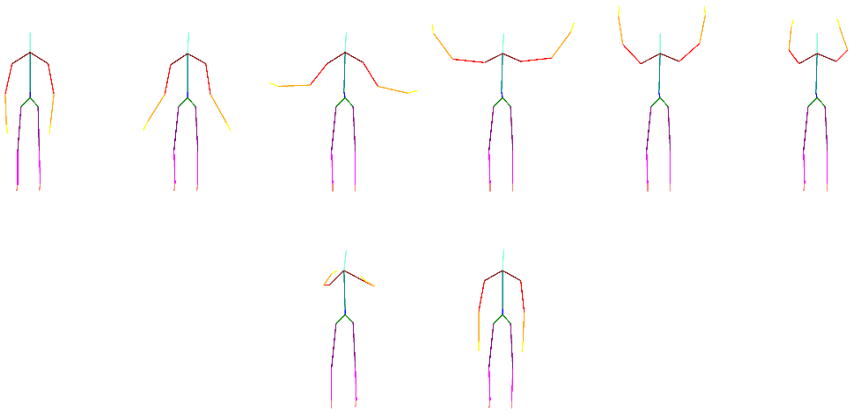
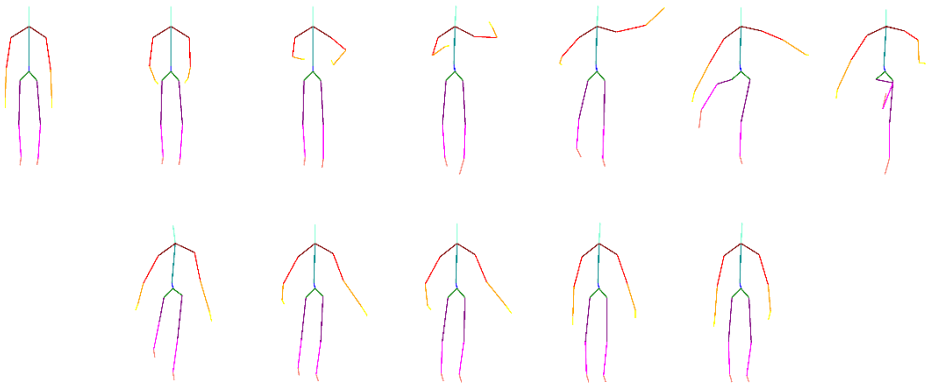
Right-hand wave A	
Raise Right hand up	Same as in Table 104
Right-hand wave B	Same as in Table 104
Surrender	Same as in Table 104
Tennis Forehand Drive	Same as in Table 104
Tennis Backhand Drive	Same as in Table 104

Table 106. Custom dataset with Swimming and Soccer (SS)

Action	Frames
Swimming Crawl	
Swimming Butterfly	
Soccer	

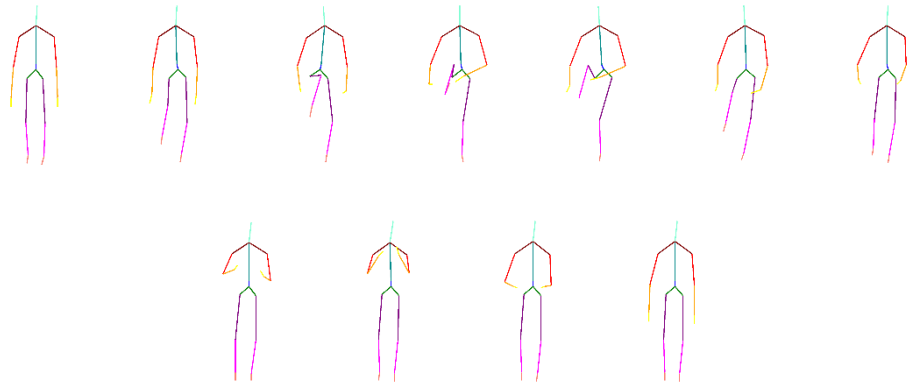
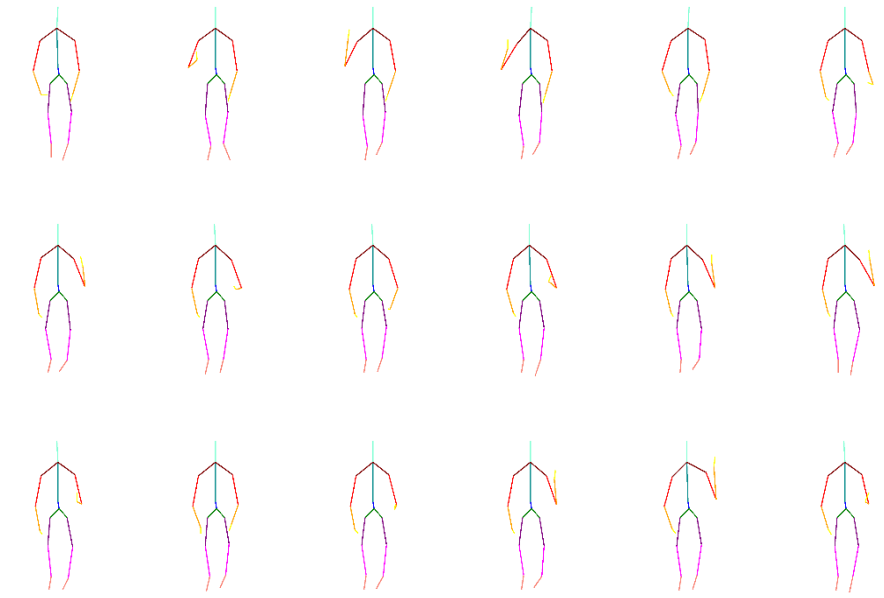
Not Soccer	

Table 107. Custom dataset with right hand up & left hand up (RL)

Action	**Frames**
Right, Left, Left, Left	

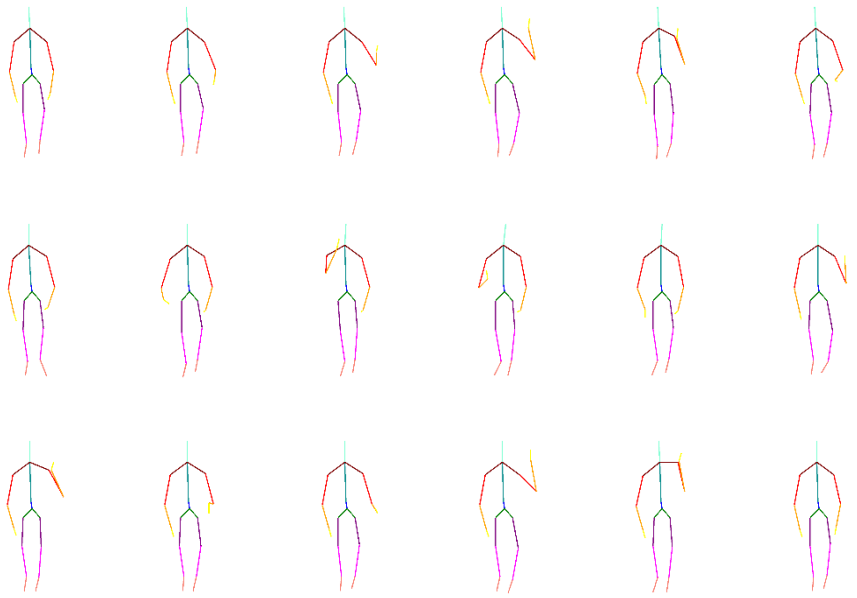
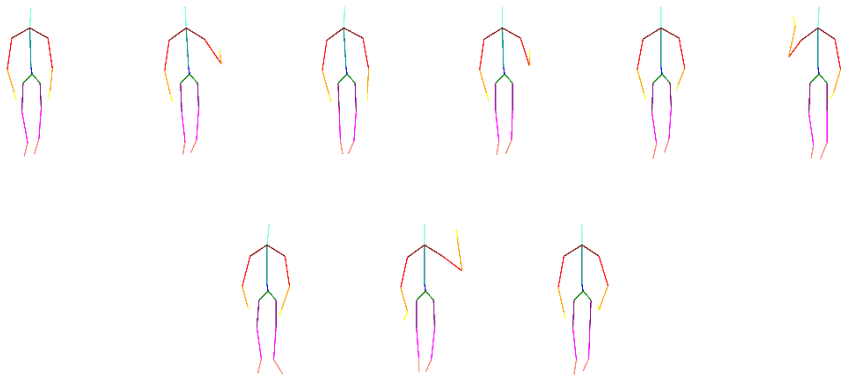
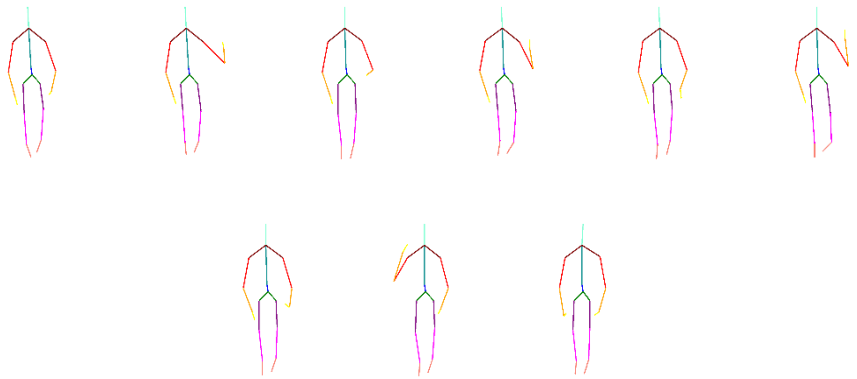
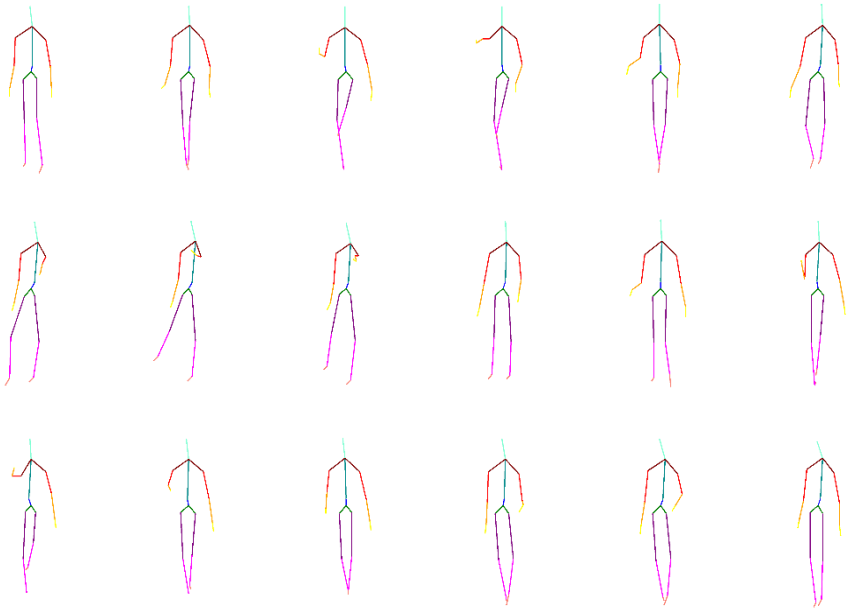
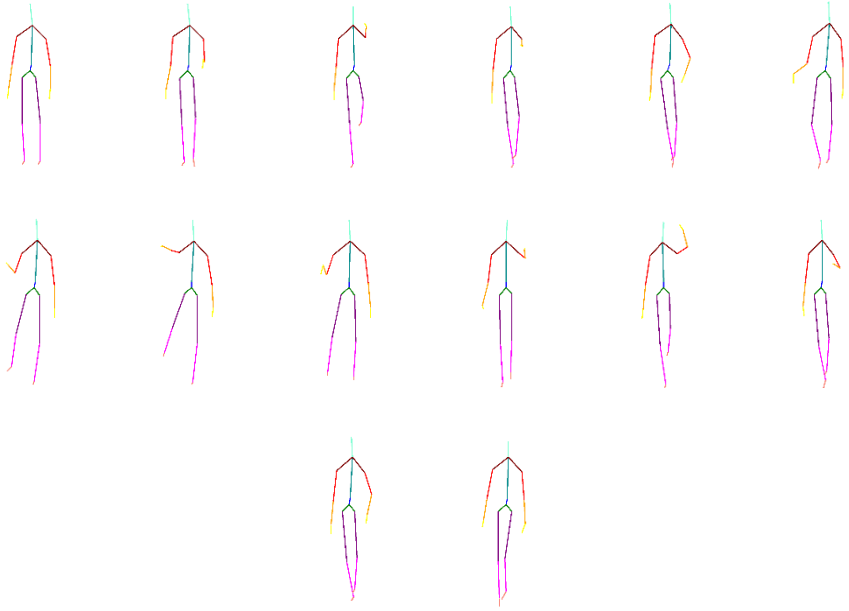
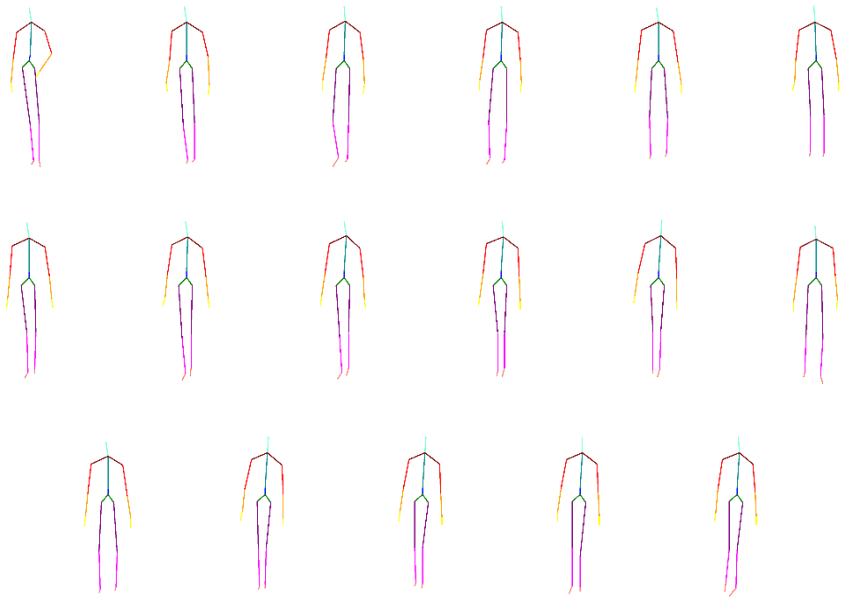
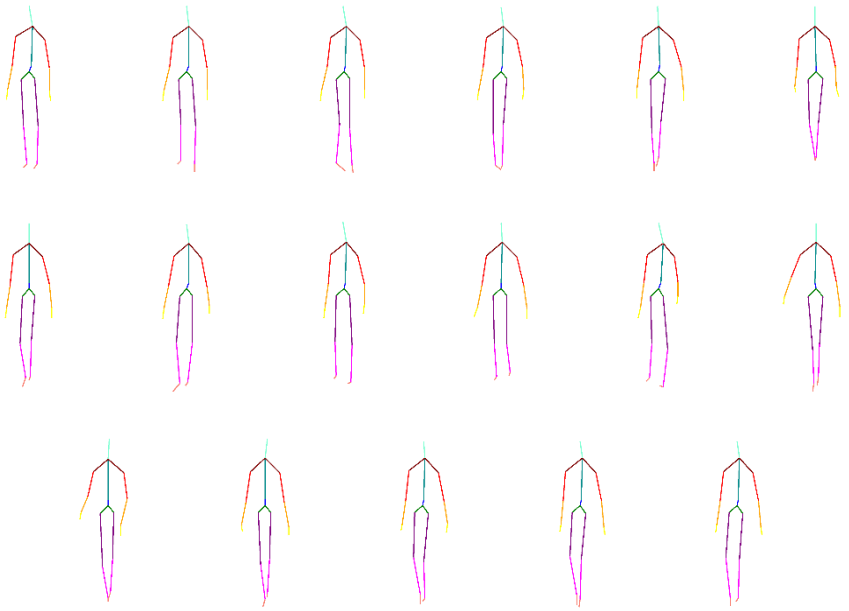
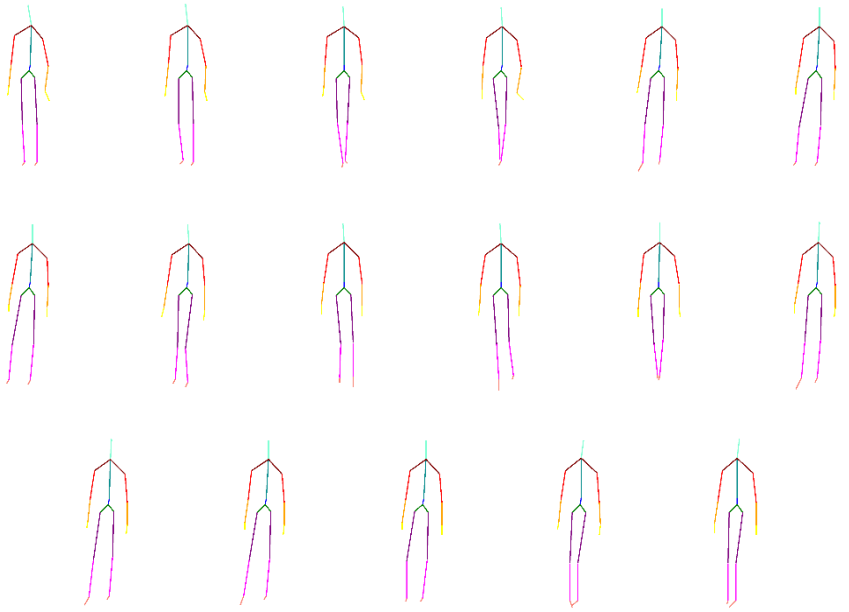
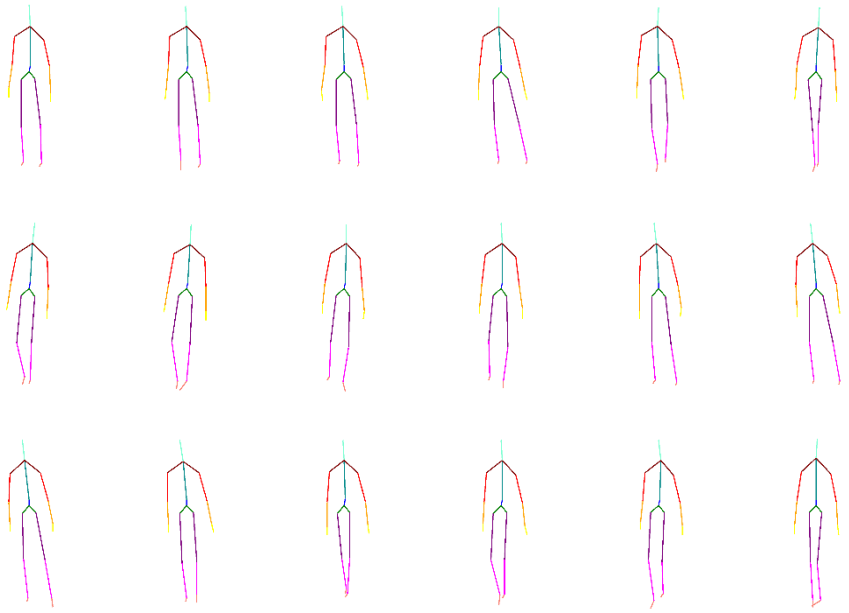
Left, Right, Left, Left	
Left, Left, Right, Left	
Left, Left, Left, Right	

Table 108. Custom gait dataset

Action	Frames
Army march	
Incorrect army march	

Parkinsonian-like shuffling	
Neurological Experiment	

Normal gait	
Right leg fracture	

Left leg fracture	

Appendix V - Open Source Contributions).

Figure 10. iClone 6 MoCap Plugin calibration in H pose.

As explained above, the solution was not consistent. Therefore, a simple solution was to calculate 2 angles (2 DOF for every joint) in 3D space using rotation matrices. It is displayed below:

According to each orientation of the bone in T position,
Calculate next vector
For each joint i in (x, y, z):
    $V_{ni} = |(next\ joint)_i - (current\ joint)_i|$
    Calculate previous vector, at T position (Unit Vector)
        $V_p = (0, |V_n|, 0)$
    $\theta_z = SignedAngle(V_{p(x,y)}, V_{n(x,y)})$
    Calculate rotation Matrix R_z
    $V'_n = V_n * R_z$
    $\theta_x = SignedAngle(V_{p(y,z)}, V_{n(y,z)})$
    The final result is:

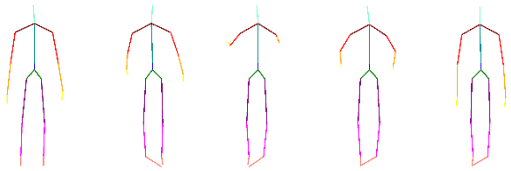
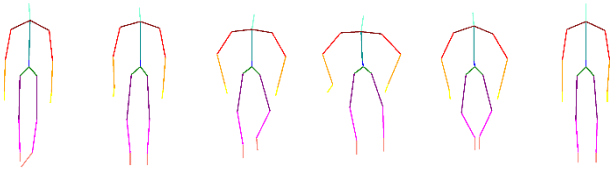
$$\vec{\theta} = (\theta_x, 1, \theta_z)$$

Algorithm 1. Calculating the DOF

Both solutions stated above (the DOF from the human Kinematics and 2 angles only) are published online as an Open Source software. (Refer to Appendix IV – Datasets

Due to graphical constraints, we only display the frames that we judge as the most relevant for an action.

Table 104. Custom dataset called CAPtured dataset (CAP)

Action	Frames
2 hands up	
Crouch	

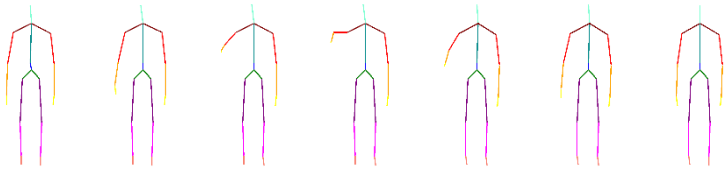
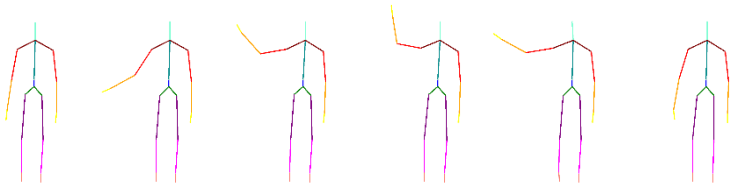
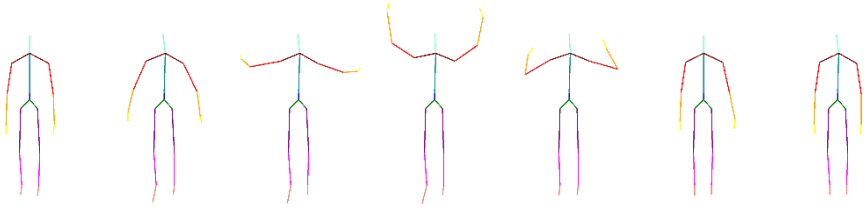
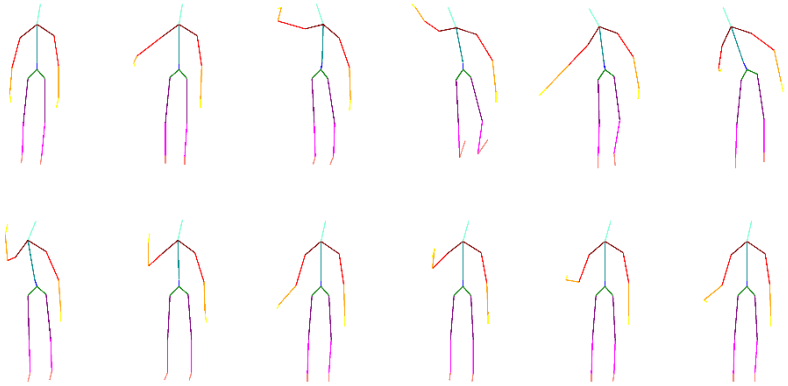
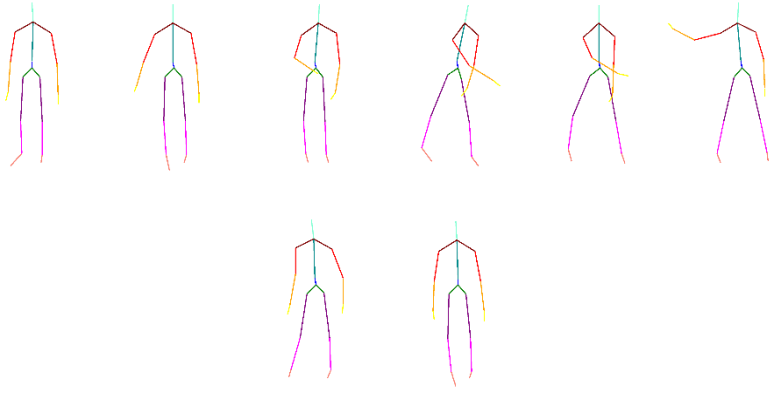
Raise right hand up	
Right Hand Wave	
Surrender	
Tennis Forehand Drive	
Tennis Backhand Drive	

Table 105. Custom dataset with right-hand wave confusion (CR)

Action	Frames

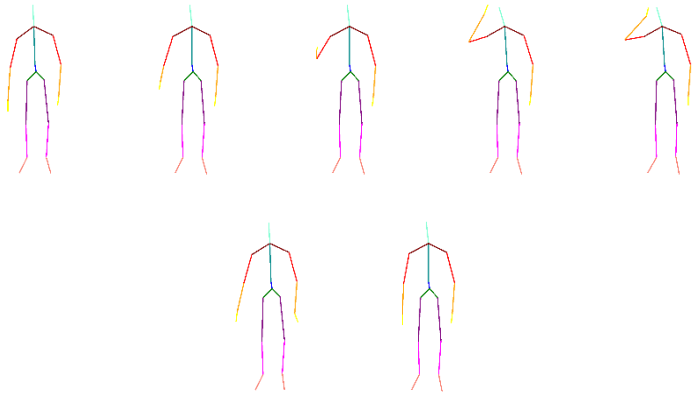
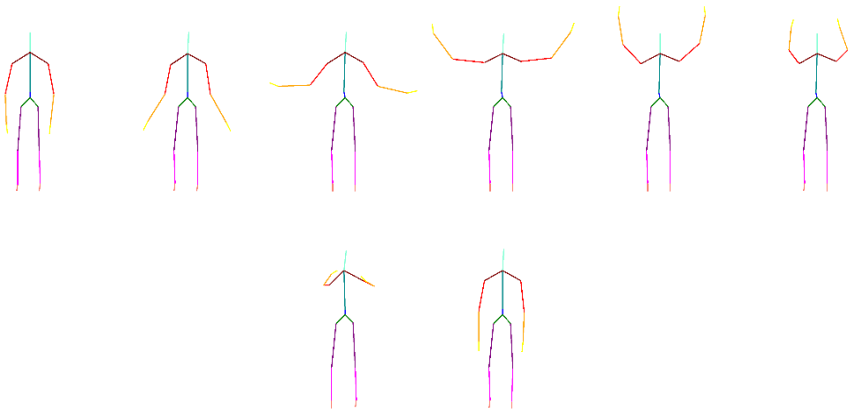
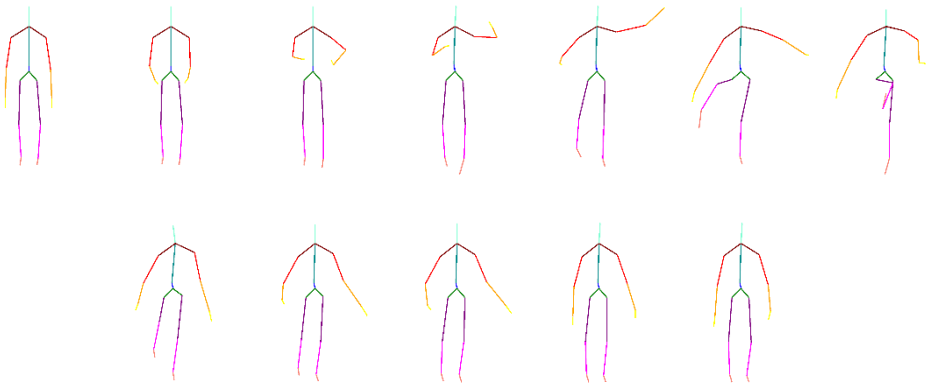
Right-hand wave A	
Raise Right hand up	Same as in Table 104
Right-hand wave B	Same as in Table 104
Surrender	Same as in Table 104
Tennis Forehand Drive	Same as in Table 104
Tennis Backhand Drive	Same as in Table 104

Table 106. Custom dataset with Swimming and Soccer (SS)

Action	Frames
Swimming Crawl	
Swimming Butterfly	
Soccer	

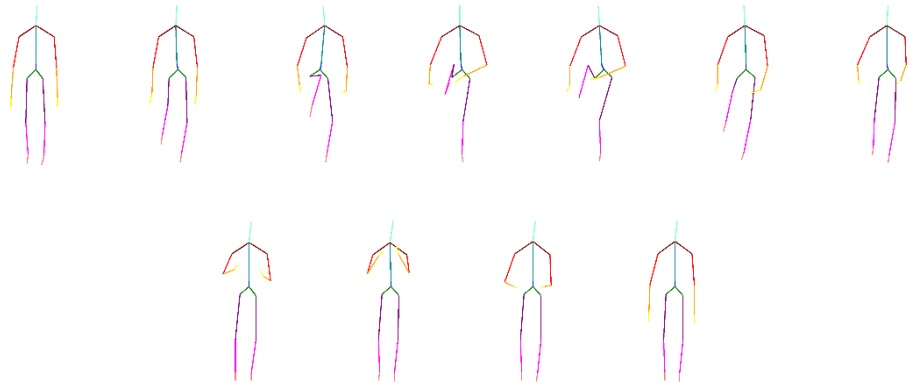
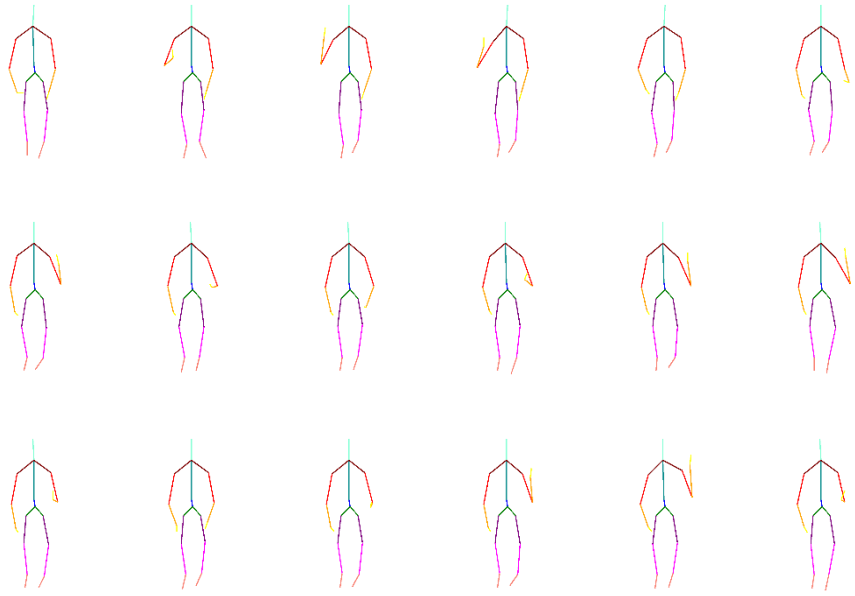
Not Soccer	

Table 107. Custom dataset with right hand up & left hand up (RL)

Action	**Frames**
Right, Left, Left, Left	

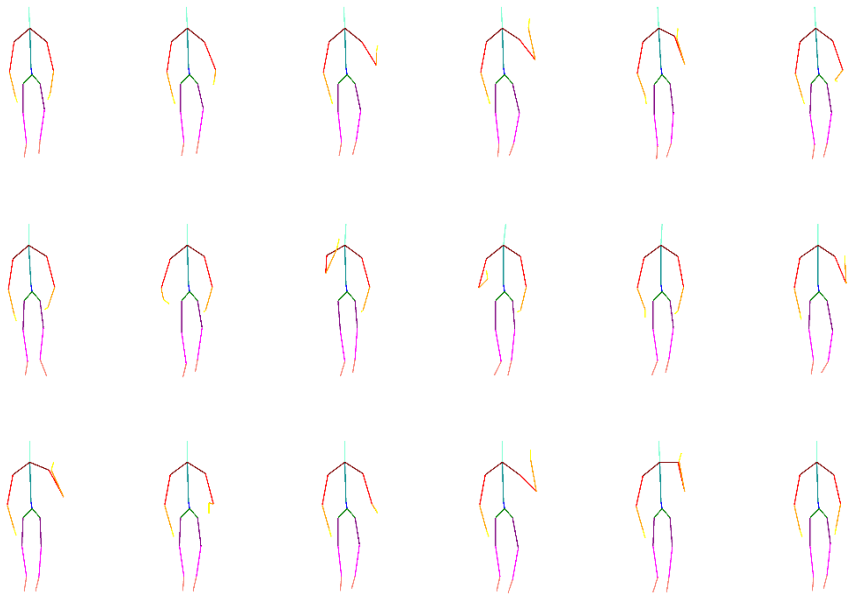
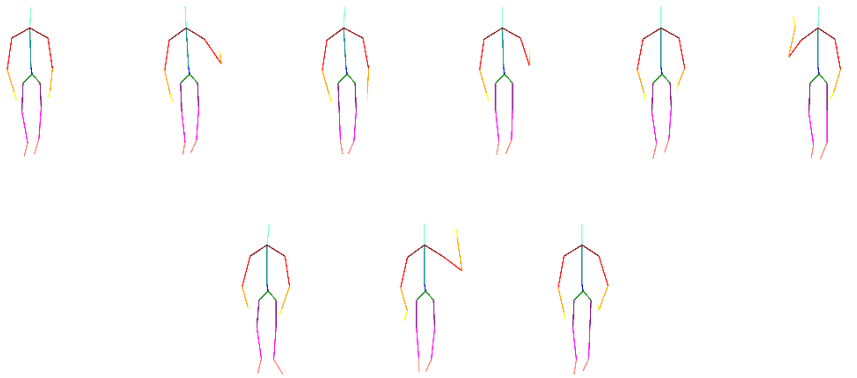
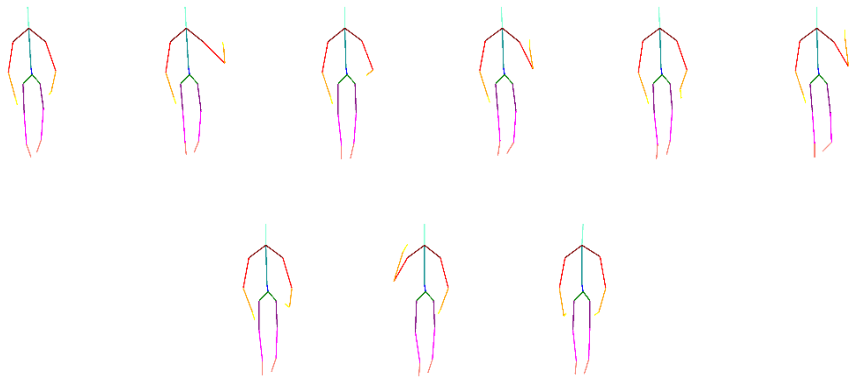
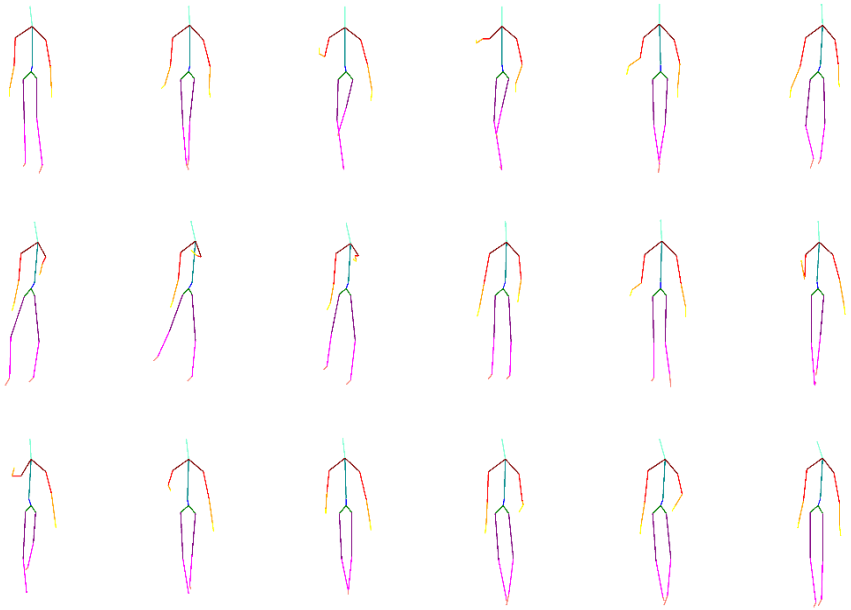
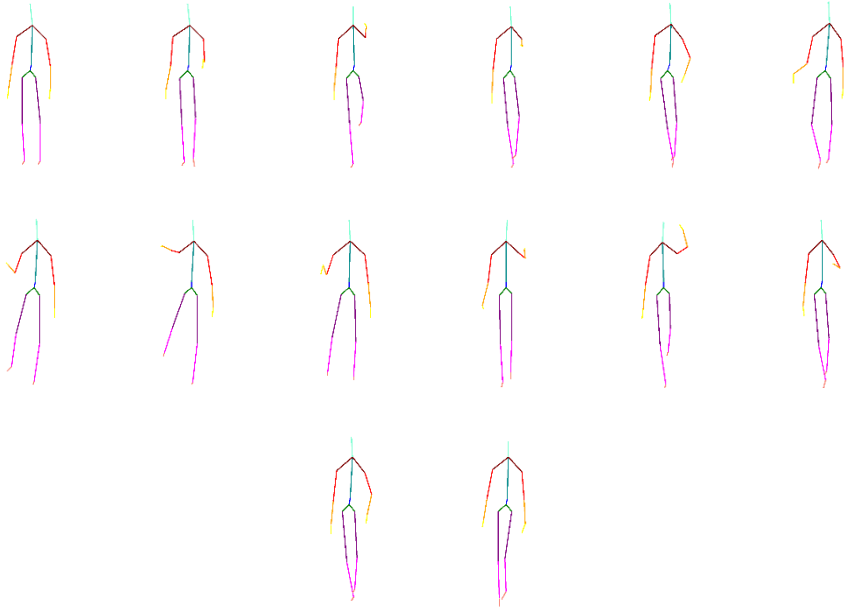
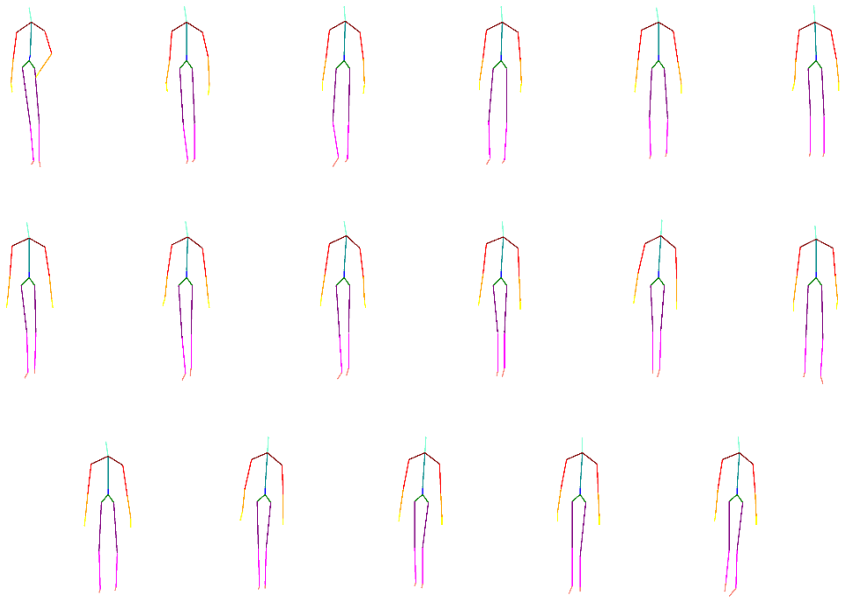
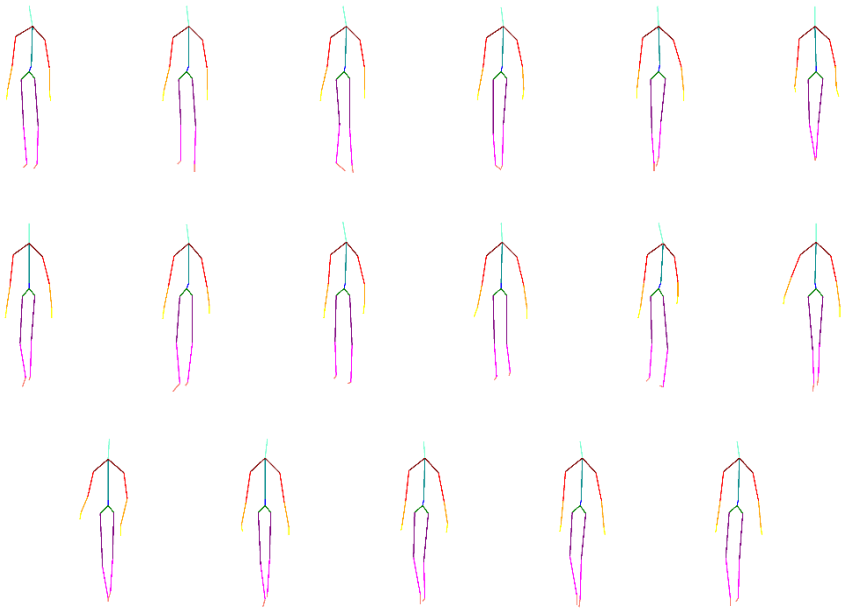
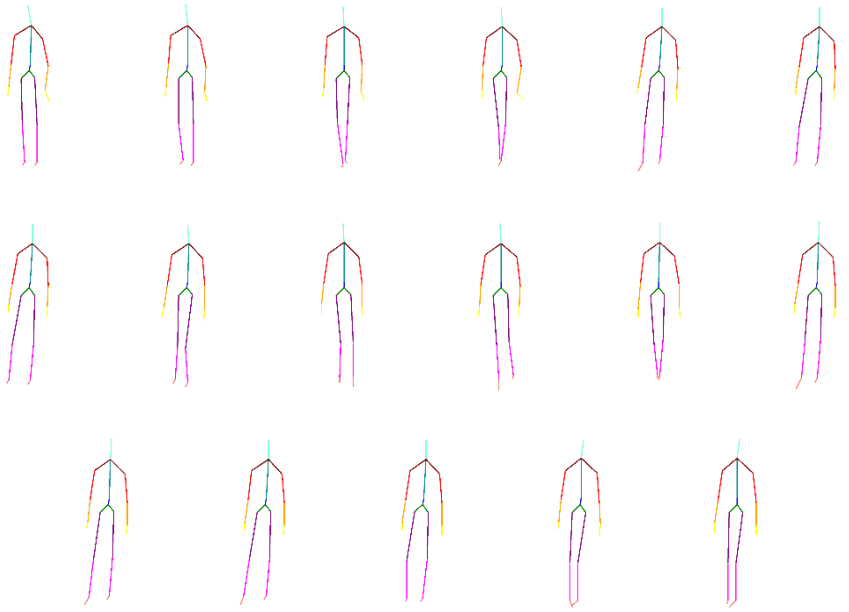
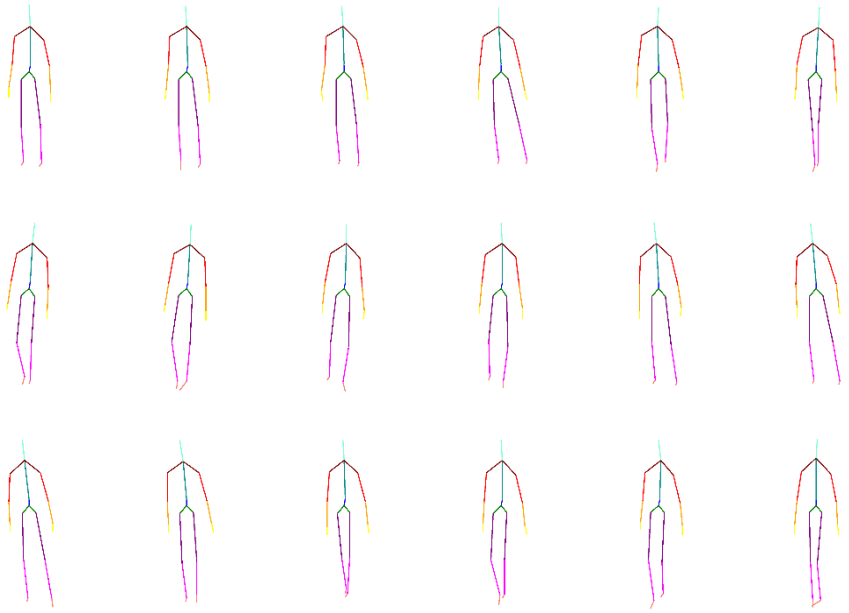
Left, Right, Left, Left	
Left, Left, Right, Left	
Left, Left, Left, Right	

Table 108. Custom gait dataset

Action	Frames
Army march	
Incorrect army march	

Parkinsonian-like shuffling	
Neurological Experiment	

Normal gait	
Right leg fracture	

Left leg fracture	

Appendix V - Open Source Contributions for more information)

b. Calculating the features

Our main classification algorithm that we will rely on to measure the performances in this thesis is a boosting algorithm (Adaboost). Hence, it is possible to generate a large number of features and leave the choice of the best features to the algorithm.

Consequently, Table 9 displays the features that are extracted from both angles and coordinates.

Table 9. Features used as input to the early classifier

Features	**Variations and comments**
Velocity	Mean max min
Acceleration	Mean max min
Signed Velocity	Mean max min
Signed Acceleration	Mean max min
Position of the Joints	Mean max min
Local maxima of the joints' position	Min mean
Local minima of the joints' position	Max mean
Extrema of the joint's position	Deviation, Standard Deviation
Voxel or Motion Volume History [88]	Time sequence of the X,Y,Z coordinates combined into a 3D Voxel model of our action

The Voxel or Motion Volume History (MVH) feature [88] is commonly used in the action recognition field and proved to give good performances. An MVH was extracted from the recording, then was projected into the 3 planes to compute the size of the covered area on the (X,Y), (Y,Z) and (X,Z) planes, as well as the total size of the MVH.

The features cited above are simple, "directly" calculated features. Derivative features, such as the Haar features in Face detection with Adaboost, improved the performances considerably. Hence, to compare simple and "derivative" features, another feature set is extracted that includes the features in Table 9, in addition to pairwise joint features. These are computed by considering a simple Euclidian distance between each joint and the others, and extracting the mean, maximum and minimum from the coordinates in time.

C. Discussion – confidence coefficient

In the following chapter, we discuss the different methods that were tried in this thesis for computing a confidence coefficient as an output of a classifier. We present our motivation behind the removal of the confidence coefficient and the adaptation of a binary result, for some uses of the classification algorithms, during the remainder of this thesis.

