universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Adaptive Gesture Recognition System, Transforming Dance Performance into Music"

verfasst von / submitted by

Evaldas Jablonskis

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2018 / Vienna 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 013
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Joint Degree Programme MEi:CogSci Cognitive Science
Betreut von / Supervisor:	Assoc. Prof. Hannes Kaufmann, Vienna University of Technology

Acknowledgements

I would like to express my sincere appreciation and gratitude to the people who made this thesis possible:

Prof. Hannes Kaufmann – for his trust and confidence in my abilities to accomplish such a technical project, for valuable coaching and advices;

Prof. Markus F. Peschl – for the second chance, for sharing his enthusiasm about phenomenology;

Elisabeth Zimmermann – for taking care of me during the journey from a lost-in-translation freshman to an assertive graduate;

Peter Hochenauer – for defeating bureaucratic challenges and answering endless questions;

Nicholas E. Gillian – for priceless assistance and fixing bugs in his EyesWeb catalogue;

Jolene Tan – for making sure the content reads fluently;

Rene Seiger – for hospitality and celebrations in Vienna;

Lora Minkova – for being an inspiring academic role model and encouragement;

Dalibor Andrijević – for abyss-deep and horizons-opening philosophical discussions;

Kirill Stytsenko – for introduction to introspective research on altered states of consciousness;

Rūta Breikštaitė – for precious partnership and companionship during all highs and lows of the last seven years;

my Mother – for unconditional support and constant care, whatever I throw myself into.

Abstract

The objective of this thesis was to develop a gesture recognition system that would transform dance to music using a machine learning algorithm. This thesis is divided into the six stages of the processing chain: *Input, Feature Extraction, Segmentation, Classification, Post-processing, Output*.

Video cameras with and without markers, wearable sensors and depth cameras were considered to provide *input* data; Microsoft Kinect v2 device was chosen as the best option. Body contour and body skeleton approaches were presented for *feature extraction*; Kinect SDK 2.0 was chosen to extract relevant features from the depth image. *Segmentation* based on music metrics was chosen over body tracking, while bar measure was chosen as the most suitable approach to split data stream to distinct gestures. For *classification*, machine learning algorithms Dynamic Time Warping (DTW), Hidden Markov Models, Support Vector Machines and Artificial Neural Network were explored; DTW was chosen as the most suitable algorithm. EyesWeb environment was chosen for *post-processing* and to build an overall "gesture engine". Ableton Live was selected to function as the *output*.

The designed system coupled virtual instruments with body parts: the system had to learn gestures of each group of body parts and know how gestures were paired with music clips in a composition. A working prototype of such a system was implemented and tested. Results supported the hypothesis of this thesis that a machine learning algorithm could be used for flexible gesture recognition.

Performance of the system under various conditions was evaluated in order to reveal its strengths and weaknesses. Measurements based on Signal Detection Theory were calculated in both fitting and cross-validation analysis. Results disclosed a very high prediction accuracy of the system: in most of the cases it was over 90%. Analysis showed that the system performed best when all predicted gestures were included in the training dataset and when each gesture had at least 16 training samples.

The implementation process provided some ideas about how the dance recognition system could be expanded to provide more features in music creation. The experience of music creation using gestures also implied that further advancements in machine learning and human-computer interfaces will not only enhance two-way interaction of dance and music but also build a closer relationship of body and mind.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

3D	Three-dimensional (horizontal, vertical and proximity)
AI	Artificial Intelligence
ANN	Artificial Neural Network
CR	Correct Rejections
DAW	Digital Audio Workstation
DTW	Dynamic Time Warping
FA	False Alarms
HMM	Hidden Markov Model
MIDI	Musical Instrument Digital Interface
NPV	Negative Predictive Value
OSC	Open Sound Control
PPV	Positive Predictive Value
RNN	Recurrent Neural Network
SDK	Software Development Kit
SEC	SARC EyesWeb Catalog
SVM	Support Vector Machines
TDNN	Time Delay Neural Network
TOF	Time Of Flight
TWSL	Triangulation With Structured Light
UI	User Interface

1. Introduction

The last few years have been witness to a ground-breaking evolution of recognition systems: smartphones can unlock by scanning our face or fingerprint, our photos get categorized by the objects in pictures, social networks suggest which friend to tag in images, home assistants can understand our questions and execute our orders, while cars can automatically avoid other cars, follow street signs and observe pedestrians (Hauert, 2017).

Even though gesture recognition has been researched for decades, there has still been no mainstream examples of its application. Microsoft Kinect was introduced in 2010 and raised high hopes (as well as the development of other devices to track a body or a hand) that we would soon interact with computers as exemplified in the science fiction film "Minority Report" released back in 2002 (Springmann, 2010).

Unfortunately, the original laser-infrared-visual camera for game consoles, designed to track body movements of a player, never crossed over "the chasm" between the stages of Early Adopters and Early Majority in the Technology Adoption Life Cycle (Figure 1.1): Microsoft discontinued production of Kinect for Windows in 2015 and for Xbox in 2017 (Gurwin, 2017), production companies abandoned development of new games for Kinect (Maiberg, 2016).

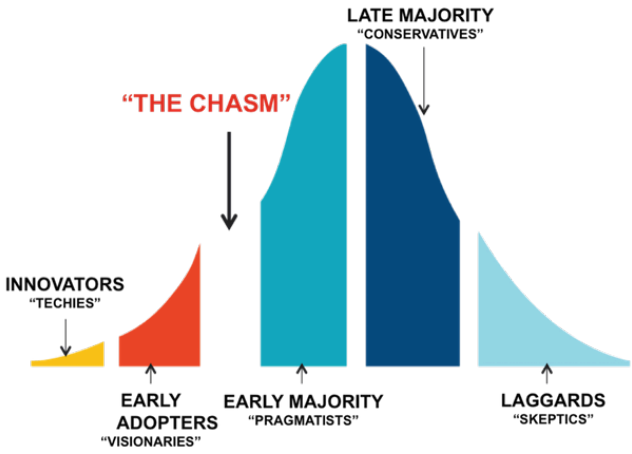

Figure 1.1. "The chasm" in the Technology Adoption Life Cycle.
Reprinted from *Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers*, by G. A. Moore, 2014. Copyright by New York HarperBusiness.