We aim to implement the following properties when computing the confidence coefficient. These will guide us throughout this thesis when calculating it:

- A confidence coefficient is usually a trusted real value that accompanies the binary or natural output of a classifier.
- It should be normalizable to a [0;1] interval, where 0 denotes that the result is not trusted and 1 that it is. If the output of the classifier is binary and is accompanied by a cc, the resulting product of the binary output and the cc is a value between -1 & 1. -1 shows that the result is trusted and is negative, 1 that it is positive and is trusted and 0 that it is not trusted.

By taking into consideration an example where the Adaboost is the main classifier, similar to [155], the confidence coefficient (cc) is calculated as follows: the distance is calculated between the feature values and the threshold that separates the positive and negative feature values during the classification of an action with an Adaboost algorithm.

1 We obtain the cc as follows:
2 For each week classifier (w) to W
3 Let $d_w = |featureValue - threshold|$ (threshold is the variable of a weak Adaboost classifier and featureValue is the feature value of a recording)
4 $\overline{d_w}$ is the average distance calculated between the threshold and the features in the dataset
5 The $UpperBound(d_w)$ is the maximum value of d_w calculated on all training recordings.
6 The $LowerBound(d_w)$ is the minimum value of d_w calculated on all training recordings.
7 Let d'_w be the distance of the feature value of the recording that is currently being tested
8 If $\overline{d_w} < d'_w \leq UpperBound(d_w)$
9 Then $cc_w = (UpperBound(d_w) - d'_w)/(UpperBound(d_w) - \overline{d_w})$
10 Else If $\overline{d_w} > d'_w \geq LowerBound(d_w)$
11 Then $cc_w = (d'_w - LowerBound(d_w))/(\overline{d_w} - LowerBound(d_w))$
12 Else $cc_w = 1$
13 End for each loop
14 $Final\ CC = \frac{\sum_w cc}{number\ of\ weak\ classifiers}$

Algorithm 2. Calculating the custom confidence coefficient for the Adaboost

In a Late Fusion architecture (II.E.b), the confidence coefficient multiplied by the binary result from the early classifier test can be considered as a Mid-level feature and is used as an input of the late classifier.

The customized cc that was calculated above was rejected in the rest of the study since we argue whether the calculated value cannot be truly trusted. In fact, the computation of the cc is missing information: calculating the confidence in the weak classifiers that includes the confidence in the choice of the features and the confidence in the cc itself. The confidence in the weak classifiers should contain information about the error that is generated by the weak classifier, and the confidence in the cc can reveal details about the performance of the cc when adapted during an evaluation phase.

We state below some studies that considered different types of confidence coefficients:

In [92], the ambiguity was added as a value that accompanied the cc. Nevertheless, the reason behind the addition of the ambiguity and the equations have not been described in details.

In [124] & [156], the output of the strong Adaboost classifier is considered as a confidence coefficient of the classifier. These studies adopted the most intuitive definition of a confidence coefficient; they stated that the result is better the closer the cc is to 1 and worse if the cc is close or equal to 0, "thus, if h(x) is close to or far from zero, it is interpreted as a low or high confidence prediction. Although the range of h may include all real numbers, we will sometimes restrict this range". This is similar to the idea of the cc that was described at the beginning of this section. Even though this is the most obvious approach, we do not consider it, since the cc cannot be trusted because it does not include in its calculation the confidence in the decision of the weak classifiers.

For the sake of diversity, we mention other fields where the confidence coefficient was studied, such as multimedia indexing [126]. The value of the cc was bounded between 0 & 1, and it was defined as a measure of "the degree of certainty of detection" of a concept. In this same research, 3 methods are proposed for computing the confidence score. One considered the distance to the decision boundary for each detector, which is closely similar to our own proposition of the confidence coefficient. Another indicated the relevance of a concept, or label, according to the multimedia document. Finally, one measured the reliability of the detector, which is calculated according to the number of samples used for training or according to a validation dataset. The method comes in parallel with a part of our vision of the confidence coefficient since the reliability can measure the "confidence in the confidence coefficient", the relevance and the proximity (or distance) from the classifier's threshold.

Nevertheless, we criticize the way the cc was normalized. In fact, it was divided by the maximum value inside the range of the ccs, and it was assumed that the ccs have a Gaussian distribution. Hence some information from the classifier might be lost.

In one of the many gesture spotting mechanism published recently [143], the testing was done by sliding windows. The final decision was taken by picking the highest confidence coefficient

from the output of the random forest classifier. In this paper, the confidence coefficient is defined as the percentage of trees that vote for that particular class.

Sometimes, the confidence coefficient of a classifier is obvious. For example, the cc of a KNN is the number of nearest neighbors labeled with the binary output label.

A large part of the classification algorithm does not have a confidence coefficient, and there is a large controversy about this issue. In fact, some researchers consider that, for example, AdaBoost does not output a confidence coefficient, hence, a custom cc is calculated, while others consider the output of the Adaboost as in a trusted cc. As stated previously, the output of the Adaboost is a real value, from which the binary decision is extracted. This algorithm does not take into consideration the confidence of the weak classifiers when taking a decision. Moreover, the real output cannot be normalized. Thus, the following question is asked: when can the confidence coefficient be trusted?

During the rest of the thesis, we consider the classifiers as “black boxes.” Consequently, we do not have any information about the adopted classification algorithm. The only information used is the binary output and the real output where the cc is obvious, as in KNN.

IV. SIMULATION

A. Introduction

The implementation of the learning process with all of the classification algorithms stated in 0 requires a large and discriminative action dataset containing a variety of data for the training phase. Therefore, the same action should be performed by multiple persons in different ways. Yet, the databases available online do not contain enough recordings for each action. In addition, it is hard to give people instructions and ask them to perform a gesture they are not familiar with. For example, it is necessary to train the classifier with a perfect dataset when trying to find a certain action or abnormal ones, nevertheless, not every person performs a tennis forehand drive correctly (tennis forehand drive is part of the CAP dataset). Therefore, it seems appropriate to remedy this problem by proposing a method to simulate recordings.

In the rest of this chapter, we present an algorithm for generating simulated recordings from a limited number of initially captured ones. These simulated recordings are expected to be relevant to train late fusion algorithms with Mid-Level features, extracted from the 20 joints of the human body, such as Adaboost, SVM, KNN and Random Forest.

We capture the data from an RGB-D sensor (as in III.A), calculate the joint angles, simulate actions, train the classifiers with the simulated actions and finally test the performances on real recordings with the late fusion algorithm previously described in II.E.b.

The purpose of this chapter is to analyze the added value of the simulation method on the overall results with the most basic implementation of the classification methods (without any update or extension to the algorithms). Consequently, the classification algorithms are trained with actions that have been pre-segmented, manually.

In resume, an action simulation algorithm is presented to reduce the dependency on public databases and to allow training with small sets of actions. The motivation in the work is to show the utility of the simulation algorithm by proving that enlarging the datasets with synthetic recordings, can improve the classification results independently of the used algorithm. The simulated recordings are learned and tested with a Late Fusion Discrete Adaboost, a KNN, a Random Forest and an SVM, using the simple feature set stated in (III.B.b).

B. Capture and analysis

The infrared-based camera uses RGB-D data to recognize the positions of the following 20 joints (refer to II.A for more information): Hip Center, Spine, Shoulder Center, Head, Shoulder Left, Elbow Left, Hand Left, Wrist Left, Shoulder Right, Elbow Right, Hand Right, Wrist Right, Hip Left, Knee Left, Ankle Left, Foot Left, Hip Right, Knee Right, Ankle Right and Foot Right. All this data is collected using the Microsoft Kinect SDK, which is an API available online for public usage [23].

By capturing recordings of an action from the RGB-D sensor, we simulate and add different variables to the initially captured actions. During the simulation process, it is impossible to generate data that are exactly similar to real recordings. Nevertheless, we aim to minimize the differences as much as possible by studying the errors. To achieve this, we first test the Kinect to find if the captured data generates a quantifiable error when tracking the joints, to determine how it should be taken into account in the simulation model.

N.B.: the experiments have been conducted with the Kinect for Windows and Kinect for Xbox, but not with newer versions of the device: Kinect for Xbox One. [23]

We find two types of error:

a. Tracking algorithm error

The first error is generated by the tracking algorithm implemented in the Kinect. We study and analyze it carefully with the technique described below:

We capture 3D joint coordinates from a mannequin in a stable position, for approximately 10 minutes from which, 100 frames are displayed in Figure 11 and Figure 12. The mannequin is standing still, 2 meters away from the RGB-D sensor in an environment with steady lighting for the whole duration of the recording.

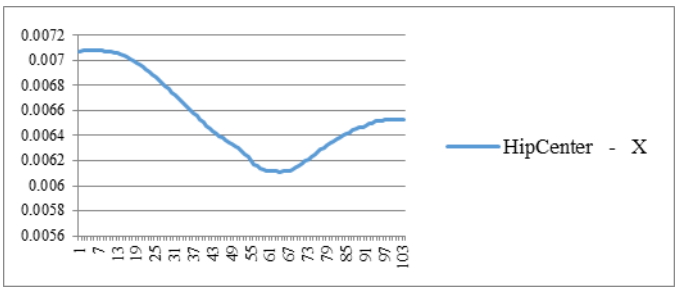


Figure 11. 100 frames taken from the recording of the abscissa coordinate, Hip Center joint. This graph plots variations in the detected hip position along the time. The scale of the horizontal axis is the frame number, and the vertical axis is the position of the joint as returned by the Kinect, in meters.

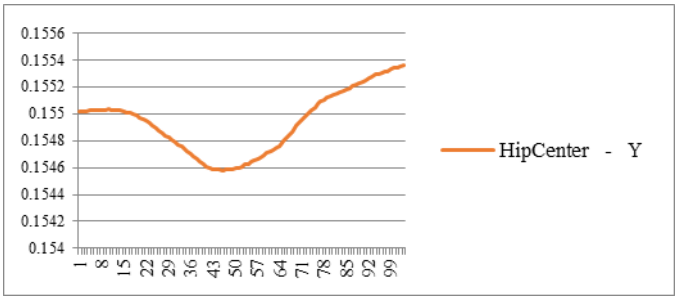


Figure 12. 100 frames taken from the recording of the ordinate coordinate, Hip Center. This graph plots variations in the detected hip position along the time. The scale of the horizontal axis is the frame number, and the vertical axis is the position of the joint as returned by the Kinect in meters.

Although a flat line was excepted, the variations of the obtained curve can be explained by the fact that a Random Forest algorithm was implemented in the Kinect device and the algorithm attempts to converge the value of the coordinates towards a realistic position in respect to the training database. The coordinates' values vary around a position with a local minima and maxima.

On the base of this experiment, we conclude that the error can be quantified in millimeters and sometimes less. We take into consideration this error and add a random number between 0 and 2 cm to the joint coordinates of the synthetic recordings.

b. Out of Field Of View Error

The second error is generated from joints that have not been tracked correctly. This occurs when a joint is concealed from the camera's field of view.

This error is not implemented as part of the algorithm; it is added implicitly when generating the recordings since the erroneous values contained in the original recordings will be included in the generation of the synthetic recordings.

C. Simulation algorithm

We present below a resume of the simulation algorithm, which will be detailed later on in this chapter:

- Capture a small set of actions using the RGB-D device, and obtain 20 3D joint coordinates from a human body.
- Convert the 3D coordinates to 2 angles using rotation matrices.
- Extract the extrema (local minima and local maxima) from the angles' sequences to solve some of the issues related to the dynamicity of the action.
- Align every sequence of joint angles' extrema using a Dynamic Time Warping (DTW) algorithm. Extract from this alignment a set of intervals.
- From these intervals, generate new points using one of the three methods: random (called *Random*), proportional to the first randomly chosen point (called *Proportionality*), or by averaging the previous two methods (called *Average*).
- Generate the full sequences of points that form the recording, convert their coordinates and angles to features (III.B.b) and train the Late Fusion algorithm using the simulated and real recordings.

a. Aligning the local minima and local maxima of the recording

k is a joint. Let i be the index of the recordings. For every recorded coordinate sequence A_{ik} (i.e. the i^{th} sequence of joint k), $\{\theta_x, \theta_y, \theta_z\}$ are the joint angles calculated as in III.B.a of a point at a temporal coordinate t in the sequence A_{ik}

We first remind the definition of an action as in (I.C.b), where it was stated that a set of recordings belonging to an action should be composed of the same gestures and that those recordings are performed with the same joint movement, yet, with changes in dynamicity and DOF amplitude. Hence, extracting the local minima and local maxima will clearly reduce the dependency from the temporal dynamicity, leaving the change in the DOF as the only parameter. As a result, we extract the set of local minima and maxima to generate a new sequence S_{ik}.

We choose a reference recording A_{Rk} randomly among all the recordings of an action in the training database and extract the sequence of extrema S_{Rk} from the recording.

We align the extrema from the sequences S_{ik}, one by one, with S_{Rk} by applying the DTW algorithm. At every alignment, a different reference recording is picked. The extrema that are aligned, as a result of finding the path with the lowest cost while performing the DTW, form a dimension of values. In Figure 13, we display in red some examples of points where two sequences S_{ik} and S_{Rk}, belonging to the action Right Hand Wave, HandRight joint, have been aligned.

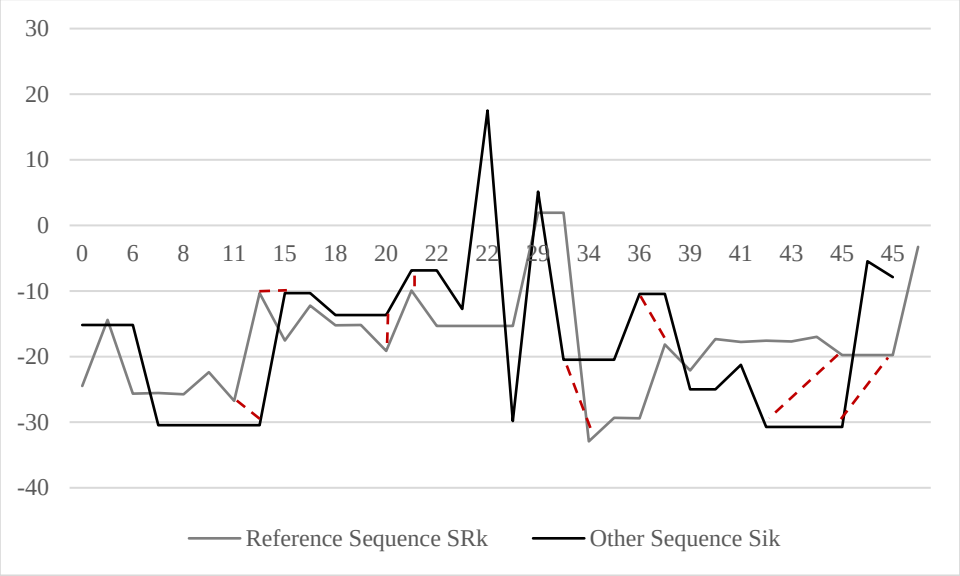


Figure 13. Alignment of the extrema sequences across time. The normal vertical and diagonal dashed red lines mark the alignments' location. The scale of the horizontal axis is the frame number, and the vertical axis is the value of the joint angle.

The dimensions of the values generated in the previous step are then joined according to the temporal coordinates of the reference recording. The resulting joined dimension will be called an interval. Figure 14 displays an enlarged sample (for a better view) from Figure 13, where two sequences have been aligned with a reference sequence. The green and red dashed line represent the resulting intervals.

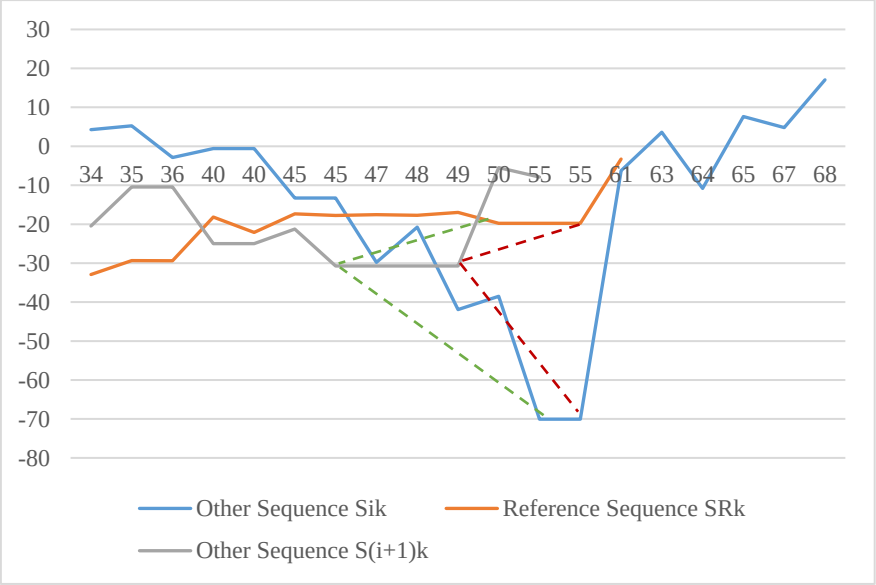


Figure 14. Enlarged samples from Figure 13. The scale of the horizontal axis is the frame number, and the vertical axis is the value of the joint angle.

At each temporal coordinate of the reference sequence S_{Rk}, we perform the union of the intervals. Hence, we obtain a final sequence of intervals where I_{tk} is an interval at a temporal coordinate t of a joint k.

b. Choosing the points

We increase the size of every I_{tk} to increase the diversity of the recordings' angles as follows:

$$I'_{tk} = [LB(I_{tk}) - K\times|UB(I_{tk}) - LB(I_{tk})|, UB(I_{tk}) + K\times|UB(I_{tk}) - LB(I_{tk})|] \quad (6)$$

Where *LB(I)* and *UB(I)* are respectively the lower and upper bounds of the interval I. *K* is a parameter that is picked arbitrarily to add the diversity. The bigger the I'_{tk}, the more the simulated recording changes from the initially captured ones. We chose *K=1* throughout the remainder of the study. Since it is hard to determine the value of *K*, it has been selected according to different experiments. Nevertheless, it is evident that *K* cannot be equal to a large value (10000), because the synthetic recordings will not output the same action as the initial actions, and *K=0* would not add diversity to the dataset.

Figure 15 is a visual representation of the increase in the interval's size, as in equation (6).

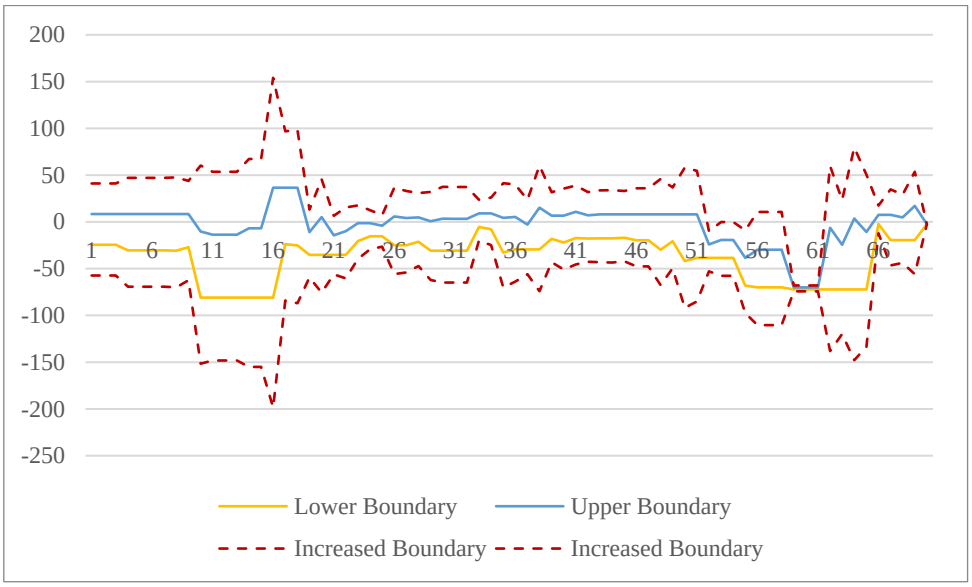


Figure 15. This figure represents the increase in the interval's size. The Bold Dashed Red line is the increased interval alignment. The scale of the horizontal axis is the frame number, and the vertical axis is the value of the joint angle.

Then, the points are generated arbitrarily by one of the three methods proposed below. The purpose of the algorithms is to pick the points inside the enlarged intervals and add randomness to the original sequence by picking at least one point as random.

At every joint, we choose a reference angle, arbitrarily, from the joint angles' collection $\{\theta_x, \theta_y, \theta_z\}$,. This angle is called θ_r, *r* being the reference angle. (This first step is standard for the three methods that will be implemented)

i. Random:

At the first interval, a point P_0 is picked randomly inside an I'_0.

The remaining of the angles at the chosen point, are calculated proportionally to the P_0.

For each interval, the points are picked the same way as the procedure above.

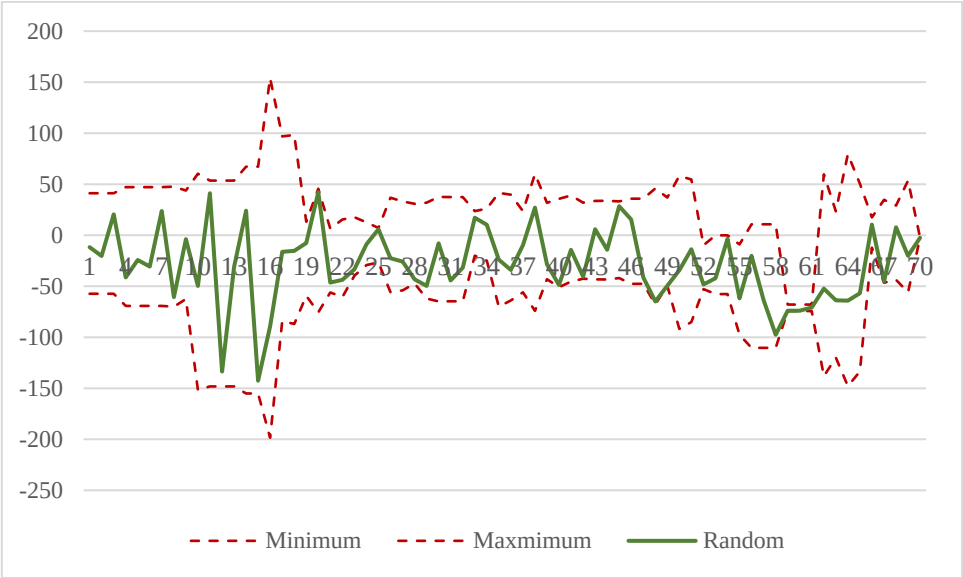


Figure 16. Example of a generated sequence with the method random. The scale of the horizontal axis is the frame number, and the vertical axis is the value of the joint angle.

ii. Proportionality:

We choose the first point P_0 randomly (as in (IV.C.e.2.1)) in I'_0, then, to improve the recording's smoothness, we calculate the rest of the points proportionally to the previous interval I'_{t-1}.

Figure 17. Example of a generated sequence with the method proportionality. The scale of the horizontal axis is the frame number, and the vertical axis is the value of the joint angle.

iii. Average:

For each interval, we calculate *P* as in (i), *P'* as in (IV.C.e.2.2) and average the result from (i) and (IV.C.e.2.2) to smooth the recording. We calculate all the intervals in the same way.

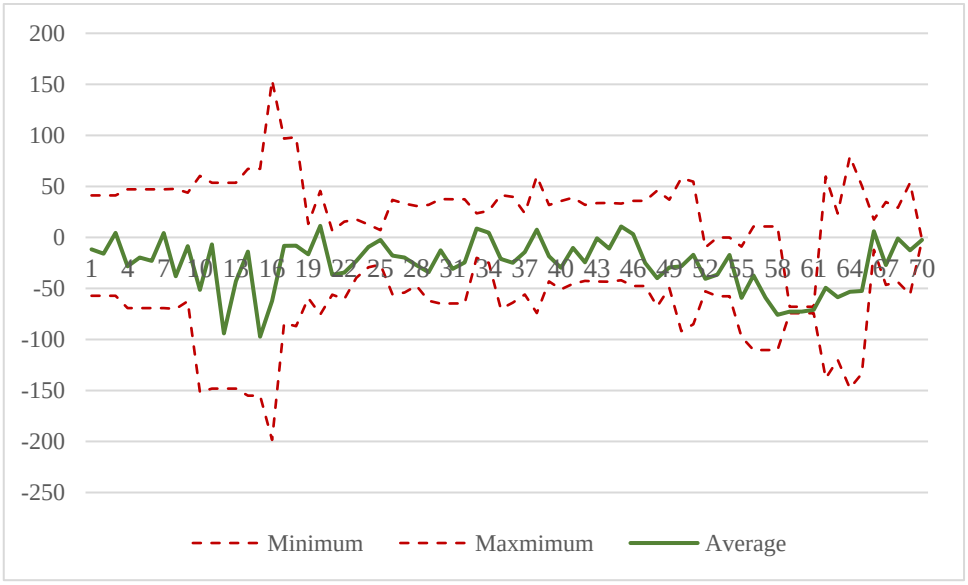


Figure 18. Example of a generated sequence with the method average. The scale of the horizontal axis is the frame number, and the vertical axis is the value of the joint angle.

c. Adding the variables

The first variable consists of changing the length of the simulated recording. Hence, we multiply the length of the sequences that we have obtained from the previous step by a random value (called *TimeWarping*) between the length of the shortest captured recording and the length of the longest one. We then divide the result by the length of the current recording, as in the formula below:

$$TimeWarping = \frac{RandomBetween(shortest\ action, longest\ action)}{current\_recording\_length} \quad (7)$$

We also add a small random error caused by the tracking algorithm as mentioned in IV.B.a.

d. Generating the recordings

After obtaining the sequence of points, the recording is generated by simple proportionality between the frames and the points.

Finally, the 3D coordinates are calculated by using rotation matrices (Reverse algorithm of III.B.a).

e. Analysis of the simulation algorithm

1. *Superfeatures*

Before explaining the concept of *superfeatures*, we recapitulate the definition of an action (I.C.b.8): recordings belonging to the same action are similar to a recording that is considered a reference if they have been performed with the same movements as the reference recording. This means that the body joints must move in the same direction and according to an order predefined by the reference recording. Yet, the recordings' time is dynamic, and the amplitude of the movements might differ. This difference between the performances of a recording generates a *margin* of difference from the reference recording that denotes the "uniqueness" of a recording. This definition is correlated to the idea of the *superfeatures*, which is clarified afterward.

The explanation in the remaining of this paragraph is based on a simple decision stump (e.g., Discrete Adaboost decision). During the classification, the decision for labeling the extracted features with different classes using a decision threshold algorithm gives a weighted error (ε). When $\varepsilon=0$, the decision might lead to an over-classification. This means that the feature is over-discriminant and is prone to miss new performances of the actions when testing features that are located outside the *margin* of a small actionset belonging to one class. In this case, these types of features are called *superfeatures*.

In fact, if the training dataset is small, the classes' margins would most probably be small, and a *superfeature* will appear. Hence, increasing the number of training recordings will increase the *margin* and eliminate the *superfeatures*. Consequently, in the rest of the Method (IV.C.e.2, we demonstrate that the simulator will generate larger margins for different classes and adds an error to the features' decision.

2. *Analysis*

The analysis of the simulator's behavior is conducted using the threshold concept of a Discrete Adaboost algorithm since its weak classifiers are easy to observe, and it compares the features' values with thresholds and errors.

Let x_i be the recording i, and y_i the class (In this case, -1 or 1, since the Adaboost is a binary classifier) of recording i. The couple (x_i, y_i) forms the input for training the Adaboost.

Since the Adaboost bases its decision on the error, it is reasonable to add one, as stated in IV.C.e.1. However, this is done with certain limits and without diverting too much from the initial recording (This will become clearer later on, with the explanation of the different simulation methods).

Let J be the number of features and $j=\{0,\ldots,J-1\}$ the feature index, as a result, a feature of a recording i is called f_{ij}. The collection of features for a single recording is f_i.

h_j is the early classifier for the feature f_j, with ε the error calculated after every separation of a set of values for f_j, by the Adaboost. One of our purposes is to eliminate *superfeatures* (where $\varepsilon=0$). Thus, the error must not be null. During training, f_j is a *superfeature* if, for every recording x_i, we have:

(In the following equations, $P(x)$ is the probability of x)

$$\forall i, if\ \varepsilon_j = 0,\ \text{then}\ h_j(x_i)\times y_i = 1 \Leftrightarrow P\big(h_j(x_i)\times y_i = 1\big) = 1 \quad (8)$$

We can be sure that we have removed all the *superfeatures*

when $\forall h_j, \varepsilon_j \neq 0 \iff$

$$\forall f_j, \exists x_i \in \text{recordings} \,/\, h_j(x_i)\times y_i = -1 \quad (9)$$

Consequently,

$$P\big(h_j(x_i)\times y_i = -1\big) \neq 0 \qquad (10)$$

Next, it is shown, by simulating new angles, that the probability of not obtaining *superfeatures* increases.

For a given feature f_j:

Let S_p be the interval of the angle values for the original recordings labeled 1, and S'_p the interval of angle for the simulated recordings labeled 1. Let S_n be the interval of the angle value values for the original recordings from the class labeled -1 and S'_n the interval of angle values for the simulated recordings labeled -1. Then:

$$S_p \subset S'_p \text{ and } S_n \subset S'_n \qquad (11)$$

If a *superfeature* f_j exists, then:

$$\varepsilon_j = 0 \Longrightarrow \forall i, h_j(x_i)\times y_i = 1 \Longrightarrow S_n \cap S_p = \emptyset \qquad (12)$$

When simulating the new angles from the intervals that we have found, using the DTW algorithm, we obtain:

$$\mathrm{P}\big(S'_n \cap S'_p = \emptyset\big) > 0 \qquad (13)$$

Consequently, variations are added to the angles as follows:

F is the total number of frames and $k \in [1;F]$ and (X_k, Y_k) are the angles at frame k

In order to eliminate the dependency on the dynamicity of the recordings, the extrema X'_k and Y'_k of the angle sequences are calculated as follows:

X_k is a local minima (noted X_k') if:

$$X_k < X_{k-1} \text{ and } X_k < X_{k+1} \qquad (14)$$

Similarly, X_k is a local maxima (noted X''_k) if:

$$X_k > X_{k-1} \text{ and } X_k > X_{k+1} \qquad (15)$$

Generating the intervals using DTW:

Let *A* and *B* be extrema sequences for the same action, different recordings but same human joint, and *(A'x_n, A'y_n)*, for $n \in [1;N]$ (Where *N* is the length of the sequence *A*) the angles of *A* and *(B'$x_{n'}$, B'$y_{n'}$)*, for $n' \in [1, N']$ (Where *N'* is the length of the sequence *B*) the angles of *B*. From the alignment of *A* and *B*, intervals are generated as follows:

K points are obtained for each angle x and y:

$$K_x(n, n') = \min(c(A'x_{n-1}, B'x_{n'-1}), c(A'x_n, B'x_{n'-1}), c(A'x_{n-1}, B'x_{n'})) \quad (16)$$

Where *c*(*Ax*, *Bx*) is the cost of the shortest path, calculated with DTW, between *Ax* and *Bx*. Same for *y'*.

At each point *K(n, n')*, the intervals $I_n=[Ax_n, Bx_{n'}]$ are the acceptable intervals at frame *n* and *n'*, according to a reference recording *A*.

In order to increase the diversity of the recordings' angles, intervals I'_n are defined by tripling the size of intervals I_n so that I_n is in the middle of I'_n. By increasing the size of the interval, additional diversity is added to the simulated actions. As a result, a large interval will generate actions that are too far from the real actions. We set the size of the interval as a user parameter.

Then, the three following methods are used to choose the simulated points (The more the points vary from the initial ones, the more it would be probable to add the error)

Let $x'_m (m \in [1;M]$, where *M* is the length of the number of intervals*)* be a collection of points chosen inside I'_n. The step of choosing the points is done using the three following methods:

2.1. Random:

$$\mathrm{P}(x'_m \in I_m) = \frac{1}{3} \; and \; \mathrm{P}(x'_m \notin I_m) = \frac{2}{3} \quad (17)$$

$$\Longrightarrow \mathrm{P}_{t\geq 0}(x'_m \in I_m) = \frac{1}{3} \quad (18)$$

2.2. Proportionality:

At the first frame, x_1 is chosen randomly

$$\mathrm{P}_{t=0}(x'_m \in I_m) = \frac{1}{3} \quad (19)$$

The following equation is used to generate the next values:

$$x'_n = x'_{n-1} \times \frac{I'_n}{I'_{n-1}} (\forall n > 1) \quad (20)$$

Since the probability that the rest of the x'_p belong to I_p depends only on the first frame, we have:

$$\mathrm{P}_{t>0}\left(x'_p \in I_p \middle| x'_{p_{t=0}} \in I_p\right) = 1 \qquad (21)$$

$$\mathrm{P}_{t>0}\left(x'_p \in I_p \middle| x'_{p_{t=0}} \notin I_p\right) = 0 \qquad (22)$$

We deduce that:

$$P\left(x'_p \in I_p\right) = \frac{1}{3} \qquad (23)$$

2.3. Average:

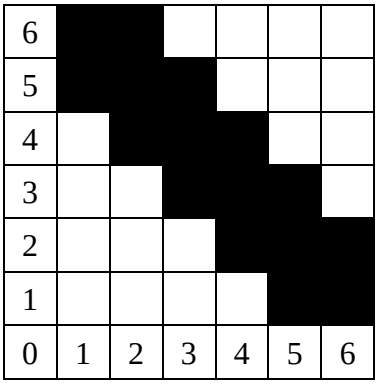


Figure 19. Matrix representing the acceptable values for the *average* method inside the interval *[3,4]*

At *p*, x'_p is defined as follows:

$$x'_p = \frac{random(x_p) + proportionality(x_p)}{2} \qquad (24)$$

Consider the interval *I=[3,4]* in the matrix in Figure 19. The interval obtained by tripling the size of I is the interval *I'=[1,6]*. Thus, the points that are obtained by applying the average method must be inside this interval. Consequently, the black cells represent the points that correspond to the initial interval *I* with a probability of $1 - \frac{white\_surface}{Total\_surface}$

$$\mathrm{P}_{t=0}\left(x'_p \notin I_p\right) = 1 - \left(\frac{2}{3}\right)^2 = \frac{5}{9} \qquad (25)$$

$$\mathrm{P}_{t=0}\left(x'_p \in I_p\right) = \frac{4}{9} \qquad (26)$$

By a simple calculation, we obtain:

$$\mathrm{P}_{t>0}\left(x'_p \notin I_p\right) = \frac{5}{9} \qquad (27)$$

Some observations about the three methods are given below:

- *Random*: this method generates the recordings that are the most divergent from the real recordings since the chosen points are the results of added noise. Therefore, it produces the highest decision error during classification and requires the simulation of a larger dataset than the other methods, for the results to converge.
- *Proportionality*: this method generates more points outside the initial interval than *average*. However, the synthetic recordings are smoother than the ones generated using the other methods, since the choice of the points depends on the first frame. Consequently, this method produces the recordings that are the most similar to the real ones. Therefore, the results are optimized compared to the other methods, when training with a large number of synthetic recordings. In this case, these recordings are the least divergent from the initial ones.
- *Average*: compared to the other methods, *average* generates the largest number of points inside the interval. Therefore, the new recordings would be the most similar to the real recordings compared to the other methods. However, since this one depends on *random*, it contains random noise. Thus, results are optimized when training with a larger number of synthetic recordings compared to the other 2 methods.

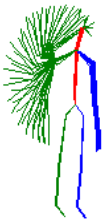
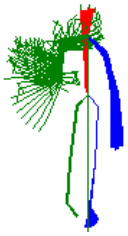
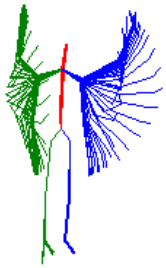
In resume, when classifying with a boosting algorithm trained with a small number of recordings, the method *proportionality* is expected to give the best results, but if the number of synthetic recordings is increased, *average* and *proportionality* will output approximately the same result. Nevertheless, *proportionality* will still be the best method, since it generates the recordings that are the closest to the real dataset.

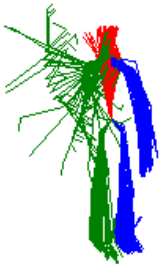
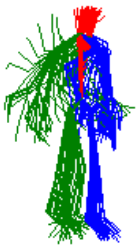
f. Experiments

As previously seen, there is randomness in the recordings that we simulate; hence, the synthetic recordings will not be the same to the real ones. It is interesting to display an example to show how the simulated recordings will look like and compare them with the real ones. Table 10 compares, from the CAP dataset, the real and simulated recordings visually.

The frames in the figures have been combined into a single image (similar to a Voxel) for easier representation.

Table 10. Visual comparison between real and simulated recordings - CAP dataset

Action	Real Recording	Simulated Recording
2 hands up		
crouch		
Raise right hand up		
Right hand wave		
Surrender		
Tennis Forehand Drive		

Tennis Backhand Drive		

The simulated actions have been generated with the proportionality method

It is obvious that the actions are "clearer" in the real recordings than in the simulated recordings, especially in right-hand wave, surrender, tennis forehand drive and tennis backhand drive.

During the experiments, all classifiers are trained with a one-vs.-all strategy, and are evaluated with the well-known F-Measure. This equation was chosen since it is a standard evaluation performance measure that shows clearly when a classifier performs well or fails to classify a label.

In this section, to show that our hypothesis is based on trusted results, the following steps and parameters are considered:

- Find the appropriate parameters for each classifier (Adaboost: number of iterations, KNN: K, Random Forest: number of trees and levels, SVM: choice of the kernel)
- Train Adaboost (AB), KNN, Random Forest (RF) and SVM with 5, 10, 20, 50, 100 and 200 synthetic recording sets.

In addition, the steps below are implemented:

- All the vectors that are inputted to the classifiers have been normalized with zero mean and unit variance.
- During training, the synthetic recordings have been merged with the real ones.

1. *Training Data*

The following datasets will be experimented on:

- A captured dataset called *captured*
- Microsoft MSRC-12 [58]
- MSR Action3D Dataset [60]

We include a variation in the datasets during training:

- Small dataset: only 2 or 3 recordings are picked from every class in the dataset
- Large dataset: all the recordings are picked from every class in the dataset

The datasets have already been introduced in details in III. Additionally, we mention an important difference between the captured and MSRC-12 databases that affects the results noticeably: the MSRC-12 recordings do not contain tracking errors from the RGB-D sensor, as

opposed to the captured dataset. This means that the recordings belonging to the Microsoft dataset are not hidden from the device's viewpoint during capture. Moreover, recordings belonging to the same class are very similar. This difference affects the DTW's alignment and consequently creates large intervals that increase the randomness in the methods and affect the results of the classification algorithms negatively.

2. *Finding the parameters*

For more accuracy, the Discrete Adaboost is trained for a number of early classifiers varying from 2 to 500. The results show that the classifier performs best between 10 and 20 early classifiers, independently from the late classifier's iterations. Thus, for performance purposes, and because the difference between the results for 10 and 20 early classifiers is small, we chose to run with 20 early classifiers. Some classification examples when running on multiple iterations are given in

Table 11.

Table 11. Adaboost results from training with real recordings, CAP dataset

Dataset Early Classifiers	F-Measure
10	0.8201
20	0.8321
50	0.8116
100	0.8235
200	0.8060
500	0.8182

To find the proper parameters for Random Forest, variating the number of trees between 50 and 200 and levels between 5 to 15, shows that the number of trees is the best at 100 and levels at 10, as in Table 12. The values have been picked arbitrarily, by taking into consideration the performances, since the number of experiments is vast.

Regarding the features, since the set is large (III.B.b) a statistical analysis of the early Adaboost classifiers allows to pick the 25 first features with the highest appearance rate, in addition to 5 random ones. We chose to train the classifier with the most discriminant features since the feature set is large and training Random Forest and SVM with it would result in poor performances. Hence, we needed to build a proper benchmark to compare the performances. As for the feature number and the number of levels in the Random Forest, those are also parameters that we set to improve the performances and provide ourselves with the required benchmark.

To perform a good recognition with Random Forest, the dataset is balanced when adding simulated recordings; before training every tree, all the real recordings are picked in addition to a number of simulated recordings to balance the one-vs-all classification. Nevertheless, the dataset is neither altered nor balanced when training with a small number of recordings because the training set does not suffice for balancing.

Table 12. Random forest results from training with real recordings, CAP dataset

Trees	50	50	50	100	100	100	200	200
Levels	5	10	15	5	10	15	5	10
F-Measure	0.4909	0.6115	0.4094	0.5938	0.6902	0.5714	0.6516	0.6471

As for KNN parameter, K is chosen arbitrarily, without carrying out any cross-fold validation or another method. However, with a small dataset, we calculate the average of the results with multiple values of K, and when training with simulated recordings, K is increased arbitrarily (e.g., K=15 for 20 recordings, K=30 for 50 recordings, K=50 for 100 recordings and K=100 for 200 recordings).

The SVM algorithm is not studied in details; the feature set is the same as for the Random Forest and, according to experimentations, the best kernel appeared to be a Gaussian function (according to .Net framework [157]).

3. *Results*

The result of the classification would certainly differ when changing the number of simulated recordings. Table 13 compares the outcome of the classification of two different datasets while using the method *proportionality* and increasing the number of simulated recordings from 5 to 200 recordings. As for Figure 20 and Figure 21, they illustrate the behavior of the classification results along the number of simulated recordings. According to Table 13, the best results appear when simulating between 50 and 200 recordings, when classifying the large dataset only. Nevertheless, according to Figure 20 and Figure 21, it is obvious that the results start to stabilize with 50 simulated recordings. A convergence and optimization are noted when training simulated recordings with the method *proportionality*. The results are optimized with a large number of simulated recordings for Adaboost and KNN, as for Random Forest, they keep degrading. With KNN, the best results appear with 50 synthetic recordings.

Consequently, according to the results and for performance purposes, 50 recordings will be simulated for training during the rest of this chapter.

Table 13. Adaboost results (F-MEASURE) from training with synthetic recordings with proportionality

	CAP dataset		**Msrc-12 dataset**
Number of simulated recordings	**Small set**	**Large set**	**Small set**
5	0.7612	0.7681	0.8919
10	0.7832	0.8133	0.9407
20	0.6357	0.8163	0.9295
50	0.7121	0.8533	0.9256
100	0.7143	0.9178	0.9300
200	0.6935	0.8919	0.9361
Real recordings	0.6379	0.8333	0.7386

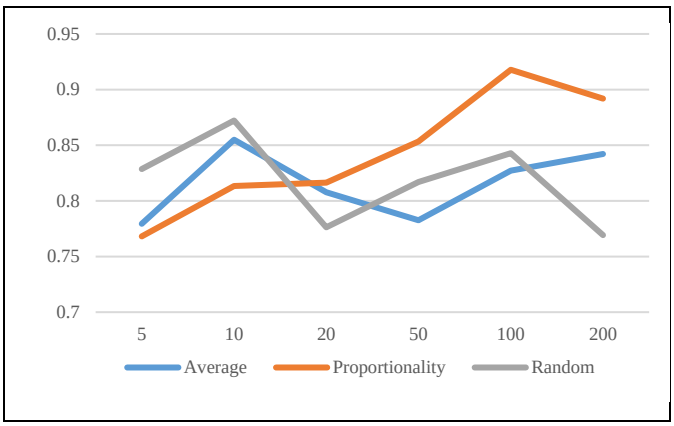


Figure 20. Plot of the variations of the F-MEASURE (vertical axis) along the number of recordings (horizontal axis) – CAP dataset – Adaboost classification

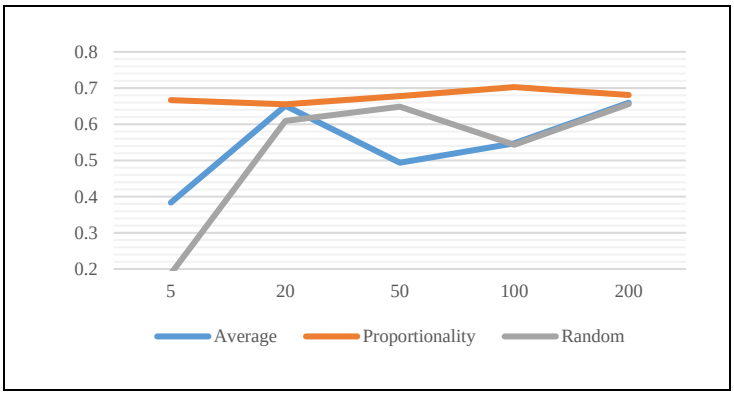


Figure 21. Plot of the variations of the F-MEASURE (vertical axis) along the number of recordings (horizontal axis) - MSRC-12 dataset - KNN classification

g. Training and classification

We note that the large dataset, even though considered as large, is still small compared to the simulated dataset. Hence, a balancing cannot be done when training with real recordings, small or large dataset.

In Table 14 to Table 16, we display the results of classifications, when training with 50 synthetic recordings per class.

Table 14. Results with CAP dataset (50 simulated recordings per class)

Simulation Method	**Small**			**Large**			
	AB	**KNN**	**RF**	**AB**	**KNN**	**RF**	**SVM**
Average	0.6815	0.4191	0.6971	0.7826	0.6552	0.6500	0.5342
Proportionality	0.7121	0.4334	0.7362	0.8533	0.6833	0.7626	0.5693
Random	0.5778	0.3900	0.5280	0.8169	0.6491	0.6929	0.4812
Real recordings	0.6379	0.2714	0.5794[(b)]	0.8321	0.3819[(a)]	0.6902[(b)]	0.3841

a. *Average of results with K={3,5,9}*

b. *Not balanced*

Table 15. Results with MSRC-12 dataset (50 simulated recordings per class)

Simulation Method	AB	KNN	RF	SVM
Average	0.8678	0.4938	0.7842	0.4352
Proportionality	0.9561	0.6776	0.8593	0.5179
Random	0.8443	0.6486	0.8288	0.4054
Real recordings	0.7386	0.6257[a]	0.2556[b]	0.2182

a. *Average of results with K={3,5}*

b. *Not balanced*

Table 16. Results with MSR Action3D dataset (50 simulated recordings per class)

Simulation Method	AB	RF	SVM
Average	0.5044	0.2634	0.1379
Proportionality	0.5937	0.2665	0.2000
Random	0.5356	0.2512	0.0976
Real recordings	0.4949	0.2222[a]	0.1661

a. *Not balanced*

As previously mentioned in IV.C.f.1, a significant difference exists between MSRC-12 and the two other databases. This affects the results noticeably: the MSRC-12 actions do not contain tracking errors generated from the RGB-D sensor, as opposed to the CAP and MSR Action3D datasets. It means that the joints in MSRC-12 are not hidden from the device's viewpoint during capture. Therefore, recordings belonging to the same class are very similar. The tracking errors affect the DTW's alignment, thus, creating large intervals that increase the randomness in the methods and degrade the classification results. Therefore, with the CAP dataset, the difference between training with simulated recordings compared to training the original ones is small, since the source of the synthetic recordings is large and noisy. As for classifying the MSRC-12 actions with synthetic recordings, the results are significantly better compared to the CAP database since the MSRC-12 dataset is "cleaner" than the captured one.

h. Limits of the Simulator

When simulating recordings from very long ones (full MSRC-12 recordings, before segmentation), we note that the resulting recordings are not very "Human-like." We present the target recordings by miming them and stating their names to three persons. We then ask them to identify three recordings chosen randomly from each simulated set. We note, for each individual, the number of recordings that are identified correctly over the total number of presented recording. The results are shown in Table 17.