Consumers were dissatisfied with the high prices, low precision and limited games, while investors were disappointed about low sales, and as a result, industry had put the development of new products on hold and even scientists seem to have abandoned this research area (Thier, 2018).

Nevertheless, the author of this thesis believes that gesture recognition systems will soon have a more successful comeback. Three related trends lead us to expect this: (1) an overload of traditional user interfaces (UI), (2) demand for less effortful interaction with devices, and (3) growing computing power and development of artificial intelligence (AI).

(1) *Overload of traditional UI*. People spend increasingly more time in everyday life interacting with various devices: they use apps on their laptops, tablets, smartphones and wearables to read news, shop, follow friends, use encyclopaedias, databases and dictionaries, watch movies, control home appliances, call for taxis, food delivery, and manage bank accounts. Most devices and applications still have traditional interaction interfaces: users have to click or tap on buttons, type text and, only recently, use voice dialog (Hollander, 2017). Users have to learn and remember how to use every app. Typing and tapping takes a lot of time and requires to focus on the device screen, leaving less attention to the rest of our surroundings.

(2) *Demand for less effortful interaction*. If a new way of human-computer interaction works as expected and makes their life easier, mass market tends to adopt it eagerly. Now it is commonplace to use a fingerprint to unlock a phone even though typing in a PIN code was the default for 20 years (Bhagavatula et al., 2015). People place smart speakers in their homes to conveniently ask their voice assistants simple questions or to perform everyday tasks (Hoy, 2018). Investors are so enthusiastic about the potential of self-driving cars that Tesla overtook General Motors as America's most valuable automaker (Randewich, 2017).

(3) *Growing computing power and AI*. Artificial intelligence has been evolving at "the speed of light" and applied in more and more areas. Big Data Analytics and Deep Learning made it possible to launch and land space rockets, fly drones and drive cars, understand human spoken and written language, fluently speak and translate to German, French or Mandarin, recognize objects in images and real environment (LeCun, Bengio, & Hinton, 2015). A symbol of AI sophistication is the victory of Google's DeepMind program AlphaGo against a human professional Go player in 2016 (Reynolds, 2017). AI develops so rapidly not only because of invention of new algorithms, which often imitate biological neural networks, but also thanks to the growth of computing power. It has been constantly increasing, although no longer as fast as predicted by Moore's law, which stated that circuit complexity in computing systems should double every year (Theis & Wong, 2017). Slow data processing, currently the main bottleneck in AI, will soon be no issue for consumer products of everyday use.

1.1. Significance of Gesture Recognition

Assuming that public demand is high and technological issues are resolved, what problems could a gesture recognition system solve and how could it improve people's lives? Applications can be grouped into the areas of (1) control, (2) monitoring and (3) analysis (Miranda et al., 2012).

(1) *Control*. Human-computer interaction could be more efficient and intuitive (Eisenstein et al., 2003). Human gestures could control not only video games, but also "smart home" environments (Pu, Gupta, Gollakota, & Patel, 2013) and multimedia systems (Lee, Sohn, Kim, Kim, & Kim, 2013), manipulate objects in virtual reality (Rautaray, 2012), as well as navigate robots (Fahn & Chu, 2011). By naturally controlling with gestures, users can focus on the task instead of learning how to use and keep track of remote controls, controllers or keyboards (Song, Demirdjian, & Davis, 2015).

(2) *Monitoring*. Automatic monitoring systems could potentially not only recognize people in the visual field but also determine what task they are doing (Gavrila, 1999). Public and domestic surveillance systems could recognize or even predict criminal activities or accidents, watch patients, children and elders (Miranda et al., 2012).

(3) *Analysis*. Gestures could convey information, which cannot be delivered or extracted by other means. Sign language could be translated to sound and text (Hernandez-Rebollar, Kyriakopoulos, & Lindeman, 2004). Service robots could track gestures to recognize human emotions (Yang, Park, & Lee, 2006). Analysis of gestures could assist doctors in diagnosis of diseases, coaches in studying the performance of athletes (Miranda et al., 2012), and athletes in the learning and practicing of correct movements (Mitra & Acharya, 2007).

This thesis explores the use of gesture recognition systems in dance performances and music creation, because this area has an exciting potential to fuse technology and art. Applications in music and dance domains can also be classified to (1) control, (2) monitoring and (3) analysis:

(1) *Control of music*. Hands and the whole body can become musical instruments, artist's gestures could trigger musical notes and loop tracks, adapt volume and tempo of virtual instruments; musical performance could interact with visualizations (Bettens & Todoroff, 2009). By interacting with virtually animated avatars, people could learn and practice dancing not just for entertainment purposes, but also for professional training (Raptis, Kirovski, & Hoppe, 2011). Dancers could express themselves better and enrich their performance by

generating with their movements an audio-visual feedback (Castellano, Bresin, Camurri, & Volpe, 2007).

Dance performance could become music creation, which is the main purpose of the gesture recognition system developed for this thesis.

The dancer could play music with a virtual set of instruments, defining relationships between body movements and sound rendering (Bevilacqua, Schnell, & Alaoui, 2011). Dance and music interaction could benefit in Dance Music Therapy, which provides a non-pharmacological treatment of anxiety, depression and aggression, aids recovery after physical traumas and diseases, as well as improves overall psychosocial and psychophysical characteristics of patients (Jeong et al., 2005).