Problems occur when simulating from the full MSRC-12 dataset because the actions are repeated multiple times in the same segment and at unknown frame positions. Hence, the DTW

will not be able to perform a proper alignment of the local maxima and minima, failing to align very different sequences in size and content.

The most flagrant confusions exist between the *Push Object with Right Hand to the Right* and the *Tennis Backhand Drive,* and between *Raise both Arms to the Sides* with *Wear Goggles*. In fact, the actions are a lot similar and were very hard to recognize visually.

Table 17. Visual results for the simulated recordings

Action	Original dataset	Person 1	Person 2	Person 3	Avg. (%)
Raise both arms to the sides	MSRC-12	1/3	1/3	1/3	33
Crouch	MSRC-12	3/3	3/3	3/3	100
Push object with right hand to the right	MSRC-12	2/3	1/3	0/3	33
Wear Goggles	MSRC-12	1/3	1/3	1/3	33
Wave hands in air	MSRC-12	2/3	3/3	3/3	89
Walk	MoCap BVH	3/3	3/3	3/3	100
Kick	MoCap BVH	3/3	1/3	2/3	67
Shoulders up	Captured	3/3	3/3	3/3	100
Tennis backhand drive	Captured	2/3	2/3	2/3	67
Tennis forehand drive	Captured	3/3	2/3	2/3	78
Surrender	Captured	3/3	3/3	2/3	89
Hand Wave	Captured	3/3	3/3	3/3	100

1/3: 1 recording identified correctly from 3 recordings.

Avg: the average of the positive result from Person 1, 2 and 3

i. Synthesis

As observed in the results, training with simulated recordings improves the performances for *proportionality* and *average* for all the classifiers. Note that with p*roportionality* (the method that generates recordings that diverge the least from the original ones), with a sufficient number of simulated recordings, the results start to stabilize, and we are able to achieve good results.

j. Comparison with SMOTE

The method is compared with the SMOTE algorithm [76] as shown in Table 18 and Table 19. The SMOTE parameter for choosing the closest neighbor is the same as the optimized parameter K that was deduced from multiple KNN classifications, consequently, K=5 is picked.

Table 18. Comparison between the current method and SMOTE, with CAP, large dataset

Algorithm	Real recordings	SMOTE, number of simulated recordings					Current Method
		5	*20*	*50*	100	200	50 simulated
AB	0.8321	0.8085	0.7891	0.8027	0.8571	0.8841	0.8533
RF	0.6902	0.7286	0.7752	0.7445	0.7075	0.7172	0.7626
KNN	0.3819	0.5926	0.5085	0.5574	0.6154	0.5902	0.6833
SVM	0.3841	0.3609	0.3750	0.5344	0.5588	0.4733	0.5693

Table 19. Comparison between the current method and SMOTE, with MSRC-12

Algorithm	Real recordings	SMOTE, number of simulated recordings						Current Method
		5	*20*	*50*	100	200	400	50 simulated
AB	0.7386	0.8638	0.8263	0.8286	0.8259	0.8774	0.8599	0.9561
RF	0.2556	0.5355	0.8071	0.8175	0.7883	0.8175	0.7754	0.8593
KNN	0.7300	0.5732	0.6082	0.6071	0.6889	0.6554	0.6322	0.6257
SVM	0.2182	0.1974	0.3451	0.3139	0.3439	0.3963	0.3318	0.5179

As observed in Table 18 and Table 19 the current approach performs better than all the results with SMOTE with 50 simulated recordings. Hence, the proposed method performances exceed the SMOTE by far.

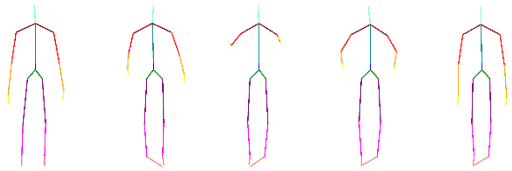
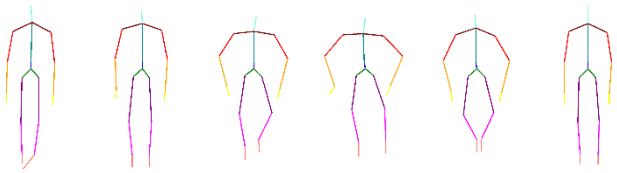
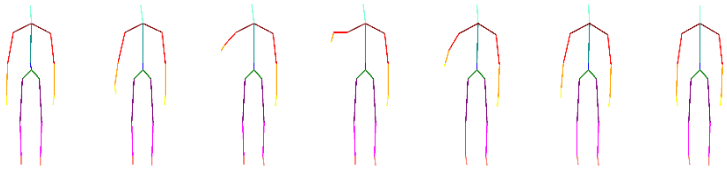
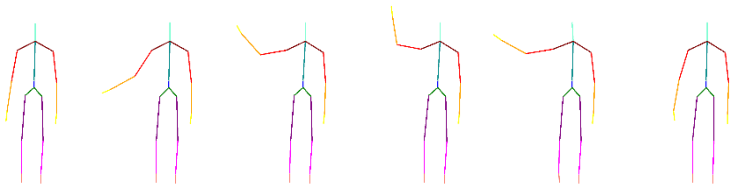
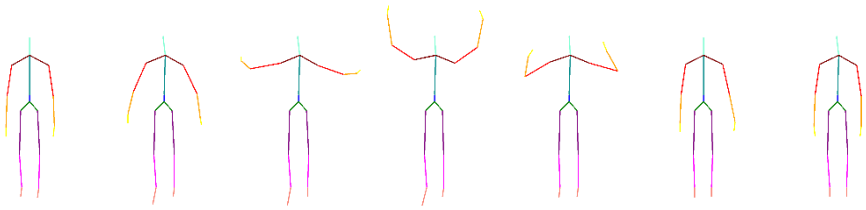
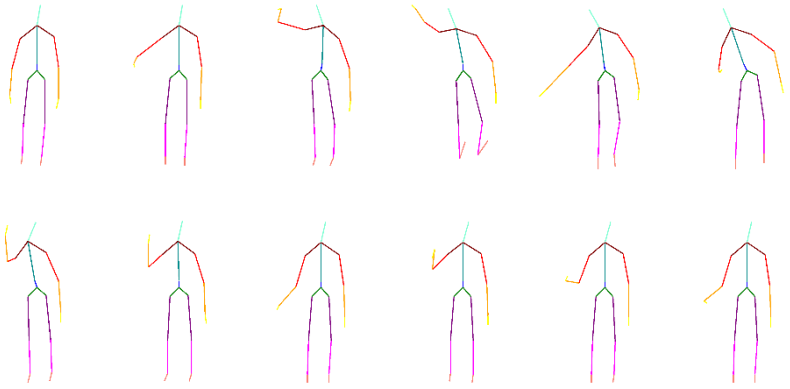
k. Application

In the scope of this study, we developed and posted online, as Open Source, the application for the simulation algorithm, an ActionViewer and a tool for converting coordinates recorded from the RGB-D camera to joint angles coordinates. These are available at [152] (Refer to Appendix IV – Datasets

Due to graphical constraints, we only display the frames that we judge as the most relevant for an action.

Table 104. Custom dataset called CAPtured dataset (CAP)

Action	Frames

2 hands up	
Crouch	
Raise right hand up	
Right Hand Wave	
Surrender	
Tennis Forehand Drive	

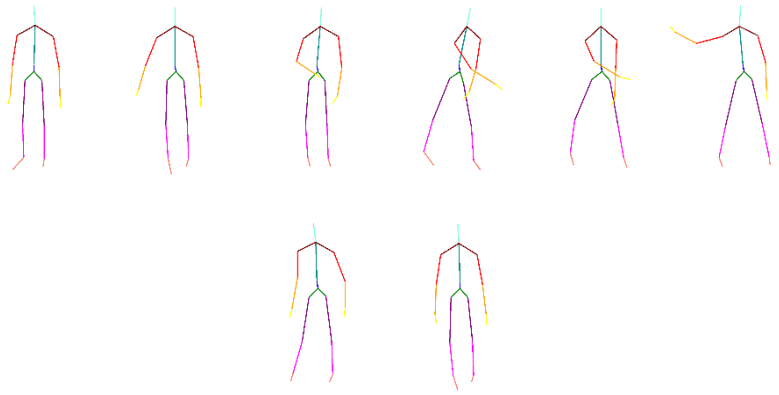
Tennis Backhand Drive	

Table 105. Custom dataset with right-hand wave confusion (CR)

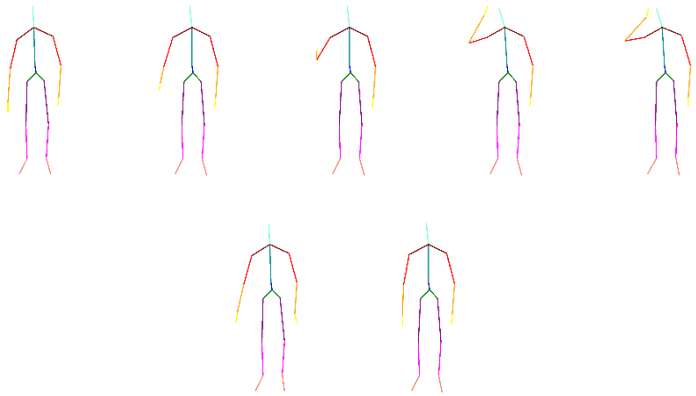
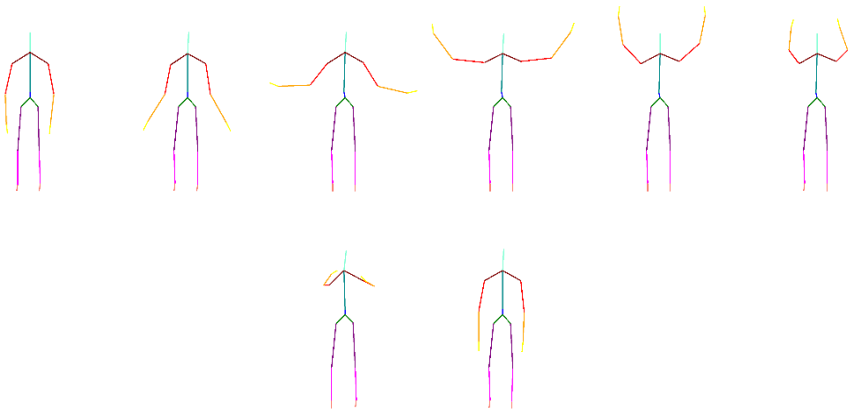
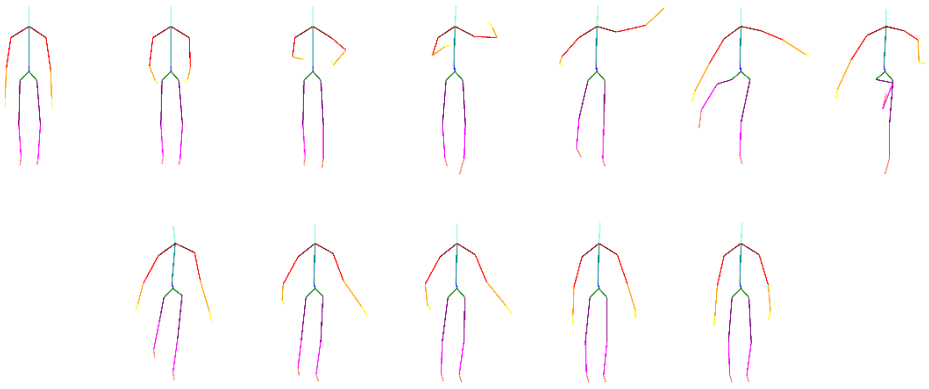
Action	Frames
Right-hand wave A	
Raise Right hand up	Same as in Table 104
Right-hand wave B	Same as in Table 104
Surrender	Same as in Table 104
Tennis Forehand Drive	Same as in Table 104
Tennis Backhand Drive	Same as in Table 104

Table 106. Custom dataset with Swimming and Soccer (SS)

Action	Frames
Swimming Crawl	
Swimming Butterfly	
Soccer	

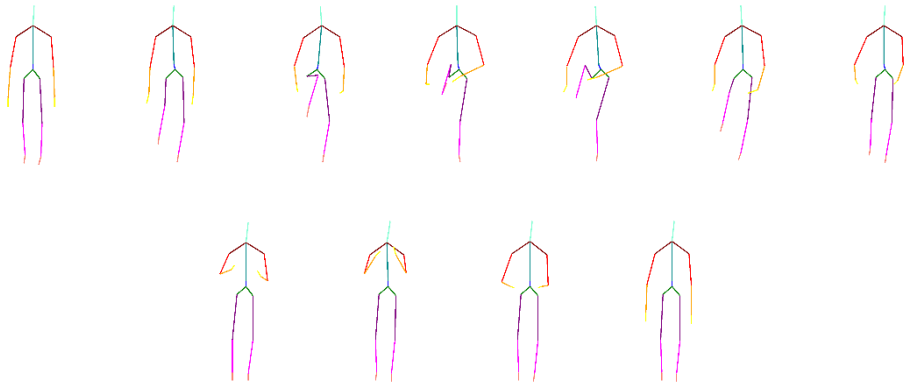
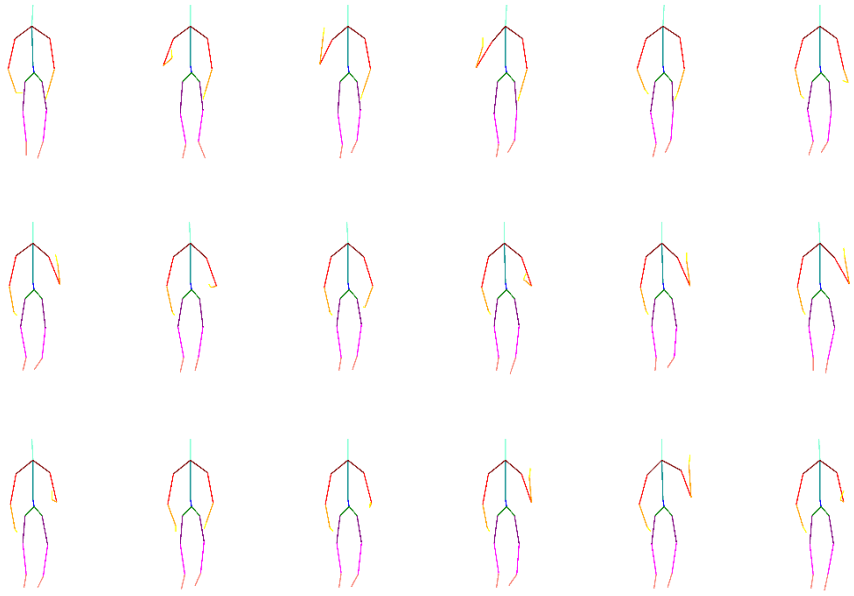
Not Soccer	

Table 107. Custom dataset with right hand up & left hand up (RL)

Action	**Frames**
Right, Left, Left, Left	

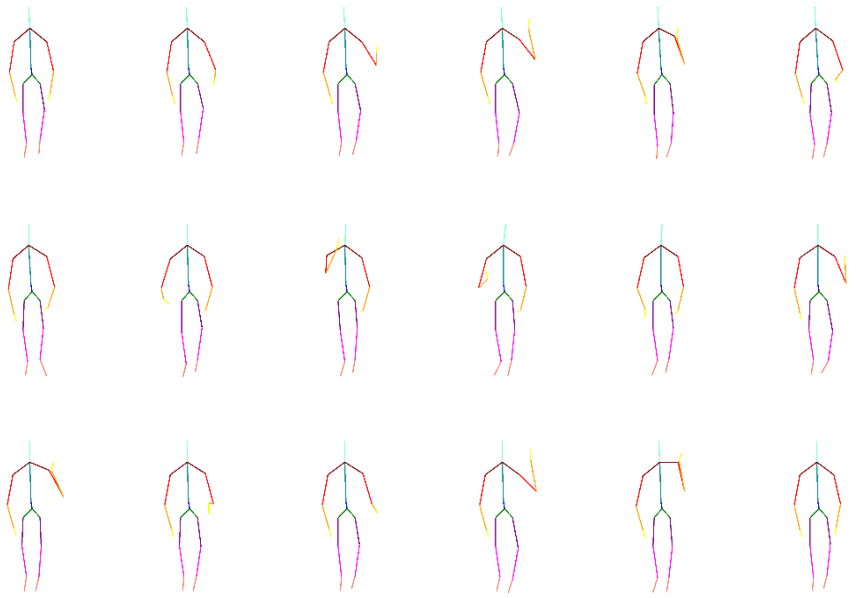
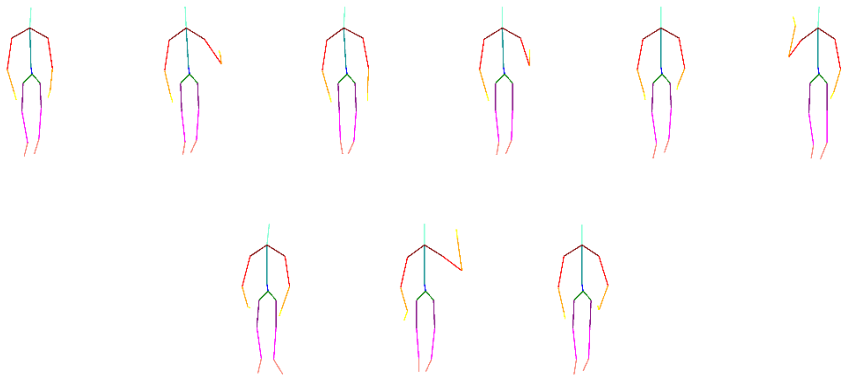
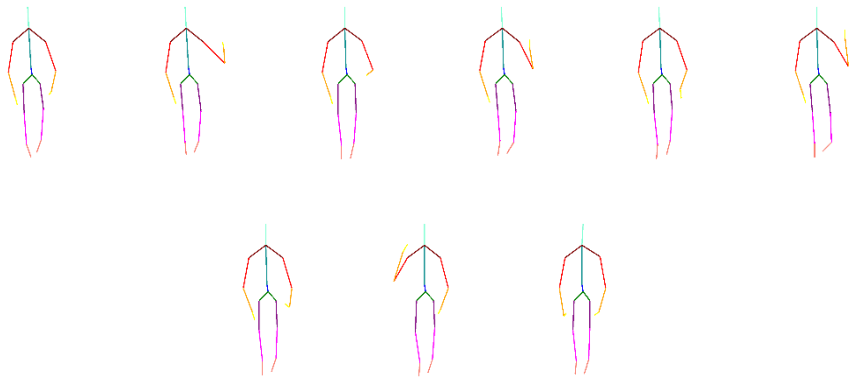
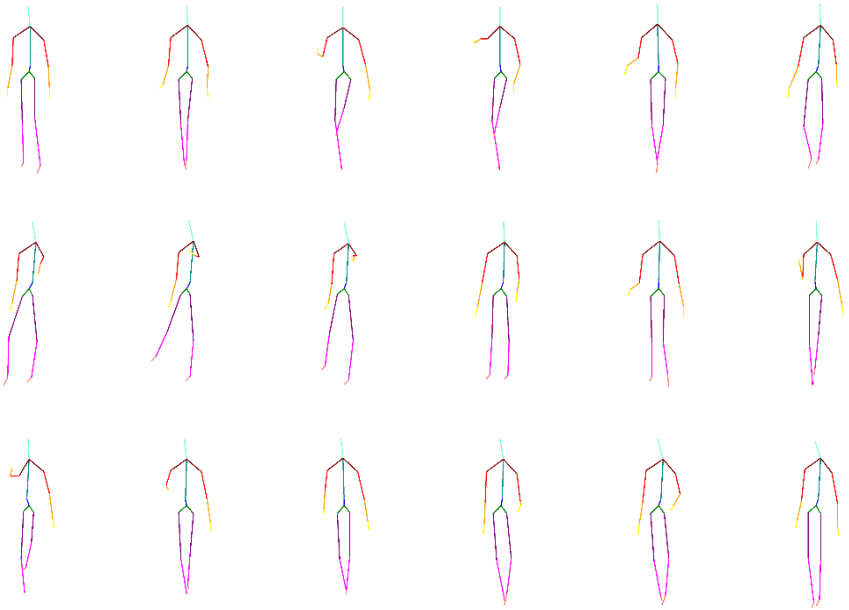
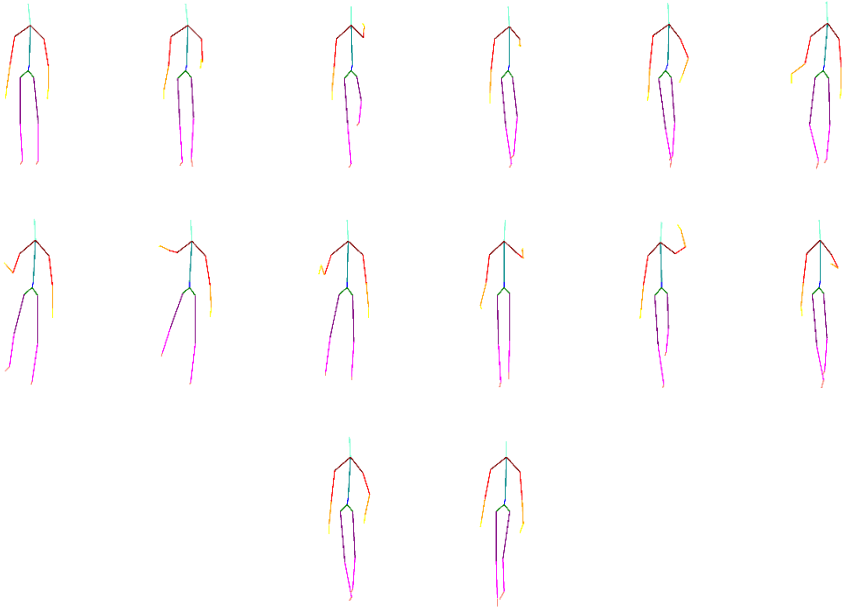
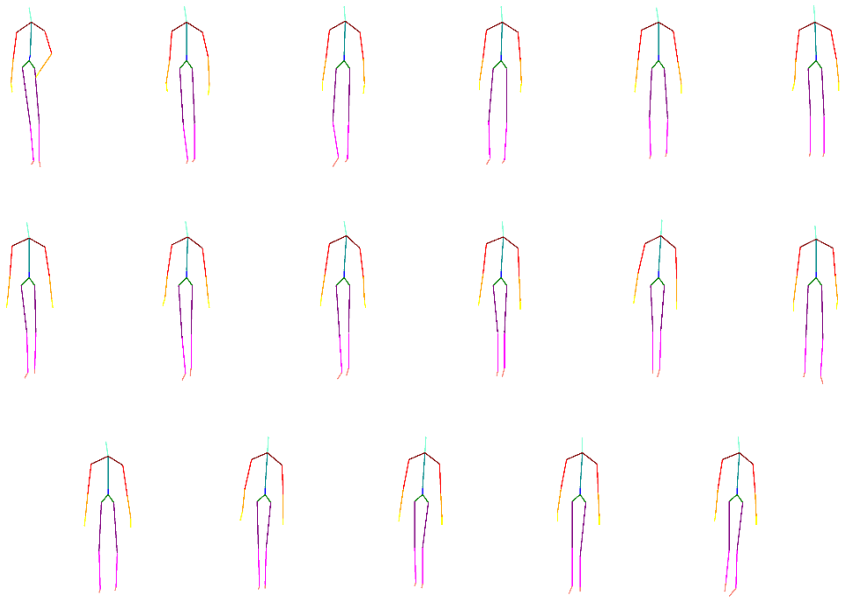
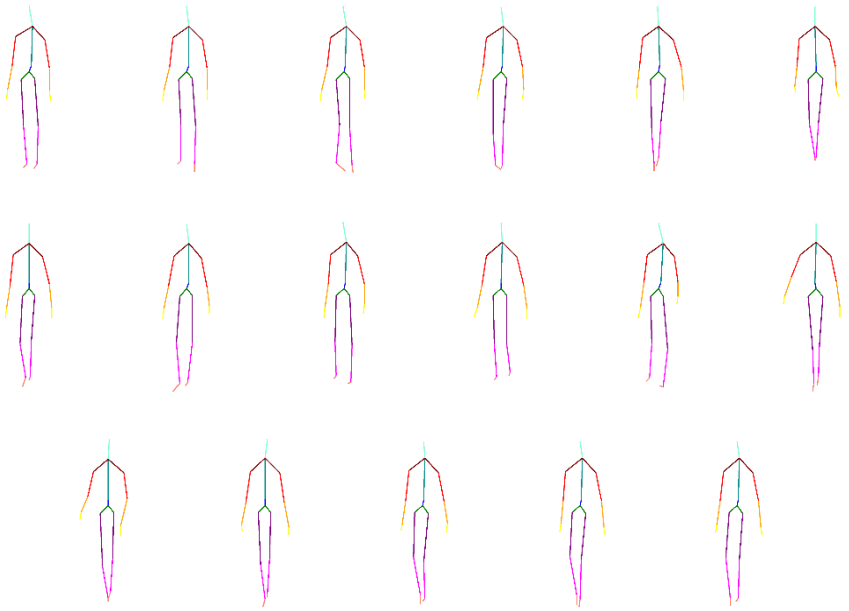
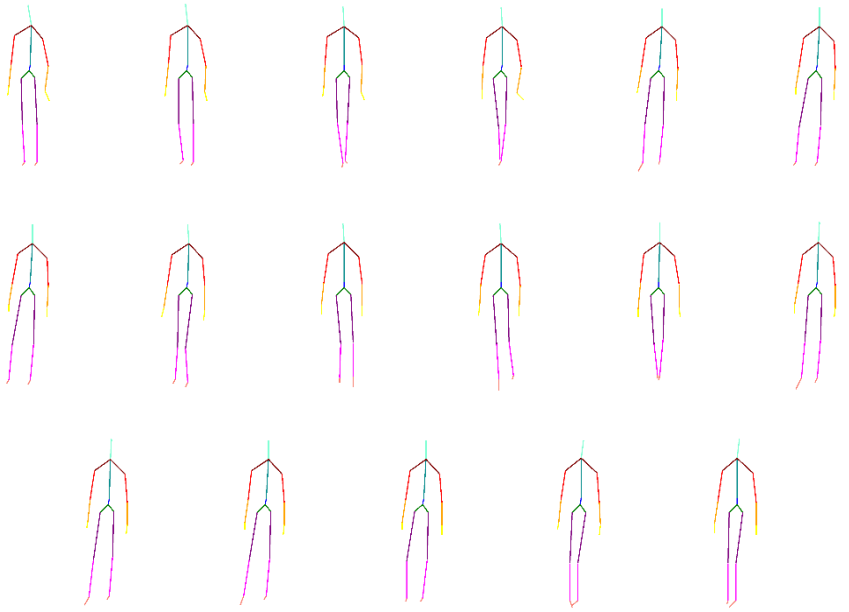
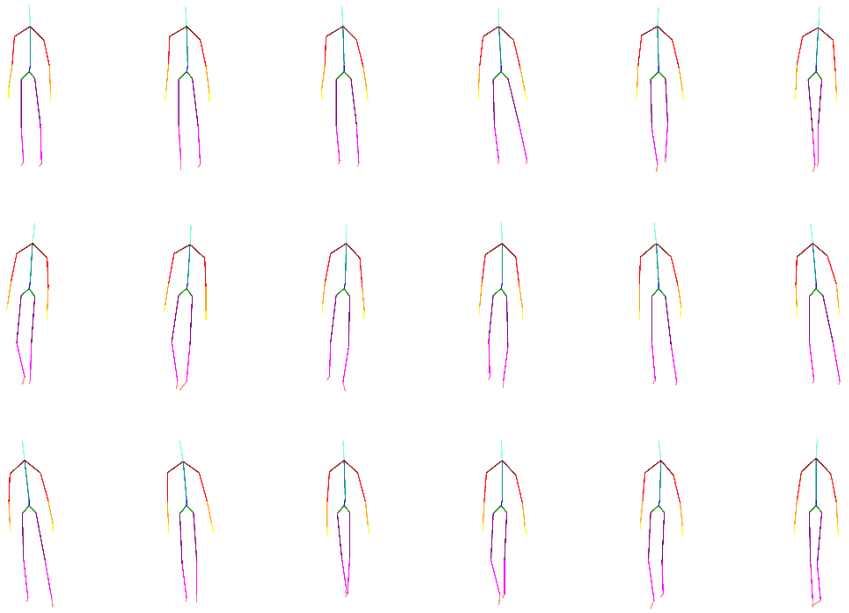
Left, Right, Left, Left	
Left, Left, Right, Left	
Left, Left, Left, Right	

Table 108. Custom gait dataset

Action	Frames
Army march	
Incorrect army march	

Parkinsonian-like shuffling	
Neurological Experiment	

Normal gait	
Right leg fracture	

Left leg fracture	

Appendix V - Open Source Contributions for more information). In addition, we have published the captured action dataset, referenced in this chapter.

l. Conclusion

The problem of non-discriminatory actions datasets for action recognition was overcome by enlarging a set of recordings performed by different persons, in different ways, and captured by an RGB-D camera. This chapter presents a novel method for generating synthetic recordings, for training action recognition algorithms. The parameters of the method are analyzed and the most appropriate one, for the different classifiers, is found. For instance, stable results were obtained with KNN, Adaboost, Random Forest and SVM when simulating with the *proportionality* method.

In addition, it was shown that removing *superfeatures*, and thereby adding noise, within an acceptable margin, contributes to improving the results significantly.

The method performed well when classifying with different algorithms while enlarging a dataset composed of a small number of recordings as well as large datasets. Consequently, the dependency from the size of the original database is reduced.

D. Extending the simulation algorithm

a. Features: Pairwise joint positions

The feature set displayed in III.B.b is not limited and can be extended further, by including the derivative features. Its extension with discriminant features can improve the results noticeably. Table 20 and

Table 21 are the average of 3 different runs of the simulation that show a distinct improvement in the performances for KNN, Adaboost, Random Forest and SVM. As noted in C.f.2, SVM and Random Forest are trained with only the 25 most discriminant features picked by the Adaboost classifier and five other random features.

Table 20. Results with relative joint positions – CAP dataset

	Adaboost	**KNN**	**Random Forest**	**SVM**
Simulated without relative joint positions	0.8533	0.6833[(b)]	0.7626	0.5693
Simulated recordings with relative joint positions (Average of 3 runs)	0.9192	0.6523[(b)]	0.8557	0.7522
Real recordings with relative joint positions	0.8308	0.5410 [(a)]	0.7241	0.6846

a. *K=9 because the set of recordings is small*

b. *K=30*

Table 21. Results with relative joint positions – MSRC-12 dataset

	Adaboost	**KNN**	**Random Forest**	**SVM**
Simulated without relative joint positions	0.9561	0.6257[(b)]	0.8593	0.5179
Simulated recordings with relative joint positions (Average of 3 runs)	0.9614	0.6478 [(b)]	0.8216	0.9506
Real recordings with relative joint positions	0.8382	0.4935 [(a)]	0.5943	0.8868

a. *K=5 because the set of recordings is small*

b. *K=30*

Adding the derivative features improves the performances in most cases. The only drops in the performances are noted with CAP KNN and Random Forest MSRC-12. Nevertheless, the loss is small compared to the improvement in the other results, for example, in all SVM tests.

Since the results have improved in most cases, we will implement the derivative features throughout the remaining of this thesis.

b. 1vs1 Classification

In this chapter, the classification has been done with a 1-vs-all strategy. The simulation algorithm proved to improve the performances when using a large number of classifiers and multiple datasets. Nonetheless, the algorithm in question is also compatible with other strategies such as 1-vs-1. Consequently, Adaboost is trained with a 1-vs-1 strategy and the results of the

different classifications are combined with a simple voting process. These are displayed in Table 22.

Table 22. Classification with a 1-vs-1 strategy

	Captured dataset	**MSRC-12 dataset**	**MSR ACTION3D dataset**
Real recordings	61/74	110/116	35/47
Simulated recordings	66/74	110/116	39/47

c. Other datasets

The simulation algorithm works well not only with the datasets that have been stated above, but has also improved the performances during the classification of the segmented Chalearn [21] dataset.

We must note that the Late Fusion algorithm has not been parameterized especially for the Chalearn dataset, and as observed in Table 23, the performances are not impressive. This is due to the nature of the Chalearn dataset where the same gesture is sometimes performed with different hands. This idea contradicts our definition of an action as in I.C.b.8, where we note that two actions are considered alike if they are performed with the same joints. In fact, Chalearn datasets is very close to the study of Sign Language (II.A.a.1) and we do not take much interest in this thesis in Sign Language, due to the fact that it requires a different definition of an action from the one we gave in I.C.b.8. Nevertheless, the combination of simulated recordings with the real ones, while balancing the training (1-vs-all performs poorly while classifying 20 labels) improves the results, as noted in Table 23 (noticeable improvement compared to training with real actions). However, since the Chalearn dataset is large and contains a lot of variations, a large increase in the size of the simulated dataset can deteriorate the performances (more than 200 recordings). Since the number of training recordings is a parameter that should be defined by the user, we refer to the study that has been done above and we simulate 50 and 100 recordings. We note that the Chalearn real dataset is already large, hence, only 5 recordings are picked arbitrarily from the original dataset (training dataset includes real and simulated recordings), and the rest is used for testing.

Table 23. Results of classification on Chalearn dataset with Adaboost

	Total	**Training with real recordings [b]**	**50 [a]**	**100 [a]**
Positive Result	1313	290	422	516
Negative Result	24947	24391	24085	23571
F-MEASURE		0.2686	0.3250	0.3220

a. *Balanced*

b. *Not balanced*

Even though we consider that the previously chosen parameters should be enough to improve the performances, other tests performed according to the datasets' properties can be in favor of the results. Hence, since the actions are very different in the Chalearn dataset, not increasing the size of the interval is helpful. Experimentations with these parameters are displayed in Table 24.

Table 24. Implementing additional Parameters

		Total	**Training with real recordings [a]**	**Interval without increase**	**Median=7 and no increase in the size of interval**
Not balanced	***Positive Result***	1313	290	415	462
	Negative Result	24947	24391	24247	23884
	F-MEASURE		0.2686	0.3418	0.3256

a. *Not balanced*

Additional changes in the simulation algorithm such as the smoothing of the synthetic coordinates with Kalman and Median filters, and replacing the DTW with the MD-DTW for the alignment have been programmed and tested. Since these implementations did not improve the performances, the results were included in this chapter, the reader can refer to Appendix I: Additional Simulation experimentations, for more information.

V. ASYNCHRONOUS LATE FUSION

A. Summary

This chapter addresses the largest part of our thesis. We propose and explore a novel method: *Asynchronous Late Fusion* that combines a mid-level model with a late fusion classification problem, across temporal sequences. With this approach, classification algorithms' performances are improved by studying binary results accompanied by confidence coefficients from early classifications, and fusing them with a previously trained model, to use it as an input of the late classification. The method is evaluated on multiple datasets and multiple classifiers and is compared with a Late Fusion synchronous application on the state of the art classification algorithms. It is clearly shown that the proposed method can improve the results independently from the classification algorithm and without performing an in-depth study of the features. The solution is applied to the problem of gesture classification.

B. Introduction and definitions

Throughout this chapter, a one-vs.-all strategy is employed with a Late Fusion classification algorithm as in the previous chapters (Chapter IV).

As stated in the summary, the recordings in question in this chapter should have the following properties:

- Temporal Sequences: the recordings should contain sequences of values defined along the time.
- Late Fusion properties: a recording that can be classified with a Late Fusion method usually contains multiple classifiers at the lower level. Hence, the different classification decisions from the sequences are combined, and the final decision is inferred at the late level. (We remind that the classifiers of the lower level sequences are called early classifiers or lower-level classifiers).

The classification of temporal sequences is still an unsolved problem, especially, when it comes to recognizing datasets with what will be described as "asynchronous late fusion properties" in the next paragraph. The decisions are generated from multiple sources and are taken at different instances of time. The concept of the Asynchronous Late Fusion (ALF) will be detailed further in the next paragraph.

a. Definition of the Asynchronous Late Fusion (ALF)

We consider multiple classes containing temporal recordings to classify. A recording contains sub-recordings that are considered more or less relevant to finding the ground truth. Examples of the sub-recordings have been mentioned in previous studies as small units of an action (II.G.a) and can indicate simple gestures.

A global feature calculated on the whole vector will be less discriminant than a feature that is extracted at the exact location where the event occurs. For example, a simple feature like the average, calculated on all time frames will be less relevant than when computed at the sub-

recording where the real change in the coordinates' values occurs. As a result, a feature vector calculated from the entire recording (e.g., average, standard deviation, minimum, maximum… are considered as features), is less discriminant than one that is calculated on windows that are extracted from the recording (which contains discriminant sub-recordings). Consequently, we choose to work with small temporal windows to reflect the local variations.

The aim is to determine the class of the recording by analyzing information extracted from the windows. We note the following during our study:

- If a window contains a sub-recording that is relevant to the class of the recording, the ground truth can be easily detected.
- The databases are annotated by class/recording, but not segmented into sub-recordings, since as explained in the related work (II.G.a), it was difficult to define the sub-units of an action properly. Hence, we choose to work with windows of fixed length. These windows will be called parts.
- We perform a 1-vs.-all classification on every part of the recording, in other words, we aim to classify a part against the "world".
- When implementing a classic classification architecture (will be called synchronous architecture or synchronous method), defining the class of the recordings is only reliable when the discriminants sub-recordings are studied.

Figure 22 compares the feature extraction methods between the synchronous and the ALF methods on a sequence. It is clear that by classifying the parts separately, the ALF method will be able to locate the discriminant sub-recordings.

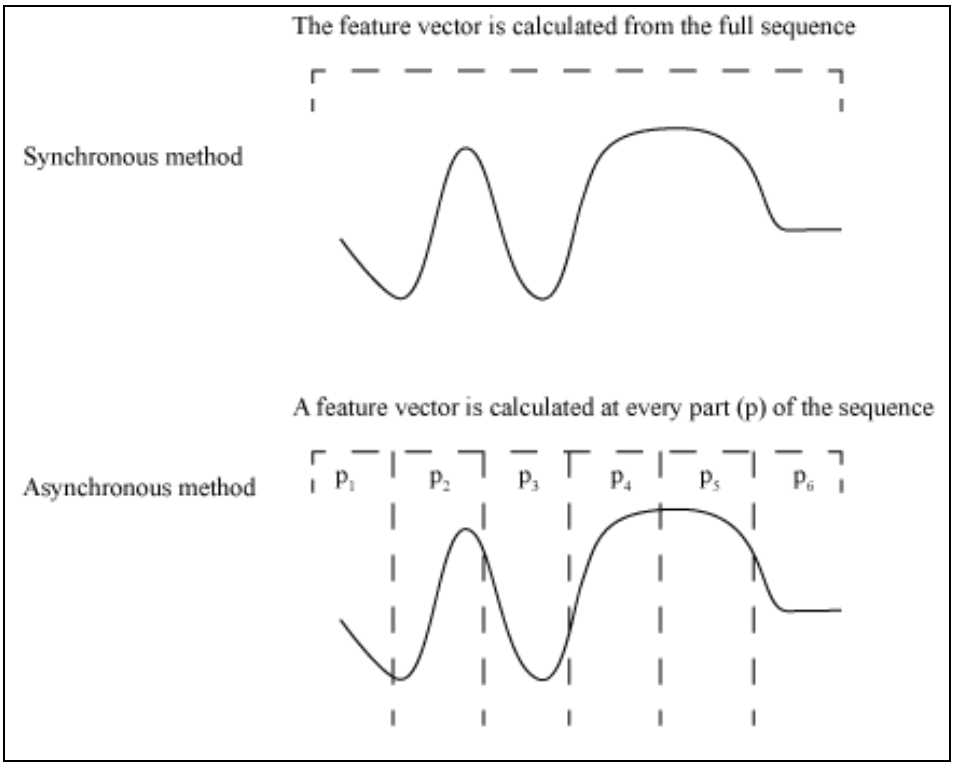


Figure 22. Extraction of the feature vector with the synchronous and asynchronous methods.

Considering a multi-dimensional signal, the discriminant sub-recordings are not found at the same instants in all dimensions (not necessarily synchronous). Hence, classifying a single part at each dimension does not output an optimal decision, because the discriminant sub-recordings are not necessarily available in every part. Moreover, all the dimensions might not be relevant in taking the final decision.

Using a classifier at every part allows the classification of the dimensions and outputs positive decisions at different time instants. Consequently, it is imperative to fuse the decisions from the parts and the dimensions to infer a final decision. The process is implemented as follows: the decisions from the parts are fused by performing the scalar product between the decision at every part and a model composed of sequential weights, called the ALF model. Afterward, the results of the scalar product, from every dimension, is inputted into a final late classifier to take the final decision.

This whole procedure will be referred to as the Asynchronous Late Fusion (ALF), as opposed to the synchronous fusion where the decision will be taken on the entire recording, at all dimensions simultaneously.

b. Dataset with ALF properties

We will refer to a dataset with asynchronous properties when the recordings contain discriminant sub-recordings that are found at different time instants. Generally, all the datasets contain asynchronous properties. Nonetheless, the asynchronous properties are graduated: a dataset is described as more or less asynchronous than another.

An example of a perfect situation where a dataset is described is highly asynchronous is when recordings belonging to two different classes have exactly the same length and the same values but shifted in time compared to the recordings in the other classes.

c. Asynchronous Late Fusion (ALF) approach

The approach aims to improve the classification of datasets that are highly asynchronous and at the same time maintain, or even improve, the results of datasets with lesser asynchronous properties.

Whatever the features are, or the classifier, the aim is to improve the classification. Hence, the classifier is considered as a “black box” and our work focuses only on its output.

After cutting the recording into sub-recordings along the temporal dimension, every part is used to train an early classifier separately. A classifier’s decision might not be trusted at a certain part, and its decision might not be as important as one taken at another part. Performance equations will be applied to the binary output of the early classifiers to build the ALF model. As a result, the ALF model is composed of values that represent the weight of the parts. Afterward, each lower level decision is combined with a weight from the ALF model. Hence, it modifies

the decision of the early classifier according to the weight that was previously calculated at every part.

The architecture of the asynchronous late fusion model is resumed in Figure 23. To describe it properly, we consider as an example two temporal sequences belonging to the same recording (e.g., two joints in action; HandLeft and HandRight). The two sequences are cut into parts (in this case, each sequence is cut into 2 parts only), converted into features, then every part is classified with an early classifier: the "black box". The result of the early classifier is processed with the asynchronous model (the model is composed of 2 values, equal to the number of parts). Afterward, the decision from every sequence is inputted into the Late Fusion classifier, which takes the final decision.

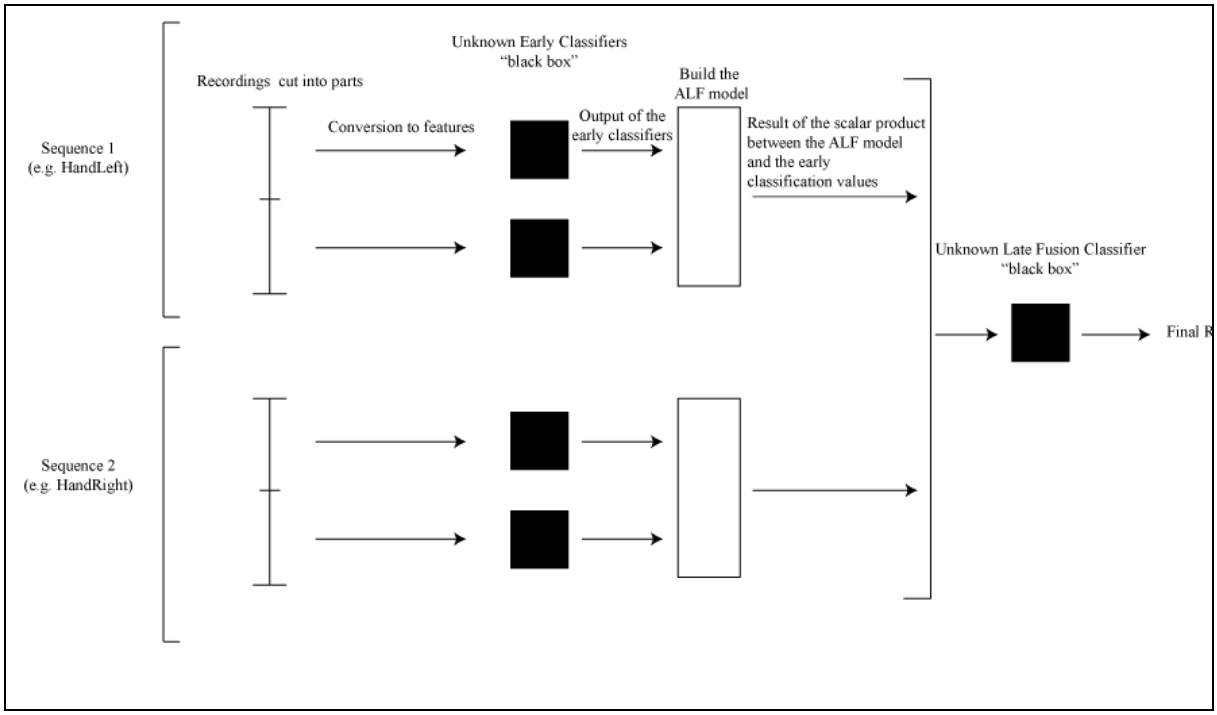


Figure 23. Asynchronous architecture

C. Chapter Outline

To show the benefits of our contribution and the purpose of the asynchronous model, we consider the following steps:

- Analyze the behavior of the early classifiers.
- State the ALF algorithm and the equations that build the ALF model.
- Study the different parameters of an asynchronous model.
- Experiment and analyze the results of classifications with multiple algorithms while variating the values of the parameters.
- Study the conditions and limits of the ALF.
- Compare the ALF with another temporal algorithm: Hidden Markov Models (HMM).
- Extend the ALF.

We test the ALF with the Adaboost, SVM, and KNN classifiers.

D. Method

a. ALF model

In this part, the following abbreviations will be used: the number of True Positive results as **TP**, True Negatives as **TN**, False Positives as **FP**, False Negatives as **FN**, the total number of positive recordings as **P**, and the total number of negative recordings as **N**.

The objective is to build the ALF model during the training phase by combining the values that are outputted at every part by an early classifier while performing tests on a training dataset. The ALF model is a vector of values expressing the reliability of the decision taken by the classifier at a time. In this vector, a high value means that the decision can be trusted at the corresponding instant of time. To generate those vectors, there are multiple ways of handling this; it is possible to propose a metric. Nevertheless, it might not work well with all cases and exceptions. Hence, we choose to work with well-known metrics, in addition to others that are extended from them, to build the proper ALF model. To this end, the following hypotheses have helped to find the most appropriate metrics to apply onto the result of early classifications:

a. The metrics should be equal to 0 when TP=0 or TN=0, to attribute the same importance to every class.

b. The result of the metrics should be normalized to a resulting value ranging between 0 & 1.

c. The metrics should be independent of the early classifier; they should not integrate any parameters concerning the early classifier. For example, they should not depend on the number of iterations of the Adaboost, or the complexity of the algorithm, because, as stated previously, in V.B.c, we consider the classifier as a "black box." Therefore, the only known information about the "black box," should be its output. Once again, we emphasize on the fact that the metrics should be generic.

d. The metrics should be symmetric i.e. they should give the same result when exchanging the P and N sets.

e. The metrics must be compatible with imbalanced datasets ($T/P>1$ or $P/T>1$). This study faces this issue since it implements a 1-vs.-all strategy.

Consequently, the binary values (accompanied by confidence coefficients or not), resulting from the decision taken on every part in the sequences, can be used to build the ALF model with the metrics described in Table 25.

Table 25 includes the most common metrics such as the recall and the precision, in addition to their variations such as the specificity, which is also known as the negative recall, and the negative precision value. We state some other metrics: the combination of the recall and the precision and their variations, such as the multiplication of both. The well-known Accuracy and F-Measure metrics are also included in our study. We researched some other less popular ones such as Matthews Correlation Coefficient and Youden's Index.

The Matthews Correlation Coefficient (MCC), like the other performance measures, is used for measuring classifiers' performances while summarizing the confusion matrix into a single value. According to [158], MCC is a reference for measuring performances on imbalanced

datasets. When the MCC value is equal to 0, the decision is random; when the value is negative, the measure indicates a perfect misclassification, and finally MCC=1 indicates a perfect classification [159]. To fit the property *b* (the metric should give a value ranging between 0 and 1) and as a negative result from the MCC reveals the absence of correlation, in this case, we set its value to 0.

Youden's Index [160] combines the specificity (recall) and the sensitivity; it is also known for its application on imbalanced datasets.

Table 25. Metrics for building the models

Symbol	Metric
R	$\boldsymbol{recall = \frac{TP}{TP + FN}}$
R-	$\boldsymbol{specifity = \frac{TN}{TN + FP}}$
P	$\boldsymbol{precision = \frac{TP}{TP + FP}}$
P-	$\boldsymbol{negative\ precision\ value = \frac{TN}{TN + FN}}$
RR	$\boldsymbol{recall \times spceficity = \frac{TP}{TP + FN} \times \frac{TN}{TN + FP}}$
PP	$\boldsymbol{precision \times negative\ predictive\ value = \frac{TP}{TP + FP} \times \frac{TN}{TN + FN}}$
A	$\boldsymbol{accuracy = \frac{(TP + TN)}{TP + TN + FP + FN}}$
Y	$\boldsymbol{Youden's\ Index = recall + specificity - 1}$
HALF	$\boldsymbol{half\ total\ error\ rate = 0.5 \times (\frac{TP}{TP + FN} + \frac{TN}{TN + FP})}$
MCC	$\boldsymbol{MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}}$ When negative => result = 0, the system does not give a
F-Measure	$\boldsymbol{F - Measure = 2 \times \frac{precision \times recall}{precision + recall}}$

The counter-examples below (in Table 26 & Table 27) help to verify the properties mentioned above.

An example of a classification of 10 recordings is considered (5 positive recordings and 5 negative recordings), and the properties *a*, *d* and *e* are applied to it. This number of recordings has been chosen arbitrarily. It is sufficient to illustrate major drawbacks of those metrics to reach a final decision. Example *e* is an exception that should be taken into consideration since few metrics are compatible with such scenarios.

The metrics are applied to the confusion matrix, and the results are displayed on the right of Table 26 and Table 27 to analyze their behavior.

We explain the reason of failure or give additional information in the 3rd column entitled: "Explanation."

Table 26. Examples of the proposed properties to find the best metric (The c*ells in gray represent the metrics that verify the property*)

Property	Example				Explanation	R	R-	P	P-	RR	PP
	TP	TN	FP	FN							
A	0	5	0	5		0	1	0	0.5	0	0
	0	4	1	5		0	0.8	0	0.44	0	0
	5	0	5	0		1	0	0.5	0	0	0
D	1	6	1	2	*P* gives more importance to the negative than the positive	0.33	0.86	0.5	0.75	0.29	0.38
	6	1	2	1	Nevertheless, we gray out the examples that do not verify the symmetrical property.	0.86	0.33	0.75	0.5	0.29	0.38
E	1	10000000	0	4		0.2	1	1	1	0.2	1
	1	5	0	4		0.2	1	1	0.56	0.2	0.56

Table 27. Examples of the proposed properties to find the best metric (The c*ells in gray represent the metrics that verify the property*)

Property	Example				A	Y	HALF	MCC	F-Measure
	TP	TN	FP	FN					
a	0	5	0	5	0.5	0	0.5	0	0
	0	4	1	5	0.4	0	0.4	0	0
	5	0	5	0	0.5	0	0.5	0	0.67
d	1	6	1	2	0.7	0.19	0.6	0.22	0.4
	6	1	2	1	0.7	0.19	0.6	0.22	0.8
e	1	10000000	0	4	1	0.2	0.6	0.45	0.33
	1	5	0	4	0.6	0.2	0.6	0.33	0.33

Since we do not have much information on how high or low the value should be in the examples proposed in property *e*, we propose the following to analyze the metrics:

- *Example e: w*e disregard all results that are approximately equal to 1 (which is considered as the maximum value according to property *b* since it does not take into consideration the class with the smallest number of recordings.

When we compare all the selected metrics for modeling the ALF model's weights of an asynchronous dataset considering a 1-vs-all classification in respect with this set of properties, we can determine that the best solution is the MCC.

b. Building the ALF model

To build the ALF method we train the lower level classifiers, then build the ALF model by applying metrics on the decisions from the early classifiers, afterward, combine the ALF model with the early decisions and then finally train the final late classifier. This whole procedure will be detailed below.

The process of building the ALF model requires three different sets recordings. One for training the early classifiers, another for building the models and a third for training the late fusion classifier.

1. *Generating the features*

Before training the early classifiers, as in every classification process, every sequence of a recording is cut into a certain number of parts. In the following example, we cut the sequences into two parts. Afterward, each part is converted into a feature vector. We remind that the parts are windows of fixed length, extracted from a recording. They represent the sub-units of the recordings. The number of parts is a parameter that will be discussed later on in this chapter, in V.D.d.1.3.1.

2. *Training of the early classifiers*

Every early classifier is trained with a different sequence of a recording as a normal late fusion. The classifiers are trained with the first set of recordings. Figure 24 displays an example of a single sequence that is cut into two parts and every part is trained with a classifier at the lower level called early classifier.

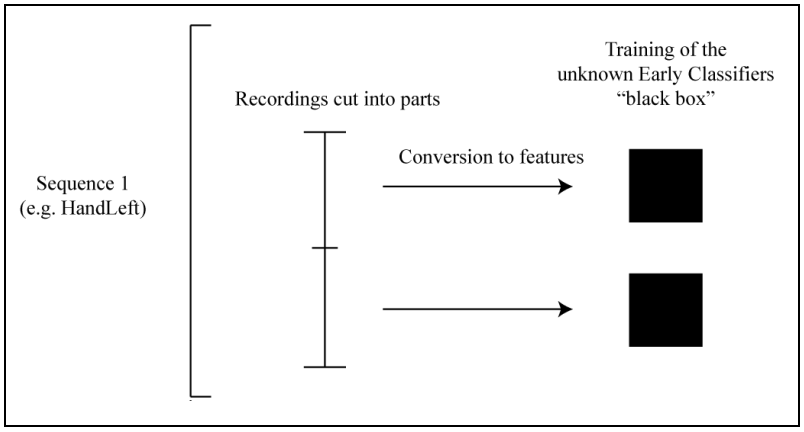


Figure 24. Lower level of the ALF solution

3. *Initialization of the ALF model*

Initializing the ALF model consists of classifying the second set of recordings with the method described in the previous steps, then applying the metrics, that have been mentioned above V.D.a, to the decision of the output to generate weights.

The ALF model is a vector of weights that is built with the following method: set the array of the ALF model to 0 with a length equal to the number of parts *P*, then run Algorithm 3.

1 For each part p=1,…,P
2 $Model[p] = Result\ of\ the\ metrics\ on\ p$
3 End for

Algorithm 3. Initializing the asynchronous model.

Figure 25 illustrates the initialization process.

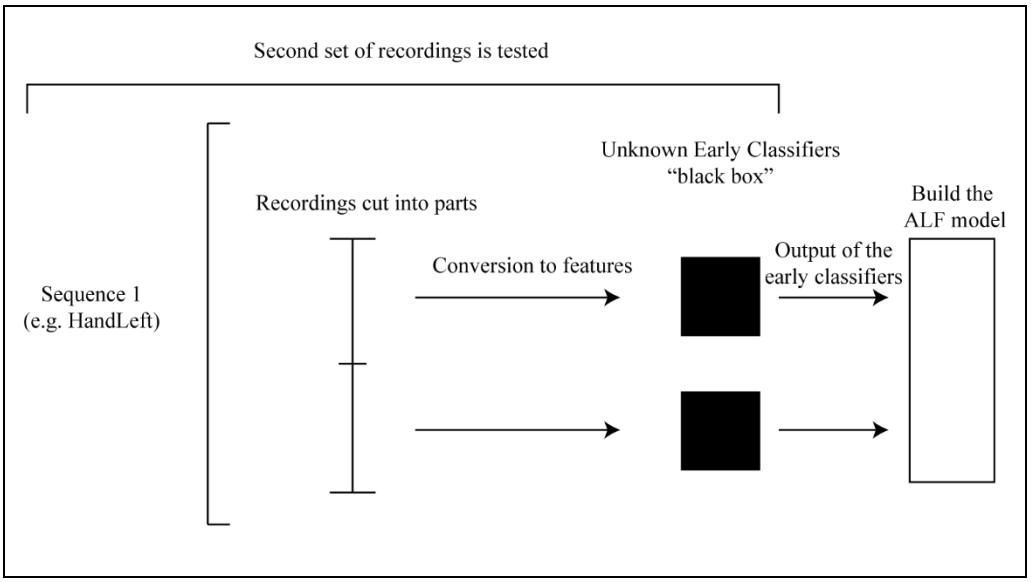


Figure 25. Initialization of the ALF model

4. *Buffering*

The buffering part fuses the model with the results from the early classifiers by performing the scalar product between the decision values outputted from the early classification and the resulting model from V.D.b.1.

Let n=1,…,N be the ALF models' index, where each model is generated from a sequence.

$$input\ of\ late\ classifier_n = \sum_{p=1}^{P} Model_n[p] \times decision_n[p] \qquad (28)$$

5. *Training of the late fusion classifier*

To train the final classifier, the decisions when classifying the third set of recordings are buffered with the ALF model and the result of the buffering is used as an input to a single classifier. Figure 26 illustrates the global process of training the late classifier.

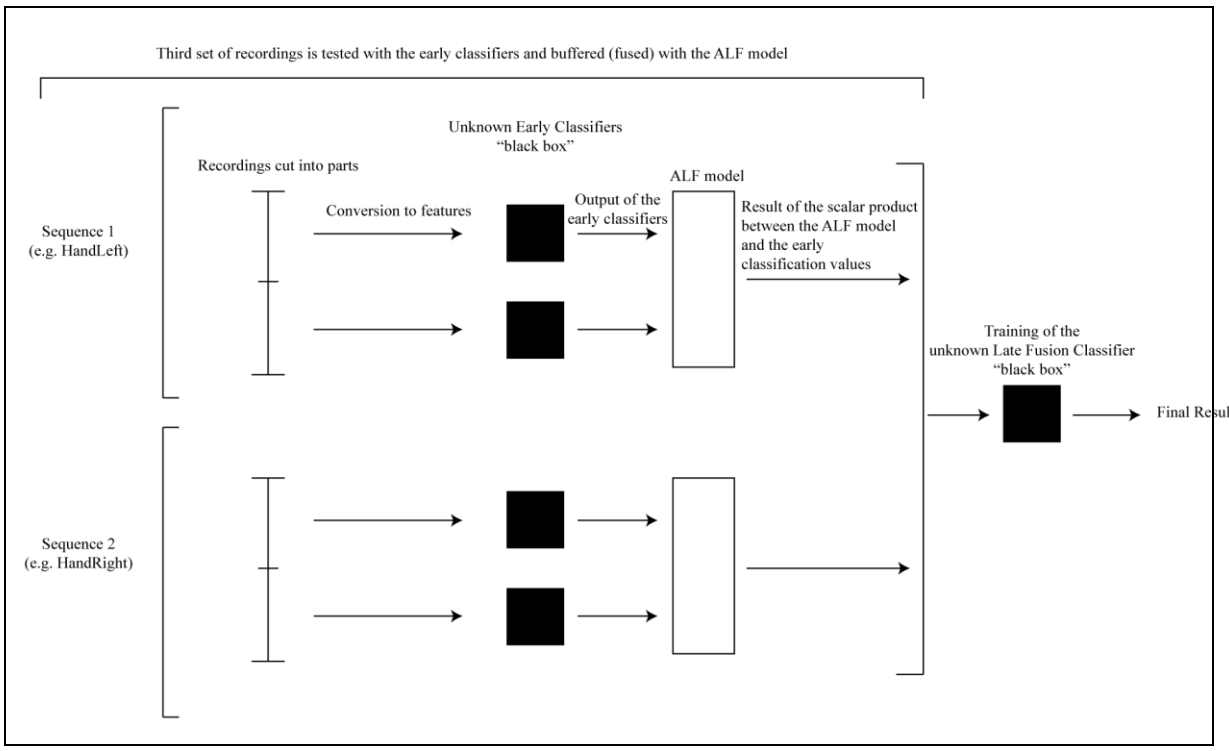


Figure 26. Training of the late fusion classifier (ALF solution)

6. *Testing*

To classify recordings, a process similar to the training of the late fusion classifier is conducted. Nevertheless, the recordings are tested with the late fusion classifier.

c. Proof of concept

In this part of the chapter, we aim to show the benefit of implementing the ALF solution by analyzing early classifiers' decisions of real life examples.

1. *Analysis of the output of the early classifiers*

It is possible to imagine a lot of action datasets with ALF properties. However, since action recognition is a recent subject, finding ready to use datasets and studies on these special cases is difficult (IV). Therefore, we take interest in that subject and apply the ALF to it.

In this chapter, as well as in the previous one, we input feature vectors into the classification algorithm, and output a binary decision, weighted by a confidence coefficient, if available. The ALF model, composed of a sequence of weights, adjusts the output of the early classifiers to improve the decision of the late classifiers. We note that we do not work on the action segmentation problem; the recordings that we use for classification are already segmented manually.

To analyze the behavior of the early classifiers, recordings of a first dataset are internally segmented using interlaced parts to train the early classifiers. A second dataset is tested with the early classifiers and the resulting values are called the mid-level decisions. The analysis of the early classifiers' decisions shows the purpose of introducing an asynchronous model. In the following experiments, tests are conducted with a Late Fusion Adaboost classification.

We study below the classification of the swimming and soccer datasets (Table 5).

In Figure 27, we display the real output of the Adaboost, along the time, of the classification of swimming butterfly Shoulder Left joint with its classifier (1-vs.-all swimming butterfly classifier) and swimming crawl Shoulder Left joint with its classifier too. The same classification is performed with the soccer and not soccer right knee classifiers, and the real output is displayed in Figure 28. Another example where the Shoulder Left and the Shoulder Right are classified with the swimming crawl classifiers is shown in Figure 29. The size of the window for cutting the recordings internally was fixed arbitrarily and is equal to 5 frames. The chosen size allows us to observe the behavior of the decisions even though this is not an appropriate size for running a proper classification. Even though the recordings do not have the same length, we combine them in the same figure for presentation purposes. The objective of these figures is to show that the decisions are taken at different time instants when comparing the joints of the same action as well as different actions.

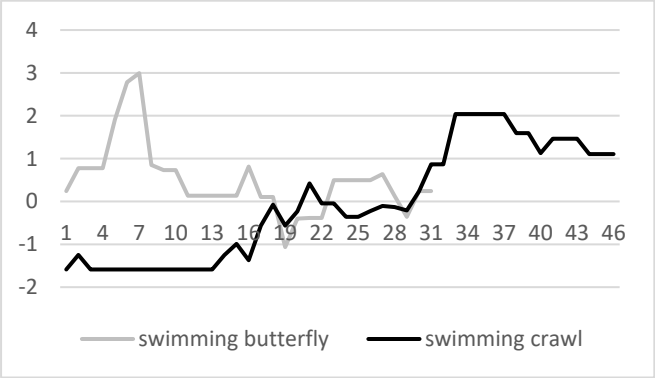


Figure 27. Real output of Shoulder Left of swimming butterfly and swimming crawl. The scale of the horizontal axis is the frame number, and the vertical axis is the value of the real output of the Adaboost.

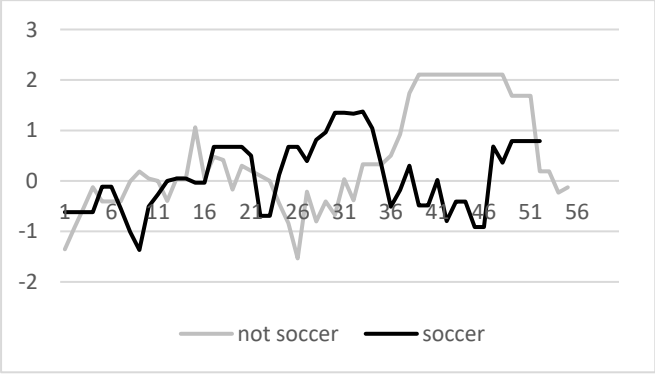


Figure 28. Real output of Knee Right of soccer and not soccer. The scale of the horizontal axis is the frame number, and the vertical axis is the value of the real output of the Adaboost.

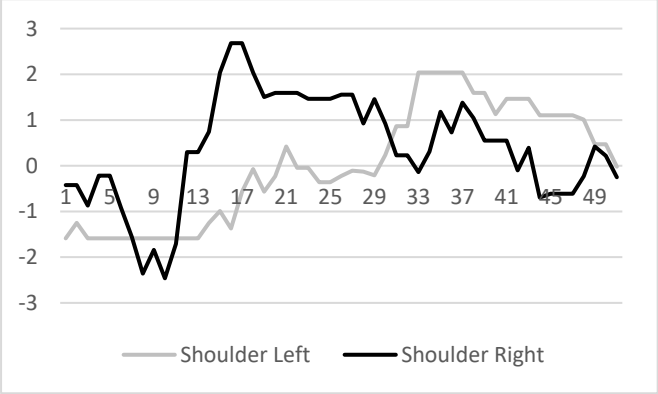


Figure 29. Real output of Shoulder Left and Shoulder Right joints, swimming crawl. The scale of the horizontal axis is the frame number, and the vertical axis is the value of the real output of the Adaboost.

In Figure 27, the classifier outputs a high value at the end of the recording during the classification of swimming crawl (at frame 33), as opposed to swimming butterfly that outputs only once a high value at frame 7. The same behavior can be observed in Figure 28 where the decision of the soccer action is taken before the "not soccer" action. Consequently, the decision is taken at different instants when comparing different actions.

We observe in Figure 29 that the swimming crawl is detected at various time instants when comparing the real output of the classification of the Shoulder Left and Shoulder Right joints. The Shoulder Left is detected with a high value of the early classifier's output at frame 16 and the Shoulder Right at frame 33. As for the remaining of the decisions, the decisions cannot be trusted where the values are approximately equal to 0 (for example, between frame 1 and frame 5).

Consequently, we deduce that the study of the output of the early classifiers at different time instants as well the combination of the decision that is taken at the different early classifiers (the late fusion) is a discriminant element for the classification.

2. *Synthetic mid-level decisions*

In this part, our objective is to analyze the decisions of the early classifiers in details and the behavior of the ALF as well as the metrics. Hence, to perform a more precise analysis while adding special variations to the recordings, we simulate synthetic recordings instead of using real life information.

When comparing the sequences in Figure 27 to Figure 29, we observe that the decisions are taken at different time instants between two actions. Consequently, we generate the synthetic decisions using multiple distributions functions that have a maximum at different time instants.

The simulation algorithm is established as follows:

i. Pseudo Code:

```
L: length of the recording.
G_l = P(TP) / l=1,…,L is the frame number (the Gaussian functions, as well as others, are considered as the probability of obtaining a True Positive result as output of the early classifier)
For each decision of the early classifier D at l=1,..,L
        R = random value between 0 and 1
        If R > G_l Then
                D_l = −R
        Else
                D_l = R
        End if
End For
```

Algorithm 4. Generating the synthetic confidence coefficients

In other words, if a randomly chosen point is located over the model, the simulated decision is negative, otherwise, if it is under the model, then it is positive, and it is equal to the random value. This algorithm results in generating more positive values in case G is above 0.5, or less positive values, if it is below 0.5.

Experimentations

To analyze the decisions of the early classifiers in details and the behavior of the ALF as well as the metrics, a dataset composed of multiple recordings belonging to 3 classes is used to train early classifiers. When testing a sequence, each early classifier outputs different positive results at different instants of time. To perform a 1-vs.-all classification, different sequences of values are simulated to train and test every 1-vs.all classifier. The recordings belonging to class A should be classified with classifiers trained to detect classes A, B & C. Consequently, distribution functions, which will be called D, are used to simulate the output of the early classifiers.

The color of the functions D denotes the early classifiers. The functions D are used to generate an output of the early classifiers, for every class.

e.g., decisions from class A:

black: EarlyClassifier_1

grey: EarlyClassifier_2

dashed: EarlyClassifier_3

Some of the models L are displayed in Figure 30.

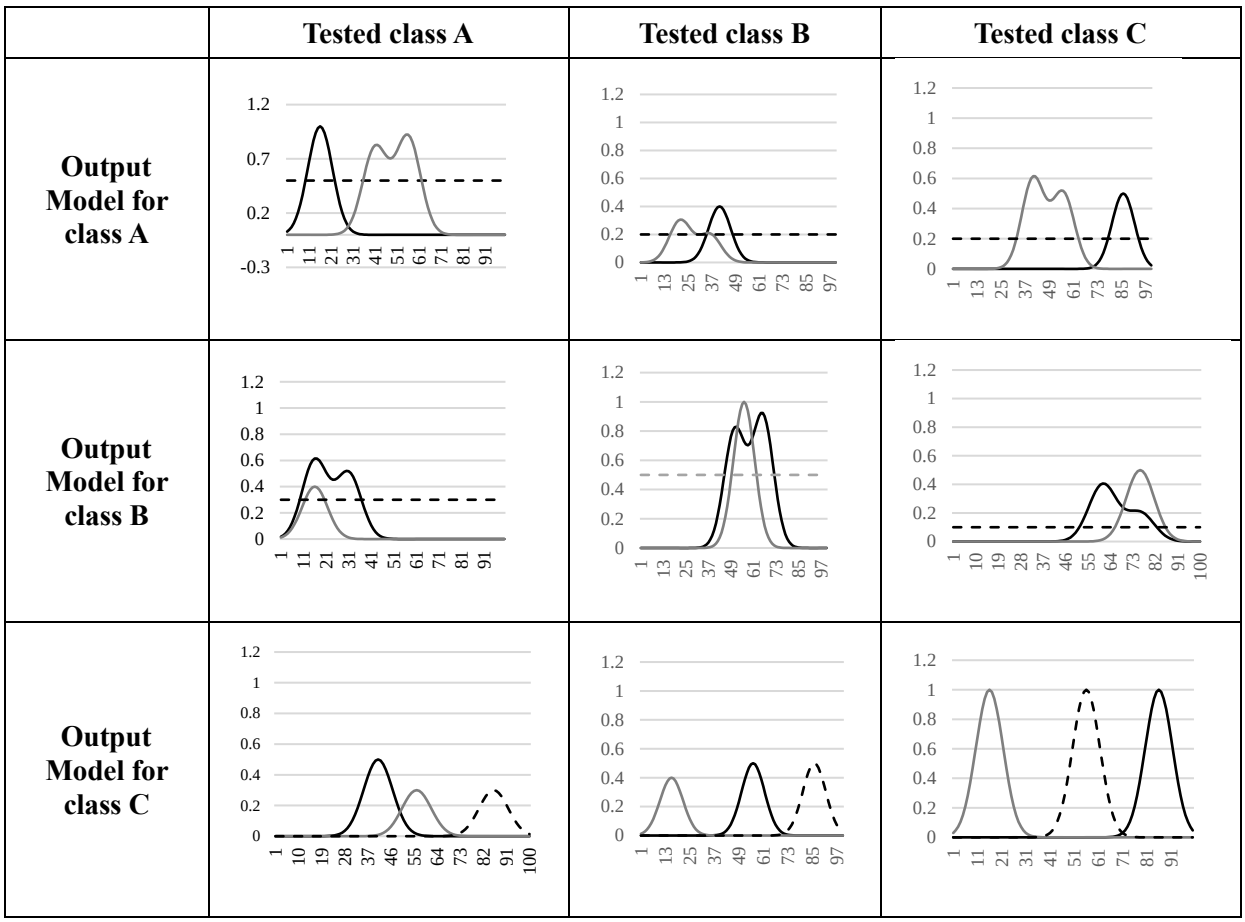


Figure 30. Gaussian function to generate the synthetic decisions. The scale of the horizontal axis is the frame number, and the vertical axis is the value of the models.

For example, the first cell on the left part represents the models used to simulate the decisions of the 3 early classifiers trained to recognize class A and tested on a recording belonging to class A. The first classifier will have a Gaussian-like behavior, the second multi-Gaussian and the third randomly generated values using a uniform distribution law.

Using the synthetic early level decisions, we train and test the ALF with an Adaboost and a KNN algorithm (considered as the unknown classifiers or "black boxes"). The model is built from the generated decisions; then the ALF model is applied, and the decisions are taken with the classifier. The results shown in Table 28 and Table 29 are obtained with models built with the metrics that have been stated in Table 26 & Table 27, and a model that is always equal to 1 (setting the model to 1 eliminates the metric, this is equivalent to the removal of the effect of the ALF). The additional metrics, which have been stated above, are derived from them and will have a similar behavior.

To analyze the metrics properly, we generate multiple datasets, which are described below and experimented on in Table 28 and Table 29. In general, the purpose of all the datasets is to produce confusion with a change of the parameters of D to deduce the most appropriate metric.

a. The sequences are generated with negative values. The functions D, which generate the sequences, are displayed in Figure 30. The purpose of this dataset is to show that the ALF model works perfectly when the decisions are taken at separate positions in time.

As noted previously, this dataset is a perfect Asynchronous situation where the decisions are taken at different time instants. The remaining of the datasets is extracted from (a) by adding certain parameters.

b. The sequences are approximately the same as in (a), with the only difference where the decisions are taken at closer time instants. This generates additional confusion and causes more misclassifications. Moreover, the sequences are cut at random positions to generate final sequences that vary between 10 and 100 frames. With this dataset, we analyze the behavior of the metrics when increasing the diversity of the decisions.
c. Same as (a), but the width of the Gaussian inspired functions is increased to also increase the confusion between the decisions.
d. Same as (a), with a change of time warping (Sequence's length are multiplied by a random number between 0.7 and 1.3). Nonetheless, the results should not differ since a small confusion is only added and the result of the scalar product while running the ALF is approximately the same as with (a).
e. The width of the functions that generate (b) is modified randomly. This results in additional confusion. The results are expected to degrade.
f. The dynamicity of the sequences in (b) is changed in this case to increase the confusion and try to degrade the results.

In Table 28 and Table 29, the best results from the classifications are highlighted in gray. The values are computed with the F-Measure.

Table 28. Results of the classification with simulated early classifier output (Classifier: Adaboost)

	Model=1	***R***	***R-***	***P***	***P-***	***PP***	***RR***	***MCC***	***A***	***Y***	***HALF***	***F-Measure***
a	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.980	1.000	1.000	1.000
b	0.2766	0.802	0.680	0.827	0.749	0.864	0.864	0.955	0.417	0.908	0.353	0.756
c	0.9867	1.000	0.997	0.993	1.000	1.000	1.000	1.000	0.976	0.979	0.990	1.000
d	0.9867	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.924	1.000	1.000
e	0.2654	0.844	0.777	0.877	0.804	0.877	0.877	0.953	0.521	0.897	0.389	0.792
f	0.3075	0.992	0.893	0.992	0.935	0.991	0.991	0.995	0.992	0.995	0.586	0.931

Table 29. Results of the classification with simulated early classifier output (Classifier: KNN)

	Model=1	*R*	*R-*	*P*	*P-*	*PP*	*RR*	*MCC*	*A*	*Y*	*HALF*	*F-Measure*
a	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.980	1.000	1.000	1.000
b	0.276	0.802	0.643	0.731	0.718	0.827	0.864	0.955	0.954	0.878	0.378	0.692
c	0.986	1.000	0.983	1.000	0.929	0.993	1.000	1.000	0.976	0.929	0.995	0.990
d	0.986	1.000	0.976	1.000	0.925	1.000	1.000	1.000	1.000	0.924	1.000	1.000
e	0.265	0.844	0.745	0.790	0.747	0.877	0.877	0.953	0.521	0.753	0.288	0.752
f	0.307	0.992	0.888	0.992	0.919	0.992	0.991	0.995	0.992	0.995	0.603	0.899

In Table 28 and Table 29, we note that most of the metrics perform well with the datasets: a and d. In fact, it contains the least diversity between the recordings: small or no changes between the simulated sequences. As for the rest of dataset, when increasing the complexity or the diversity of the simulated decisions, few metrics still perform as excepted.

With both Adaboost and KNN, only the MCC column is entirely in gray. As a result, we clearly see that the MCC metric overcomes all the other ones on simulated data. Those observations have to be consolidated by some experiments on actual datasets.

d. Experimentations

To show the advantages of using the asynchronous algorithm, we conduct the following tests on multiple classification problems.

1. *Action classification*

1.1. Datasets

During our experimentations in this section, we adopt datasets that are more or less compatible with the ALF solution than others, described previously (V.B.b). In fact, the movement in the actions from the SS dataset (Table 5) does not happen at the same time; when performing Swimming Butterfly, the subject moves both hands at the same time during the full length of the action, as opposed to Swimming Crawl where the subject moves his right hand and the left one afterwards. A similar situation is observed when performing the soccer and not soccer (throw the ball and then shoot it and shoot and throw afterward). Consequently, a highly asynchronous property is observed compared to the CAP dataset (Table 3) where most of the actions happen at the same time and the CR dataset (Table 4) that contains lots of confusion between the actions. In this case, the SS dataset is said to be more asynchronous than the others. Finally, we experiment with the RL dataset (Table 6) where the asynchronous properties are obvious. As a matter of fact, all subjects are performed in a sitting position, only the right and left hand are moved up and down once in every recording. Hence, the positions of the joints in the recordings are noise-free, and the early classifiers take the decisions at different time instants without additional confusion between the decisions.

In this chapter, since the main focus is on the ALF method and comparing it with the classic Synchronous method, independently from the size of the dataset or the classification algorithm, it is necessary to get rid of all extensions and additions to algorithms or datasets to study the behavior of our method properly. Hence, the training datasets are considered large enough for training. As a result, the training datasets are split into four groups of equal sizes. The first group is used to train the early level (Figure 24), the second for building the ALF model (Figure 25) and the third to train the final classifier (Figure 26). Finally, the benchmarks are computed with the four groups.

1.2. Synchronous classification (Figure 7)

To show the gain of implementing the ALF, we compare the results with the synchronous solution. The main datasets that are experimented on in this thesis (CAP, CR, SS) are classified with an Adaboost and a KNN, and the results are displayed in Table 30. The values that should be remembered for comparison are the F-Measure, which is displayed in the last column of the table.

The scores in Table 30 are computed using a one-vs-all classification with Adaboost and KNN. The parameters of Adaboost are the same as the ones that were set in the first chapter: 50 iterations on both the early and late classifiers. Regarding the KNN parameters, K has been arbitrary fixed (K=9 for training with real recordings and K=30 when simulating with 50 recordings, same as in IV.C.f).

Table 30. Scores with a synchronous solution

	Type	**TP**	**P**	**TN**	**N**	**F-Measure**
Adaboost CAP[(a)]	Real	37	49	291	294	0.8315
	Simulated	40	49	290	294	0.8602
Adaboost SS[(c)]	Real	25	39	104	117	0.6494
	Simulated	27	39	110	117	0.7397
Adaboost CR[(b)]	Real	41	51	247	255	0.8200
	Simulated	40	51	254	255	0.8696
KNN CAP[(a)]	Real	25	49	291	294	0.6494
	Simulated (K=30)	29	49	288	294	0.6905
KNN SS[(c)]	Real	26	39	107	117	0.6933
	Simulated (K=30)	28	39	109	117	0.7467
KNN CR[(b)]	Real	26	51	244	255	0.5909
	Simulated (K=30)	28	51	241	255	0.6022

a. *CAP refers the Captured simple dataset (Table 3)*

b. *CR references the datasets containing right-hand wave confusion (Table 4)*

c. *SS refers the Swimming & Soccer dataset (Table 5)*

1.3. Asynchronous classification

To build the frameworks of our experiments, we identify two types of parameters for the asynchronous model:

- The number of parts: to generate the model, the recordings should be cut into an equal number of parts.
- The overlap: since the recordings are cut into parts, an overlap between the parts might be efficient to add redundant information between the parts.

In a first step, we want to evaluate in which way these parameters have an impact on the results.

1.3.1. Impact of the number of parts on the Asynchronous and the Synchronous classification

As explained previously, the temporal model is built using decisions calculated through time. Those decisions are computed by cutting the recording into parts of equal size. Nevertheless, choosing the right number of parts is a problem.

As stated before, since we do not consider the real output value from the Adaboost as a trusted confidence coefficient, the scalar product is the combination of the binary output of the classifier on each part and the weight attributed to this part, by the ALF model. We test a different approach with KNN, by multiplying the binary output of the early classifiers with the number of votes.

Below is a description of Table 31 to Table 34, Table 36, Table 41 & Table 42.

- The grayed out cells across the number of parts point out the cases where the synchronous performances have been improved. For example, according to the results of a synchronous classification, in Table 30, the F-Measure when testing with Adaboost, CAP dataset, is equal to 0.8315, which is lower than any of the values in the grayed out cells in Table 31.
- The grayed out cells with row title average, maximum, minimum and standard deviation point out at the results that fall within a range smaller than 0.02 from the maximum value of the row. (The 0.02 is chosen arbitrarily to exclude the metric that can fall the farthest from the highest value and to show that the results are very close with the different metric)
- We give the results only for a small number of a restricted number of parts, to summarize the results.

An additional experiment is conducted to check whether the number of parts is consistent when classifying recordings from datasets that have been captured in a different situation than the one where the recordings have been captured initially; change of location of the camera, change of person who is performing the action and change of the camera. Hence, an additional set of recordings belonging to the CAP dataset is captured and classified with the same classifier

that is trained to provide the benchmarks for comparing the metrics. The results are displayed in the last column of Table 31, where it is noted that the MCC metric is used to build the ALF model, and the results are compared to the synchronous benchmark in the last row of the table.

Table 31. F-Measure results with different metrics (Adaboost) – CAP datset

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1	MCC (Additional Dataset)
2	0.7368	0.7447	0.7368	0.7447	0.7368	0.7368	0.7447	0.7368	0.7368	0.7097	0.8242
4	0.8736	0.8605	0.8506	0.8235	0.8409	0.8095	0.8605	0.8095	0.8736	0.7778	0.8776
6	0.8046	0.7907	0.8140	0.7907	0.8372	0.8000	0.8046	0.7955	0.8095	0.6863	0.7857
8	0.8132	0.8315	0.8090	0.8132	0.8222	0.8222	0.8315	0.8444	0.8132	0.8182	0.7381
10	0.8421	0.8333	0.8454	0.8333	0.8454	0.7835	0.8333	0.7917	0.8333	0.8000	0.8370
12	0.7551	0.7551	0.7551	0.7475	0.7475	0.7629	0.7551	0.7708	0.7347	0.7579	0.8261
14	0.8261	0.8352	0.8298	0.8261	0.8211	0.8478	0.8352	0.8387	0.8261	0.8478	0.7978
16	0.8119	0.8387	0.8155	0.8571	0.8000	0.7677	0.8298	0.8125	0.8081	0.8172	0.8085
Average	0.8079	0.8112	0.8070	0.8045	0.8064	0.7913	0.8118	0.8000	0.8044	0.7769	0.8119
Maximum	0.8736	0.8605	0.8506	0.8571	0.8454	0.8478	0.8605	0.8444	0.8736	0.8478	0.8776
Minimum	0.7368	0.7447	0.7368	0.7447	0.7368	0.7368	0.7447	0.7368	0.7347	0.6863	0.7381
Standard Deviation	0.0442	0.0425	0.0408	0.0406	0.0422	0.0358	0.0412	0.0352	0.0473	0.0560	0.0408
	Synchronous with additional dataset										0.8333

Table 32. F-Measure results with different metrics (Adaboost) – CR dataset

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.7872	0.7723	0.8043	0.7872	0.8043	0.8043	0.7579	0.7629	0.7723	0.8090
4	0.7692	0.7912	0.8211	0.7816	0.8261	0.8132	0.7957	0.7692	0.7778	0.8298
6	0.8081	0.7879	0.8119	0.7629	0.8119	0.8163	0.8000	0.7921	0.8000	0.8041
8	0.8632	0.8660	0.8600	0.8511	0.8515	0.8600	0.8660	0.8889	0.8485	0.8155
10	0.8387	0.8298	0.7778	0.7872	0.7692	0.7879	0.8511	0.8041	0.7865	0.8421
12	0.7843	0.8298	0.7525	0.7447	0.7238	0.7742	0.8041	0.7640	0.8081	0.7677
14	0.7872	0.7723	0.8043	0.7872	0.8043	0.8043	0.7579	0.7629	0.7723	0.8090
Average	0.8085	0.8128	0.8046	0.7858	0.7978	0.8093	0.8125	0.7969	0.7989	0.8114
Maximum	0.8632	0.8660	0.8600	0.8511	0.8515	0.8600	0.8660	0.8889	0.8485	0.8421
Minimum	0.7692	0.7723	0.7525	0.7447	0.7238	0.7742	0.7579	0.7629	0.7723	0.7677
Standard Deviation	0.0360	0.0350	0.0370	0.0360	0.0452	0.0295	0.0396	0.0480	0.0278	0.0256

Table 33. F-Measure results with different metrics (Adaboost) – SS dataset

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.7213	0.7213	0.7213	0.7213	0.7213	0.7213	0.7213	0.7213	0.7213	0.7213
4	0.9315	0.9315	0.9315	0.9315	0.9315	0.9315	0.9315	0.9315	0.9315	0.9315
6	0.9167	0.9167	0.9041	0.9167	0.9041	0.9167	0.9167	0.9167	0.9167	0.9167
8	0.9211	0.9211	0.9211	0.9211	0.9211	0.9211	0.9211	0.9211	0.9211	0.9211
10	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333
12	0.8919	0.8919	0.8919	0.9041	0.9067	0.8919	0.8919	0.8919	0.8919	0.8919
14	0.8378	0.8378	0.8378	0.8493	0.8378	0.8378	0.8378	0.8378	0.8378	0.8378
16	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333
Average	0.8859	0.8859	0.8843	0.8888	0.8861	0.8859	0.8859	0.8859	0.8859	0.8859
Maximum	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333	0.9333
Minimum	0.7213	0.7213	0.7213	0.7213	0.7213	0.7213	0.7213	0.7213	0.7213	0.7213
Standard Deviation	0.0739	0.0739	0.0733	0.0731	0.0737	0.0739	0.0739	0.0739	0.0739	0.0739

The tables above allow us to compare the synchronous vs. asynchronous classification, the metrics and the effect of the choice on the number of parts.

As observed in the different tables, when classifying actions with asynchronous and synchronous properties, the results are the best when using an asynchronous model. In fact, the results of the Adaboost when training real recordings, with all datasets, in Table 30, are lower than the grayed-out cells of any of the columns in Table 31. The differences between the results when classifying a synchronous dataset and an asynchronous are obvious; the improvement is a lot more significant with the asynchronous dataset.

As we notice by comparing the classification of the different datasets, there is not a fixed number of parts that always gives the best results. Consequently, we decided that this parameter should be fixed by the user. Nevertheless, when classifying datasets that are targeted for the ALF solution, or have high ALF properties (review V.D.d.1), we observe that the ALF solution outputs the best performances independently of the number of parts. For example, this behavior can be easily observed by comparing the performances in Table 33 and Table 31, where the classification of SS does not depend on the number of parts, as opposed to the classification of the CAP dataset where the number of parts should be fixed.

It is important to mention that the optimal number of parts remains the same and gives the same result regardless of the captured recordings that are tested. As mentioned previously, to show that this parameter is stable, a second test dataset is recorded with the same actions as the CAP database and the results are recorded in Table 31, in the last column. The metric that is used is the MCC. The performances have been improved in fact with 4 and 10 parts compared to the synchronous dataset, same as with the initially tested dataset. The same results are

observed when operating the classification with KNN on the additional dataset in "Appendix II – ALF additional experimentations," where they have been considerably improved at 8, 10 and 12 parts compared to the synchronous results.

Important note: we only display performances with Adaboost classification, the rest of the results with KNN can be checked in Appendix II – ALF additional experimentations. They show the same results than the ones obtained with the Adaboost and can be analyzed in the same way.

1.3.2. Metrics

To build the ALF model, the metrics have been analyzed in V.D.a and in V.D.c.2 where it the performances were optimal when building the ALF model with the MCC metric. In this section, we analyze the metric results in Table 31, Table 32 and Table 33 with the captured datasets.

To analyze the results, the maximum value of the F-Measure calculated across the number of parts is the value that interests us the most, since choosing an optimal number of parts always gives the best results when performing any classification.

Most of the metrics give approximately the same results, in fact, the difference between the maximum of the F-Measure and its lowest value does not exceed 4%. Moreover, the values of the standard deviation are a lot similar.

As a result, we rely on the explanations in Table 26 and Table 27 to decide which metric will be adopted. Hence, we choose the MCC and consider it as the most appropriate metric for building the ALF model in our future experiments.

We are aware that the metrics listed in Table 25 may not be the most appropriate choice, as there are numerous possibilities for combining the results into a model. Yet, the table allows us to present a solution for picking the most appropriate one.

The columns in the tables in this part, entitled Model=1, are a special case where the model is equal to 1 on all parts. In other words, we remove any possible benefit of the ALF model in the final results.

1.3.3. Overlap

An additional parameter is introduced and tested to increase the probability of finding the same information in different parts: the overlap size between the parts *w*.

The new window size is calculated as follows:

$$new\ window\ size = (window\ size\ wihtout\ overlap) \times w \times 2 + (window\ size\ wihtout\ overlap) \quad (29)$$

We consider three values for *w* during our experimentations: 0, 0.5 and 1. The additional frames are concatenated to both sides of the original window.

In Figure 31, we display an example of a recording that is segmented into three windows with the three types of overlap. When w=0.5 and w=1 the window without an overlap is slid, and

additional frames are equally appended to its start and end. The dot lines are the extensions to the original windows to reach an overlap of w% of the neighbor windows.

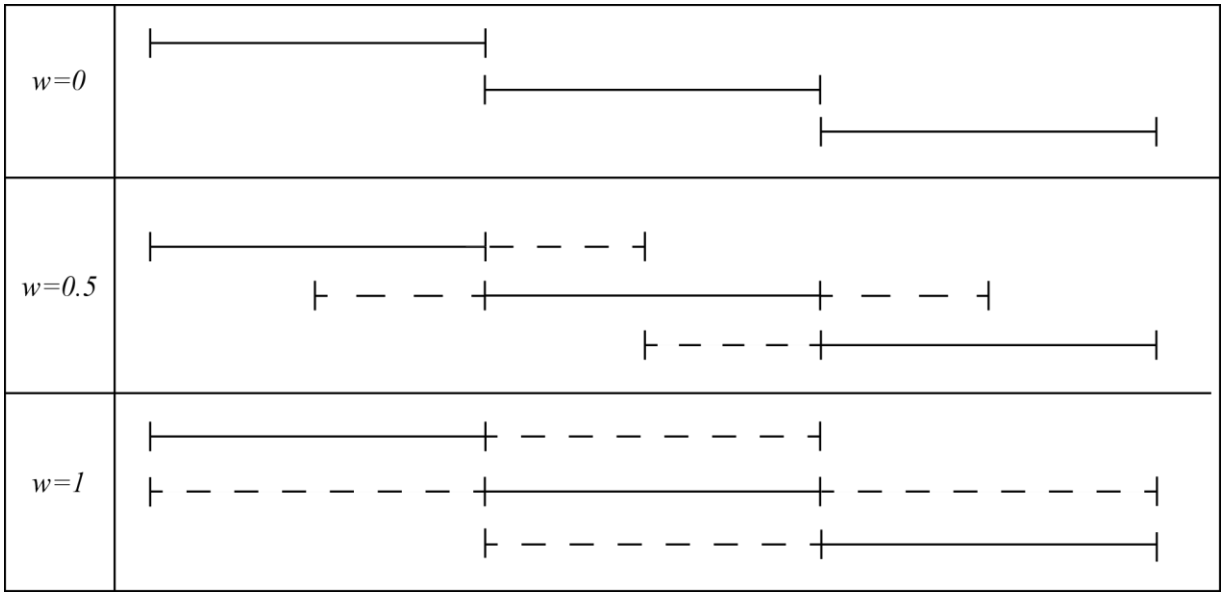


Figure 31. Example of an overlap of *w=0.5* & *w=1* on a sequence that has been segmented into 3 parts

In Table 34, we display the performances of different numbers of parts when classifying the CAP, CR, and SS datasets, when changing the overlap size with the ALF solution. The values are then compared by observing the results of each dataset separately.

Table 34. Different overlap sizes (F-Measure) – MCC metric – Adaboost

	CAP			**CR**			**SS**		
Number of parts	***w=0***	***w=0.5***	***w=1***	***w=0***	***w=0.5***	***w=1***	***w=0***	***w=0.5***	***w=1***
2	0.7368	0.8381	0.8315	0.7872	0.7921	0.6593	0.7213	0.3636	0.3636
4	0.8736	0.7009	0.8352	0.7692	0.8247	0.5977	0.9315	0.8421	0.7945
6	0.8046	0.7791	0.7727	0.8081	0.8632	0.7191	0.9167	0.8889	0.9041
8	0.8132	0.7440	0.8696	0.8632	0.8200	0.7234	0.9211	0.9333	0.9315
10	0.8421	0.8387	0.8696	0.8387	0.8132	0.7327	0.9333	0.9333	0.9459
12	0.7551	0.8512	0.8539	0.7843	0.8043	0.7083	0.8919	0.8947	0.9189
14	0.8261	0.8255	0.8636				0.8378	0.8378	0.8493
16	0.8119	0.8381	0.8000				0.9333	0.9459	0.9315
Average	0.8079	0.8019	0.8385	0.8085	0.8196	0.6901	0.8859	0.8300	0.8299
Maximum	0.8736	0.8512	0.8696	0.8632	0.8632	0.7327	0.9333	0.9459	0.9459
Minimum	0.7368	0.7009	0.7727	0.7692	0.7921	0.5977	0.7213	0.3636	0.3636
Standard Deviation	0.0442	0.0548	0.0370	0.0360	0.0243	0.0521	0.0739	0.1928	0.1951

In Table 34, the maximum and the standard deviation are the values that interest us, in order to analyze the best result and difference across the number of parts. The most flagrant values reveal the failure of *w=1* with CR and, both *w=0.5* and *w=1* with SS. *w=0* partially fails with Adaboost SS (The asynchronous dataset) where the difference between the maximum results for the different values of *w* is 0.01, which can be disregarded.

Consequently, we do not add an overlap when performing the remaining of the experimentations. Table 35 resumes the analysis of the results, where an overlap is labeled as "bad" when its standard deviation or maximum value is very different from the ones obtained with the other overlaps.