(2) *Monitoring of dance*. Gesture recognition could be used to observe and describe dance movements. Motiongrams – charts summarizing dancer's movement to music in a video recording – would be useful in navigating large databases of video material in order to get a quick overview of long dance performance videos (Jensenius, 2006). Recognition systems could assist in choreography to create notations for dance, ballet, and theatre movements (Gavrila, 1999). Notations are dance descriptions that currently take a lot of time to prepare using very sophisticated vocabulary of symbols therefore technology that converts images of body gestures to notations automatically would make the preparation process much easier (Boukir & Chenevière, 2004).

(3) *Analysis of performance*. Digital processing of movements expands the possibilities to research music and dance performances. It would help specialists to study musical intentions and expressions, analyse musicians' ability to communicate emotional states and attitudes with body articulation and evaluate high-level cognitive skills needed for mental and physical control over a musical instrument (Desmet et al., 2012). Topological Gesture Analysis – looking for shared geometrical elements in music and dance – could be used to study dance forms of different cultures (Naveda & Leman, 2010). Automated recognition techniques could identify cues that convey emotional content to study natural emotional movement expression in modern dance (Camurri, Lagerlöf, & Volpe, 2003).

1.2. Objective & Hypothesis

The author of this thesis was particularly intrigued by the possibility to create music with dance among this huge variety of current and potential applications of gesture recognition.

For a course at Vienna University of Technology (TU Wien) he had created a video game *Dance2Music*, where a dancer could play various audio tracks depending on which body part was being moved: shaking hips up and down started a drum-beat track, swinging hips left and right switched to another drum-beat track; kicking legs or stepping to the front triggered one bass-line track, stepping to a side, another. Furthermore, hand movements could play melodies: jiggling arms to sides started one melody, stretching arms to the front started another, while raising a hand above the head played some sound effects and percussion (Figure 1.2).

Figure 1.2. Screenshot of the promotional video for Dance2Music video game.

The game was developed using Microsoft Kinect motion sensor, Unity3D game engine and Ableton Live digital audio workstation. The game algorithm processed coordinates of a player's 15 "skeleton" points and outputted a signal when coordinates change beyond predefined time and space thresholds. The game worked well and was in the final shortlist at the worldwide contest "Xtion PRO Developer Challenge" by ASUS (Aigner, 2011).

Nevertheless, the development of the game revealed significant limitations of hard-coded definitions of gestures that have to be recognized. First, it was very difficult to adjust the thresholds to suit each person's unique body height and length of limbs as well as to suit each person's unique performance of even the simplest gestures. Another issue was that only simple gestures could be predefined (e.g. moving left-right, up-down) which stands in contrast to the sophistication of real-life aesthetic dance movements.

The *objective* of this thesis was to solve these problems and develop a gesture recognition system that could serve users of any age, gender or body shape and could recognize any dance movements.

The author *hypothesized* that exploitation of machine learning algorithms, used in AI to recognize human gestures, could provide a satisfying solution. This solution came with a

challenge though, because machine learning algorithms have more requirements than a simple threshold-based recognition system if the system were to be kept easy to use.

1.3. Research Questions & Scope

The project of this thesis was divided into the stages of the processing chain for a generic gesture recognition system (Gillian, 2011): (1) Input, (2) Feature Extraction, (3) Segmentation, (4) Classification, (5) Post-processing, (6) Output. Each stage raised specific *research questions*, needed to be answered in order to implement the system:

(1) *Input*. What is the best way to track a dancer's movements? Various tools from 2D video image processors to Wi-Fi signal change detectors are available to detect a human body in space, translate the signals into 3D coordinates (measurements of horizontal, vertical and depth position) in the timeline and send this data to pre-processing. Chapter 2.1 of this thesis discusses their strengths and weaknesses and explains the choice of Microsoft Kinect device.

(2) *Feature extraction*. Should the dimensionality of the data or the number of samples be reduced? What kind of features should be extracted instead of the raw data? Different machine learning algorithms require different data formats to process. Chapter 2.2 argues that down-sampled coordinates of Body Skeleton joints could be used to feed the system.

(3) *Segmentation*. How will the system identify the beginning and the end of a continuous gesture? The system could exploit the music rhythm (e.g., split gestures on every bar) or the repetition of dance movements (e.g., trim a gesture when it closes in the loop). Chapter 2.3 explores the possibilities and explains the choice of segmentation by rhythm.

(4) *Classification*. How should the gestures be assigned, remembered and recognized? There are several machine learning algorithms that have been successfully used for gesture recognition. The algorithm has to be multi-class (recognize several gestures), real-time (for live performances), sparse (require short training) and adaptive (learn recent gestures). Algorithms are described and compared in the chapter 2.4. The chapter explains the use of Dynamic Time Warping algorithm in the system's prototype.

(5) *Post-processing*. Should the system assist the dancer to make better music? The system could observe the music being performed and start the right tracks at the right time. This is discussed in the chapter 2.5.

(6) *Output*. How should audio tracks be played? There are music creation workstations that accept signals to start and stop audio tracks. Chapter 2.6 justifies the use of Ableton Live software.

The focus of this thesis was to design the system and choose the most promising tools based on literature review and theoretical analysis, as well as to build a working system's prototype and to evaluate several aspects of the system's strengths and weaknesses. Interpersonal testing, statistical data analysis and comparison with other systems developed by other researchers using representative experimental data is beyond the scope of this thesis.

1.4. Innovative Aspects

As already discussed, there are tools and methods already available to track and classify human gestures but they have not yet been applied to many promising areas. This thesis explores one of the possibilities hoping to stimulate this area of research and development.

Probably the most innovative aspect is *the purpose* of developed system. While existing gesture recognition systems either passively observe and assess a dance (Desmet et al., 2012; Naveda & Leman, 2010; Camurri et al., 2003) or actively create and conduct music (Bettens & Todoroff, 2009; Castellano et al., 2007), here, both tasks are combined in an unusual way: dance creates music, reversing the traditional music-dancer interaction.