Table 35. Overlap analysis resume

		w=0	***w=0.5***	***w=1***
Adaboost	C	Good	Bad	Bad
	Cr	Good	Good	Bad
	SS	Good	Bad	Bad
KNN	Cr	good	Good	Bad
	CR	good	Bad	Bad
	SS	good	Bad	Bad

Same as in (V.D.d.1.3.1) above, not all the results are shown in Table 34; all the different metrics have been combined with the variations of the overlap size with KNN and Adaboost, and a huge set of results can be found in Appendix II – ALF additional experimentations.

1.3.4. Experimentations' resume

Table 36 and Table 37 present a resume of the tables above allowing us to compare the synchronous and asynchronous solutions, and show the purpose of using the asynchronous model, especially when classifying datasets with asynchronous properties.

The gray cells in Table 36 represent the F-Measure results of the classifications with ALF that outperform the synchronous solution. We note that KNN gives the best results with the ALF, especially that, as in the discussions of the confidence coefficients III.C, the output of the early classifiers is multiplied by the number of nearest neighbors labeled with the binary output, hence decreasing the confusion in the classifiers.

Table 36. Comparison between synchronous and asynchronous – all datasets - MCC metric

	Adaboost			KNN		
Number of parts	**CAP**	**CR**	**SS**	**CAP**	**CR**	**SS**
2	0.7368	0.7872	0.7213	0.6400	0.6053	0.7941
4	0.8736	0.7692	0.9315	0.6923	0.7356	0.9189
6	0.8046	0.8081	0.9167	0.7586	0.7045	0.8780
8	0.8132	0.8632	0.9211	0.7907	0.7033	0.8810
10	0.8421	0.8387	0.9333	0.7816	0.7310	0.7711
12	0.7551	0.7843	0.8919	0.7765	0.7174	0.8675
14	0.8261		0.8378	0.8046		0.8276
16	0.8119		0.9333	0.7529		0.9383
Average	0.8079	0.8085	0.8859	0.7497	0.6995	0.8596
Synchronous	0.8315	0.8200	0.6494	0.6494	0.6933	0.5909

Finally, in Table 37 we compare the results obtained when classifying the different datasets with the Asynchronous and Synchronous late fusion. It is clear that results are better with the ALF solution.

Table 37. F-Measure comparison of Asynchronous and Synchronous late fusion

	ALF solution	**Synchronous solution**
Adaboost C	0.8736	0.8315
Adaboost SS	0.9333	0.6494
Adaboost CR	0.8632	0.8200
KNN C	0.8046	0.6494
KNN SS	0.8276	0.6933
KNN CR	0.7273	0.5909

1.4. Additional datasets: ALF compatible datasets

As seen in the previous experimentations, the ALF solution can perform very well with some types of datasets; for example, the SS dataset in Table 33. The ALF solution outperforms the synchronous solution independently from the number of parts. In this part, we will experiment on two additional datasets: the RL (introduced in Table 6) and the Gait dataset (introduced in Table 7) to show that the ALF solution is not only restricted to a few datasets.

1.4.1. Gait dataset (III.A, Table 7)

The gait dataset is hard to recognize since there are few differences between the actions, in particular between the normal gait [10], the neurological experiment [151], and the Parkinsonian-like shuffling [150]. Nevertheless, we observe asynchronous properties in the dataset, in particular between the following actions: army march and incorrect army march left leg fracture and right leg fracture. In the stated actions, the legs move in opposite directions.

The major difference that exists between the normal gait, the neurological experiment, and the Parkinsonian-like shuffling is the dynamicity of the steps when performing the action. The Parkinsonian-like shuffling is performed slower than the others. This change in the dynamicity induces compatibility with the ALF since the sub-recordings (steps) will be shifted.

We cut the recordings between 2 and 16 parts, and we compare the ALF results to the synchronous solution in Table 38.

Table 38. Gait classification across the number of parts

Number of Parts	***F-Measure***
2	0.691729
4	0.778626
6	0.788732
8	0.723077
10	0.791667
12	0.706667
14	0.689189
16	0.723684
Synchronous	0.6423

It is clear that the ALF solution improves the performances regardless of the number of parts. We note a 15% increase in the performances when choosing the optimal number of parts.

1.4.2. RL dataset

We classify the RL dataset that is a perfectly asynchronous situation; the recordings have approximately the same length and the same number of sub-recordings, but the hands are raised at different time instants in all actions, with the right one shifted in time (it is composed of 4 classes, as described in Table 6). Table 39 shows the results of the classification of the RL dataset. The recordings are long. Consequently, it is possible to cut them to up to 36 parts. The performances when classifying with the ALF solution are compared to the synchronous solution.

Table 39. RL classification across the number of parts

Number of Parts	*F-Measure*
2	0.524
4	1
6	1
8	1
10	1
12	1
14	1
16	1
18	0.955
20	0.952
22	0.977
24	0.955
26	0.930
28	0.933
30	0.952
32	0.930
34	0.930
36	0.913
Synchronous	0.4

The difference between the ALF and the synchronous classification is noticeable; the ALF performances are perfect when cutting the recordings with a number of parts between 4 and 16, and higher than 0.9 with a larger number of parts, as opposed to the synchronous classification that has much lower classification performances. When classifying a perfect asynchronous dataset such as the RL, the gain of using the ALF is obvious, no matter the number of parts.

2. *Misclassification on the early level (temporal dynamicity)*

A confusion occurs in a special case: large changes in temporal dynamicity; when the dynamicity of a temporal sequence changes a lot between two recordings belonging to the same class, the same part might contain information that can be found in the same parts of the other class.

When the location of the gestures differs inside recordings that belong to a similar class, the parts that are extracted from the recording might not always contain the same information. e.g., the action surrender from the CAP dataset, and 2 recordings of 100 frames each. If the surrender action starts in a recording at frame 10 and ends at 50, and in another recording, it starts at frame

40 and ends at 80, by cutting the recordings into 4 parts the information inside each parts is not similar.

e. Comparison with HMM

The main difference between the model in this thesis and HMM classification is that the states in an HMM are dependent. Hence, the probability of an event, in HMM, depends on the previous states, as opposed to the asynchronous model where the decision that was taken by the classifier at the early level is independent of the one taken at the next or previous part. Another difference is that the HMM does not combine the confidence coefficient and the resulting decision.

The HMM is tested with the datasets that are mentioned in this chapter, by considering every part as a state and performing a PCA on the features before training or testing a dataset. The HMM classifier is tested with an early fusion classification, then with a late fusion, both multi-class (using the Accord.Net framework [157]).

Figure 32 displays the early fusion architecture; the recordings are cut into parts and the similar parts from every joint are concatenated into a vector before applying PCA and inputting the vector to the HMM. Every vector is considered as a single state.

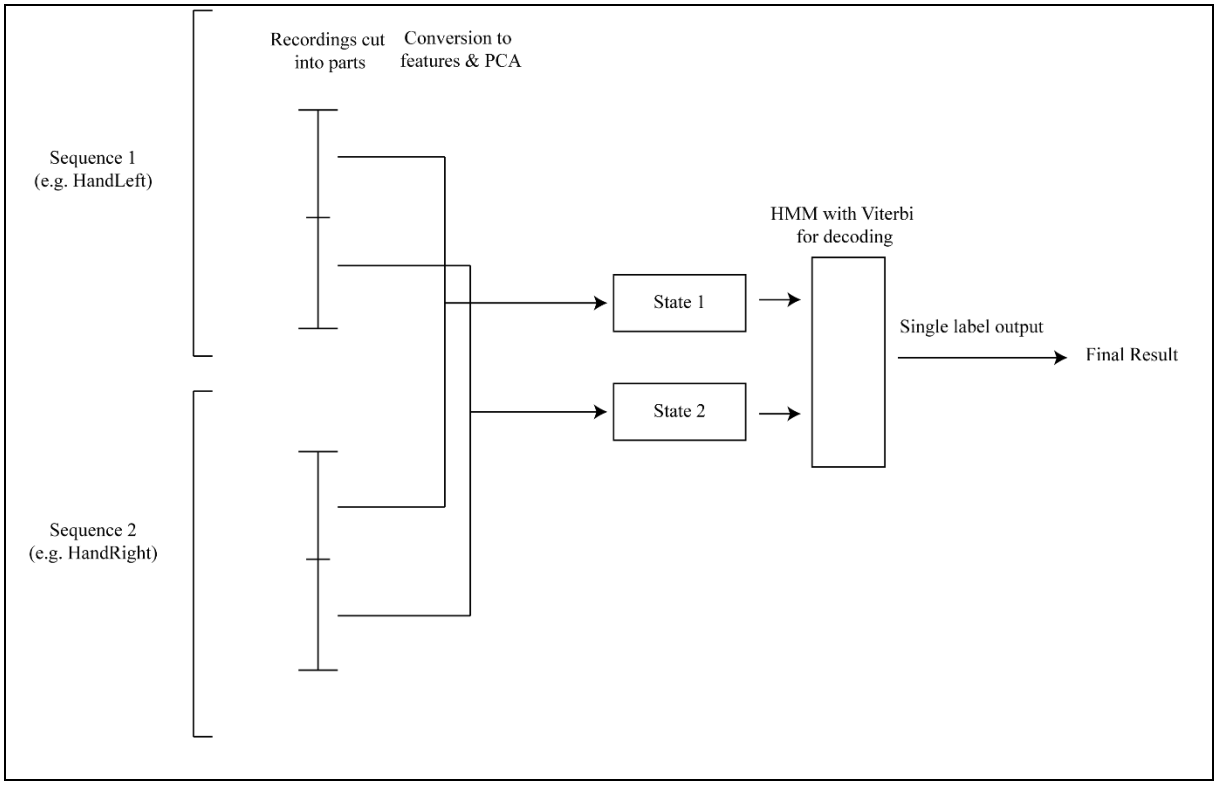


Figure 32. HMM early fusion architecture

Figure 33 shows the late fusion architecture that was adopted. As in the early fusion architecture, every part is considered as a state, but in this case, an HMM (with Viterbi for decoding) is built for every joint sequence. The results from the HMM are combined using a simple voting strategy to extract the final decision.

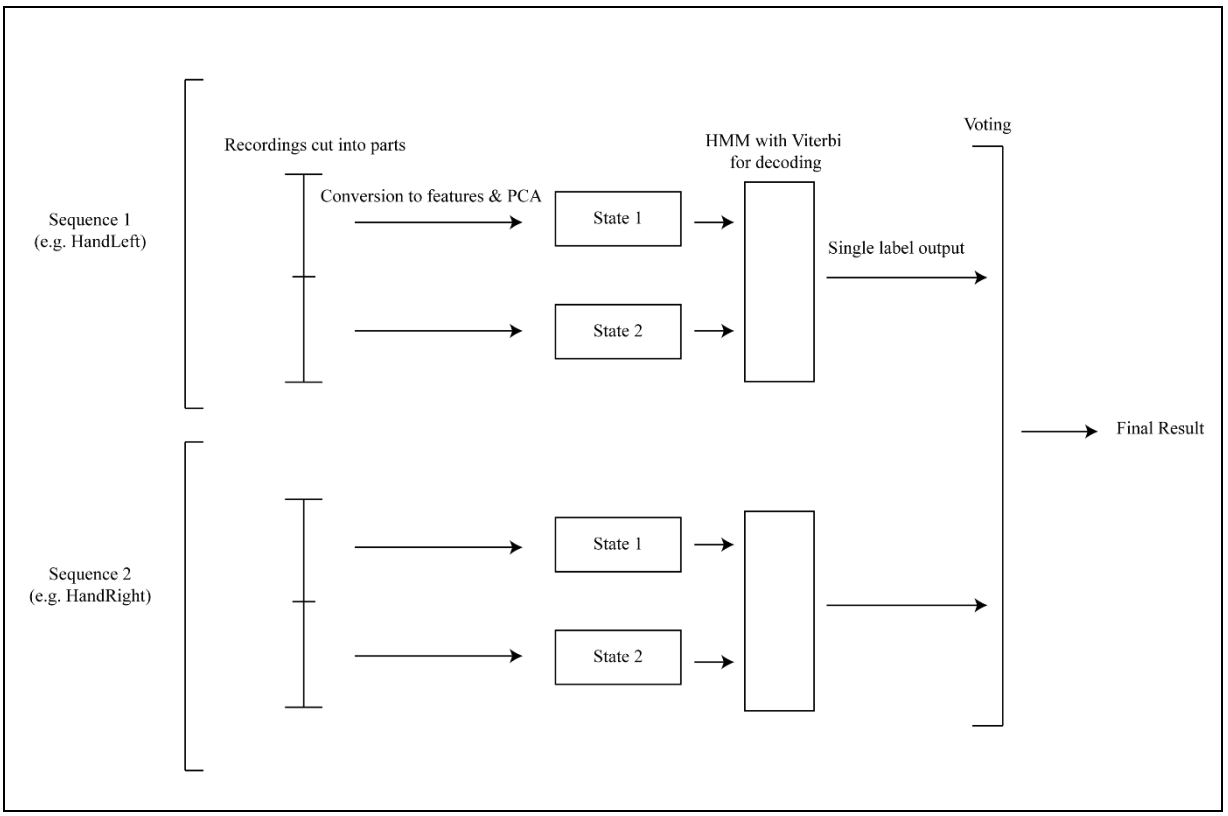


Figure 33. HMM Late Fusion architecture

Since the HMM classification outputs multi-class results, to compare the performances with the one-vs-all classification that we have obtained while classifying with the ALF solution, only the recordings that have been attributed with a single label are considered as True Positive.

The results of the classification with HMM using an early fusion and a late fusion are displayed in Table 40 across the number of parts. For comparison purposes, the classifications with Adaboost are mentioned in the right column of the table.

Table 40. HMM classification of CAP dataset (parts as HMM states)

	HMM Early Fusion			HMM Late Fusion			Adaboost ALF		
Number of parts	**TP**	**Total Recordings**	**Result**	**TP**	**Total Recordings**	**Result**	**TP**	**Total Recordings**	**Result**
2	16	49	0.3265306	18	49	0.367347	36	49	0.7347
4	24	49	0.4897959	18	49	0.367347	37	49	0.7551
6	24	49	0.4897959	18	49	0.367347	40	49	0.8163
8	25	49	0.5102041	19	49	0.387755	39	49	0.7959
10	24	49	0.4897959	14	49	0.285714	38	49	0.7755
12	20	49	0.4081633	19	49	0.387755	39	49	0.7959
14	21	49	0.4285714	18	49	0.367347	34	49	0.6939
16	16	49	0.3265306	14	49	0.285714	40	49	0.8163
Average			0.433673			0.352040			0.7730
Adaboost Synchronous Late Fusion							35	49	0.714286

By comparing the column called results and the average of the results, we note that the ALF solution with both Adaboost and KNN (Table 81) outperforms the HMM early and Late fusion classification.

f. Extension of the evaluation framework

We improve the results of the asynchronous model by generating simulated recordings using the same method as in chapter IV.

1. *ALF method with simulated recordings*

We consider that the datasets that we are working on are not very diverse and contain very few recordings. Consequently, we apply the simulation algorithm (IV), to generate new recordings and enrich our dataset. As seen previously, the classification of Adaboost CAP with synchronous actions is improved with a sufficient number of simulated recordings (we picked 50 simulated recordings according to the results Table 13), using the proportionality method IV.C.i.

Experiments have been conducted in Table 41 to show the consequence of including simulated recordings in the datasets when applying the ALF method. The cells marked in gray denote the best values when comparing the classification of the CAP dataset, when training with simulated recordings, in the following cases:

- Removal of the effect of the ALF model (Model=1) and using the model (MCC)
- Synchronous solution and ALF solution

We note that we classify the CAP dataset with 2 algorithms: KNN and Adaboost. The classification results with both algorithms are compared separately.

Table 41. Asynchronous model with simulated recordings – CAP dataset

Number of parts	Adaboost Model=1	Adaboost MCC	KNN Model=1	KNN MCC
2	0.8714	0.9200	0.6862	0.7955
4	0.9128	0.9333	0.8381	0.8791
6	0.8889	0.9028	0.8151	0.8298
8	0.9178	0.9178	0.8016	0.8261
10	0.8859	0.9252	0.8151	0.8261
12	0.8961	0.9020	0.8016	0.8222
14	0.9178	0.9028	0.7998	0.8222
16	0.9103	0.8947	0.7998	0.8090
Average	0.9001	0.9123	0.7946	0.8262
Maximum	0.9178	0.9333	0.8381	0.8791
Minimum	0.8714	0.8947	0.6862	0.7955
Standard deviation	0.0171	0.0136	0.0457	0.0242
Synchronous with simulated	0.8696		0.6905	

By observing the results in the table above, we conclude that:

- The combination of the ALF method and the simulated recordings, outperforms the results of the classification with a synchronous method, regardless of the number of parts.
- The combination of the ALF method and the simulated recording improves the performances of the classification with the ALF method when training with real recordings.
- Removing the effect of the ALF model (Model=1) while classifying with simulated recordings does not outperform the usage of a model (MCC).

2. *Real output of the Adaboost*

Even though, we did not consider the real output from the Adaboost as a trusted confidence coefficient (III.C). During the testing phase, we modify the input of the late classifier by multiplying the confidence coefficient (real output of the classifiers) by the binary decision of the classifiers and the weight attributed by the ALF model. As a result, the standard schema of the ALF solution is modified, and the previous statement is expressed in the equation below (equation 29 is an update of equation 28)

Let n=1,…,N be the ALF models' index, where each model is generated from a sequence and p=1,…,P the number of parts.

$$input\ of\ late\ classifier_n = \sum_{p=1}^{P} Model_n[p] \times real\_decision_n[p] \times binary\_decision_n[p] \quad (30)$$

Multiplying the real output by the scalar product adds a difference between the values used as an input of the Late Fusion. Nevertheless, this only improves the classification performances of the CAP and CR datasets. Even though the results with SS degrade compared to using a binary output, the performances are variable depending on the data intrinsic properties. The results are displayed in Table 42.

Table 42. Asynchronous solution – Adaboost – MCC – multiply the asynchronous model fusion with Adaboost real output

	Adaboost					
Number of parts	**CAP**		**CR**		**SS**	
	cc=bin	cc=real	cc=bin	cc=real	cc=bin	cc=real
2	0.7368	0.7879	0.7872	0.8333	0.7213	0.7838
4	0.8736	0.9167	0.7692	0.7961	0.9315	0.8919
6	0.8046	0.8406	0.8081	0.8125	0.9167	0.8493
8	0.8132	0.9041	0.8632	0.8980	0.9211	0.8947
10	0.8421	0.8947	0.8387	0.8163	0.9333	0.9067
12	0.7551	0.8235	0.7843	0.7647	0.8919	0.9067
14	0.8261	0.8000			0.8378	0.8108
16	0.8119	0.9189			0.9333	0.9091
Average	0.8079	0.8608	0.8085	0.8202	0.8859	0.8691
Maximum	0.8736	0.9189	0.8632	0.8980	0.9333	0.9091
Minimum	0.7368	0.7879	0.7692	0.7647	0.7213	0.7838
Standard Deviation	0.0442	0.0539	0.0360	0.0446	0.0739	0.0488

E. Resume

In conclusion, the asynchronous late fusion is the temporal decisions schema applied on **predefined windows** for finding a certain class when using a classification algorithm, and when different components of the studied item (action) react at different time instants. This is what we call the asynchronous properties of a dataset.

We introduced this ALF model for improving temporal events classification applied on late fusion classification algorithms. We showed the reason behind the use of an asynchronous model when classifying datasets with temporal properties. Then, we introduced the algorithm behind the asynchronous model and the parameters that were used to tune it.

Finally, according to computed performances from different algorithms and datasets, we showed that the Asynchronous Late Fusion improves the results of a simple Synchronous solution in most of the cases.

VI. A FRAMEWORK FOR THE ASYNCHRONOUS MODEL

In this chapter, the term series will be used multiple times. Hence, we remind that a series is a long recording composed of a succession of actions where the recordings might not belong to the same action.

A. Asynchronous Index & Asynchronous Index on the Parts (ASI & ASIP)

a. Objective

As observed in the results obtained in V.D.d, some datasets are highly compatible with the ALF solution (SS dataset) compared to others (CAP and CR dataset). In this chapter, we provide the users a tool to identify some of these datasets. It consists of calculating an indicator that compares datasets by extracting statistical information from the recordings. In other words, the compatibility with the ALF method is a measure that can only be used to compare datasets between each other.

We only study a couple of properties in the asynchronous datasets. In the definition of the ALF (V.B.a), we stated that the discriminant sub-recordings are not found at the same instants in all dimensions. An example where this perfect asynchronous situation occurs is considered: recordings that belong to different classes from the same dataset are similar (similar values) but shifted in time (translation). We extract statistical information concerning the similarity and the translation from the dataset with these special properties to analyze the compatibility of the datasets with the ALF.

Figure 34 represents an example of the asynchronous dataset, mentioned in the paragraph above, where the two properties can be found. In the figure, the same dimension from two different recordings A & B is displayed. A belongs to a different class than B. The recordings are extracted from a series and cut into 6 parts (p_1 to p_6). The recordings are taken as examples from the datasets to compare the different classes. We only take two recordings to explain this simply.

On one hand, when observing A and B in full, before cutting them, they are similar. In fact, they are only translated. The average, the local minima and the local maxima, calculated on all their values are equal.

On the other hand, A is translated in time to obtain B. When cutting the recordings into 6 parts, at the same time instants, the local maxima and minima are not located in the same parts. For example, we note that the local maximum that is located in the 2nd part of recording A is located in the 3rd part of recording B.

In resume, we focus on two points only in this part to compare the compatibility of the datasets with the ALF: the similarity and the translation in time between recordings, belonging to different classes in the same dataset.

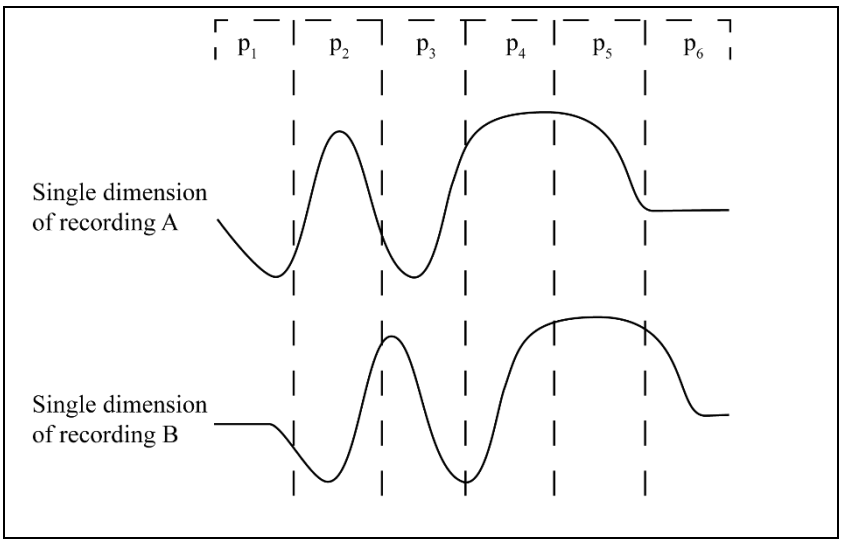


Figure 34. Perfect situation of a dataset compatible with the ALF solution. A & B are similar but shifted in time

b. Similarity (ASI)

The similarity information is studied by extracting information from the full recordings (without cutting them): the average, the local maxima, and the local minima. These three values are called the ASynchronous Index or the ASI.

Since we are working with multiple dimensions when applying the late fusion solution, the sum of the averages and the average of the local minima and maxima are calculated to end up with 3 values for each dataset.

To extract the ASI, the algorithm below is applied:

Let N be the number of recordings in a dataset D with R_i the *i-th* recording belonging to D with label c_i, with $c_i \in \{1,\ldots,C\}$ *and* $i \in \{1,\ldots,N\}$, and R_k the k-th recording belonging to D with label c_k, with $c_k \in \{1,\ldots,C\} \setminus c_i$

j is a skeleton joint with $j \in \{HandLeft,\ldots, HipCenter,\ldots\}$

The difference between the averages and the extrema are calculated between every 2 recordings, from different classes. Then the differences are summed.

Algorithm 5 details the calculation of the values described above.

1 For every class c_i

2 For every class c_k

3 Calculate the average of the coordinates (X,Y,Z) of every recording:

$$\mu_{R_i} = \sum_j \frac{sum_X + sum_Y + sum_Z}{number\ of\ frames}$$ and similarly for R_k

4 Calculate the sum of the differences between the averages

$$\mu_{(c_i,c_k)} = \sum_{i,k}^{N} |\mu_{R_i} - \mu_{R_k}|$$

5 Calculate the sum of the difference between the average of the local minima. (We calculate their average because the number of local minima is unknown.)

$$local\_minima_{(c_i,c_k)} = \sum_{i,k}^{N} \sum_j \Big(\Big| \mu(local_{minima_{R_i}}(X)) - \mu(local_{minima_{R_k}}(X)) \Big| + \Big| \mu(local_{minima_{R_i}}(Y)) - \mu(local_{minima_{R_k}}(Y)) \Big| + \Big| \mu(local_{minima_{R_i}}(Z)) - \mu(local_{minima_{R_k}}(Z)) \Big| \Big)$$

6 We do the same for the local maxima

7 End For

8 End For

9 After comparing the classes, we calculate, to obtain a single value for each of the average, local minima and local maxima, we calculate the average of the values obtained at line 4, 5 & 6 of the algorithm:

10 $$\mu_{ASI} = \frac{\sum_{i,k} \mu_{(c_i,c_k)}}{number\ of\ class\ combinations} = \frac{\sum_{i,k} \mu_{(c_i,c_k)}}{C(C-1)/2}$$

$$local\_minima_{ASI} = \frac{local\_minima_{(c_i,c_k)}}{number\ of\ class\ combinations} = \frac{local\_minima_{(c_i,c_k)}}{C(C-1)/2}$$

We do the same for the local maxima

Algorithm 5 ASI

When two full recordings are similar or approximately similar, the differences between the averages and the extrema are equal to 0 or very small.

c. Translation (ASIP)

The ASynchronous Index calculated on the Parts (ASIP) detects the translations between the recordings of different classes. The ASIP is an ASI that is computed at every part of the recordings (the parts that have been extracted with the ALF method.) and compares the parts between the classes of a dataset. In other words, the ASIP is a similarity between the parts of a

dataset. Figure 34, above, is an example of the comparison between two recordings that are shifted in time.

To extract the ASIP, Algorithm 5 is modified to obtain the one below in Algorithm 6

1 For every part p where p={1,…,P}

2 For every class c_i

3 For every class c_k

4 Calculate the average of the coordinates (X,Y,Z) of every part p in the recording:

$$\mu_{R_i p} = \sum_j \frac{sum_X + sum_Y + sum_Z}{number\ of\ frames}$$

5 Calculate the sum of the differences between the averages

$$\mu_{(c_i,c_{i'})\mathrm{p}} = \sum_{i,k}^{N} |\mu_{R_i p} - \mu_{R_k p}|$$

6 Calculate the sum of the differences between the average of the local minima. (We calculate their average because the number of local minima is unknown.)

$$local\_minima_{(c_i,c_k)\mathrm{p}} = \sum_{i,k}^{N} \sum_j \left|\mu(local\_minima_{R_i p}(X)) - \mu(local\_minima_{R_k p}(X))\right| + \left|\mu(local\_minima_{R_i p}(Y)) - \mu(local\_minima_{R_k p}(Y))\right| + \left|\mu(local\_minima_{R_i p}(Z)) - \mu(local\_minima_{R_k p}(Z))\right|$$

7 We do the same for the local maxima

8 End For

9 End For

10 After comparing the classes, we calculate, to obtain a single for each of the average, local minima and local maxima, we calculate the average of the values obtained at line 4, 5 & 6 of the algorithm:

11 $$\mu_{ASI\ at\ p} = \frac{\sum_{i,k} \mu_{(c_i,c_k)\mathrm{p}}}{number\ of\ class\ combinations} = \frac{\sum_{i,k} \mu_{(c_i,c_k)\mathrm{p}}}{C(C-1)/2}$$

$$local\_minima_{ASI\ at\ p} = \frac{local\_minima_{(c_i,c_k)p}}{number\ of\ class\ combinations} = \frac{local\_minima_{(c_i,c_k)\mathrm{p}}}{C(C-1)/2}$$

12 We do the same for the local maxima

13 End For

14 Calculate the average of the ASI that have been obtained at every part to get only 3 values for every dataset:

$$\mu_{ASIP} = \frac{\sum_{p=1}^{P} \mu_{ASI\ at\ p}}{P}$$

$$local\_minima_{ASIP} = \frac{\sum_{p=1}^{P} local\_minima_{ASI\ at\ p}}{P}$$

We do the same for the local maxima

Algorithm 6. ASIP

In addition to the ASI, when two recordings are similar or approximately similar but shifted in time, the differences between the averages and the extrema calculated at every part is high.

d. End result (ASIv)

To compare two datasets, the ASI and ASIP are computed on both. The results from both datasets are compared to one another. A low value of the ASI indicates that the full recordings of two datasets are similar and a high value of the ASIP reveals a dissimilarity between the parts of a dataset. The dataset where the lowest value of the ASI and the highest value of the ASIP appear signify that it is not compatible with a synchronous solution, but should be classified with the ALF to enhance the results.

Each of the ASI and the ASIP is composed of 3 values. The comparison of two datasets requires comparing 12 values. To take a final decision the user needs to have a general view of all values. Consequently, we proceed with a simple voting solution where we compute a final index (ASIv) for every dataset. We calculate the following between every 2 datasets:

- Whenever a value from the ASI of the first dataset is lower than the corresponding one in the ASI of the second dataset, the index is increased by 1.
- Whenever a value from the ASIP of the first dataset is higher than the corresponding one in the ASIP of the second dataset, the index is increased by 1.
- When the opposite of the above points occurs, the index is decreased by 1.

The ASIv is the sum of the values that are obtained above, per dataset.

Finally, the dataset with the highest ASIv is a dataset that should be classified with the ALF solution.

We note that when the ASIv is low (ASI high and ASIP low) when comparing the datasets, it is impossible to predict the behavior of the dataset when applying the ALF solution. In fact, the dataset might verify other properties of the ALF (e.g. sub-recordings that are considered more or less relevant to finding the ground truth can be found in some parts of the recordings).

e. Experimentations

An example of the application of the algorithms above is displayed in Table 43 & Table 44, where we compare the CAP, CR, and SS datasets.

Table 43. ASI to compare the CAP, CR, and SS datasets

Dataset	Average	Local maxima	Local minima
CAP	2.97	111.55	78.59
CR	3.87	93.35	73.63
SS	1.97	83.93	57.98

Table 44. ASIP to compare the CAP, CR, and SS datasets

Dataset	Average	Local maxima	Local minima
CAP	2.60	126.18	97.11
CR	3.67	110.50	89
SS	3.17	141.62	101.45

Table 45. ASIv to compare the CAP, CR, and SS datasets

Dataset	CAP	CR	SS	ASIv
CAP	-	0	-6	-6
CR	0	-	-4	-4
SS	6	4	-	10

According to Table 43, the SS dataset has the lowest ASI values, and according to Table 44, it has the highest ASIP. As a result, according to the ASIv in Table 45, SS is more compatible with the ALF solution than the CAP and CR datasets. The values of the indexes when computed on the CAP and CR are both lower than SS and show that both do not perform as well as the SS dataset. In fact, the ALF method improves the classification of the CAP and CR only when the optimal number of parts is picked.

B. Action Segmentation

a. General introduction

During this thesis, the classification has only been applied to segmented recordings. Nonetheless, it is possible to apply on series segmentation.

In the state of the art, to perform the segmentation of series, we observe that there is a schema that is often used to perform segmentation [117] [113]:

1. Train the classifiers with a pre-segmented dataset
2. Choose the best size of windows for segmenting the series.
3. Segment the series with the chosen window size (different overlapping methods can be applied).
4. Classify the windows.
5. Filter the results and classify the recordings inside the series.

Otherwise, the classification can be performed on the whole recording and the location deduced according to the value of the confidence coefficients. In this chapter, we consider the latter.

b. Segmentation

To perform the segmentation, we are inspired from the general procedure mentioned above to classify series from the CAP and SS datasets. As in the previous chapters, the training dataset is different from the testing dataset, and the training dataset is composed of three groups of recordings.

1. *Training with additional label*

The training of the classifier is performed as it was done in V, nonetheless, when performing the final segmentation and classification with different window sizes, new actions may appear while sliding the windows on the series. In order to train the classifier and find these actions, new "random" recordings are extracted from the training dataset to enhance the negative class. A large series containing multiple actions is cut randomly to generate as many samples as needed. The number of recordings is a user parameter. It is recommended to take into consideration the type of classification (1-vs.all, 1-vs.1, imbalanced datasets…)

1	Concatenate all the original recordings into one large series (S)
2	For every class (C) in the original dataset
3	Calculate the minimum (min) and maximum (max) length of the recordings from class C
4	Pick a random number R1 between min and max as the recordings' width
5	Pick a random number pos between 0 and (length of S-R1) as the start of the recording
6	Cut the series at Position=pos and extract R1 skeletons

7 End for

8 Finally, we populate every set of recordings (one for training the early level and one for the late fusion) with the recordings generated above.

Algorithm 7. Generating garbage recordings for segmentation training

The above algorithm is applied to the CAP and SS dataset, by joining all the samples that were previously used for training into a large sequence, in the previous chapters, when classifying segmented recordings.

2. *Choosing the best window for segmenting the series*

To perform a proper segmentation of the series, the main parameter to set is the size of windows that will segment a large series containing multiple recordings. Considering that the number of parts has been fixed during the training of the ALF, it is possible to compute an average window size from the training dataset. We propose to use this value. Nevertheless, to confirm the previous statement, we perform multiple experimentations considering the minimum, the maximum and the average of the size of the parts.

3. *Segment the series with the chosen window size*

After choosing the window size, the series is cut into smaller interlaced windows. When classifying a recording with the ALF solution, the parts should not be interlaced and overlapped, according to the results obtained in the previous experimentations. Consequently, the windows that correspond to the parts are picked from the series will be sequential and non-interlaced while segmenting. Figure 35 explains clearly how the windows are picked from a series when working without an overlap. The example in the figure consists of an ALF model composed of two parts and a recording that starts and ends at unknown locations. The interlaced windows that are cut from the series are shown on the second line (we only show 3 interlaced windows in the figure) and the parts are picked from the interlaced windows on the 3rd line.

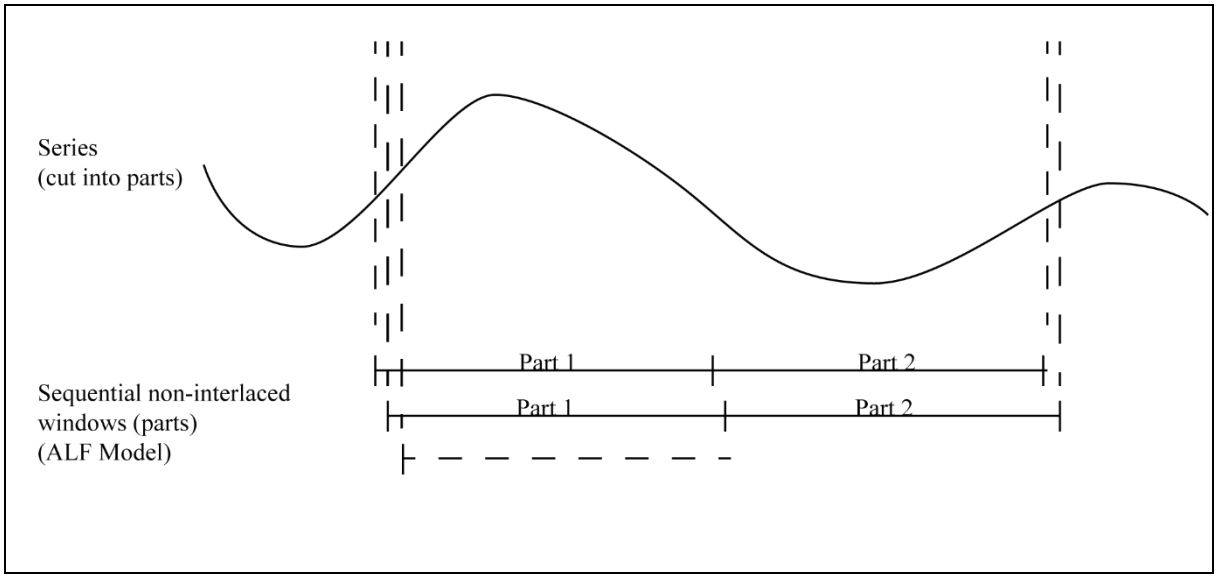


Figure 35. ALF segmentation: picking the parts from the windows

4. *Classify the windows*

When segmenting with a synchronous method, every window is tested disjointedly from the others. As for the segmentation with the asynchronous method, a number of sequential windows (the number of windows that are picked is equal to the number of parts) is tested (as in Figure 35). The scalar product is calculated afterward between the model and the decision at every part (similar to the standard ALF solution), and the final decision is outputted by the late fusion classifier of the ALF solution. We run the classification with a 1-vs.-all strategy for as many classes as there are in the dataset. The decision of every classification is binary, hence, the target label of the 1-vs.-all classification is considered as the decision. These are attributed to the starting frame of the ALF model, which is considered as the start of the recording since the size of the recording is unknown. The procedure to deduce the end of the recording will be described in the next paragraphs.

An example of the classification procedure when running both the ALF and the synchronous solutions are displayed in Figure 36.

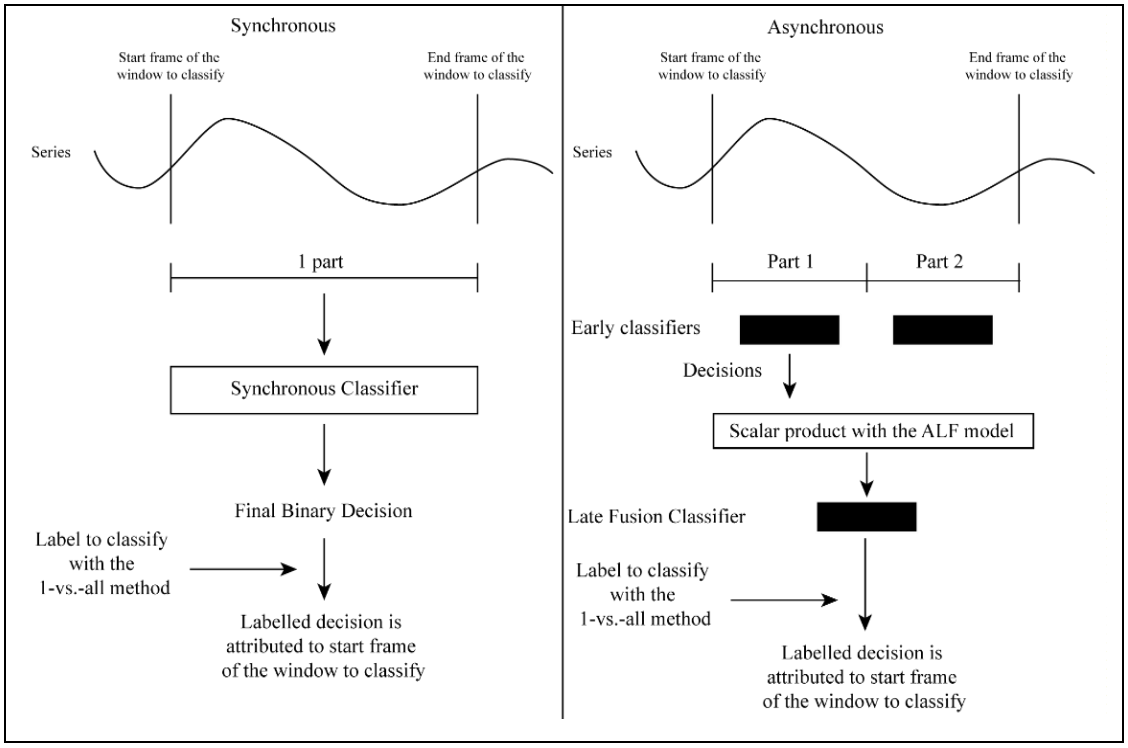


Figure 36. Segmentation - Synchronous vs. an example of ALF on 2 parts: classifying the windows

5. *Filter the results and take the final decision*

In the following part, we filter and analyze the decisions that have been taken previously to output the final decision, which consists of segmented recordings and label. To this end, the start and end of the segmented recordings should be located. This method is implemented when applying the synchronous and asynchronous solutions.

As mentioned previously, because of the 1-vs.-all strategy, multiple decision labels might be taken, therefore, when two decisions or more are found in a single frame, the classifier's decision is not considered as trusted and the decision is eliminated.

The first label where an action is detected is considered as the start of a recording. We need then a procedure to find the end of a recording. The end frame is found by the following simple algorithm:

1	For each frame between the minimum and the maximum length of the recordings that were used for training
2	Find the label (L) that appears the most
3	The end location of the recording is the average position of the frames where L is found
4	End For
5	If there is no label detected between the minimum and maximum, the average value of the length of the training recordings is considered as the end of the recording.

Algorithm 8. Finding the end of a recording

In Figure 37, we display an application of Algorithm 8 where we consider that the start of the recording, indicated in the figure by S, has already been found, and its end will be calculated by averaging the positions of two labels, which are detected: L_1 & L_2 at multiple positions. The timeline in the figure is fictive and has only been drawn for illustration purposes. L_1 is the label that appears the most and by averaging its 3 positions, we find the final end position.

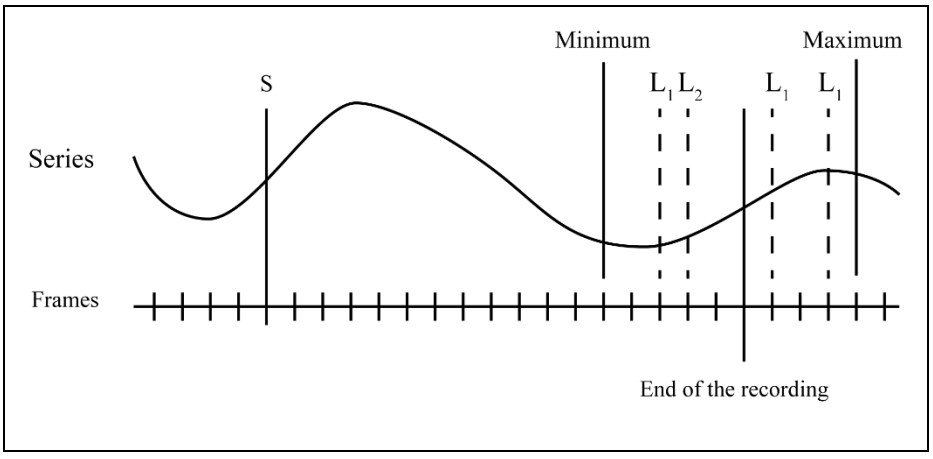


Figure 37. Extracting the end of the recording

To evaluate the performances of the segmentation, we analyze the results with the Jaccard Index, the same performance measure that was proposed in Chalearn 2014 [21]:

$$Jaccard\ Index = \frac{Ground\ Truth \cap Result}{Ground\ \ Truth \cup Result} \qquad (31)$$

In other words, the equation calculates the segmentation error by dividing the frames that have been detected as true positive by the total frames that have been detected (Ground Result + False Positive detection). A perfect decision would be equal to 1.

6. *Experimentations*

We display in Table 46 the results of the application of the ALF segmentation on the CAP and SS datasets, while changing the window size, and compare it to the synchronous late fusion segmentation. Three series are tested; each one is composed of approximately 4000 frames. The first one contains recordings that belong to the CAP dataset (CAP with resting). It contains the actions that have been segmented and tested in the previous chapter. Nonetheless, in this part, they are joined into a series with the addition of a synthetic resting position between the recordings. The resting position is a repetition of the final frame of the recording. The second test is performed on a new series that belong to the CAP dataset (real CAP). It is not related to the datasets in the previous chapter. The third, and final series, contains SS actions that are joined into a single series.

Usually, when performing a segmentation, the window size to segment the series is considered as the average size of the training recordings. We question the choice of this parameter and segment the series with different window sizes. We calculate the minimum, maximum and average size of the parts of the recordings for each class when training the ALF and the same values on the full recordings when training a synchronous solution. Hence, during the segmentation and classification of the series, a different window size is considered for every class.

The largest confusion will, of course, appear in the SS dataset since the actions have asynchronous properties and are the inverse of each other, e.g., a soccer action starts by throwing a ball and then kicking it, as opposed to the not soccer that starts with a kick in the air and then throws the ball. When multiple soccer recordings are joined, the classifier will recognize the action soccer at the first frame for example, and in its middle, the action not soccer will be detected.

Table 46. Segmentation of series from the CAP dataset

	Asynchronous method			**Synchronous method**		
	Window Size			***Window Size***		
	Average	***Minimum***	***Maximum***	***Average***	***Minimum***	***Maximum***
CAP with resting	0.46	0.39	0.42	0.30	0.29	0.23
Real CAP	0.42	0.37	0.40	0.27	0.17	0.26
SS	0.66	0.68	0.43	0.36	0.36	0.33

The values in the table correspond to the Jaccard Index that is computed when cutting the series with different window sizes.

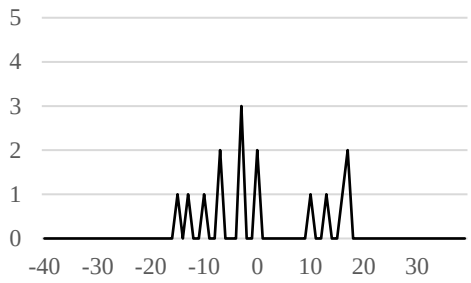
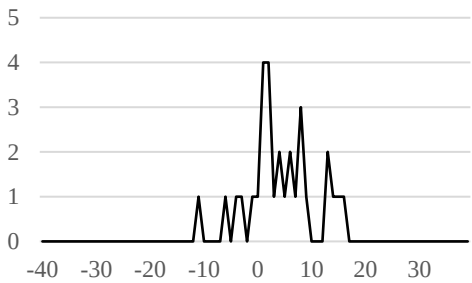
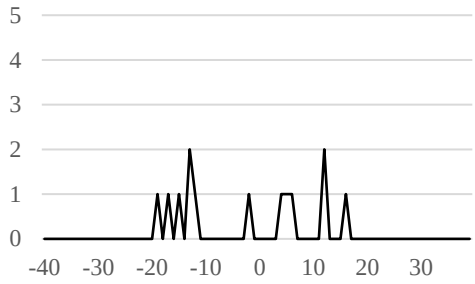
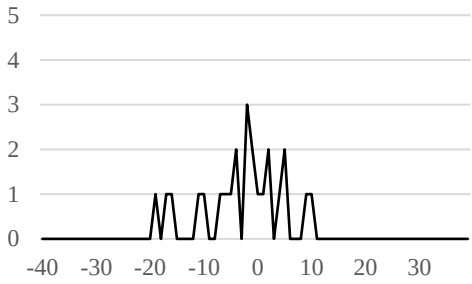
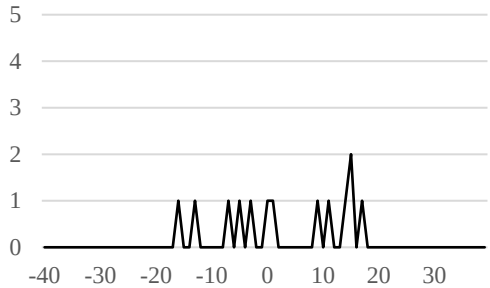
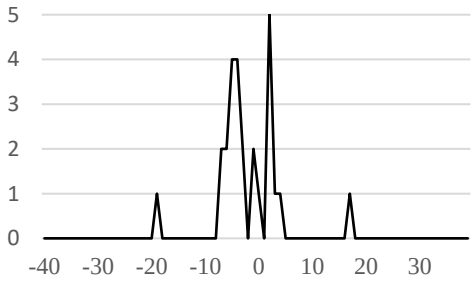
According to the results in the table above, it is clear that no matter the size of the window, the results of the ALF solution outperforms the Synchronous method. In fact, all the Jaccard

Indexes are larger with the Asynchronous method. Moreover, we note that in most of the tests, the windows size that outputs the best performances is the average size of the original recordings.

To display additional information concerning the segmentation, we compute the number of detections of the start and end of the recordings that are located around the real start and end of the recordings. These values help to visualize the accuracy of the segmentation. Accurate results are obtained when a peak appears at the exact position of the start and end of the recording. The exact start and end positions are marked by the value 0 and 40 frames are displayed around this position.

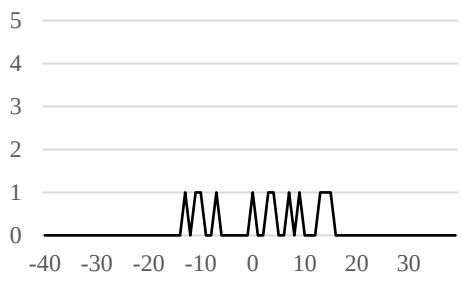
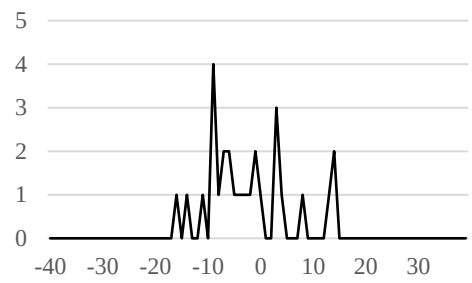
We obtain the results of the detection of the start and end of the recordings in the tables below.

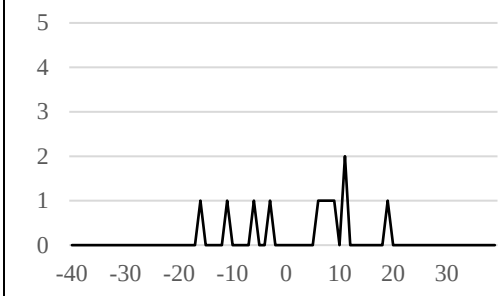
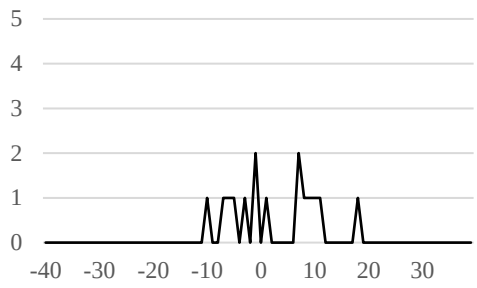
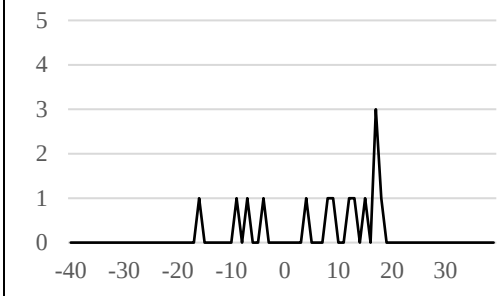
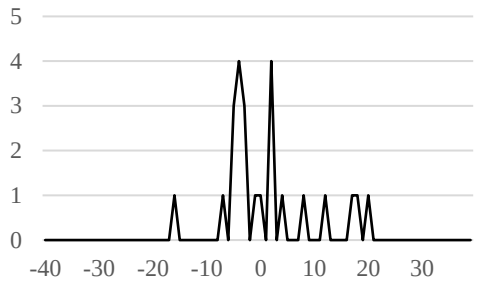
Table 47. Detected location of the start of the recordings

	Synchronous	Asynchronous
CAP with resting		
Real CAP		
SS		

The scale of the horizontal axis is the frame number around the exact start and end positions of the action, and the vertical axis is the number of detections around the start and end positions. The start and end are the frame 0.

Table 48. Detected location of the end of the recordings

	Synchronous	Asynchronous
CAP with resting		

Real CAP		
SS		

The scale of the horizontal axis is the frame number around the exact start and end positions of the action, and the vertical axis is the number of detections around the start and end positions. The start and end are the frame 0.

When applying the asynchronous method, the decisions are gathered around the true start and end of the recording, as opposed to the segmentation with the synchronous method where the results are scattered and rarely located at the true starting position of the recordings. Moreover, in most of the segmentations, a peak appears at the true position. Hence, the asynchronous is more accurate while segmenting than the synchronous method.

c. Adaboost and full MSRC-12 dataset

We extend the experiments that we have conducted in the segmentation section to well-known datasets such as the MSRC-12.

1. *Segmented recordings*

First of all, a simple but long test was performed on the full-segmented MSRC-12 action dataset:

- The full recordings were segmented using the following annotated dataset: [161].
- Approximately 6150 recordings were verified manually.
- 20 recordings per class were used to train 12 classes

  The performances were evaluated on the remaining recordings and are noted in Table 49:

Table 49. Full MSRC-12 dataset performances

	Number of Parts	TP	TN	P	N	F-Measure
Synchronous (real output as input of late classifiers)	-	3401	56076	5189	57079	0.7091
ALF	2	3504	56521	5189	57079	0.7575
	4	3273	56224	5189	57079	0.7026
	6	3402	56222	5189	57079	0.7202
	8	3502	56222	5189	57079	0.7336
	10	3423	56486	5189	57079	0.7437
	12	3368	56472	5189	57079	0.7351

The cells that are highlighted in gray indicate the values that outperform the synchronous method.

According to the right column, which displays the results with the F-Measure, it is obvious that by finding the best number of parts, the ALF is better than the Synchronous method.

2. *Full series*

Next, we segment the MSRC-12 series. The performances are evaluated according to the annotations mentioned previously VI.B.c.1 and the Jaccard Index is displayed in Table 50. The results are the average computed from of 3 trials.

We choose the number of parts according to the 1-vs-all results for each label. As a result, every action is tested with a different number of parts.

N.B.: the series in this section are evaluated previously in the experimentations of the previous chapters as segmented recordings.

Table 50. Jaccard Index from the segmentation of MSRC-12 recording streams

Action class	Synchronous	ALF
Start	0.396	0.64
Crouch	0	0
Push Right	0.83	0.87
Put Goggles on	0	0
Wind it up (Gym)	0.07	0
Shoot	0.4895	0.515
Bow	0	0.77
Throw	0.1	0.34
Surrender	0.03	0.11
Change Weapon	0.10	0.03
Beat Both Hands	0.51	0.88
Kick	0.54	0.57

As observed in Table 50, the algorithm that is implemented does not output the best performances. Nevertheless, it is clear that the ALF outperforms the simple synchronous solution, except for the cases where both fail totally.

Wind it up, Change Weapon, Crouch and Put on Goggles are cases where both the synchronous classification and the ALF fail. The performances of the classification of these four actions are poor, because the recordings for a single class are very different, justifying the misclassification.

C. Additional dataset: power consumption

This part shows that the asynchronous model can be used in other fields than action recognition.

The datasets on which we could apply the ALF is large (e.g. commodity market, traffic congestion...). Nowadays, one of the major social issues targeted by research program is the Household Energy Saving. Considering this, we are interested in a dataset that was developed for this purpose. We took such a one from the UCI repository [160]: Individual household electric power consumption Data Set. It was taken from the UCI repository [162]: Individual household electric power consumption Data Set. The dataset has the following properties:

- Every record in the data is labeled with a date and a time.
- A sample is recorded every minute
- 7 variables are recorded

We consider that the dataset in question is not fully annotated and aim to perform this task. It is evident that the power consumption differs between the days of the week. For example, during the weekends, the household power consumption should be different from the weekdays since most people will be working during the weekdays. It is expected that the ALF detects the fluctuations, that a synchronous does not find, during the full day since a full day is cut into small part and every part is analyzed separately at a first stage.

The dataset is segmented manually, and the following tasks are identified. They are listed below:

1. Weekend detection: the goal is to separate the weekends from the weekdays. A simple binary classification problem is presented that should be an easy task for classification algorithms. This task should be the most compatible with the ALF because as we have mentioned above, most people will be working during the day. Hence, the power consumption should be the lowest during this time and the ALF should be able detect this change easily. We apply a 1 vs. 1 classification strategy. The training and test datasets consist of days that are picked randomly from the year 2007. The training dataset contains 20 days per class, per level (early, model and late) and the test dataset contains 60 days per class.
2. Day detection for years 2007, 2008, 2009 and 2010 (labeled 2007-2010): the goal is to recognize the day of the week. This task is harder than the binary classification because the weekdays should have approximately the same power consumption values (same for the weekends). Nevertheless, our goal is to find out if the ALF can perform well with this complexity. We apply a 1 vs. all classification strategy. The training dataset contains only the days from the year 2007. The test dataset is composed of daily recordings for the years 2008 to 2010. The training dataset contains 20 days per class, per level and the test dataset contains 140 days per class.
3. Day detection for year 2007: the goal is to recognize the day of the week. This task should be simpler than the "day detection for years 2007-2010" since there should be less fluctuation throughout one year than across several years. We apply a 1 vs. all classification strategy. The training and test datasets contain only the days from the year 2007. The training dataset contains 20 days per class, per level and the test dataset contains 20 days per class.

The features that are extracted from the recordings are the following:

Table 51. Features used as input to the early classifier

Features	Variations and comments
Absolute value of the Gradient	Mean max min
Absolute second derivative	Mean max min
Signed Gradient	Mean max min
Signed second derivative	Mean max min
Values of the variables	Mean max min
List of the local maxima calculated on the variables	Min mean
List of the local minima calculated on the variables	Max mean
List of the local minima and local maxima calculated on the variables	Standard deviation
Difference between the first value and the last value of the sample (called F_{diff})	Signed, Unsigned
$\frac{F_{diff}(\text{variable A})}{F_{diff}(\text{variable B})}$	Signed, Unsigned Between all the variables A and B cited previously where A and B are different

The results of the classification, with an Adaboost algorithm, are displayed in Table 52. The result is calculated on a number of parts that varies between 2 and 20, and the best number of parts is only shown in this table (this parameter is displayed in the second column). The remaining results can be checked in Appendix II – ALF additional experimentations.

Since we consider that the confidence coefficient of the Adaboost is not trusted (III.C), we input a binary value to the late fusion classifier (Table 52). Even without the usage of a confidence coefficient, with this poor information, we still show that the ALF solution outperforms the synchronous method.

The Adaboost algorithm is parameterized as follows:

- The classification problem is hard; hence the number of iterations is increased to 50. As a result, the performances improve.
- The classes are balanced since the power consumption dataset is large.

In the table below, below the performances of the classification of the 2nd and 3rd datasets have been computed with the F-Measure, since there are more than 2 classes in the dataset, as opposed to the 1st classification where 2 classes are only available (weekend and not weekend)

Table 52. Classification results of the power consumption dataset

	Best number of parts	Confidence coefficient used as input to the late classifier (for synchronous method only)	Synchronous	ALF solution
Classification 1 Weekend detection	15	Binary	0.5286	0.6989
		Real	0.5464	-
Classification 2 Day detection 2007-2010	20	Binary	0.2449	0.2811
		Real	0.2427	-
Classification 3 Day detection 2007	10	Binary	0.243	0.3045
		Real	0.2407	-

Even though the classifications did not perform very well (especially classification 3), according to the table above, we note that the ALF solution improves the classification. In fact, the performances outperform the synchronous method when classifying the 3 datasets. This shows that the ALF solution can be applied to other domains than action recognition.

At the beginning of this chapter, we proposed a tool for the users of the ALF to identify the compatibility of the datasets with the ALF solution. It consists of computing indexes: the ASI, ASIP, and ASIv. The dataset with the lowest ASI and the highest ASIP will be the one that is the most compatible with the ALF solution. The ASI and ASIP are afterward combined into a single value: the ASIv. The dataset with the highest ASIv is the dataset that is the most compatible with the ALF solution. To show that this tool can be used with additional datasets such as the power consumption, we display in Table 53, Table 54 & Table 55 the results from the ASI and the ASIP applied on the power consumption.

Since the 2007-2010 dataset is large, the indexes are calculated on a small amount of the training recordings in addition to some other recordings from the years 2008 to 2010.

Table 53. ASI to compare the power consumption datasets

Dataset	Average[a]	Local maxima[a]	Local minima[a]
Weekend detection	137	307960	306813
Day detection 2007-2010	155	308769	308704
Day detection 2007	151	307320	307848

a. *Values that are extracted from the full recordings and that represent the ASI (sum of the averages, average of the local minima and local maxima) (Algorithm 5)*

Table 54. ASIP to compare the power consumption datasets

Dataset	Average[a]	Local maxima[a]	Local minima[a]
Weekend detection	242	308282	306230
Day detection 2007-2010	267	307440	304558
Day detection 2007	241	308311	306175

a. *Values that are extracted from the full recordings and that represent the ASIP (sum of the averages, average of the local minima and local maxima calculated on the parts) (Algorithm 6)*

Table 55. ASIv to compare the power consumption datasets

Dataset	Weekend	Day 2007-2010	Day 2007	ASIv
Weekend	-	4	2	6
Day 2007-2010	-4	-	-4	-8
Day 2007	-2	4	-	2

Table 53 shows that the weekend dataset is the most compatible with the ALF solution since the values of the ASI are the lowest, afterward, the 2007 dataset. Nevertheless, when looking at Table 54, it is difficult to conclude. Hence, Table 55 resumes both the decisions of the ASI and ASIP, where the weekend dataset has the highest ASIv: 4, then the 2007 and finally the 2007-2010.

D. Defining an action

We gave a general definition of an action in the introduction I.C.b.8 and in this part, we question this definition and we improve it. We propose a "visual definition" of an action where we put forth the most discriminants joints and parts of an action in comparison to the other actions in the database. With the aid of the ALF model, we focus on these parts and joints and display them in a voxel-inspired image (as previously seen in Table 10). Consequently, the display is simplified and will help the viewer describing and identifying the actions correctly by looking at a single image.

The ALF model delivers information concerning an action by attributing weights to different parts of the recordings (for more information, review the ALF model in chapter V). Consequently, in this part, we match the ALF model and the recordings by finding the most discriminant parts in the recordings, in other words, the parts that have the highest weights in the ALF model.

The first step is to locate the discriminant joints. To this end, we compute the sum of the ALF model's values at every joint and disregard the values that are below the average of the sums.

N.B.: the average is only a parameter that we consider. Of course, the higher the value, the less the number of discriminant joints is. Hence, the precision is increased.

The second step is to find the most discriminant parts. We perform the same operation that was conducted on the joints, but this time, on the parts; the sum is calculated for every part. Afterward, the sums that are lower than the average of the sums are disregarded.

Table 56 shows a sample of the process that was described above applied on the action crouch, taken from the CAP dataset, with a 1-vs-all classification. The values above the average are marked in gray.

Table 56. ALF model for cap dataset, crouch action – 4 parts

Joint	Part 1	Part 2	Part 3	Part 4	Sum calculated on the joints
AnkleLeft	0.44	1.00	1.00	1.00	3.44
AnkleRight	0.71	1.00	1.00	0.90	3.61
ElbowLeft	0.64	0.78	1.00	1.00	3.42
ElbowRight	0.76	0.88	1.00	0.58	3.22
FootLeft	0.88	1.00	1.00	1.00	3.88
FootRight	0.71	1.00	1.00	0.76	3.47
HandLeft	0.78	0.76	1.00	0.83	3.36
HandRight	0.71	1.00	1.00	0.88	3.60
Head	0.90	0.88	1.00	0.90	3.69
HipLeft	0.61	0.88	0.90	0.90	3.30
HipRight	0.90	1.00	1.00	0.90	3.80

KneeLeft	1.00	0.88	1.00	1.00	3.88
KneeRight	1.00	1.00	0.83	1.00	3.83
ShoulderCenter	0.64	0.76	1.00	0.76	3.15
ShoulderLeft	0.90	0.64	0.90	1.00	3.44
ShoulderRight	0.76	0.78	0.88	0.67	3.08
Spine	0.88	0.78	1.00	0.76	3.42
WristLeft	0.64	0.64	0.90	0.55	2.73
WristRight	0.76	1.00	1.00	0.90	3.67
Sum calculated on the parts	14.63	16.66	18.42	16.30	

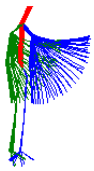
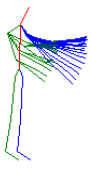
It is clear in Table 56 that the joints related to the lower part of the body are considered as discriminant, as well as the 2 middle parts of the action. In fact, when performing the crouch action, the significant changes in the joints' positions are observed on the legs. Moreover, when comparing the recordings in the dataset, the difference between the recordings is mostly observed in the middle parts, since at the end and at the start, the joints move from and to their initial, stable position.

To show that the discriminant parts and joints from the recordings are the only ones that have been retained, we block the movement of the joints that are labeled in gray in Table 56 for recordings that belong to the CAP, SS, and MSRC-12 dataset. Some of the resulting recordings are displayed in voxel-inspired images, in Table 57 to Table 59.

In Table 57, the surrender action originally moves both hands at the same time but after blocking the joints, we observe that the left hand is the only joint that is moving. In fact, the right hand is the only discriminant joint in the right-hand wave action and has a similar behavior to the movement of the right hand in the surrender action. As a result, to identify the right-hand wave properly, the right hand is attributed with a lower weight in the ALF model of 1-vs.-all classification of the surrender action and the left-hand remains. This allows the algorithm to minimize the confusion between the two actions.

An interesting exception occurs in the CAP dataset: in right hand up, all the joints have been blocked. This does not appear as a problem since only the right hand in the surrender action is blocked. Hence, by comparing the right hand up blocked to the remaining of the dataset, the action is detected easily.

Table 57. Captured dataset blocked joints & parts

Action type	**Without blocking**	**with blocking**
2 hands up		

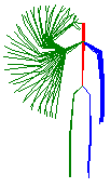
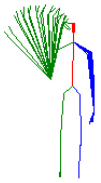
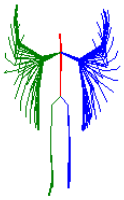
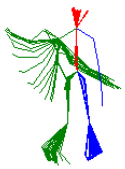
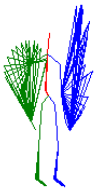
Crouch		
Right-hand up		
Right-hand wave		
Surrender		
Tennis forehand drive		
Tennis backhand drive		

Table 58. SS dataset blocked joints & parts

Action type	Without blocking	with blocking
Swimming butterfly		

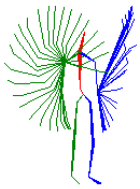
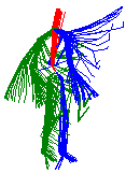
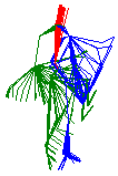
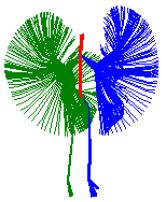
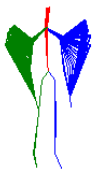
Swimming crawl		
Soccer		
Not Soccer		

Table 59. MSRC-12 dataset blocked joints & parts (5 actions picked randomly)

Action type	Without blocking	With blocking
Bow		
Kick		
Wind it up (Gym)		
Goggles		

Shoot		

In some actions, joints that are not moving (stable joints) can be discriminant, for example, when performing the action surrender, the legs do not move. When a joint is blocked, a visual confusion arises between them and the original stable joints, hence, to fix this issue, it is possible to color the joints that have been blocked in a different color than the ones that are considered to be stable or moving.

To ensure that these are still close to the real actions, we ask 10 persons to recognize them. The resulting confusion matrices are obtained (Table 60, Table 61 & Table 62).

Table 60. Performances of the manual classification of the blocked recordings – CAP dataset

	2 hands up	**Crouch**	**Right-hand up**	**Right-hand wave**	**Surrender**	**Tennis forehand drive**	**Tennis backhand drive**
2 hands up	100	0	0	0	0	0	0
Crouch	0	100	0	0	0	0	0
right hand up	50	0	0	30	20	0	0
right hand wave	0	0	0	100	0	0	0
Surrender	0	0	0	0	100	0	0
tennis forehand drive	0	0	0	0	0	100	0
tennis backhand drive	0	0	0	0	0	0	100

Table 61. Performances of the manual classification of the blocked recordings – SS dataset

	Soccer	**not soccer**	**swimming crawl**	**swimming butterfly**
Soccer	100	0	0	0
not soccer	0	100	0	0
swimming crawl	0	0	100	0

swimming butterfly	0	0	0	100

Table 62. Performances of the manual classification of the blocked recordings – MSRC-12 dataset

	Start	**crouch**	**push right**	**goggles**	**wind it up**	**shoot**	**bow**	**throw**	**had enough**	**change weapon**	**beat both hands**	**kick**
Start	100	0	0	0	0	0	0	0	0	0	0	0
Crouch	0	100	0	0	0	0	0	0	0	0	0	0
push right	0	0	0	0	0	0	0	0	0	0	0	0
Goggles	0	0	0	100	0	0	0	0	0	0	0	0
wind it up	0	0	0	0	100	0	0	0	0	0	0	0
Shoot	0	0	0	0	0	100	0	0	0	0	0	0
Bow	0	0	0	0	0	0	100	0	0	0	0	0
Throw	0	0	0	0	0	0	0	80	0	0	20	0
had enough	0	0	0	30	70	0	0	0	0	0	0	0
change weapon	0	0	0	0	40	0	0	40	0	20	0	0
beat both hands	0	0	0	0	0	0	0	0	0	0	100	0
Kick	0	0	0	0	0	0	0	0	0	0	0	100

We note the following observations:

- Most of the actions have been clearly recognized from the first guess (these are the fields that are labeled in dark gray with a 100% values in the confusion matrices above).
- After showing all the actions to a person, she identified the mistakes that she made and corrected them, e.g., the classifier blocks the joints on the right arm for the action raise right-hand up. Consequently the skeleton is approximately stable. After showing all the actions to a subject, she remembered that he did not see the right hand up and attributed the label right hand up to the stable skeleton. Nevertheless, we do not add these corrections to the result; only the first guess is computed.

E. Additional Classifiers

In this part, the asynchronous model is applied to the SVM algorithm when classifying the action. This only purpose here is to show that the ALF is not restricted to a classification algorithm and that any can be implemented as the "black box."

Table 63. Synchronous classification of action datasets with SVM

Dataset	F-Measure
CAP	0.840909
SS	0.507042

Table 64. ALF Classification of action dataset with SVM

Dataset	Number of parts	F-Measure
CAP	2	0.863158
	4	0.893617
	6	0.863158
	8	0.836735
	10	0.833333
	12	0.777778
	14	0.709677
	16	0.712644
SS	2	0.931507
	4	0.931507
	6	0.901408
	8	0.849315
	10	0.805556
	12	0.938272
	14	0.658824
	16	0.631579

By comparing the results between the synchronous method (Table 63) and the ALF method

Table 64, the ALF method outperforms the synchronous one when the optimal number of parts is chosen for the CAP dataset (2, 4 and 6 parts). Nevertheless, the ALF method outperforms the synchronous one with SS (which is more asynchronous than CAP) no matter the number of parts.

Consequently, the results show the benefit of using the ALF model with the SVM classifier.

VII. CONCLUSION

Our work in this thesis has led to improvements in the classic classification approach. We implemented an additional level to the late fusion architecture. We called this method the Asynchronous Late Fusion (ALF). This study has been applied to the human gesture recognition domain. We proposed a method to enrich action datasets that were used later on, for the classification and for measuring the performance of our algorithm. Moreover, its implementation in several known problems has led to interesting results as well to noticeable performance improvements.

A. Kinesiology

Since we implemented our study on the action recognition field, it was essential to gather all available datasets beforehand to train our classification algorithms. In order to pick the datasets and their actions, it was necessary to define what an action is. The definitions that we found were general.

Therefore, we considered a kinesiological approach to establish a proper definition of an action. We described the correlation between action recognition in machine learning and computer vision, and Kinesiology. Our research also targeted biology, physics, arts...

We combined all these fields and ended up with the following definition (more information in I.C.b.8): an action is a predefined sequence of concatenated simple gestures. Recordings belonging to a similar action should have the same characteristics as a reference recording: same ROM, DOF, and their joints must move in the same direction. The recordings' time is dynamic, and the amplitude of the movements might defer (within certain boundaries). This difference denotes the uniqueness of an action when performed by different subjects.

Once we established the definition, we were able to capture recordings to build our datasets and perform the experimentations in this thesis. We were also able to evaluate the action datasets that are available to the public and finally, the definition allowed us to propose an action simulation algorithm.

B. Simulation

To perform action recognition, it was important to gather large datasets with discriminant recordings. We found that the datasets available online as well as the ones we have captured were insufficient to perform the classification of actions. Hence, we developed an algorithm to simulate synthetic actions in chapter IV.

The problem of non-discriminatory action datasets for action recognition was overcome by enlarging a set of captured recordings performed by different persons. We proposed a synthetic action generation method, for training action recognition algorithms.

The action simulation algorithm was inspired by the definition of an action, which states that recordings denoting the same action must follow certain characteristics of the reference recording (as explained previously). Nevertheless, the recordings are unique, and a small margin appears between these performances. We take into consideration the previous statement to generate synthetic recordings.

We analyze different methods for building the dataset and find the most appropriate ones. For example, we have adopted a method called proportionality, where we simulate recordings by creating a small margin surrounding the observation values. The classifiers that we train using these synthetic datasets perform much better than when experimenting on multiple classifiers.

Stable results were obtained with KNN, Adaboost, Random Forest and SVM when simulating with proportionality method.

In addition, we showed that removing what we call the *superfeatures* (features that are very discriminant on small training datasets and tend to misclassify recordings from a different test dataset), and thereby adding noise, within an acceptable margin, contributes to a significant improvement of the results.

Moreover, the method performed well with a dataset containing a large number of recordings, and when enlarging a small dataset. As seen in Table 13, we have simulated recordings from a limited number of original ones (2 or 3 recordings) from the CAP and MSRC-12 datasets and trained an Adaboost algorithm. The results were improved by 13% and 21% respectively, compared to the results obtained when training with the original recordings. According to the results in Table 14, the application of the simulation on the small CAP dataset, improved the results of a KNN algorithm by 13% when enlarging a small dataset and by 30% when enlarging a large one. In addition, we enhanced the performances with RF by 7% and with SVM on the CAP large dataset by 18%.

Consequently, the dependence on the size of the original training database is reduced.

C. Asynchronous Late Fusion

The core of this thesis is the Asynchronous Late Fusion study. The ALF is a temporal decision schema for finding a particular class while applying a classification algorithm on predefined windows from temporal sequences. The studied item (basic action classifier) outputs different decisions at different time instants. Some of the decisions and the temporal sequences might not be discriminant. Hence, this schema is designed to process and improve the classification of datasets where the previous case occurs. For example, we observe the presence of a certain action for specific joints, at different points in time, when performing action recognition.

At the first level of the schema, we cut the recordings in the datasets into parts of equal sizes and then we classify every part with a different classifier. We add a mid-level to the standard late fusion classification, where we combine the decisions from the parts. Those are weighed with a confidence coefficient and help to build a model (ALF model). The previous procedure is performed at every temporal sequence (joints) and the resulting decisions from these sequences are combined with a late fusion classifier to deduce the final decision.

We studied a lot of parameters to tune the ALF and to determine optimal strategies: the parts should not overlap, the number of parts should be picked as a user-defined parameter, and the MCC metric is used to build the ALF model.

When applying the ALF algorithm on our datasets, we noted significant performance improvements when classifying datasets that are compatible with the ALF, such as the SS and RL datasets. We noted a 29% increase in the results when comparing the synchronous and asynchronous solution (Adaboost classifier) in Table 33, a 15% increase during the classification of the quasi-medical Gait dataset in Table 38 and a 60% increase in the classification of the perfectly asynchronous RL dataset in Table 39.

When classifying other datasets that are less compatible with the ALF solution, we observed a 4% increase with the CAP and CR datasets.

We have also observed a similar improvement when working with other algorithms such as KNN; 16% with CAP and 22% with SS, and SVM; 5% with CAP and 43% with SS.

In addition, we have used the simulation algorithm from chapter IV and trained the ALF solution with the synthetic datasets. The results showed an increase of the performances when compared with the synchronous solution that was also trained with the simulated recordings.

D. A framework for the asynchronous model

a. Asynchronous Indexes

As it can be difficult for the user of the ALF solution to determine which datasets are compatible with the method, we built indicators that compare the datasets by extracting statistical information from the recordings. The compatibility differs between the datasets.

We developed two indexes: the ASI and the ASIP. These are combined into a final index (the ASIv) to provide information concerning the compatibility of the dataset with the ALF.

The ASI focuses on the similarity between the recordings inside the dataset and the ASIP on the translation between them. The ASI calculates statistical information between the recordings, and the ASIP uses the same calculation as the ASI on the parts that have been cut with the ALF method. Finally, the ASIv is a single final value that combines the ASI and the ASIP to provide the user with an easier understanding of the comparison between the datasets. As a result, when comparing two datasets, the one with the highest ASIv is considered as the most compatible with the ALF solution.

By extracting the ASIv from the CAP, CR and SS datasets, it was clear that the SS was the most compatible with the ALF solution.

In resume, the ALF improved the performances of the synchronous classification, and ASIv confirmed the compatibility of the datasets with the ALF.

b. Additional applications

The ALF solution has additional applications, such as the segmentation of actions in series and can be implemented in fields other than action recognition.

We were inspired by previous work on the segmentation of actions to build our algorithm. We trained the classifier by adding a negative class. The additional recordings were generated by first joining multiple recordings into a series, then extracting parts at random locations. The procedure was performed to disregard the windows that contain nonrelated actions the original dataset. After training the classifiers, a window was chosen to cut the sequences. We applied the ALF algorithm on sequential and non-interlaced windows. Finally, the results were filtered to obtain the start and end of the recordings.

We evaluated the performances with the Jaccard Index and compared the results between the synchronous and ALF solution. The method that we proposed increased the performances by an average of 15% when segmenting a CAP series containing a resting position between the recordings, by 16% when segmenting a newly recorded CAP series and finally by 25% when segmenting an SS series where the recordings have been joined end to start.

Moreover, we implemented the solution on three datasets containing power consumption recordings: weekend classification, 2007 day classification and 2007 to 2010 day classification. The performances showed an average increase of 9% when comparing the ALF solution to the synchronous solution.

We showed once again that when the ALF solution is applied, the performances can be increased. In addition, the detection of the start and end during the segmentation of the recordings with ALF is more precise than the segmentation with the synchronous solution (Table 57 & Table 58).

c. Human Action Visual Representation

We analyzed the human movement and gave a general definition of an action. Later, we improved this definition and proposed a "visual definition" of an action.

With the aid of the ALF model, we focus on the parts and joints of an action that are the most discriminant and display them in a voxel-inspired image.

The ALF model delivers information concerning an action by attributing weights to different parts of the recordings (for more information, review the ALF model in chapter V). Consequently, we match the ALF model and the recordings by finding the most discriminant parts and joints in the recordings. In other words, we find the parts that have the highest weights in the ALF model and the joints where the sum of the ALF model is the highest. Afterward, the parts and joints that do not agree with the previous statement are blocked completely, and the recordings are displayed in the form of a voxel in an image. The resulting actions are supposed to be easy to identify.

We have obtained single and straightforward images that revealed the most discriminant parts and joints of an action. We asked 10 persons to identify the actions by showing them the

recording once. They were able to identify 86% of the CAP actions, 100% of the SS actions and 75% of the MSRC-12 actions.

The output was interesting and resulted in a simple visual representation of an action, which highlighted the discriminant joints allowing us to define and recognize a movement in a dataset.

VIII. FUTURE STUDIES

A. Short Term

a. Number of parts

One of the parameters of the ALF is the number of parts in a recording. We set it as a user parameter in (V.D.d.1.3) when discussing the "impact of the number of parts on the Asynchronous and the Synchronous classification" (V.D.d.1.3.1).

We propose to go even more in depth in the matter towards an automatic process to set the value.

The ASIP (VI.A.c) has previously helped determining the compatibility of the datasets with the ALF solution. It would be interesting to apply the ASIP as a comparison index between the parts. For example, the first part could be compared with the second part...

The ASIP could be calculated on the recordings of a single dataset by modifying the number of parts. Afterward, a voting process could be performed, as previously done with the ASIv (VI.A.d) to find the optimal number of parts.

b. Complexity

The algorithms that we implemented in this thesis tend to be complex and time-consuming. For example, all the late fusion classification algorithms have to be run on all the joints. Also, the simulation algorithm aligns the coordinate sequences at all the joints to extract the intervals. Hence, it is possible to optimize and speed up the processes if the discriminant joints are found.

As seen in VI.D, some of the joints and parts of the recordings have been blocked. According to the visual representation, the actions were visually "clear". As a result, only the discriminant joints and parts of the recordings were left.

The possibility that the ALF model offers, in terms of information processing analysis, is an interesting property; it might for example allow us to discard the least discriminant joints to infer the final decision, thus reducing the amount of calculations. In addition, it might allow us to analyze thoroughly the discriminance of features and thus can eliminate them as input of the classification, consequently, reduce the complexity.

c. In-depth study of the MD-DTW

The decision to discard the MDDTW depended on the features used for alignment. The features were picked according to the literature on the subject. Nevertheless, these might not be the most appropriate ones. Hence, it is possible to take advantage of a boosting algorithm applied on the ALF to find the most discriminant ones.

As a result, the simulation algorithm will run as follows:

1. Perform an initial classification of the dataset containing the real actions with a boosting algorithm.
2. Extract the model from the ALF.
3. Apply the model to the actions as in V.D.b.4.
4. Extract the discriminant features picked by the boosting algorithm
5. Align the MDDTW with the discriminant features
6. Compare the alignment cost with the DTW
7. Generate the simulated action by aligning with the new features.
8. Compare the final results after simulation

Algorithm 9. Finding new features to align the MDDTW

B. Long Term

a. Deep Architecture

The Deep architecture became increasingly popular in the last few years. It is now one of the most used algorithms in the computer vision domain. We did not include this kind of algorithm as a classifier in our thesis since it would have raised some difficulties in identifying a specific architecture and it might have heavily increase the training steps for our experiments.

Nevertheless, it is possible to implement it for comparison purposes only:

- Consider a deep algorithm (for example Deep Neural Network, DNN) as a black box and run similar experiments as in our thesis.

- Compare the performances between the deep architecture and the ALF architecture by analyzing the parts of the recordings with both solutions. Every part of the recording will be inputted as a separate vector to a DNN, and an experiment will be conducted to find out if the algorithm might have a similar behavior as the ALF architecture and focus on the discriminant parts. Thus, it will be interesting to understand the behavior and compare the performances of the algorithm with the ALF solution.

b. Confidence Coefficient

The confidence coefficient has been an obstacle throughout this thesis. As seen in the literature review and the discussions, the calculation of the confidence coefficient always relied on the distance between a recording and the training recordings, or a vote from sub-decisions of the classifier (e.g., with the trees of a random forest). Nevertheless, these studies are still ambiguous and do not help to implement a proper confidence coefficient.

We built a custom algorithm for calculating the confidence coefficient, by finding the distance between the different recordings in the training dataset and a decision threshold (e.g., the weak classifier threshold in Adaboost). We then calculated the distance between the obtained value and the feature value of the recording that is being tested (Algorithm 1). We did not find that this was enough to trust the cc's values. Nevertheless, when utilizing the cc, that was calculated from simple algorithms (such as KNN), the classification performances improved considerably with both the synchronous and asynchronous solutions.

Consequently, in future studies, we will go further into details on the cc subject. We will explore procedures to calculate it and set properties to generalize a confidence coefficient onto as much classification algorithms as possible. As a preview to the subject, we refer to an interesting study that was discussed in chapter III.C.

As a result, the cc should combine the following values:

- The relevance of the recording according to the classifier: the distance to the decision boundary for each classifier.
- The relevance of the recording in the dataset: the distance between the recordings.
- The reliability of the classifier: calculated according to the number of recordings used for training or according to a validation dataset….

C. Minor paths

a. Increase the speed of the alignment algorithm

During the simulation process, we performed the alignment with DTW and MD-DTW only. Nevertheless, the time cost (complexity) behind both alignment algorithms is large.

It is possible to implement additional algorithms and compare the alignment's cost. The large variety of alignment algorithms extends to the implementation of multiple sequence alignment at the same time, such as the ones we can find in DNA sequence decoding problems. Consequently, it will be possible to align all joints at the same time. This method could decrease the time to perform the alignment while taking into consideration that the alignment cost does not degrade compared to the performances of DTW.

IX. APPENDIX I: ADDITIONAL SIMULATION EXPERIMENTATIONS

This part contains extensions to the simulation algorithm and implementations that have been programmed. Since, most of these have failed, nevertheless, for reference purposes, we developed the procedures in the Appendix.

A. Simulation algorithm

a. Aligning with MD-DTW

Certain basic algorithms were chosen when simulating the recordings, such as the DTW for aligning the extrema. The choice has been made according to the background on action recognition [163]. We are aware that there are numerous sequence decoding and classification algorithms such as Viterbi [164]. Nevertheless, we chose to test our method with the DTW because it is simple to implement, to observe and to handle. One of its variations, which has been previously applied for action recognition in [165], is the MD-DTW. In fact, the MD-DTW is used as a replacement of the straightforward and basic DTW.

With the MD-DTW, we align the sequences according to their features and their coordinates by doing the following:

- Store the coordinates, the instant velocity, average velocity and acceleration into a multi-dimensional vector (MD-vector).
- Normalize the MD-vector to the zero mean and unit variance.
- In the original algorithm, a Gaussian filter has been used to smooth the sequences before computing the distance matrix. Nevertheless, the alignment was bad. As a result, we skipped the smoothing step.
- Calculate a distance matrix with the Euclidean distance between the different dimensions of the vectors.
- Apply a normal DTW on the final distance matrix.

Before running the experiments with the simulated recordings, we extract some statistics from the alignment algorithm. The statistics are simple and consist of computing the sum of the intervals while running the simulation process. When simulating 50 recordings per class with the captured and the MSRC-12 datasets, the sum of the intervals is much larger when aligning with MD-DTW than with DTW.

The approximate sum with DTW is 2,500,000, as for the MD-DTW the sum is between 3,500,000 and 4,000,000. (Note: the values are large and do not make a lot of sense, but we mention them as an example of the large difference and the misalignment with MD-DTW)

Moreover, since the algorithm picks the reference recording randomly at every iteration IV.C.b, the sum differs a lot between the runs of the algorithm. Table 65 0to Table 66 display performance comparisons between the classification algorithms that have been used throughout this chapter, when aligning with MD-DTW and with DTW, on 3 different runs of the simulation

algorithm. Note that the simulation is done with the best parameters that have been found previously in this chapter: proportionality method, 50 simulated recordings, and the classifiers' specific parameters.

Table 65. Results with DTW with relative joint positions – CAP dataset

	Adaboost	**KNN (K=30)**	**Random Forests**	**SVM**	**Sum of the intervals during simulation**
Run 1	0.9103	0.6777	0.8690	0.7419	2461541
Run 2	0.9231	0.6667	0.8649	0.7273	2458438
Run 3	0.9241	0.6126	0.8333	0.7874	2529592
Average	0.9192	0.6523	0.8557	0.7522	

Table 66. Results with MD-DTW with relative joint positions – CAP dataset

	Adaboost	**KNN (K=30)**	**Random Forests**	**SVM**	**Sum of the intervals during simulation**
Run 1	0.8429	0.6609	0.8828	0.5641	3714225
Run 2	0.8936	0.6667	0.8707	0.7288	4144843
Run 3	0.9220	0.5833	0.8082	0.7143	3643200
Average	0.8862	0.6370	0.8539	0.6691	

Table 67. Results with DTW with relative joint positions – MSRC-12 dataset

	Adaboost	**KNN (K=30)**	**Random Forests**	**SVM**	**Sum of the intervals during simulation**
Run 1	0.9664	0.6235	0.7855	0.9502	6082631
Run 2	0.9565	0.6957	0.8365	0.9604	6051329
Run 3	0.9614	0.6243	0.7957	0.9412	6099888
Average	0.9614	0.6478	0.8059	0.9506	

Table 68. Results with MD-DTW with relative joint positions – MSRC-12 dataset

	Adaboost	**KNN (K=30)**	**Random Forests**	**SVM**	**Sum of the intervals during simulation**
Run 1	0.9412	0.6630	0.7379	0.9459	9376639
Run 2	0.8927	0.6556	0.7124	0.9321	9410222
Run 3	0.9136	0.6047	0.7883	0.9364	9510737
Average	0.9158	0.6411	0.7462	0.9381	

In Table 66, the difference between the runs is large. This is due to the large intervals that are generated while aligning with the MD-DTW, also described as a misalignment. Moreover, it is caused by the fact that the simulation algorithm contains randomness with all the methods (average, proportionally and random) when choosing the reference sample and the reference coordinate. By simply comparing the tables above, it is obvious that the alignment with DTW is much more stable than with MD-DTW and the performances are better. Consequently, as seen in this thesis, the DTW was the only algorithm that was implemented in the simulation algorithm.

b. Smoothing and noise reduction

We explore two well-known algorithms: the Median filter and Kalman filter, in order to know whether a smoothing of the generated angles' sequences can affect the results. We are aware there a lot of smoothing algorithms. Nevertheless, we study the ones that have been stated previously, since they are popular and easy to observe.

To improve the recordings visually, it is possible to smooth the simulated recording with a Median filter or a Kalman filter. Consequently, the recordings would appear more human-like. Yet, the variations disappear, resulting in the reappearance of the *superfeatures*.

1. *Median filter*

A large median filter removes discriminant data but improves the recordings visually. We do not go into many details to find the best window for the Median filter, we increase its size and observe the resulting recordings. We find that a window of size 7 or 9 frames does the job.

Again, 3 training sets of 50 recordings for every class are simulated, trained with a 1-vs-all strategy and the performances are evaluated with the F-Measure.

Table 69. Results of the classification after smoothing the simulation recordings with median filter (window=7) – CAP dataset

	Adaboost	**KNN (K=30)**	**Random Forests**	**SVM**
Run 1	0.8725	0.6829	0.8671	0.7460
Run 2	0.8794	0.6071	0.8652	0.7520
Run 3	0.8859	0.6207	0.8472	0.7559
Average	0.8793	0.6369	0.8599	0.7513
Average of classification without median filter (comparison purposes)	0.9192	0.6523	0.8557	0.7522

Table 70. Results of the classification after smoothing the simulated recordings with median filter (window=7) – MSRC-12 dataset

	Adaboost	**KNN (K=30)**	**Random Forests**	**SVM**
Run 1	0.9652	0.6592	0.7751	0.9065
Run 2	0.9614	0.6092	0.7197	0.8223
Run 3	0.9478	0.6592	0.7327	0.8774
Average	0.9581	0.6425	0.7425	0.8687
Average of classification without median filter (comparison purposes)	0.9614	0.6478	0.8059	0.9506

We compare the performances between the classification with and without the median filter, as in Table 69 & Table 70. We note that the difference is generally small (0.01%) when the classification with the median filter improves the performances. Consequently, the median filter was not implemented.

2. *Kalman filter*

As with the median filter, the parameters are set according to visual results, and the performances are displayed in Table 71. After simulating the CAP dataset 3 times and obtaining the average result, it is clear that the classification's performances did not improve. Therefore, the simulation has only been done 2 times to classify the MSRC-12 dataset, and the conclusion was the same: the Kalman filter did not improve the results.

Table 71. Results of the classification after smoothing the simulated recordings with Kalman filter – CAP dataset

	Adaboost	**KNN (K=30)**	**Random Forests**	**SVM**
Run 1	0.8456	0.5385	0.8000	0.7000
Run 2	0.8129	0.5505	0.7943	0.6610
Run 3	0.9252	0.6250	0.8082	0.6724
Average	0.8612	0.5713	0.8008	0.6778
Average of classification without the Kalman filter (comparison purposes)	0.9192	0.6523	0.8557	0.7522

Table 72. Results of the classification after smoothing the simulated recordings with Kalman filter – MSRC-12 dataset

	Adaboost	**KNN (K=30)**	**Random Forests**	**SVM**
Run 1	0.9407	0.5244	0.7552	0.8416
Run 2	0.9038	0.6036	0.7534	0.7447
Average	0.9222	0.5640	0.7543	0.7931
Average of classification without the Kalman filter (comparison purposes)	0.9614	0.6478	0.8059	0.9506

B. Asynchronous Late Fusion (Face Expression Recognition)

This is an example of a dataset where the results were not efficient with the ALF. In fact, the dataset that was classified is the CK dataset [166] [167] contains different facial emotions that have been performed by different subjects, by starting at a neutral face expression and finishing at the peak of the emotion. Moreover, the multiple recordings in the dataset have a length of 4 frames. Consequently, the recordings can only be partitioned into 2 parts and sequence of values is always incremental, making this dataset synchronous.

a. Feature Extraction

The features that were inputted to the classifier are stated briefly in Table 73.

Table 73. Features used as input to the early classifier

Features	Variations and comments
Difference between the first and last frame of the sum of the neighborhood for R, G, and B	Minimum, maximum
Difference between the first value and the last value of the sample (called f_{diff})	Signed, Unsigned
$\frac{F_{diff}(\text{variable A})}{F_{diff}(\text{variable B})}$	Signed, Unsigned Between all the variables A and B cited previously where A and B are different

b. Classification

We divide the dataset into 2 groups. In these groups, the performed emotions are slightly different:

- Classification 1: classifies the emotions: Angry, Contempt, Discuss
- Classification 2: classifies the emotions: Happiness, Surprise, Fear

As observed in Table 74, the result did not improve.