The system incorporates a *unique combination* of devices, frameworks and algorithms, defined in research questions and described in the processing chain. The thesis attempts to acknowledge the best practices of previous research and discover new solutions where improvements are possible.

Another novel aspect is the *adaptive feature* of the gesture recognition model. Even though the model needs to adapt to changing body movements (as discussed earlier), such feature has been rarely implemented in related works.

1.5. Interdisciplinary Aspects

Dancing to music relates to the paradigm of *embodied music cognition*: it couples perception and action; physical environment and subjective experiences (Leman, 2012). Music created by dance is based on the same general framework: music's properties like pulse (pattern of beats) and tempo (speed of the pulse) must occur in synchronization with repetitive body movements (Burger, Thompson, Luck, Saarikallio, & Toiviainen, 2013). The process here requires even higher cognitive abilities, because the body takes on both active and passive role – a dancer needs to initiate music rhythms and respond to it.

According to S. Mitra et al. (2007), "gesture recognition is an ideal example of *multidisciplinary* research. There are different tools for gesture recognition, based on the approaches ranging from statistical modelling, computer vision and pattern recognition, image processing, connectionist systems, etc." (p. 213).

Connectionist approaches like Neural Networks have been widely used in pattern recognition (Bishop, 1995). Artificial Neural Networks is a computational model for information processing inspired by biological nervous systems like human or animal brains (Kiran, Chan, Lai, Ali, & Khalifa, 1996).

Artificial Neural Networks evolved to Deep Neural Networks and in recent years became the state-of-the-art solution to detect and recognize visual objects, dramatically improved other domains such as speech recognition, drug discovery and genomics (LeCun et al., 2015). Even though *deep learning* methods is another promising solution, it goes beyond the scope of this thesis which will focus on considering the classic Artificial Neural Network as a gesture recognition algorithm.

In the next chapter, theoretical background is presented to understand which elements are necessary for a gesture recognition system and which options are available for every element.

2. Theoretical Background & Related Work

The project of this thesis was divided into stages, which were adapted from the processing chain for a generic gesture recognition system (Gillian, 2011). Gillian's processing chain has 5 stages: (1) Input, (2) Feature Extraction, (3) Classification, (4) Post-processing, (5) Output. Gillian admitted, that a real-time recognition system of temporal gestures requires one more stage – Segmentation (Figure 2.1).

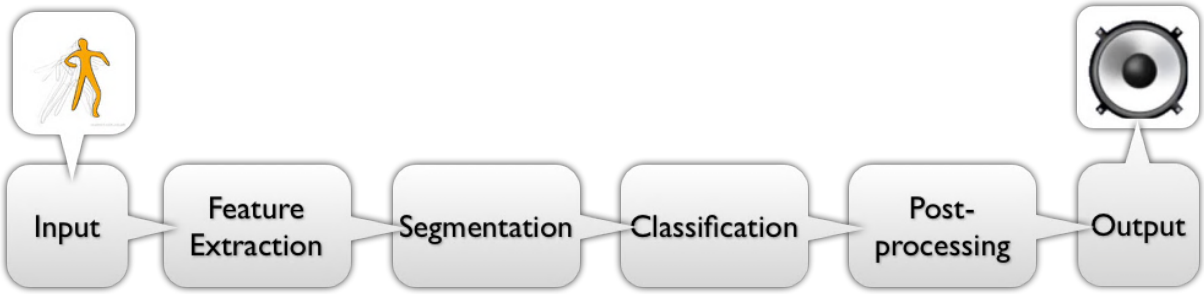

Figure 2.1. The processing chain for a dance gesture recognition system.

In this chapter, the author clarifies each stage's purpose and requirements, as well as reviewed available tools and methods to execute each stage based on related work in the literature.

2.1. Input

The first stage in the gesture recognition processing chain is to obtain the dancer's body tracking data from an input device. Available tools could be classified to (1) video cameras without markers, (2) video cameras with markers, (3) wearable sensors, and (4) depth cameras.

2.1.1. Marker-less Video Cameras

Video input without markers is the most natural way to track a body in space, because it uses optical sensors (just like humans do with their eyes) and is not intrusive as it does not require any special gear to be worn to be detected (Figure 2.2). Every frame in the video stream has to be processed to detect the body (i.e. to separate it from background) and to identify the location of specific body parts (Gavrila, 1999). Modern computer vision algorithms can extract 3D coordinates of a moving object in a video, but it takes a lot of processing power and may lack information due to occlusion of body parts (Liu & Kavakli, 2010). To solve these issues,

3D camera setups are used to get more reliable depth (i.e., proximity to the camera) data (Malassiotis, Aifanti, & Strintzis, 2002).

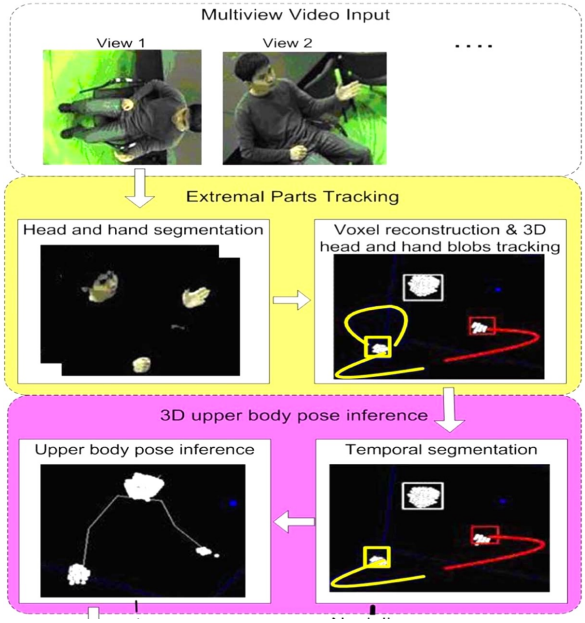

Figure 2.2. Processing of marker-less video input to detect the body.
Adapted from *3-D posture and gesture recognition for interactivity in smart spaces*, by C. Tran, & M. M. Trivedi, 2012. Copyright by IEEE Transactions on Industrial Informatics.

Video camera sensors are precise, cost effective and convenient to use: high resolution and frame-rate video cameras are relatively inexpensive, easy to setup and calibrate. But this method is vulnerable due to the environment's conditions: cluttered background, inconspicuous colour of clothes, poor lighting conditions make it difficult to track the body and its limbs (Ren, Yuan, & Zhang, 2011).

2.1.2. Video Cameras with Markers

The drawbacks of traditional video camera sensors can be drastically reduced using markers on the body. Video recordings with markers are widely used in cinema and game industries to track the bodies of actors, in virtual reality systems to track headset and controllers (Figure 2.3). A tracked object must have distinctly coloured spots or infrared light reflectors or emitters in relevant places. A human body can wear over 40 markers to mark all edges visible from all sides (Li & Prabhakaran, 2005). Multiple cameras are often used to make sure no marker of body movements is occluded at any time, so that the markers are easily detected by

computer vision algorithms in the video stream and their 3D coordinates can be calculated using triangulation (Bevilacqua et al., 2011).

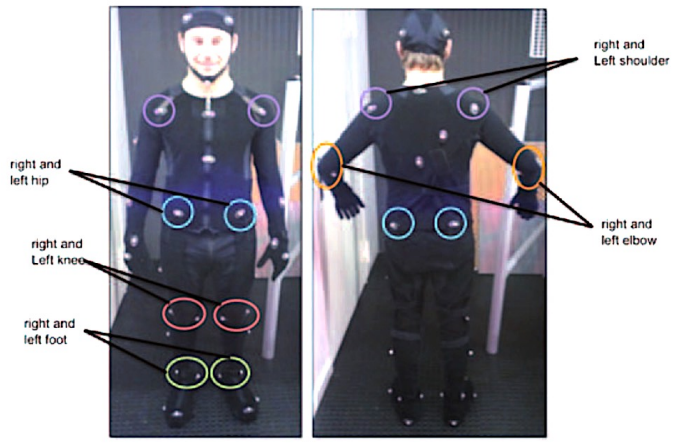

Figure 2.3. Placement of markers on the actor's body.
Reprinted from *Biomechanical Validation of Upper-body and Lower-body Joint Movements of Kinect Motion Capture Data for Rehabilitation Treatments*, by A. Fernández-Baena et al., 2012.
Copyright by IEEE Computer Society.

This method keeps the advantages of video camera sensors – high resolution and frame-rate, relatively low cost – and avoids disadvantages: image processing algorithms easily detect the markers in the image frames and it is much less prone to errors due to environmental conditions (Burger et al., 2013). Nevertheless, compared to marker-less video sensors, it is more intrusive (markers have to be worn) and less convenient (sophisticated setup and calibration are required).

Image processing still requires a lot of computational power because every pixel in high-definition video has to be analysed, therefore it is more often used for "offline" analysis of a recorded video (Desmet et al., 2012).

2.1.3. Wearable Sensors

Sensors like accelerometers and gyroscopes were proposed to reduce the need for high computational power (Figure 2.4). Just like markers, they have to be equipped on the tracked body, but do not require image processing to extract 3D coordinates (Junker, Amft, Lukowicz, & Tröster, 2008). Multiple sensors provide relative location of body parts based on distances between sensors, as well as change of sensor's position and angle (Aylward & Paradiso, 2006).

Figure 2.4. Accelerometers attached to participant's lower and upper arm.
Adapted from *Gesture spotting with body-worn inertial sensors to detect user activities*,
by H. Junker et al., 2008. Copyright by Elsevier Ltd.

To receive not only relative but also absolute location of a tracked body part, accelerometers and gyroscopes are used in combination with other sensors like infrared cameras. Wii game console is a good example of such system: Wii controller has an accelerometer, a gyroscope and an infrared emitter, tracked by the console (Schlömer, Poppinga, Henze, & Boll, 2008).

Wearable sensors do not need a powerful computer to process the data, but convenience is a trade-off (Ren et al., 2011). It is uncomfortable to wear the sensors, takes time to prepare the absolute position tracking system and requires frequent re-calibration (Patsadu, Nukoolkit, & Watanapa, 2012).

2.1.4. Depth Cameras

Depth-sensing systems have advantages of both wearable sensors and video cameras: they require less data processing to detect an object and are able to provide its position without markers (Fernández-Baena et al., 2012). Depth cameras are usually equipped with infrared lasers and sensors. There are two technologies to get the depth information: triangulation with structured light (TWSL) and time-of-flight (TOF) (Lun & Zhao, 2015).

Based on the TWSL technology, the laser projects a constant speckled pattern of infrared light to the whole field of view, which looks like a sky of stars; the infrared sensor observes the disparity of this pattern caused by objects and uses trigonometry to calculate distance to every "star" of projected pattern (Zhang, 2012).

Using the TOF technology, frequent pulses of infrared laser light up the whole field of view while the sensor detects reflections from all area and calculates distances between every

reflecting surface based on time the light travelled from the emitter to the sensor (Noonan, Howard, Hallett, & Gunn, 2015).

The depth camera outputs 3D coordinates of every pixel in the field of view, which can be visualized as a grayscale image, where darker objects are closer to the camera. Depth information can be combined with video stream to increase the precision of object tracking and add colour information for every pixel (Figure 2.5).