Table 74. Classification results of emotion dataset (F-Measure)

	Confidence coefficient used as input to the late classifier (For synchronous method only)	**Synchronous**	**Asynchronous**
Classification 1	Binary	0.64	0.56
	Real	0.57	-
Classification 2	Binary	0.74	0.69
	Real	0.76	-

X. APPENDIX II – ALF ADDITIONAL EXPERIMENTATIONS

A. Adaboost Asynchronous classification with different metrics and overlap sizes

a. Overlap: w=0.5

Table 75. F-MEASURE results with different metrics (Adaboost) – CAP dataset - *w=0.5*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.8381	0.8506	0.8506	0.8605	0.8506	0.8506	0.8605	0.8506	0.8506	0.8636
4	0.7009	0.7609	0.7609	0.7640	0.7368	0.7609	0.7727	0.7556	0.7391	0.7865
6	0.7791	0.8125	0.8125	0.8571	0.8211	0.8211	0.7957	0.8478	0.8172	0.8444
8	0.7440	0.8211	0.8211	0.7742	0.8723	0.8000	0.7527	0.8222	0.7789	0.8000
10	0.8387	0.7872	0.7872	0.8571	0.7872	0.8667	0.8791	0.8571	0.8571	0.7742
12	0.8512	0.8791	0.8791	0.8636	0.8696	0.8723	0.8636	0.8632	0.8889	0.8043
14	0.8255	0.8889	0.8889	0.8791	0.8889	0.8222	0.9032	0.8817	0.8182	0.8182
16	0.8381	0.8571	0.8571	0.8571	0.8478	0.7708	0.8000	0.8333	0.8315	0.8409
Average	0.8019	0.8322	0.8322	0.8391	0.8343	0.8206	0.8284	0.8389	0.8227	0.8165
Maximum	0.8512	0.8889	0.8889	0.8791	0.8889	0.8723	0.9032	0.8817	0.8889	0.8636
Minimum	0.7009	0.7609	0.7609	0.7640	0.7368	0.7609	0.7527	0.7556	0.7391	0.7742
Standard Deviation	0.0548	0.0447	0.0447	0.0439	0.0507	0.0417	0.0550	0.0382	0.0469	0.0310

Table 76. F-MEASURE results with different metrics (Adaboost) – CR dataset - *w=0.5*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.7921	0.7789	0.7789	0.8039	0.8039	0.7789	0.8081	0.7921	0.7789	0.7160
4	0.8247	0.8247	0.8163	0.8125	0.8000	0.7961	0.8247	0.8119	0.8283	0.8155
6	0.8632	0.8632	0.8200	0.8750	0.8632	0.8400	0.8632	0.8454	0.8163	0.8454
8	0.8200	0.8454	0.8155	0.8515	0.8113	0.8454	0.8247	0.8632	0.8333	0.8454
10	0.8132	0.8182	0.8041	0.8222	0.8163	0.8211	0.8182	0.8261	0.8352	0.7959
12	0.8043	0.8444	0.8400	0.8315	0.8571	0.8485	0.8352	0.8421	0.8182	0.7810
Average	0.8196	0.8291	0.8125	0.8328	0.8253	0.8217	0.8290	0.8301	0.8184	0.7999
Maximum	0.8632	0.8632	0.8400	0.8750	0.8632	0.8485	0.8632	0.8632	0.8352	0.8454
Minimum	0.7921	0.7789	0.7789	0.8039	0.8000	0.7789	0.8081	0.7921	0.7789	0.7160
Standard Deviation	0.0243	0.0294	0.0202	0.0264	0.0276	0.0286	0.0190	0.0255	0.0208	0.0485

Table 77. F-MEASURE results with different metrics (Adaboost) – SS dataset - *w=0.5*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636
4	0.8421	0.8267	0.8831	0.8267	0.8831	0.8421	0.8267	0.8267	0.8421	0.7945
6	0.8889	0.9014	0.8767	0.8889	0.8767	0.8889	0.9014	0.9014	0.9014	0.9041
8	0.9333	0.9333	0.9315	0.9189	0.9189	0.9333	0.9333	0.9333	0.9333	0.9315
10	0.9333	0.9459	0.9333	0.9459	0.9333	0.9333	0.9459	0.9333	0.9333	0.9459
12	0.8947	0.9189	0.8947	0.9041	0.8947	0.8947	0.9189	0.8947	0.8947	0.9189
14	0.8378	0.8378	0.8378	0.8378	0.8732	0.8378	0.8378	0.8378	0.8378	0.8493
16	0.9459	0.9315	0.9333	0.9315	0.9333	0.9459	0.9459	0.9315	0.9315	0.9315
Average	0.8300	0.8324	0.8318	0.8272	0.8346	0.8300	0.8342	0.8278	0.8297	0.8299
Maximum	0.9459	0.9459	0.9333	0.9459	0.9333	0.9459	0.9459	0.9333	0.9333	0.9459
Minimum	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636
Standard Deviation	0.1928	0.1946	0.1921	0.1920	0.1919	0.1928	0.1957	0.1922	0.1922	0.1951

b. Overlap: w=1

Table 78. F-MEASURE results with different metrics (Adaboost) – CAP dataset - *w=1*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.8315	0.8315	0.8315	0.8315	0.8315	0.8315	0.8315	0.8315	0.8315	0.8315
4	0.8352	0.8046	0.8222	0.8444	0.8222	0.8222	0.8315	0.8222	0.8352	0.7742
6	0.7727	0.7727	0.8000	0.8352	0.7727	0.8090	0.7727	0.8090	0.8000	0.8471
8	0. 8696	0.8817	0.8542	0.8421	0.8632	0.8367	0.8817	0.8298	0.8696	0.8817
10	0.8696	0.8571	0.8958	0.8571	0.8842	0.8696	0.8571	0.8696	0.8696	0.8636
12	0.8539	0.8539	0.8222	0.8636	0.8222	0.8132	0.8539	0.8090	0.8667	0.8387
14	0.8636	0.8636	0.8539	0.8636	0.8764	0.8539	0.8636	0.8636	0.8636	0.7955
16	0.8000	0.8315	0.7957	0.8315	0.7957	0.8352	0.7955	0.8222	0.8261	0.8444
Average	0.8385	0.8371	0.8344	0.8461	0.8335	0.8339	0.8359	0.8321	0.8453	0.8346
Maximum	0. 8696	0.8817	0.8958	0.8636	0.8842	0.8696	0.8817	0.8696	0.8696	0.8817
Minimum	0.7727	0.7727	0.7957	0.8315	0.7727	0.8090	0.7727	0.8090	0.8000	0.7742
Standard Deviation	0.0370	0.0351	0.0328	0.0136	0.0390	0.0203	0.0365	0.0229	0.0259	0.0349

Table 79. F-MEASURE results with different metrics (Adaboost) – CR dataset - *w=1*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.6593	0.6593	0.6522	0.5647	0.6737	0.6400	0.7059	0.6250	0.6222	0.5773
4	0.5977	0.5977	0.6667	0.6087	0.6364	0.7158	0.7835	0.6522	0.6364	0.6947
6	0.7191	0.7191	0.7033	0.6889	0.6809	0.7273	0.8350	0.7400	0.7253	0.7000
8	0.7234	0.7234	0.7475	0.7143	0.7400	0.7273	0.7955	0.7327	0.7097	0.7400
10	0.7327	0.7327	0.7129	0.6667	0.7200	0.7083	0.7573	0.7071	0.6800	0.7083
12	0.7083	0.7083	0.7010	0.6598	0.6869	0.6804	0.7692	0.6939	0.6667	0.6804
Average	0.6901	0.6901	0.6973	0.6505	0.6896	0.6998	0.7744	0.6918	0.6734	0.6835
Maximum	0.7327	0.7327	0.7475	0.7143	0.7400	0.7273	0.8350	0.7400	0.7253	0.7400
Minimum	0.5977	0.5977	0.6522	0.5647	0.6364	0.6400	0.7059	0.6250	0.6222	0.5773
Standard Deviation	0.0521	0.0521	0.0340	0.0548	0.0364	0.0340	0.0429	0.0453	0.0402	0.0557

Table 80. F-MEASURE results with different metrics (Adaboost) – SS dataset - *w=1*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636	0.2928	0.3636	0.3636
4	0.7945	0.8267	0.8831	0.8267	0.8831	0.8421	0.8267	0.7730	0.8421	0.7945
6	0.9041	0.9014	0.8767	0.8889	0.8767	0.8889	0.9014	0.8799	0.9014	0.9041
8	0.9315	0.9333	0.9315	0.9189	0.9189	0.9333	0.9333	0.9135	0.9333	0.9315
10	0.9459	0.9459	0.9333	0.9459	0.9333	0.9333	0.9459	0.9135	0.9333	0.9459
12	0.9189	0.9189	0.8947	0.9041	0.8947	0.8947	0.9189	0.8614	0.8947	0.9189
14	0.8493	0.8378	0.8378	0.8378	0.8732	0.8378	0.8378	0.7896	0.8378	0.8493
16	0.9315	0.9315	0.9333	0.9315	0.9333	0.9459	0.9459	0.9144	0.9315	0.9315
Average	0.8299	0.8324	0.8318	0.8272	0.8346	0.8300	0.8342	0.7923	0.8297	0.8299
Maximum	0.9459	0.9459	0.9333	0.9459	0.9333	0.9459	0.9459	0.9144	0.9333	0.9459
Minimum	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636	0.3636	0.2928	0.3636	0.3636
Standard Deviation	0.1951	0.1946	0.1921	0.1920	0.1919	0.1928	0.1957	0.2093	0.1922	0.1951

B. KNN Asynchronous classification with different metrics and overlap sizes

a. Overlap: w=0

Table 81. F-MEASURE results with different metrics (KNN) – CAP dataset - *w=0*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	MCC1	F-Measure	Model=1	MCC (Additional Dataset)
2	0.6400	0.6575	0.6988	0.6667	0.7294	0.6667	0.6667	0.6842	0.6753	0.6111	0.6579	0.4966
4	0.6923	0.7013	0.7250	0.6753	0.7250	0.7529	0.6842	0.7529	0.7500	0.7013	0.7674	0.6584
6	0.7586	0.7143	0.7778	0.7442	0.7778	0.7500	0.7381	0.7273	0.7416	0.7500	0.7356	0.7419
8	0.7907	0.7619	0.7955	0.7765	0.7955	0.7640	0.7407	0.7640	0.7727	0.8046	0.7273	0.7717
10	0.7816	0.7529	0.7674	0.7674	0.7816	0.7778	0.7529	0.7727	0.7816	0.7816	0.7778	0.7889
12	0.7765	0.7619	0.7765	0.7529	0.7674	0.7442	0.7619	0.7619	0.7381	0.7674	0.7442	0.8177
14	0.8046	0.7765	0.7765	0.7765	0.7619	0.7529	0.7765	0.7907	0.7619	0.8046	0.7529	0.8111
16	0.7529	0.7586	0.7619	0.7586	0.7619	0.7294	0.7674	0.7529	0.7191	0.7529	0.6988	0.8136
Average	0.7497	0.7356	0.7599	0.7398	0.7626	0.7422	0.7361	0.7508	0.7425	0.7467	0.7327	0.7375
Maximum	0.8046	0.7765	0.7955	0.7765	0.7955	0.7778	0.7765	0.7907	0.7816	0.8046	0.7778	0.8177
Minimum	0.6400	0.6575	0.6988	0.6667	0.7250	0.6667	0.6667	0.6842	0.6753	0.6111	0.6579	0.4966
Standard Deviation	0.0559	0.0407	0.0319	0.0439	0.0245	0.0336	0.0398	0.0324	0.0337	0.0642	0.0388	0.1107
	Synchronous (k=3)											0.7045

Table 82. F-MEASURE results with different metrics (KNN) – CR dataset - *w=0*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.6053	0.5783	0.6098	0.6444	0.6341	0.7059	0.6154	0.6512	0.6588	0.6744
4	0.7059	0.6512	0.7556	0.6813	0.7174	0.7556	0.7294	0.7312	0.7045	0.7391
6	0.7045	0.6818	0.6809	0.6818	0.6737	0.6882	0.6897	0.6809	0.6742	0.6882
8	0.7033	0.6742	0.7033	0.6667	0.6809	0.6667	0.6591	0.6737	0.6957	0.6522
10	0.7273	0.7191	0.7273	0.7356	0.7356	0.7253	0.7059	0.7609	0.7333	0.7391
12	0.7174	0.7253	0.7097	0.7333	0.7174	0.7033	0.7045	0.7111	0.7097	0.7191
Average	0.6939	0.6716	0.6977	0.6905	0.6932	0.7075	0.6840	0.7015	0.6960	0.7020
Maximum	0.7273	0.7253	0.7556	0.7356	0.7356	0.7556	0.7294	0.7609	0.7333	0.7391
Minimum	0.6053	0.5783	0.6098	0.6444	0.6341	0.6667	0.6154	0.6512	0.6588	0.6522
Standard Deviation	0.0444	0.0536	0.0498	0.0367	0.0374	0.0306	0.0408	0.0406	0.0265	0.0360

Table 83. F-MEASURE results with different metrics (KNN) – SS dataset - *w=0*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.7941	0.7813	0.8000	0.8182	0.8000	0.7826	0.7879	0.8116	0.8235	0.7714
4	0.9189	0.9189	0.9189	0.9014	0.9189	0.8919	0.9041	0.9041	0.9167	0.8947
6	0.8780	0.9114	0.8675	0.9000	0.8706	0.8810	0.8750	0.8571	0.8780	0.8706
8	0.8810	0.8500	0.8706	0.8148	0.8706	0.8916	0.8250	0.8706	0.8916	0.8889
10	0.7711	0.7805	0.8312	0.7901	0.9024	0.9024	0.7619	0.8810	0.8000	0.8780
12	0.8675	0.8974	0.9250	0.8608	0.9250	0.9250	0.8642	0.9250	0.9000	0.9367
14	0.8276	0.7912	0.9000	0.7955	0.9114	0.8974	0.7692	0.8780	0.8471	0.8974
16	0.9383	0.8889	0.9620	0.8571	0.9744	0.9744	0.8571	0.9620	0.9383	0.9620
Average	0.8596	0.8524	0.8844	0.8422	0.8967	0.8933	0.8306	0.8862	0.8744	0.8875
Maximum	0.9744	0.9189	0.9620	0.9014	0.9744	0.9744	0.9041	0.9620	0.9383	0.9620
Minimum	0.7711	0.7805	0.8000	0.7901	0.8000	0.7826	0.7619	0.8116	0.8000	0.7714
Standard Deviation	0.0661	0.0601	0.0528	0.0441	0.0511	0.0536	0.0528	0.0452	0.0474	0.0560

b. Overlap: w=0.5

Table 84. F-MEASURE results with different metrics (KNN) – CAP dataset - *w=5*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.7073	0.7000	0.6829	0.6988	0.6747	0.6667	0.7000	0.6914	0.6667	0.6341
4	0.6500	0.6667	0.7059	0.6329	0.6905	0.7073	0.6835	0.7229	0.7073	0.7073
6	0.7556	0.6667	0.7556	0.6582	0.7556	0.7294	0.6835	0.6977	0.7294	0.7381
8	0.7126	0.6977	0.7586	0.6977	0.7529	0.7442	0.7317	0.7442	0.7442	0.7442
10	0.7640	0.7778	0.7955	0.7865	0.7955	0.7727	0.7640	0.7727	0.7727	0.7586
12	0.7470	0.7000	0.7529	0.7000	0.7381	0.7229	0.6835	0.7317	0.7229	0.7229
14	0.7816	0.8046	0.7586	0.7907	0.7586	0.7529	0.8046	0.7765	0.7529	0.7529
16	0.7442	0.7381	0.7586	0.7381	0.7586	0.7500	0.7381	0.7586	0.7500	0.7273
Average	0.7328	0.7189	0.7461	0.7129	0.7406	0.7308	0.7236	0.7370	0.7308	0.7232
Maximum	0.7816	0.8046	0.7955	0.7907	0.7955	0.7727	0.8046	0.7765	0.7727	0.7586
Minimum	0.6500	0.6667	0.6829	0.6329	0.6747	0.6667	0.6835	0.6914	0.6667	0.6341
Standard Deviation	0.0416	0.0504	0.0352	0.0562	0.0395	0.0328	0.0445	0.0321	0.0328	0.0396

Table 85. F-MEASURE results with different metrics (KNN) – CR dataset - *w=0.5*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.6118	0.5517	0.6265	0.5476	0.6506	0.7111	0.5301	0.6593	0.5747	0.6374
4	0.6824	0.6747	0.7033	0.6824	0.6889	0.7368	0.6341	0.7312	0.7333	0.7312
6	0.6531	0.6383	0.7216	0.6667	0.7143	0.7255	0.6316	0.6990	0.6667	0.7255
8	0.6882	0.7010	0.6869	0.7143	0.6939	0.7	0.6875	0.6931	0.6804	0.6800
10	0.6735	0.7083	0.6667	0.6667	0.6667	0.6596	0.7021	0.6465	0.6875	0.6667
12	0.6809	0.6737	0.6667	0.6947	0.6598	0.6947	0.6522	0.6804	0.6737	0.6947
Average	0.6649	0.6580	0.6786	0.6621	0.6790	0.7046	0.6396	0.6849	0.6694	0.6892
Maximum	0.6882	0.7083	0.7216	0.7143	0.7143	0.7368	0.7021	0.7312	0.7333	0.7312
Minimum	0.6118	0.5517	0.6265	0.5476	0.6506	0.6596	0.5301	0.6465	0.5747	0.6374
Standard Deviation	0.0288	0.0576	0.0333	0.0589	0.0240	0.0271	0.0607	0.0302	0.0520	0.0357

Table 86. F-MEASURE results with different metrics (KNN) – SS dataset - *w=0.5*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.7222	0.6269	0.7397	0.7143	0.7222	0.7568	0.7143	0.7467	0.6471	0.7368
4	0.7945	0.7143	0.7945	0.7945	0.7778	0.8267	0.7778	0.8108	0.7429	0.8421
6	0.8974	0.7945	0.8974	0.8919	0.8974	0.9351	0.8767	0.9351	0.8500	0.9114
8	0.8434	0.8919	0.8235	0.8434	0.8140	0.8861	0.8537	0.8750	0.8675	0.8861
10	0.8043	0.8750	0.8261	0.8043	0.8261	0.8315	0.8043	0.8352	0.8506	0.8571
12	0.8537	0.8222	0.8642	0.8000	0.8780	0.8642	0.8000	0.8750	0.8780	0.8608
14	0.8090	0.8293	0.8276	0.7527	0.8372	0.8140	0.7778	0.8090	0.8276	0.8537
16	0.8352	0.7609	0.8837	0.8261	0.8837	0.8941	0.8132	0.9157	0.8409	0.9383
Average	0.8200	0.7894	0.8321	0.8034	0.8296	0.8510	0.8022	0.8503	0.8131	0.8608
Maximum	0.8974	0.8919	0.8974	0.8919	0.8974	0.9351	0.8767	0.9351	0.8780	0.9383
Minimum	0.7222	0.6269	0.7397	0.7143	0.7222	0.7568	0.7143	0.7467	0.6471	0.7368
Standard Deviation	0.0514	0.0873	0.0508	0.0542	0.0591	0.0555	0.0496	0.0620	0.0788	0.0597

c. Overlap: w=1

Table 87. F-MEASURE results with different metrics (KNN) – CAP dataset - *w=1*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.6500	0.6582	0.6506	0.6410	0.6506	0.6914	0.6494	0.6914	0.6500	0.6341
4	0.7160	0.6829	0.7442	0.6914	0.7442	0.7229	0.6750	0.7073	0.6988	0.7073
6	0.7527	0.7209	0.7333	0.7209	0.7333	0.7143	0.7209	0.7059	0.7692	0.7381
8	0.7391	0.6966	0.7391	0.6966	0.7391	0.7126	0.6818	0.7045	0.7253	0.7442
10	0.7609	0.7556	0.7742	0.7556	0.7742	0.7674	0.7556	0.7674	0.7692	0.7586
12	0.7191	0.7209	0.7191	0.7294	0.7191	0.7209	0.7356	0.7126	0.7045	0.7229
14	0.7126	0.7209	0.7045	0.7209	0.7045	0.7294	0.7294	0.7209	0.7045	0.7529
16	0.7356	0.6977	0.7500	0.6977	0.7500	0.7500	0.7209	0.7500	0.7356	0.7273
Average	0.7233	0.7067	0.7269	0.7067	0.7269	0.7261	0.7086	0.7200	0.7197	0.7232
Maximum	0.7609	0.7556	0.7742	0.7556	0.7742	0.7674	0.7556	0.7674	0.7692	0.7586
Minimum	0.6500	0.6582	0.6506	0.6410	0.6506	0.6914	0.6494	0.6914	0.6500	0.6341
Standard Deviation	0.0343	0.0295	0.0371	0.0339	0.0371	0.0235	0.0359	0.0257	0.0395	0.0396

Table 88. F-MEASURE results with different metrics (KNN) – CR dataset - *w=1*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.6593	0.6593	0.6522	0.5647	0.6737	0.6400	0.6047	0.6250	0.6222	0.5773
4	0.5977	0.5977	0.6667	0.6087	0.6364	0.7158	0.5909	0.6522	0.6364	0.6947
6	0.7191	0.7191	0.7033	0.6889	0.6809	0.7273	0.6742	0.7400	0.7253	0.7000
8	0.7234	0.7234	0.7475	0.7143	0.7400	0.7273	0.7045	0.7327	0.7097	0.7400
10	0.7327	0.7327	0.7129	0.6667	0.7200	0.7083	0.7500	0.7071	0.6800	0.7083
12	0.7083	0.7083	0.7010	0.6598	0.6869	0.6804	0.6526	0.6939	0.6667	0.6804
Average	0.6901	0.6901	0.6973	0.6505	0.6896	0.6998	0.6628	0.6918	0.6734	0.6835
Maximum	0.7327	0.7327	0.7475	0.7143	0.7400	0.7273	0.7500	0.7400	0.7253	0.7400
Minimum	0.5977	0.5977	0.6522	0.5647	0.6364	0.6400	0.5909	0.6250	0.6222	0.5773
Standard Deviation	0.0521	0.0521	0.0340	0.0548	0.0364	0.0340	0.0602	0.0453	0.0402	0.0557

Table 89. F-MEASURE results with different metrics (KNN) – SS dataset - *w=1*

Number of parts	MCC	RR	PP	R	P	A	Y	HALF	F-Measure	Model=1
2	0.6571	0.6667	0.6944	0.6667	0.6849	0.7027	0.6667	0.6757	0.7059	0.6087
4	0.7778	0.7500	0.7671	0.7671	0.7467	0.7397	0.7246	0.7397	0.7708	0.7353
6	0.8312	0.8421	0.8684	0.8312	0.8684	0.8684	0.8462	0.8684	0.8039	0.7692
8	0.9000	0.8889	0.8780	0.8780	0.8889	0.9000	0.8675	0.9000	0.8222	0.8182
10	0.8235	0.8537	0.8434	0.8095	0.8537	0.8642	0.8193	0.8537	0.7767	0.8000
12	0.8916	0.8780	0.8605	0.8571	0.8605	0.8571	0.8675	0.8810	0.7692	0.7879
14	0.8000	0.7865	0.8372	0.7865	0.8372	0.8235	0.7865	0.8276	0.8974	0.8235
16	0.7629	0.7872	0.8352	0.7789	0.8444	0.8605	0.7551	0.8444	0.9060	0.7761
Average	0.8055	0.8066	0.8230	0.7969	0.8231	0.8270	0.7917	0.8238	0.8065	0.7649
Maximum	0.9000	0.8889	0.8780	0.8780	0.8889	0.9000	0.8675	0.9000	0.9060	0.8235
Minimum	0.6571	0.6667	0.6944	0.6667	0.6849	0.7027	0.6667	0.6757	0.7059	0.6087
Standard Deviation	0.0774	0.0746	0.0620	0.0653	0.0700	0.0692	0.0724	0.0769	0.0677	0.0691

d. Resume: Different overlap sizes

Table 90. Different overlap sizes (F-MEASURE) – MCC metric - KNN

	CAP			CR			SS		
Number of parts	**w=0**	**w=1.5**	**w=2**	**w=0**	**w=1.5**	**w=2**	**w=0**	**w=1.5**	**w=2**
2	0.6400	0.7073	0.6500	0.6053	0.6118	0.6593	0.7941	0.6571	0.6571
4	0.6923	0.6500	0.7160	0.7356	0.6824	0.5977	0.9189	0.7778	0.7778
6	0.7586	0.7556	0.7527	0.7045	0.6531	0.7191	0.8780	0.8312	0.8312
8	0.7907	0.7126	0.7391	0.7033	0.6882	0.7234	0.8810	0.9000	0.9000
10	0.7816	0.7640	0.7609	0.7310	0.6735	0.7327	0.7711	0.8235	0.8235
12	0.7765	0.7470	0.7191	0.7174	0.6809	0.7083	0.8675	0.8916	0.8916
14	0.8046	0.7816	0.7126				0.8276	0.8000	0.8000
16	0.7529	0.7442	0.7356				0.9383	0.7629	0.7629
Average	0.7497	0.7328	0.7233	0.6995	0.6649	0.6901	0.8596	0.8055	0.8055
Maximum	0.8046	0.7816	0.7609	0.7356	0.6882	0.7327	0.9744	0.9000	0.9000
Minimum	0.6400	0.6500	0.6500	0.6053	0.6118	0.5977	0.7711	0.6571	0.6571
Standard Deviation	0.0559	0.0416	0.0343	0.0480	0.0288	0.0521	0.0661	0.0774	0.0774

After performing all the asynchronous tests, we tried to optimize even more the choice of the K parameters. The best results for multiple values of K are noted in Table 91. The resulting value is compared to the ones in the asynchronous solution and shows that the performances are still better when implementing the ALF.

Table 91. Finding the best K for KNN

	K	**F-Measure with synchronous classification**	**With ALF, F-Measure average of 3 results (K=9)**
CAP	3	0.7045	0.7467
	5	0.6914	
	7	0.5714	
	9	0.6494	
SS	3	0.7123	0.8956
	5	0.6944	
	7	0.7123	
	9	0.6933	
CR	3	0.5747	0.6809
	5	0.6353	
	7	0.5934	
	9	0.5909	

C. Power consumption

In VI.C, we have applied the ALF on a power consumption dataset with asynchronous properties to show that the asynchronous solution can be applied to other fields. The remainder of the results that were displayed are shown in the following table:

Table 92. Classification results of the power consumption dataset (F-Measure)

Number of parts	Weekend detection	Day detection 2007-2010	Day detection 2007
2	0.470588	0.223654	0.253308
3	0.464497	0.275727	0.238361
4	0.518033	0.240567	0.240458
5	0.642202	0.243992	0.210526
6	0.49387	0.258378	0.244009
7	0.644483	0.256432	0.27051
8	0.512241	0.280737	0.238411
9	0.619231	0.252029	0.26087
10	0.611212	0.265407	0.304545
11	0.614286	0.273966	0.25
12	0.698835	0.255739	0.266055
13	0.624561	0.259477	0.299287
14	0.579545	0.251242	0.299065
15	0.698925	0.278607	0.262626
16	0.679104	0.255697	0.265306
17	0.670251	0.248118	0.2746
18	0.629252	0.256876	0.293976
19	0.631193	0.252361	0.246787
20	0.555347	0.281071	0.272277

XI. Appendix III – Discussion – Performance Measure

Throughout this work/thesis, a lot of discussions were conducted around the performance measures. Every measure has its own properties: the F-measure and the MCC output a high value when the tested classes perform well and penalize the result when the classes have been misclassified but do not attribute exactly the same weight to the classes, as opposed to the Half Total Error Rate (HER). The measures are numerous, and all of them can be proved inefficient. We give some examples below.

We consider three 1-vs-all classification problems where, of course, the number of positives is smaller than the number of negatives as in Table 93 (and as seen throughout this thesis):

Table 93. Comparison of different performance metrics

	Positive	**Negative**	**MCC**	**F-Measure**	**HER**
Problem 1	54/74	416/444	0.6391	0.6923	0.8333
Problem 2	40/74	440/444	0.6671	0.6780	0.7658
Problem 3	0/74	440/444	-0.0360	0	0.4955

The F-Measure and the MCC can emphasize on the class that contains the largest number of samples. As for the Half Total Error Rate, it does not penalize the results if the classification fails to detect a certain label completely.

Some performance measures such as the recall only evaluate one class and are usually accompanied by the precision metric, or combined into a final value (F-Measure, Youden's Index...). Hence, we did not agree on a performance measure that is able to reveal the true nature of the results with one value only. In the end, we adopted the F-Measure as a performance measure since it is one of the most known in the research field.

Finally, some studies consider the confusion matrix to compare the results. Nevertheless, this would not be really efficient for us since the number of tests is large. For informational purposes, we display in the appendix some of the results as confusion matrices.

Since we adopt a 1-vs-all strategy in most of the classifications, to convert the results into a confusion matrix, we are forced to introduce an "unknown" label for every classification. Some results for the classification of the CAP dataset are compiled below into a confusion matrix:

Table 94. CAP dataset, training with real actions

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	100	0	0	0	0	0	0
2	0	100	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	100	0	0	0
5	0	0	0	0	100	0	0
6	0	0	0	0	0	100	0
7	0	0	0	0	0	0	100
-1	0	0	100	0	0	0	0

Table 95. CAP dataset, training with simulated actions

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	100	0	0	0	0	0	0
2	0	100	0	0	0	0	0
3	0	0	100	0	0	0	0
4	0	0	0	100	0	0	0
5	37.5	0	0	0	62.5	0	0
6	0	0	0	0	0	100	0
7	0	0	0	0	0	0	100
-1	0	0	0	0	0	0	0

The tables containing the classification results obtained with the asynchronous dataset are displayed below.

Table 96. CAP dataset, ALF – 2 parts

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	100	0	0	0	0	0	0
2	0	100	0	0	0	0	0
3	0	0	100	0	0	0	0
4	0	0	0	100	0	0	0
5	0	0	0	0	100	0	0
6	0	0	0	0	0	100	0
7	25	0	0	0	0	25	50
-1	0	0	0	0	0	0	0

Table 97. CAP dataset, ALF – 4 parts

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	85.71	0	14.29	0	0	0	0
2	0	100	0	0	0	0	0
3	0	0	100	0	0	0	0
4	0	0	14.29	85.71	0	0	0
5	0	0	0	0	100	0	0
6	0	0	0	0	0	87.5	12.5
7	0	0	0	0	0	0	100
-1	0	0	0	0	0	0	0

Table 98. CAP dataset, ALF – 6 parts

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	100	0	0	0	0	0	0
2	0	100	0	0	0	0	0
3	0	0	100	0	0	0	0
4	0	0	0	100	0	0	0
5	0	0	0	0	100	0	0
6	0	0	0	0	0	100	0
7	0	0	0	0	0	20	80
-1	0	0	0	0	0	0	0

Table 99. CAP dataset, ALF – 8 parts

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	100	0	0	0	0	0	0
2	0	100	0	0	0	0	0
3	0	0	100	0	0	0	0
4	0	0	22.22	77.78	0	0	0
5	0	0	0	28.57	71.43	0	0
6	0	0	0	0	0	100	0
7	25	0	0	0	0	33.33	66.67
-1	0	0	0	0	0	0	0

Table 100. CAP dataset, ALF – 2 parts

	1	2	3	4	5	6	7
1	100	0	0	0	0	0	0
2	0	100	0	0	0	0	0
3	0	0	66.67	0	0	0	0
4	0	0	0	33.33	0	0	0
5	0	0	12.5	87.5	100	0	0
6	0	0	0	0	0	100	0
7	25	0	0	0	0	0	100
-1	0	0	0	0	0	0	0

Table 101. CAP dataset, ALF – 4 parts

	1	2	3	4	5	6	7
1	100	0	0	0	0	0	0
2	0	100	0	0	0	0	0
3	0	0	100	0	0	0	0
4	0	0	16.67	83.33	0	0	0
5	0	0	0	0	100	0	0
6	0	0	0	0	0	100	0
7	0	0	0	0	0	25	75
-1	0	0	0	0	0	0	0

Table 102. CAP dataset, ALF – 6 parts

	1	2	3	4	5	6	7
1	100	0	0	0	0	0	0
2	0	100	0	0	0	0	0
3	0	0	66.67	33.33	0	0	0
4	0	0	0	100	0	0	0
5	0	0	0	0	100	0	0
6	0	0	0	0	0	100	0
7	25	0	0	0	0	25	50
-1	0	0	0	0	0	0	0

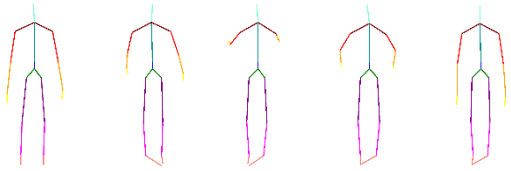
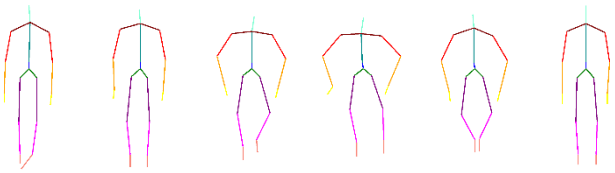
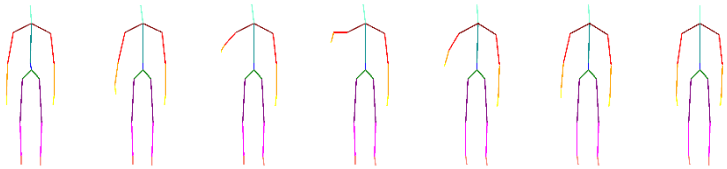
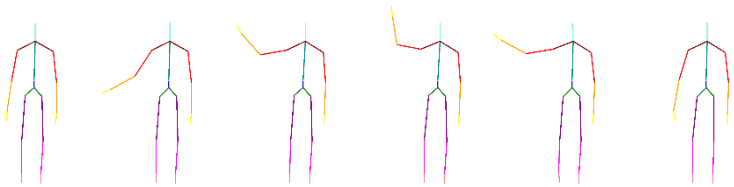
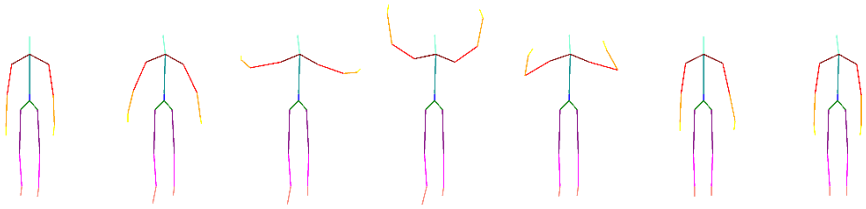
Table 103. CAP dataset, ALF – 8 parts

	1	2	3	4	5	6	7
1	100	0	0	0	0	0	0
2	0	100	0	0	0	0	0
3	0	0	100	0	0	0	0
4	0	0	0	100	0	0	0
5	0	0	0	0	100	0	0
6	0	0	0	0	0	100	0
7	25	0	0	0	0	60	40
-1	0	0	0	0	0	0	0

XII. Appendix IV – Datasets

Due to graphical constraints, we only display the frames that we judge as the most relevant for an action.

Table 104. Custom dataset called CAPtured dataset (CAP)

Action	Frames
2 hands up	
Crouch	
Raise right hand up	
Right Hand Wave	
Surrender	

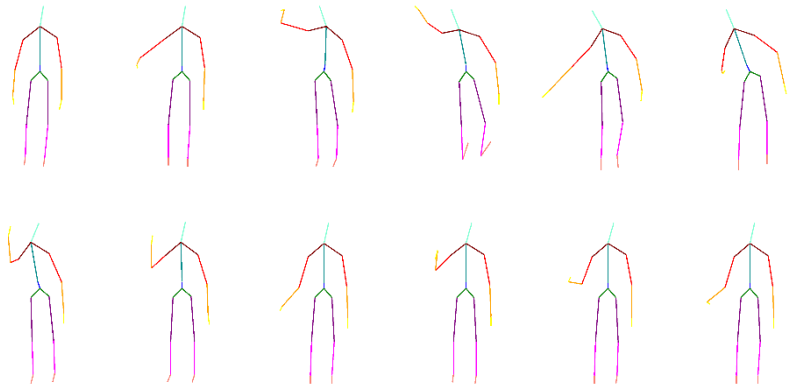
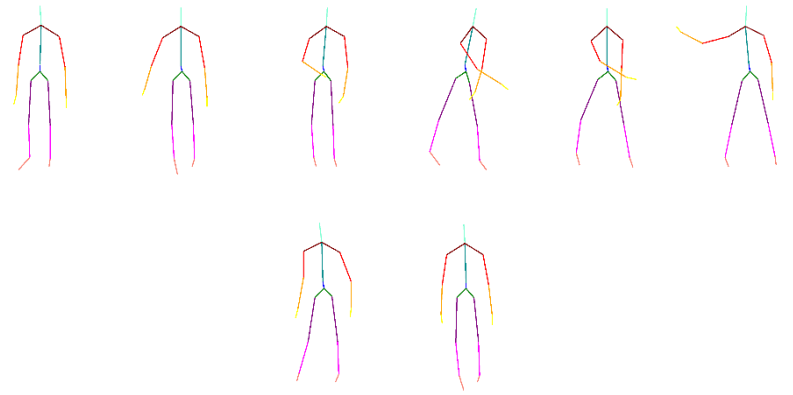
Tennis Forehand Drive	
Tennis Backhand Drive	

Table 105. Custom dataset with right-hand wave confusion (CR)

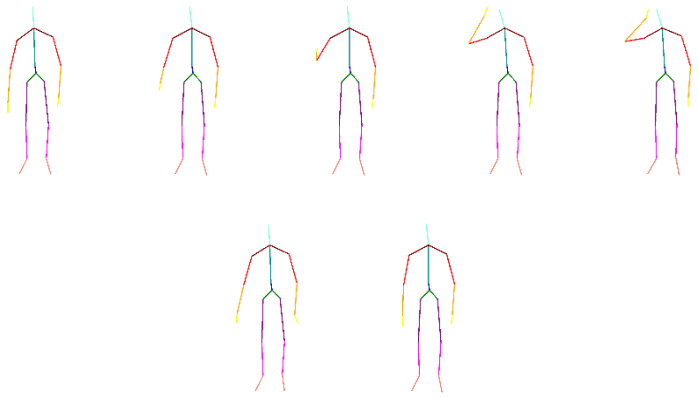
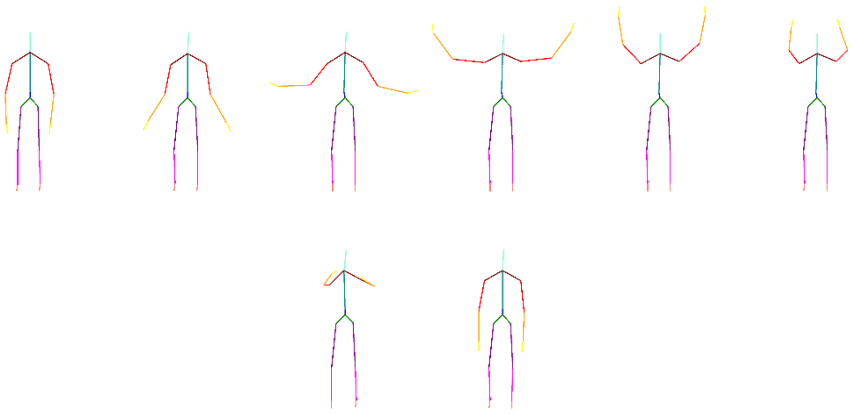
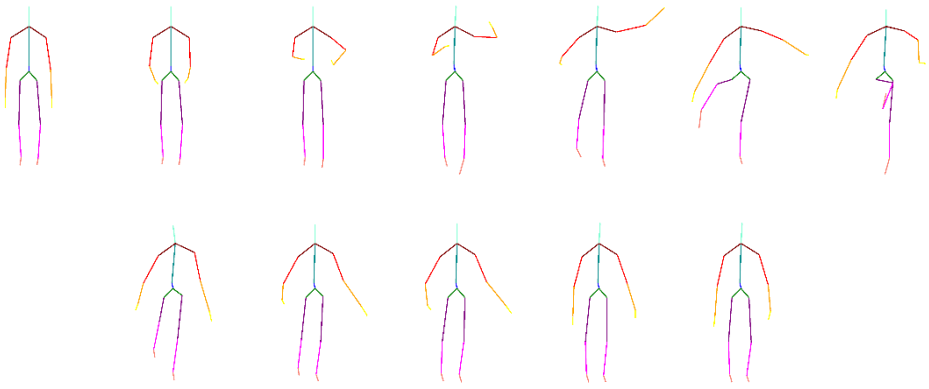
Action	Frames
Right-hand wave A	
Raise Right hand up	Same as in Table 104
Right-hand wave B	Same as in Table 104
Surrender	Same as in Table 104
Tennis Forehand Drive	Same as in Table 104
Tennis Backhand Drive	Same as in Table 104

Table 106. Custom dataset with Swimming and Soccer (SS)

Action	Frames
Swimming Crawl	
Swimming Butterfly	
Soccer	

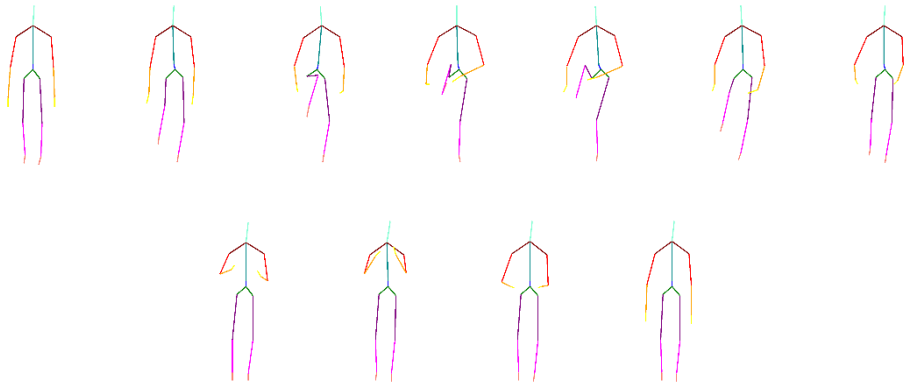
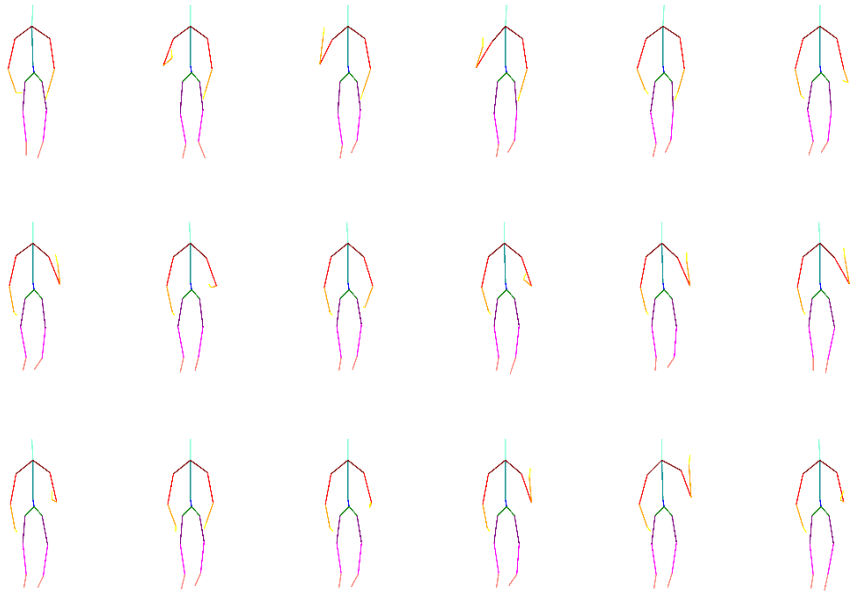
Not Soccer	

Table 107. Custom dataset with right hand up & left hand up (RL)