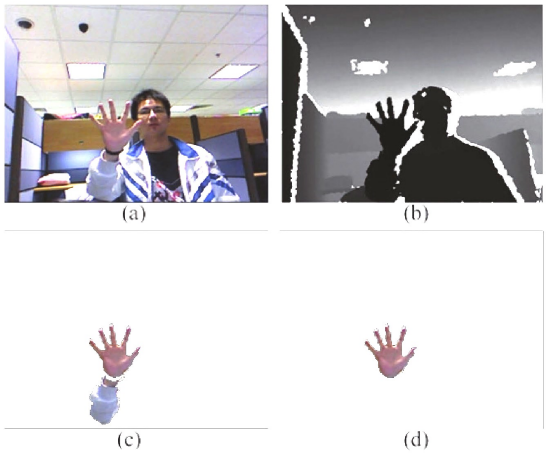

Figure 2.5. Hand detection based on colour and depth image captured by the Kinect sensor. Adapted from *Depth camera based hand gesture recognition and its applications in Human-Computer-Interaction*, by R. Zhou et al., 2011. Copyright by IEEE.

Just like video cameras, depth cameras suffer from the loss of tracking data due to occlusion of body parts. This problem can be solved using multiple depth cameras from different angles and merging the tracking data (Schönauer & Kaufmann, 2013).

The depth-sensing technology relies on the reflection of infrared light from the surface of tracked object, therefore tracking is disturbed by other sources of infrared light (like the sun) or unevenly reflective surfaces (like glass) (Wasenmüller & Stricker, 2017).

Depth cameras provide lower resolution and frame-rate than video cameras or wearable sensors. TOF technology enables higher resolution, but infrared sensors which have to detect light pulses are very expensive (Han, Shao, Xu, & Shotton, 2013).

More technologies to track gestures are being researched, such as electric fields or Wi-Fi signals. Human body absorbs and reflects radiation in spectrum of radio frequency, therefore movements could be detected measuring the strength of the electric field, when the user is between transmitting and receiving electrodes (Pun, 2006). Wi-Fi routers and mobile devices, common in home and work environments, could be exploited for a gesture recognition system, it would not require special sensors to wear or cameras to set up and could track movements

through walls without occlusion (Pu et al., 2013). The system could achieve it by detecting a distortion of Wi-Fi signal, caused by an in-air body movement (Abdelnasser, Youssef, & Harras, 2015). However, these technologies are still in its infancy and need more exhaustive research and development to be widely applied.

Depth-sensing technology seems to be a good trade-off between precision, convenience and need for computational power required for dance recognition. Dance gestures are executed by the whole body, which is a large object to track and therefore does not require a high-resolution sensor (like hand gesture recognition systems do). Dance is a dynamic long-lasting rhythmical activity and wearing markers or sensors would be uncomfortable and disturbing – which depth cameras do not need it. Depth sensors provide 3D coordinates of every pixel in the field of view, eliminating the step of computationally expensive image pre-processing to extract this data and making a real-time recognition system more responsive. For the project of this thesis, the depth-sensing device *Microsoft Kinect v2* was chosen as an input device.

First generation Kinect v1 was introduced to mass market in 2010 and caused an explosion of research and experimentation thanks to its low price and simple setup (Biswas & Basu, 2011). It was based on TWSL technology, therefore had a speckled-pattern-laser-light emitter and an infrared sensor, as well as an RGB camera and a microphone (Zhang, 2012). Due to limitations of used technology, the device was making low fidelity depth measurements and was sensitive to lighting conditions (Lun & Zhao, 2015).

In 2014 Microsoft switched to a more advanced TOF technology and launched the second-generation *Kinect v2* (Wasenmüller & Stricker, 2017). It has higher specifications than v1 in many aspects and comes with an official *Software Development Kit 2.0* (SDK 2.0), which can even accomplish feature extraction discussed in the next stage of processing chain (Wang, Kurillo, Ofli, & Bajcsy, 2015).

2.2. Feature Extraction

The depth sensor of Microsoft Kinect v2 has a resolution of 512 by 424 pixels and the RGB sensor resolves “full HD” with 1920 by 1080 pixels at a frequency of 30 frames per second (Lun & Zhao, 2015). It means that the device provides *3 position values* (horizontal, vertical and proximity coordinates) of 307,200 pixels and *3 colour values* (red, green and blue) of 2,073,600 pixels at a rate of 30 cycles per second. It is not reasonable to feed this huge amount of data directly into classification algorithms, because it would take too much time to process (not suitable for real-time recognition systems), cause overfitting errors (some data is noise and

should be ignored) and some algorithms simply cannot handle high-dimensional data (Al-Ali, Milanova, Al-Rizzo, & Fox, 2015).

The amount of input data should be reduced in the way that it keeps the key information needed to achieve the following tasks: (1) detect a human body and (2) label the body parts. The process of reduction of data amount retaining relevant information is called feature extraction (Chaaraoui, Padilla-López, & Flórez-Revuelta, 2013).

(1) *Detection of human body*. Identifying a human body in the field of view is relatively easy when both depth and colour information is available. First step is to remove background pixels which have larger depth (proximity to the camera) values (Beyl et al., 2013). High-resolution colour image, merged with depth map, can improve the accuracy of object and background separation, if their colour patterns are different (Han et al., 2013).

Some interactive dance systems do not require further identification: they track the whole body and trigger audio-visual events based on its location and speed of movement in space (Camurri, Mazzarino, Ricchetti, Timmers, & Volpe, 2004).

The system of this thesis has to identify and track individual body parts in order to connect every part to a specific audio track, therefore detection of human body is not sufficient.

(2) *Labelling of body parts.* There are two approaches to segment body parts to meaningful labels, keeping relevant information on their position: body contour and body skeleton.

2.2.1. Body Contour

Body contour, also called "3D shape" or a "silhouette" (Figure 2.6), is a large set of dots/vectors with coordinates in 3D space, which represent the surface of the body (Han et al., 2013).

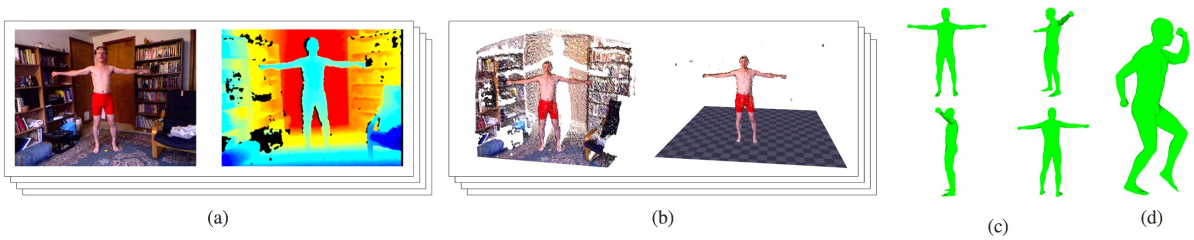

Figure 2.6. Body contour, recovered from colour and depth images captured by the Kinect sensor. Reprinted from *Home 3D Body Scans from Noisy Image and Range Data*, by A. Weiss et al., 2011. Copyright by IEEE.

For some gesture recognition systems, it is important to know the exact location of the body surface, for example: the position of finger tips is needed for a convincing "touch" interaction in virtual environments; accurate tracking of body surface is crucial in cinema and game production, where it is remodelled to an animated avatar (Licsár & Szirányi, 2005).

2.2.2. Body Skeleton

The body skeleton approach is a suitable choice for interactive dance systems, because it does not require the knowledge about the body's surface. Body skeleton consists of a small number of dots/vectors in 3D space, which represent the main limbs and joints of a human body: head, neck, palms, elbows, shoulders, waist, hips, knees and feet (Patsadu et al., 2012).

The task of further processing is greatly simplified, when only a handful of vectors (further called as "joints") need to be analysed without losing essential information (Kurakin, Zhang, & Liu, 2012). For example, a hand movement can be represented by the change of the 3D position and rotation of two joints – palm and elbow – in relation to the third joint – shoulder. This information would be insufficient for e.g. a sign language recognition system, but is satisfactory for interactive dance systems.

Researchers have created many algorithms to cluster depth and colour pixels and classify to skeleton joints (Chiu, Blanke, & Fritz, 2011). Microsoft's official Kinect SDK provides the skeleton information out-of-the-box (Figure 2.7).

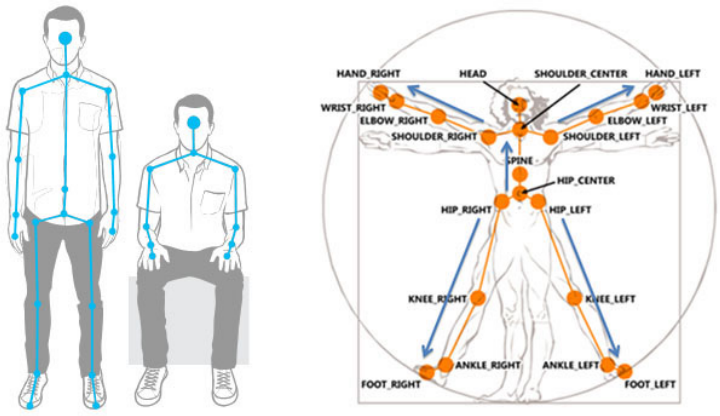

Figure 2.7. Body skeleton, provided by Kinect SDK 2.0.
Adapted from *Human Interface Guidelines for Kinect v2*,
by Microsoft Corporation, 2014. Copyright by Microsoft Corporation.

According to the creators of Kinect v1, the training data was collected generating hundreds of thousands "realistic synthetic depth images of humans of any shapes and sizes in

highly varied poses sampled from a large motion-capture database" which was used to train "a deep randomized decision forest classifier" (Zhang, 2012). The proprietary algorithm analyses input from depth sensor, assigns every pixel to a body joint and every joint to a 3D skeleton. The classifier has low latency – processes every frame in few milliseconds – because it is able to exploit the computer's graphical processing unit (Zhang, 2012). The method of skeleton tracking in Kinect SDK 2.0 which comes with the second-generation device Kinect v2 has not been disclosed, but it's believed to be similar to the original (Wang et al., 2015).

The body skeleton approach was chosen for the system of this thesis, exploiting Microsoft's original Kinect SDK 2.0 to extract temporal 3D coordinates of the dancer's body parts.

Some gesture classification algorithms may require additional feature extraction to reduce the amount of data and/or transform it to a format which they are able to process. Feature extraction methods applicable to specific algorithms will be discussed in the chapter 2.4.

2.3. Segmentation

Dance is a continuous stream of motion, where gestures are not clearly separated – there is no idle time between gestures – which makes segmentation a challenging task. Gesture segmentation is a process where data stream is divided into chunks which are accepted by classification algorithms as discrete cases (Kahol, Tripathi, & Panchanathan, 2004). Segmentation can be done based on (1) "internal" data of body tracking or (2) "external" data of music metrics.

2.3.1. Based on Body Tracking

The first approach relies solely on the data received from a depth sensor. Movement has to be deconstructed to primitive elements, then the pattern of these elements is being compared to known sequences looking for similarities. This method is similar to a deconstruction of a sentence to words in speech recognition (Wang, Shum, Xu, & Zheng, 2001). The size of a pattern has to be defined either by the number of last frames, milliseconds or deconstructed elements (Naveda & Leman, 2008). Primitive elements can be defined explicitly (e.g. as key poses) or implicitly (e.g. as "periods" of higher activity) (Kahol et al., 2004).

Popular dance has an advantageous characteristic: it consists of repetitive movements, differently from ballet or modern dance where movements are supposed to reflect emotion or

mood rather than rhythm (Naveda & Leman, 2008). If a gesture is repeated rhythmically, its trajectory must represent a closed loop. This feature allows distinction of a gesture without its prior knowledge: a segmentation algorithm must only find when the same sequence of the gesture's primitive elements begins again (Figure 2.8).