Action	**Frames**
Right, Left, Left, Left	

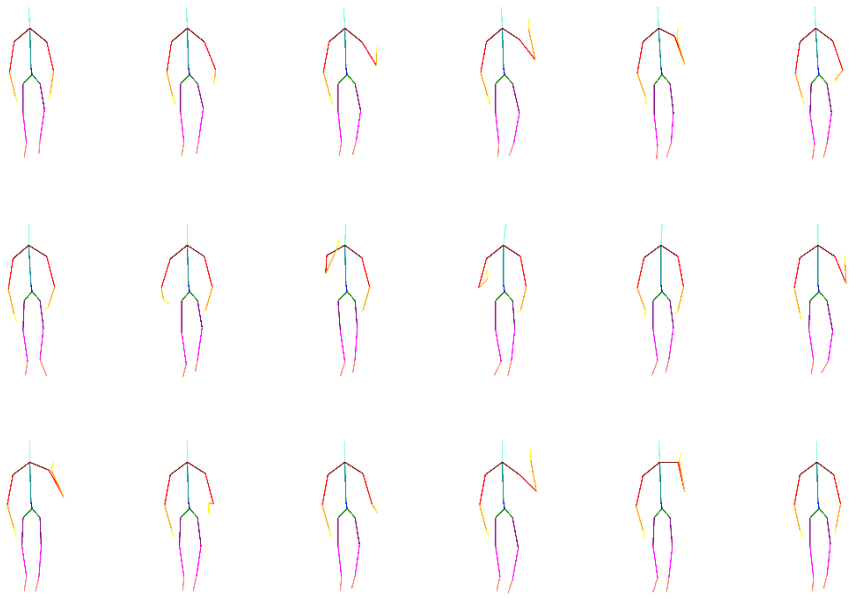
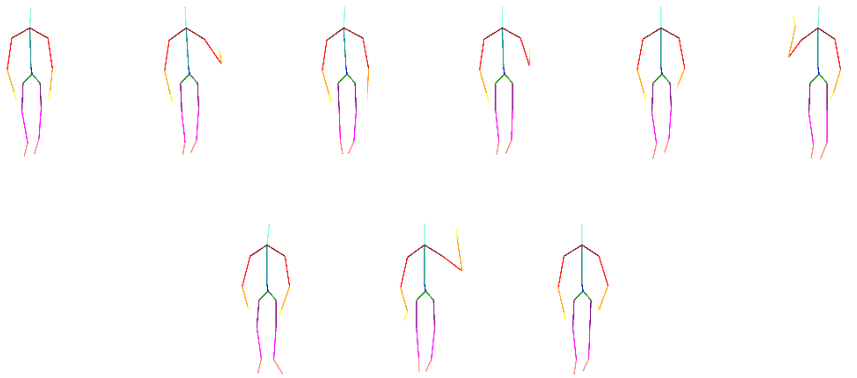
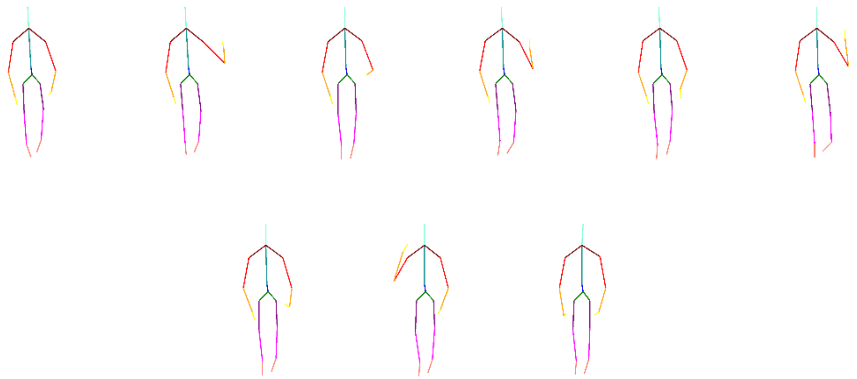
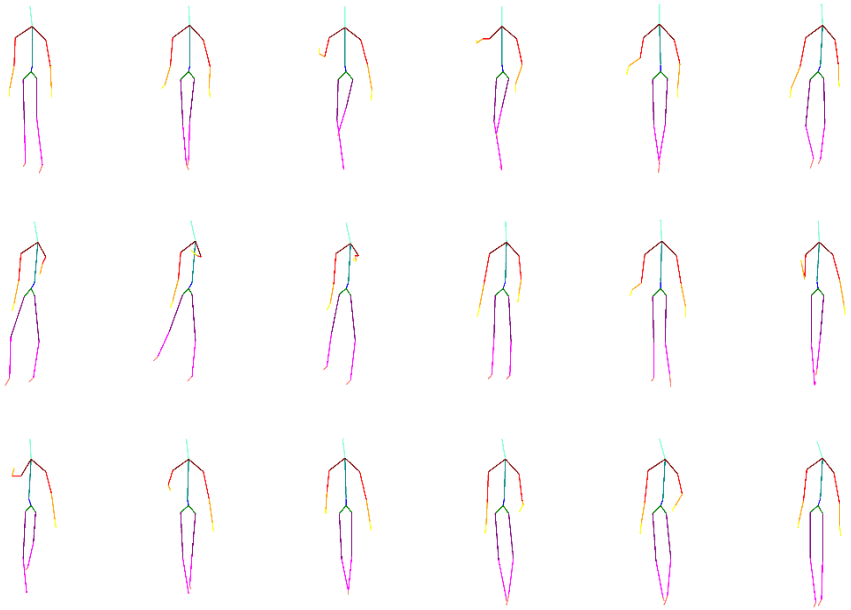
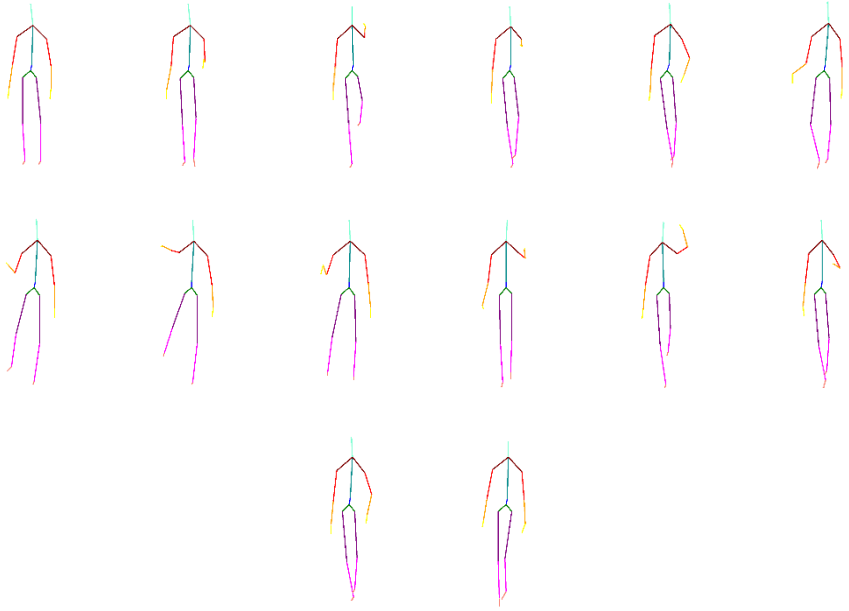
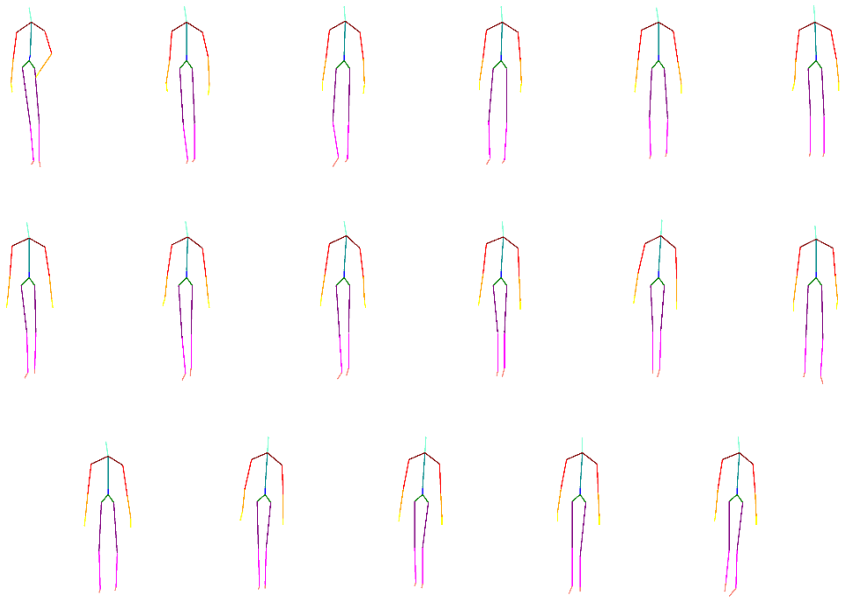
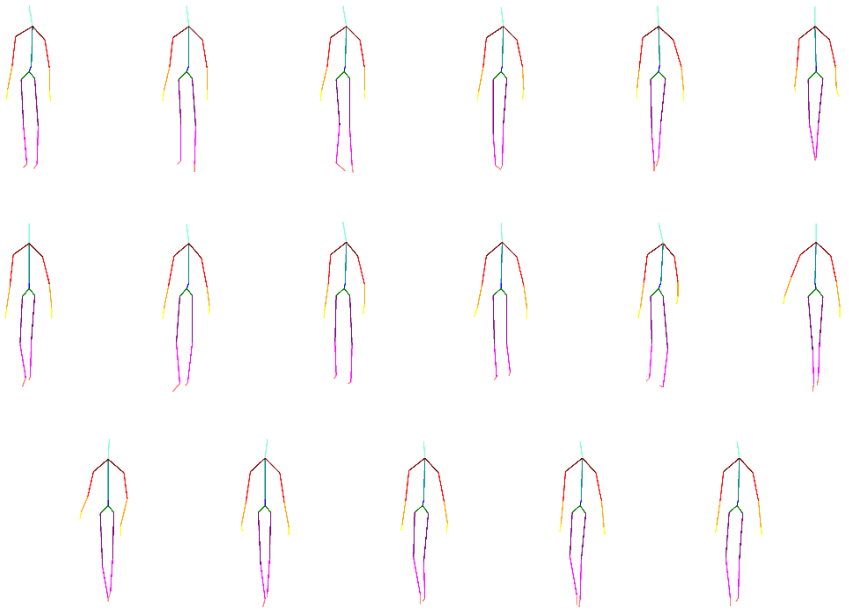
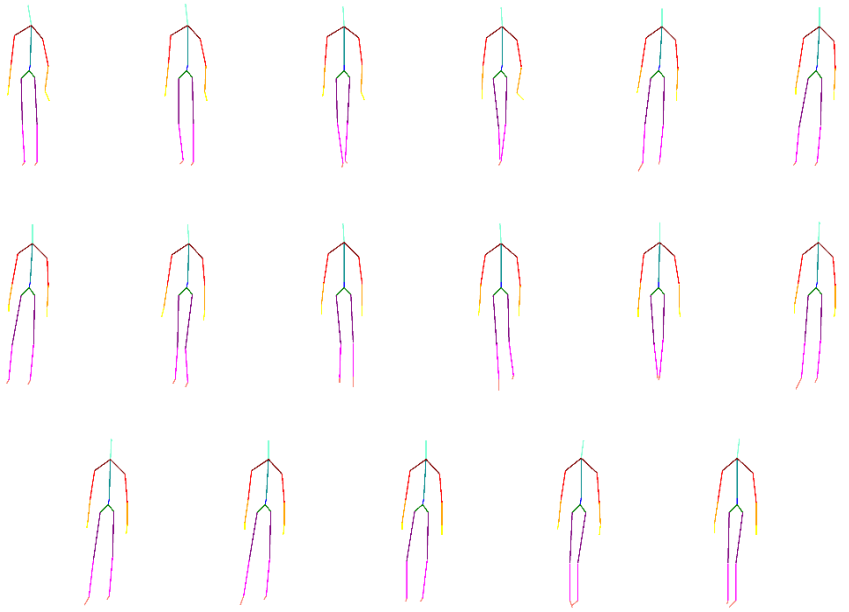
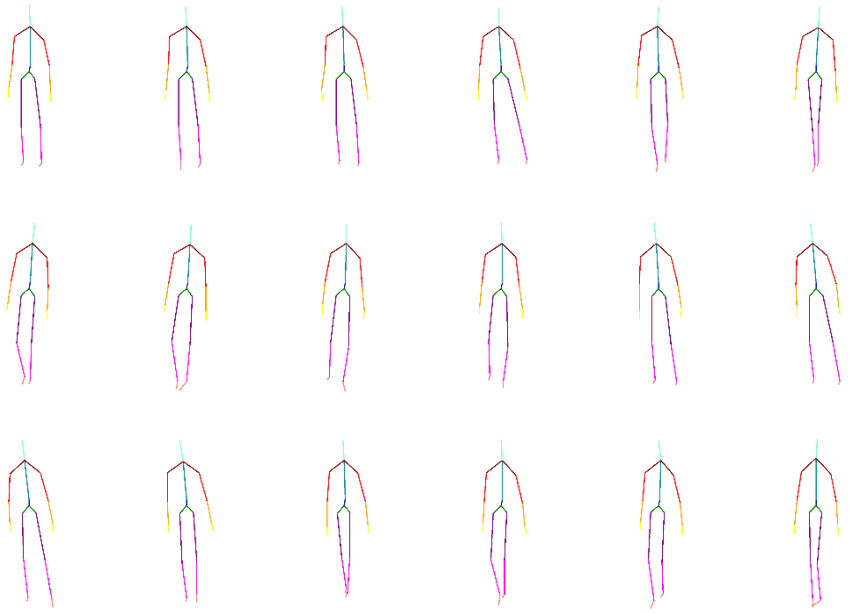
Left, Right, Left, Left	
Left, Left, Right, Left	
Left, Left, Left, Right	

Table 108. Custom gait dataset

Action	Frames
Army march	
Incorrect army march	

Parkinsonian-like shuffling	
Neurological Experiment	

Normal gait	
Right leg fracture	

Left leg fracture	

XIII. APPENDIX V - OPEN SOURCE CONTRIBUTIONS

As part of the work on the Joint Angles and the simulation algorithm, we uploaded 2 applications that convert the joint coordinates to angles, as well as an action viewer and a part of the CAP dataset, as open source, online at *http://computing-technologies.com/action_viewer*.

Knowing that the action recognition community (from the Microsoft Kinect) is small, the number of downloads and interests in the uploaded software was interesting. The software was downloaded more than 190 times within 1 month of release, and we received approximately 15 inquiries concerning updates or support requests.

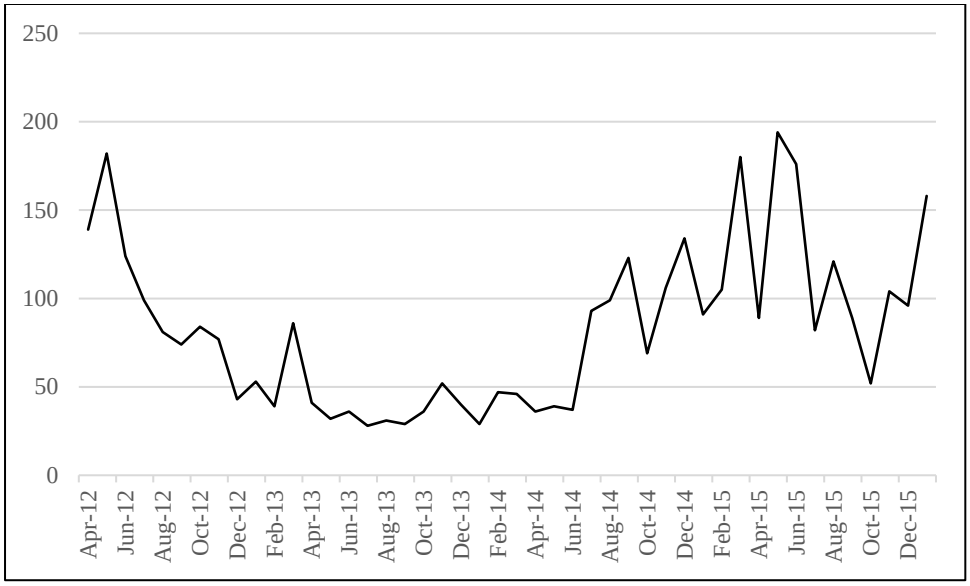


Figure 38. Number of downloads of the Kinect Joint Angles software per month

The releases are described below:

A. Kinect Joint Angles 1.0

This software, coded in C#, allows users to calculate the Joint Angles by capturing the data from Microsoft Kinect for Xbox 360. The code has been made available for download and public use under the MIT license.

a. Features:

- Joint angles detection: detects the joint angles for the Degrees Of Freedom (DOF) of 14 Joints using the coordinates from the Kinect Skeleton.
- Export Skeleton data to XML.
- Exports the Skeleton coordinates and Joints Angles to XML or CSV.
- Depth and RGB video recording using AForge library

b. Requirements:

- .Net Framework 4.0
- Microsoft Kinect
- Microsoft Kinect driver

B. Kinect Joint Angles 2.0

This software, coded in C#, allows users to calculate the Joint Angles by capturing the data from Microsoft Kinect for Xbox 360 and Microsoft Kinect for Windows. The code has been made available for download and public use under the MIT license.

a. Features:

- Joint coordinates conversion to angles and export: converts the joint coordinates to 2 angles and export them to a custom format, which is viewable in the ActionViewer.
- Export Skeleton data to XML.
- Exports the Skeleton coordinates and Joints Angles to XML.
- Convert from Kinect Joint Angles 1.0 to Kinect Joint Angles 2.0 file format
- Start and stop the export with voice commands: "START" & "STOP."

b. Requirements:

- .Net Framework 4.5
- Microsoft Kinect
- Microsoft Kinect driver

XIV. BIBLIOGRAPHY

[1] Google Inc., "Speech Recognition - Research at Google," Google Inc., [Online]. Available: http://research.google.com/pubs/SpeechProcessing.html. [Accessed 2015].

[2] V. Milanés, D. F. Llorca, B. M. Vinagre, C. González, M. Sotelo and others, "Clavileño: Evolution of an autonomous car," in *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, 2010.

[3] E. Blanzieri and A. Bryl, "A survey of learning-based techniques of email spam filtering," *Artificial Intelligence Review*, vol. 29, no. 1, pp. 63-92, 2008.

[4] Shazam, "Shazam - Music Discovery, Charts & Song Lyrics," Shazam, [Online]. Available: http://www.shazam.com/. [Accessed 2015].

[5] M. Curtiss, K. Bharat and M. Schmitt, *Systems and methods for improving the ranking of news articles*, Google Patents, 2005.

[6] R. Plamondon and S. N. Srihari, "Online and off-line handwriting recognition: a comprehensive survey," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 1, pp. 63-84, 2000.

[7] S. Mori, H. Nishida and H. Yamada, Optical Character Recognition, 1st ed., New York, NY, USA: John Wiley \& Sons, Inc., 1999.

[8] M. Turk, A. P. Pentland and others, "Face recognition using eigenfaces," in *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, 1991.

[9] P. Ekman and W. V. Friesen, Unmasking the face: A guide to recognizing emotions from facial clues, Ishk, 2003.

[10] L. S. Lippert, Clinical kinesiology and anatomy, FA Davis, 2011.

[11] E. E. Stone and M. Skubic, "Fall detection in homes of older adults using the Microsoft Kinect," *Biomedical and Health Informatics, IEEE Journal of*, vol. 19, no. 1, pp. 290-301, 2015.

[12] M. Hvilshøj, S. Bøgh, O. Madsen and M. Kristiansen, "The mobile robot "Little Helper": concepts, ideas and working principles," in *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*, 2009.

[13] B. Burger, I. Ferrané and F. Lerasle, "Multimodal interaction abilities for a robot companion," in *Computer Vision Systems*, Springer, 2008, pp. 549-558.

[14] Microsoft Corporation, "Kinect | Xbox 360," Microsoft Corporation, [Online]. Available: http://www.xbox.com/en-US/xbox-360/accessories/kinect. [Accessed 2015].

[15] CrunchBase, "PrimeSense | CrunchBase," Apple Inc., 200. [Online]. Available: https://www.crunchbase.com/organization/primesense. [Accessed 2015].

[16] B. James, "Xbox One to oevertake PS4 in US by 2015, says analyst," 13 05 2014. [Online]. Available: http://www.gamesindustry.biz/articles/2014-05-13-xbox-one-to-overtake-ps4-in-us-by-2015-says-analyst.

[17] Sony Computer Entertainment America LLC, "Playstation Move," Sony Computer Entertainment America LLC, 2015. [Online]. Available: https://www.playstation.com/en-us/explore/accessories/playstation-move/. [Accessed 2015].

[18] Creative Applications Network, "Kinect Projects | CreativeApplications.Net," Creative Applications Network, 23 June 2014. [Online]. Available: http://www.creativeapplications.net/kinect/. [Accessed 22 March 2016].

[19] Kinect for Windows Team, "Windows Store provides new market for Kinect apps | Kinect for Windows Product Blog," Microsoft Corporation, 18 March 2015. [Online]. Available: https://blogs.msdn.microsoft.com/kinectforwindows/2015/03/18/windows-store-provides-new-market-for-kinect-apps/. [Accessed 22 March 2016].

[20] KinectEDucation, "KinectEDucation," [Online]. Available: http://www.kinecteducation.com/.

[21] "ChaLearn Looking at People @ ECCV2014: Challenge and Workshop on Pose Recovery, Action and Gesture Recognition," 2014. [Online]. Available: http://gesture.chalearn.org/2014-looking-at-people-challenge.

[22] Y. Guo, G. Xu and S. Tsuji, "Tracking human body motion based on a stick figure model," *Journal of Visual Communication and Image Representation*, vol. 5, no. 1, pp. 1-9, 1994.

[23] Microsoft Corporation, "Kinect - Windows app develoment," Microsoft Corporation, 2015. [Online]. Available: https://dev.windows.com/en-us/kinect. [Accessed 2015].

[24] Boundless Biology, "Human Appendicular Skeleton," Boundless , 21 07 2015. [Online]. Available: https://www.boundless.com/biology/textbooks/boundless-biology-textbook/the-musculoskeletal-system-38/types-of-skeletal-systems-215/human-appendicular-skeleton-814-12055/. [Accessed 29 09 2015].

[25] Boundless Biology, "Human Axis Skeleton," Boundless, 21 07 2015. [Online]. Available: https://www.boundless.com/biology/textbooks/boundless-biology-textbook/the-musculoskeletal-system-38/types-of-skeletal-systems-215/human-axial-skeleton-813-12054/. [Accessed 29 09 2015].

[26] S. J. Spaulding, Meaningful motion: biomechanics for occupational therapists, Elsevier Health Sciences, 2005.

[27] R. C. Schafer, Clinical biomechanics: musculoskeletal actions and reactions, Williams \& Wilkins, 1987.

[28] L. Zhang, J. Sturm, D. Cremers and D. Lee, "Real-time human motion tracking using multiple depth cameras," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012.

[29] L. W. Campbell and A. E. Bobick, "Recognition of human body motion using phase space constraints," in *Computer Vision, 1995. Proceedings., Fifth International Conference on*, 1995.

[30] A. Knust, "An introduction to kinetography laban (labanotation)," *Journal of the International folk music council*, pp. 73-76, 1959.

[31] M. Z. Uddin, N. D. Thang, J. T. Kim and T.-S. Kim, "Human activity recognition using body joint-angle features and hidden Markov model," *Etri Journal*, vol. 33, no. 4, pp. 569-579, 2011.

[32] K. Koster, N. O'reilly, D. Jackson and S. Buxton, "Gait - Physiopedia, universal access to physiotherapy knowledge.," Physiopedia, [Online]. Available: http://www.physio-pedia.com/Gait. [Accessed 29 09 2015].

[33] M. Marszalek, I. Laptev and C. Schmid, "Actions in Context," in *IEEE Conference on Computer Vision & Pattern Recognition*, 2009.

[34] H. Wang and C. Schmid, "Action recognition with improved trajectories," in *Computer Vision (ICCV), 2013 IEEE International Conference on*, 2013.

[35] D. Roetenberg, H. Luinge and P. Slycke, "Xsens MVN: full 6DOF human motion tracking using miniature inertial sensors," *Xsens Motion Technologies BV, Tech. Rep*, 2009.

[36] C. Wu and H. Aghajan, "Model-based human posture estimation for gesture analysis in an opportunistic fusion smart camera network," in *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*, 2007.

[37] H. Aghajan and C. Wu, "Layered and collaborative gesture analysis in multi-camera networks," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, 2007.

[38] S. Ramagiri, R. Kavi and V. Kulathumani, "Real-time multi-view human action recognition using a wireless camera network," in *Distributed Smart Cameras (ICDSC), 2011 Fifth ACM/IEEE International Conference on*, 2011.

[39] Microsoft Corporation, "JointType Enumeration," Microsoft Corporation, [Online]. Available: https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx. [Accessed 29 09 2015].

[40] Microsoft Corporation, "Tracking Users with Kinect Skeletal Tracking," Microsoft Corporation, 2013. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131025.aspx. [Accessed 29 09 2015].

[41] N. Pugeault and R. Bowden, "Spelling it out: Real-time ASL fingerspelling recognition," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, 2011.

[42] M. W. Kadous, "Temporal classification: Extending the classification paradigm to multivariate time series," 2002.

[43] M. Gonzalez Preciado, "Computer vision methods for unconstrained gesture recognition in the context of sign language annotation," 2012.

[44] C. 2015, "ChaLearn LaP @ ICCV2015: Challenge and Workshop on Apparent Age Estimation and Cultural Event Recognition," Chalearn, 2015. [Online]. Available: http://gesture.chalearn.org/. [Accessed 2051].

[45] "Human Activity Video Datasets," The University of Texas at Austin, [Online]. Available: https://www.cs.utexas.edu/~chaoyeh/web_action_data/dataset_list.html. [Accessed 27 09 2015].

[46] C. Schuldt, I. Laptev and B. Caputo, "Recognizing human actions: a local SVM approach," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, 2004.

[47] M. Blank, L. Gorelick, E. Shechtman, M. Irani and R. Basri, "Actions as Space-Time Shapes," in *The Tenth IEEE International Conference on Computer Vision (ICCV'05)*, 2005.

[48] M. Hoai, Z.-Z. Lan and F. De la Torre, "Joint segmentation and classification of human actions in video," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, 2011.

[49] D. Weinland and E. Boyer, "Action recognition using exemplar-based embedding," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 2008.

[50] M. Grundmann, F. Meier and I. Essa, "3D shape context and distance transform for action recognition," in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, 2008.

[51] I. Laptev, M. Marszalek, C. Schmid and B. Rozenfeld, "Learning realistic human actions from movies," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 2008.

[52] J. Yuan, Z. Liu and Y. Wu, "Discriminative subvolume search for efficient action detection," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 2009.

[53] A. Stoian, A. Ferecatu, J. Benois-Pineau and M. Crucianu, "Fast action localization in large-scale video archives," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 10, pp. 1917--1930, 2016.

[54] S. Karaman, J. Benois-Pineau, V. Dovgalecs, R. Mégret, J. Pinquier, R. André-Obrecht, Y. Gaêstel and J.-F. Dartigues, "Hierarchical Hidden Markov Model in detecting activities of daily

living in wearable videos for studies of dementia," *Multimedia tools and applications,* vol. 69, no. 3, pp. 743-771, 2014.

[55] S. Singh, S. Velastin, H. Ragheb and others, "Muhavi: A multicamera human action video dataset for the evaluation of action recognition methods," in *Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on*, 2010.

[56] D. Lau, "The Science Behind Kinects or Kinect 1.0 versus 2.0," 27 11 2013. [Online]. Available: http://www.gamasutra.com/blogs/DanielLau/20131127/205820/The_Science_Behind_Kinects_or_Kinect_10_versus_20.php. [Accessed 22 03 2016].

[57] K. Khoshelham, "Accuracy analysis of kinect depth data," in *ISPRS workshop laser scanning*, 2011.

[58] S. Fothergill, H. M. Mentis, P. Kohli and S. Nowozin, "Instructing people for training gestural interactive systems," in *CHI*, 2012.

[59] P. Gomes, S.-M. Morgens and S.-R. Smith, *Gesture Classification from Kinect Data,* Santa Cruz: CMPS242: Machine Learning, 2012.

[60] W. Li, Z. Zhang and Z. Liu, "Action recognition based on a bag of 3d points," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, 2010.

[61] Y. Junsong, Z. Liu and Y. Wu, "Msr action recognition datasets and codes," [Online]. Available: http://research.microsoft.com/en-us/um/people/zliu/actionrecorsrc/default.htm.

[62] Carnegie Mellon University, "CMU Graphics Lab Motion Capture Database," Carnegie Mellon University, [Online]. Available: http://mocap.cs.cmu.edu/. [Accessed 2015].

[63] "UMD-Telluride Kinect Dataset," [Online]. Available: http://www.umiacs.umd.edu/research/POETICON/telluride_dataset/.

[64] "G3D: A Gaming Action Dataset," 28 5 2012. [Online]. Available: http://dipersec.king.ac.uk/G3D/.

[65] "Personal Robotics," Cornell University, 2009. [Online]. Available: http://pr.cs.cornell.edu/humanactivities/data.php#format.

[66] K. Yun, J. Honorio, D. Chattopadhyay, T. L. Berg and D. Samaras, "Two-person Interaction Detection Using Body-Pose Features and Multiple Instance Learning," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, 2012.

[67] T. F. Cootes, G. J. Edwards and C. J. Taylor, "Active appearance models," *IEEE Transactions on Pattern Analysis \& Machine Intelligence,* no. 6, pp. 681-685, 2001.

[68] R. Min, N. Kose and J.-L. Dugelay, "KinectFaceDB: A Kinect Database for Face Recognition," *Systems, Man, and Cybernetics: Systems, IEEE Transactions on,* vol. 44, no. 11, pp. 1534-1548, Nov 2014.

[69] LIRIS, "ICPR - HARL 2012," 2012. [Online]. Available: http://liris.cnrs.fr/.

[70] J. Alon, V. Athitsos, Q. Yuan and S. Sclaroff, "A unified framework for gesture recognition and spatiotemporal gesture segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 31, no. 9, pp. 1685-1699, 2009.

[71] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook and R. Moore, "Real-time human pose recognition in parts from single depth images," *Communications of the ACM,* vol. 56, no. 1, pp. 116-124, 2013.

[72] L. Breiman, J. Friedman, C. J. Stone and R. A. Olshen, Classification and regression trees, CRC press, 1984.

[73] M. B. F. O. &. S. Lichman, "Waveform Database Generator (Version 1) Data Set," University of California, Irvine, School of Information and Computer Sciences, 2013. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Waveform+Database+Generator+%28Version+1%29.

[74] M. B. F. O. &. S. Lichman, "Waveform Database Generator (Version 2) Data Set," UCI Machine Learning Repository, 1984. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Waveform+Database+Generator+%28Version+2%29.

[75] E. J. K. a. M. J. Pazzani, "Pseudo Periodic Synthetic Time Series Data Set University of California, Irvine, California 92697 USA," Department of Information and Computer Science , [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Pseudo+Periodic+Synthetic+Time+Series.

[76] N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, no. 1, pp. 321-357, 2002.

[77] A. Yao, J. Gall, G. Fanelli and L. J. Van Gool, "Does Human Action Recognition Benefit from Pose Estimation?.," in *BMVC*, 2011.

[78] A. Klaser, M. Marszalek and C. Schmid, "A spatio-temporal descriptor based on 3d-gradients," in *BMVC 2008-19th British Machine Vision Conference*, 2008.

[79] N. Dalal, B. Triggs and C. Schmid, "Human detection using oriented histograms of flow and appearance," in *Computer Vision--ECCV 2006*, Springer, 2006, pp. 428-441.

[80] P. Dollár, V. Rabaud, G. Cottrell and S. Belongie, "Behavior recognition via sparse spatio-temporal features," in *Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 2005. 2nd Joint IEEE International Workshop on*, 2005.

[81] A. Kovashka and K. Grauman, "Learning a hierarchy of discriminative space-time neighborhood features for human action recognition," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010.

[82] J. R. Uijlings, I. Duta, N. Rostamzadeh and N. Sebe, "Realtime video classification using dense HOF/HOG," in *Proceedings of International Conference on Multimedia Retrieval*, 2014.

[83] T.-S. Kim and Z. Uddin, Silhouette-based Human Activity Recognition Using Independent Component Analysis, Linear Discriminant Analysis and Hidden Markov Model, INTECH Open Access Publisher, 2010.

[84] G. Willems, T. Tuytelaars and L. Van Gool, "An efficient dense and scale-invariant spatio-temporal interest point detector," in *Computer Vision--ECCV 2008*, Springer, 2008, pp. 650-663.

[85] L. Yeffet and L. Wolf, "Local trinary patterns for human action recognition," in *Computer Vision, 2009 IEEE 12th International Conference on*, 2009.

[86] E. H. Adelson and J. R. Bergen, "Spatiotemporal energy models for the perception of motion," *JOSA A*, vol. 2, no. 2, pp. 284-299, 1985.

[87] A. Yilmaz and M. Shah, "Actions sketch: A novel action representation," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 2005.

[88] D. Weinland, R. Ronfard and E. Boyer, "Free viewpoint action recognition using motion history volumes," *Computer Vision and Image Understanding*, vol. 104, no. 2, pp. 249-257, 2006.

[89] A. Laurentini, "The visual hull concept for silhouette-based image understanding," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 16, no. 2, pp. 150-162, 1994.

[90] N. D. Thang, T.-S. Kim, Y.-K. Lee and S. Lee, "Estimation of 3-D human body posture via co-registration of 3-D human model and sequential stereo information," *Applied Intelligence*, vol. 35, no. 2, pp. 163-177, 2011.

[91] A. W. Vieira, E. R. Nascimento, G. L. Oliveira, Z. Liu and M. F. Campos, "On the improvement of human action recognition from depth map sequences using Space--Time Occupancy Patterns," *Pattern Recognition Letters*, vol. 36, pp. 221-227, 2014.

[92] J. Wang, Z. Liu, Y. Wu and J. Yuan, "Mining actionlet ensemble for action recognition with depth cameras," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012.

[93] G. Evangelidis, G. Singh and R. Horaud, "Skeletal quads: Human action recognition using joint quadruples," in *Pattern Recognition (ICPR), 2014 22nd International Conference on*, 2014.

[94] W. Li, Z. Zhang and Z. Liu, "Expandable data-driven graphical modeling of human actions based on salient postures," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 18, no. 11, pp. 1499-1510, 2008.

[95] X. Yang and Y. Tian, "Eigenjoints-based action recognition using naive-bayes-nearest-neighbor," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, 2012.

[96] W.-L. Lu and J. J. Little, "Simultaneous tracking and action recognition using the pca-hog descriptor," in *Computer and Robot Vision, 2006. The 3rd Canadian Conference on*, 2006.

[97] B. A. Draper, K. Baek, M. S. Bartlett and J. R. Beveridge, "Recognizing faces with PCA and ICA," *Computer vision and image understanding*, vol. 91, no. 1, pp. 115-137, 2003.

[98] F. Perronnin, J. Sánchez and T. Mensink, "Improving the fisher kernel for large-scale image classification," in *Computer Vision--ECCV 2010*, Springer, 2010, pp. 143-156.

[99] G. Evangelidis, G. Singh and R. Horaud, "Continuous gesture recognition from articulated poses," in *ChaLearn Looking at People Workshop in conjunction with ECCV 2014--European Conference on Computer Vision*, 2014.

[100] A. Lopes, R. S. Oliveira, J. M. de Almeida, D. A. Araujo and others, "Spatio-temporal frames in a bag-of-visual-features approach for human actions recognition," in *Computer Graphics and Image Processing (SIBGRAPI), 2009 XXII Brazilian Symposium on*, 2009.

[101] S. Ali, A. Basharat and M. Shah, "Chaotic invariants for human action recognition," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, 2007.

[102] A. Efros, A. C. Berg, G. Mori, J. Malik and others, "Recognizing action at a distance," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, 2003.

[103] Y. Wang and G. Mori, "Human action recognition by semilatent topic models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 10, pp. 1762-1774, 2009.

[104] V. N. Vapnik and V. Vapnik, Statistical learning theory, vol. 1, Wiley New York, 1998.

[105] H. Kim, S. Lee, D. Lee, S. Choi, J. Ju and H. Myung, "Real-Time Human Pose Estimation and Gesture Recognition from Depth Images Using Superpixels and SVM Classifier," *Sensors*, vol. 15, no. 6, pp. 12410-12427, 2015.

[106] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123-140, 1996.

[107] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5-32, 2001.

[108] G. Yu, J. Yuan and Z. Liu, "Unsupervised random forest indexing for fast action search," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, 2011.

[109] A. Yao, J. Gall and L. Van Gool, "A hough transform-based voting framework for action recognition," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010.

[110] A. Yao, J. Gall and L. Van Gool, "Coupled action recognition and pose estimation from multiple views," *International journal of computer vision*, vol. 100, no. 1, pp. 16-37, 2012.

[111] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *Computational learning theory*, 1995.

[112] M. Jones and P. Viola, "Fast multi-view face detection," *Mitsubishi Electric Research Lab TR-20003-96*, vol. 3, p. 14, 2003.

[113] C. Monnier, S. German and A. Ost, "A multi-scale boosted detector for efficient and robust gesture recognition," in *ECCV Workshops*, 2014.

[114] A. F. a. G. Mori, *Action Recognition by Learning Mid-level Motion Features*, Burnaby: School of Computing Science Simon Fraser University, 2008.

[115] L. Bourdev and J. Brandt, "Robust object detection via soft cascade," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 2005.

[116] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137-154, 2004.

[117] N. Neverova, C. Wolf, G. Taylor and F. Nebout, "Multi-scale deep learning for gesture detection and localization," in *ECCV Workshops*, 2014.

[118] S. Lawrence, C. L. Giles, A. C. Tsoi and A. D. Back, "Face recognition: A convolutional neural-network approach," *Neural Networks, IEEE Transactions on*, vol. 8, no. 1, pp. 98-113, 1997.

[119] G. Chéron, I. Laptev and C. Schmid, "P-CNN: Pose-based CNN Features for Action Recognition," *arXiv preprint arXiv:1506.03607*, 2015.

[120] K. Mikolajczyk and H. Uemura, "Action recognition with motion-appearance vocabulary forest," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 2008.

[121] G. Duncan, "Open source Kinect gesture recognition project, Kinect DTW," 24 8 2011. [Online]. Available: http://channel9.msdn.com/coding4fun/kinect/Open-source-Kinect-gesture-recognition-project-Kinect-DTW.

[122] G. Ten Holt, M. Reinders and E. Hendriks, "Multi-dimensional dynamic time warping for gesture recognition," 2007.

[123] U. Niaz and B. Merialdo, "Fusion methods for multi-modal indexing of web data," in *Image Analysis for Multimedia Interactive Services (WIAMIS), 2013 14th International Workshop on*, 2013.

[124] A. Fathi and G. Mori, "Action recognition by learning mid-level motion features," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 2008.

[125] Y.-L. Boureau, F. Bach, Y. LeCun and J. Ponce, "Learning mid-level features for recognition," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010.

[126] J. R. Smith, M. Naphade and A. Natsev, "Multimedia semantic indexing using model vectors," in *Multimedia and Expo, 2003. ICME'03. Proceedings. 2003 International Conference on*, 2003.

[127] G. Ye, D. Liu, I.-H. Jhuo, S.-F. Chang and others, "Robust late fusion with rank minimization," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012.

[128] C. Plagemann, V. Ganapathi, D. Koller and S. Thrun, "Real-time identification and localization of body parts from depth images," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010.

[129] A. Baak, M. Müller, G. Bharaj, H.-P. Seidel and C. Theobalt, "A data-driven approach for real-time full body pose reconstruction from a depth camera," in *Consumer Depth Cameras for Computer Vision*, Springer, 2013, pp. 71-98.

[130] M. Raptis, D. Kirovski and H. Hoppe, "Real-time classification of dance gestures from skeleton animation," in *Proceedings of the 2011 ACM SIGGRAPH/Eurographics symposium on computer animation*, 2011.

[131] X. Wei, P. Zhang and J. Chai, "Accurate realtime full-body motion capture using a single depth camera," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 6, p. 188, 2012.

[132] J. Taylor, J. Shotton, T. Sharp and A. Fitzgibbon, "The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012.

[133] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Robotics-DL tentative*, 1992.

[134] P. Kohli and J. Shotton, "Key developments in human pose estimation for kinect," in *Consumer Depth Cameras for Computer Vision*, Springer, 2013, pp. 63-70.

[135] M. Ye and R. Yang, "Real-time simultaneous pose and shape estimation for articulated objects using a single depth camera," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, 2014.

[136] G. Guerra-Filho and Y. Aloimonos, "A language for human action," *Computer*, vol. 40, no. 5, pp. 42-51, 2007.

[137] R. D. Green and L. Guan, "Quantifying and recognizing human movement patterns from monocular video images-part i: a new framework for modeling human motion," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 14, no. 2, pp. 179-190, 2004.

[138] L. Goncalves, E. Di Bernardo and P. Perona, "Movemes for modeling biological motion perception," in *Seeing, Thinking and Knowing*, Springer, 2004, pp. 143-170.

[139] K. Kulkarni, E. Boyer, R. Horaud and A. Kale, "An unsupervised framework for action recognition using actemes," in *Computer Vision--ACCV 2010*, Springer, 2011, pp. 592-605.

[140] C.-H. Lee, F. K. Soong and B.-H. Juang, "A segment model based approach to speech recognition," in *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, 1988.

[141] L. Rabiner and B.H. Juang, "Fundamentals of speech recognition," 1993.

[142] J. Alon, V. Athitsos and S. Sclaroff, "Accurate and efficient gesture spotting via pruning and subgesture reasoning," in *Computer Vision in Human-Computer Interaction*, Springer, 2005, pp. 189-198.

[143] A. Joshi, "A random forest approach to segmenting and classifying gestures," 2014.

[144] S. Bengio, "An asynchronous hidden markov model for audio-visual speech recognition," 2002.

[145] M. Al-Hames and G. Rigoll, "Reduced complexity and scaling for asynchronous HMMs in a bimodal input fusion application," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, 2006.

[146] T. Li, A. Ekpenyong and Y.-F. Huang, "Source localization and tracking using distributed asynchronous sensors," *Signal Processing, IEEE Transactions on*, vol. 54, no. 10, pp. 3991-4003, 2006.

[147] X. Lin, Y. Bar-Shalom and T. Kirubarajan, "Multisensor multitarget bias estimation for general asynchronous sensors," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 41, no. 3, pp. 899-921, 2005.

[148] S. Verdu, "Minimum probability of error for asynchronous Gaussian multiple-access channels," *Information Theory, IEEE Transactions on,* vol. 32, no. 1, pp. 85-96, 1986.

[149] S. Essa, "Gait Analysis," King Khalid University, 23 12 2012. [Online]. Available: http://www.slideshare.net/shimaa2022/gait-analysis-15743497.

[150] M. E. Morris, F. Huxham, J. McGinley, K. Dodd and R. Iansek, "The biomechanics and motor control of gait in Parkinson disease," *Clinical biomechanics,* vol. 16, no. 6, pp. 459-470, 2001.

[151] S. Russel and M. Triola, "The Precise Neurological Exam," New York University School of Medicine, [Online]. Available: https://informatics.med.nyu.edu/modules/pub/neurosurgery/coordination.html.

[152] P. Saadé, "Action Viewer," 2014-2015. [Online]. Available: http://computing-technologies.com/action_viewer.

[153] Reallusion, "iClone 6 Tutorial - Kinect Motion Capture Editing," 13 05 2015. [Online]. Available: iClone 6 Tutorial - Kinect Motion Capture Editing.

[154] Reallusion, "3D Animation and 2D Cartoons Made Simple - Reallusion Animation Software," [Online]. Available: www.reallusion.com.

[155] P. Saade, P. Joly and A. Awada, "Simulating actions for learning," in *Electronics, Control, Measurement, Signals and their application to Mechatronics (ECMSM), 2013 IEEE 11th International Workshop of,* 2013.

[156] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine learning,* vol. 37, no. 3, pp. 297-336, 1999.

[157] C. R. Souza, *Accord. net framework,* 2013.

[158] G. Jurman, S. Riccadonna and C. Furlanello, "A comparison of MCC and CEN error measures in multi-class prediction," *PloS one,* vol. 7, no. 8, p. e41882, 2012.

[159] M. Vihinen, "How to evaluate performance of prediction methods? Measures and their interpretation in variation effect analysis," *BMC Genomics,* vol. 13, no. 4, pp. 1-10, 2012.

[160] W. J. Youden, "Index for rating diagnostic tests," *Cancer,* vol. 3, no. 1, pp. 32-35, 1950.

[161] D. M. E. Hussein, "Dr. Mohamed E. Hussein's Home Page - MSRC-12 Re-Annotation," 08 2013. [Online]. Available: http://eng.staff.alexu.edu.eg/~mehussein/msrc12_annot4rec/index.html. [Accessed 24 02 2014].

[162] G. Hebrail and A. Baillard, "UCI Machine Learning Repository: Individual household electric power consumption Data Set," EDF R&D, 30 08 2012. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption.

[163] "Kinect SDK Dynamic Time Warping (DTW) Gesture Recognition," 30 7 2011. [Online]. Available: https://kinectdtw.codeplex.com/.

[164] G. D. Forney Jr, "The viterbi algorithm," *Proceedings of the IEEE,* vol. 61, no. 3, pp. 268-278, 1973.

[165] G. A. ten Holt, M. J. Reinders and E. Hendriks, "Multi-dimensional dynamic time warping for gesture recognition," in *Thirteenth annual conference of the Advanced School for Computing and Imaging,* 2007.

[166] T. Kanade, J. F. Cohn and Y. Tian, "Comprehensive database for facial expression analysis," in *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on,* 2000.

[167] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar and I. Matthews, "The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression," in

Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on, 2010.

THESIS SUMMARY

In this thesis, we took interest in human action recognition. Thus, it was important to define an action. We proposed our own definition: an action is a predefined sequence of concatenated simple gestures. The same actions are composed of the same simple gestures. Every performance of an action (recording) is unique. Hence, the body and the joints will perform the same movements as the reference recording, with changes of dynamicity of the sequence and amplitude in the DOF. We note that the variations in the amplitude and dynamicity must not exceed certain boundaries in order not to lead to entirely different actions.

For our experiments, we captured a dataset composed of actions containing basic variations. We merged some of those recordings with other actions to form a second dataset, consequently inducing more confusion than the previous one during the classification. We also captured three other datasets with properties that are interesting for our experimentations with the ALF (Asynchronous Late Fusion).

We overcame the problem of non-discriminatory actions datasets for action recognition by enlarging a set of recordings performed by different persons and captured by an RGB-D camera. We presented a novel method for generating synthetic recordings, for training action recognition algorithms. We analyzed the parameters of the method and identified the most appropriate ones, for the different classifiers. The simulation method improved the performances while classifying different datasets.

A general overview of data classification starting from the audio-visual context led to the ALF idea. In fact, most of the approaches in the domain classify sound and video streams separately with different tools. Every temporal sequence from a recording is analyzed distinctly, as in audiovisual stream analysis, where the classification outputs decisions at various time instants. Therefore, to infer the final decision, it is important to fuse the decisions that were taken separately, hence the idea of the asynchronous fusion. As a result, we found it interesting to implement the ALF in temporal sequences.

We introduced the ALF model for improving temporal events classification applied on late fusion classification algorithms. We showed the reason behind the use of an asynchronous model when classifying datasets with temporal properties. Then, we introduced the algorithm behind the ALF and the parameters used to tune it.

Finally, according to computed performances from different algorithms and datasets, we showed that the ALF improves the results of a simple Synchronous solution in most of the cases.

As it can be difficult for the user of the ALF solution to determine which datasets are compatible with the ALF, we built indicators to compare the datasets by extracting statistical information from the recordings. We developed indexes: the ASI and the ASIP, combined into a final index (the ASIv) to provide information concerning the compatibility of the dataset with the ALF.

We evaluated the performances of the ALF on the segmentation of action series and compared the results between synchronous and ALF solutions. The method that we proposed increased the performances.

We analyzed the human movement and gave a general definition of an action. Later, we improved this definition and proposed a "visual definition" of an action. With the aid of the ALF model, we focus on the parts and joints of an action that are the most discriminant and display them in an image.

In the end, we proposed multiple paths as future studies. The most important ones are :

- Working on a process to find the ALF's number of parts using the ASIv.
- Reducing the complexity by finding the discriminant joints and features thanks to the ALF properties
- Studying the MD-DTW features in-depth since the algorithm depends on the choice of the features
- Implementing a DNN for comparison purposes
- Developing the confidence coefficient.

RÉSUMÉ DE THÈSE

Dans cette thèse, nous nous intéressons à la reconnaissance de l'activité humaine. Nous commençons par proposer notre propre définition d'une action : une action est une séquence prédéfinie de gestes simples et concaténés. Ainsi, des actions similaires sont composées par les mêmes gestes simples. Chaque réalisation d'une action (enregistrement) est unique. Le corps humain et ses articulations vont effectuer les mêmes mouvements que celles d'un enregistrement de référence, avec des variations d'amplitude et de dynamique ne devant pas dépasser certaines limites qui conduiraient à un changement complet d'action.

Pour effectuer nos expérimentations, nous avons capturé un jeu de données contenant des variations de base, puis fusionné certains enregistrements avec d'autres actions pour former un second jeu induisant plus de confusion au cours de la classification. Ensuite, nous avons capturé trois autres jeux contenant des propriétés intéressantes pour nos expérimentations avec la Fusion Tardive Asynchrone (ou Asynchronous Late Fusion notée ALF).

Nous avons surmonté le problème des petits jeux non discriminants pour la reconnaissance d'actions en étendant un ensemble d'enregistrements effectués par différentes personnes et capturés par une caméra RGB-D. Nous avons présenté une nouvelle méthode pour générer des enregistrements synthétiques pouvant être utilisés pour l'apprentissage d'algorithmes de reconnaissance de l'activité humaine. La méthode de simulation a ainsi permis d'améliorer les performances des différents classifieurs.

Un aperçu général de la classification des données dans un contexte audiovisuel a conduit à l'idée de l'ALF. En effet, la plupart des approches dans ce domaine classifient les flux audio et vidéo séparément, avec des outils différents. Chaque séquence temporelle est analysée séparément, comme dans l'analyse de flux audiovisuels, où la classification délivre des décisions à des instants différents. Ainsi, pour déduire la décision finale, il est important de fusionner les décisions prises séparément, d'où l'idée de la fusion asynchrone. Donc, nous avons trouvé intéressant d'appliquer l'ALF à des séquences temporelles.

Nous avons introduit l'ALF afin d'améliorer la classification temporelle appliquée à des algorithmes de fusion tardive tout en justifiant l'utilisation d'un modèle asynchrone lors de la classification des données temporelles. Ensuite, nous avons présenté l'algorithme de l'ALF et les paramètres utilisés pour l'optimiser.

Enfin, après avoir mesuré les performances de classifications avec différents algorithmes et jeux de données, nous avons montré que l'ALF donne de meilleurs résultats qu'une solution synchrone simple.

Etant donné qu'il peut être difficile d'identifier les jeux de données compatibles avec l'ALF, nous avons construit des indicateurs permettant d'en extraire des informations statistiques. Nous avons développé des indices : l'ASI et l'ASIP, combinés en un indice final (ASIv) afin de fournir des informations concernant la compatibilité des données avec l'ALF.

Nous avons comparé les résultats entre la solution synchrone et l'ALF sur la segmentation de série d'enregistrements. Ceux-ci ont montré que l'ALF améliore les performances.

Nous avons analysé le mouvement humain et, après avoir donné une définition générale d'une action, nous avons amélioré cette définition et proposé une "définition visuelle". Ainsi, grâce à l'ALF, nous avons pu identifier les parties et les articulations d'une action les plus discriminantes et les afficher dans une image.

Nous avons proposé en perspectives quelques points importants dont :

- Définition d'un processus pour identifier le nombre de parties de l'ALF à l'aide du ASIv
- Réduction de la complexité en repérant les articulations et les caractéristiques discriminantes grâce à l'ALF
- Etude du choix des descripteurs de la MD-DTW puisque l'algorithme en dépend
- Mise en œuvre d'un DNN à des fins de comparaison
- Développement formel d'un coefficient de confiance.