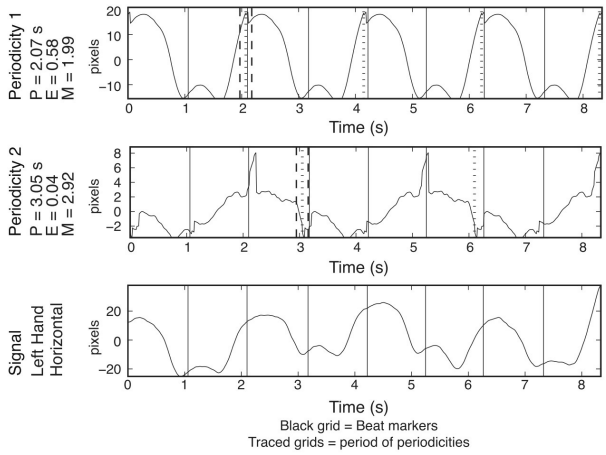

Figure 2.8. Gesture segmentation based on hand's movement pattern.
Adapted from *A Cross-modal Heuristic for Periodic Pattern Analysis of Samba Music and Dance*, by L. Naveda & M. Leman, 2009. Copyright by Journal of New Music Research.

It is difficult to identify these elements by position and rotation of skeleton joints, because body movement is never exactly the same even though the performer's intention is to repeat a gesture – it can be slower or quicker, joints can have slightly different path or amplitude. Derivative data could be used to ignore this deviation – speed, acceleration, direction – which could be further processed observing distribution, frequency (e.g. with *Fourier transform*) and sequence (e.g. with *Dynamic Time Warping*) of these values (Bettens & Todoroff, 2009). Motion cues are also used such as peaks of energy (speed multiplied by mass of a body part), fluency (change of movement direction), impulsiveness (change of state from idle to rapid movement) describing kinetic characteristics of a body (Camurri et al., 2004; e.g. Figure 2.9).

Dance is movement synchronized with music, therefore music metrics could be exploited to reduce the amount of data needed for quick and precise segmentation: movement indicators could be overlaid with music metrics and peaks of energy ignored if they do not match with peaks of pitch (Naveda & Leman, 2009).

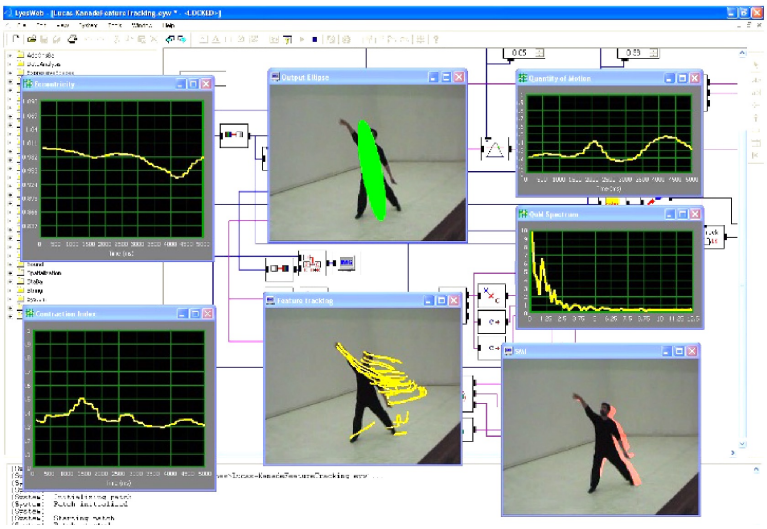

Figure 2.9. Extraction of motion cues using EyesWeb framework.
Reprinted from *Multimodal Analysis of Expressive Gesture in Music and Dance Performances*, by A. Camurri et al., 2004. Copyright by Springer.

2.3.2. Based on Music Metrics

The second approach relies solely on music metrics. Popular music genres have a stable tempo and time signature (Bahn, Hahn, & Trueman, 2001). The tempo is the song's speed described in beats (pulses) per minute, while the time sequence specifies the number of beats in each bar (measure) and the length of a beat (Burger et al., 2013). Bar (measure) is a musical notation that describes a time segment of one length with the same number of beats, perceived by a dancer as a rhythm (Burger et al., 2013). Rhythmical dance should match the song's time signature, which provides an opportunity to split the dance into separate gestures by the time period of a bar (Raptis et al., 2011; e.g. Figure 2.10).

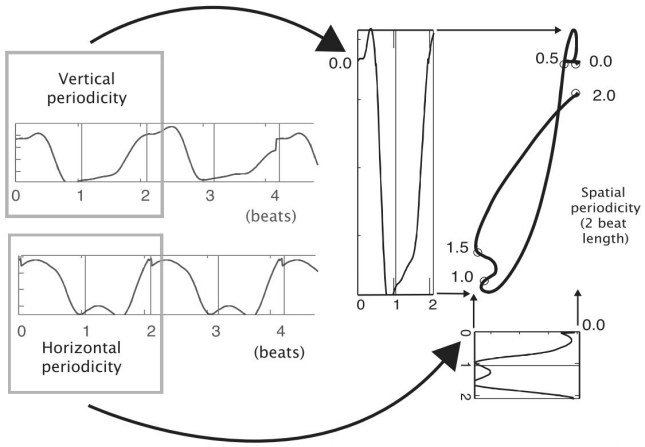

Figure 2.10. Gesture segmentation based on the beat length.
Reprinted from *Representation of Samba dance gestures, using a multi-modal analysis approach*, by L. Naveda & M. Leman, 2008. Copyright by Edizione ETS.

The second approach is robust, easy to implement and does not require data processing, it is therefore an attractive option, if dance gestures are repetitive and match the music's rhythm. If both of these requirements cannot be met, the first approach should be used.

In the project of this thesis a gesture segmentation by music bar was chosen because it was expected that a dancer would repeat gestures on every bar.

2.4. Classification

Classification is a crucial part in the system's processing chain, because this is where the actual recognition happens. Recognition of a gesture is successful when the system labels it with the correct class.

In the previous chapter "segmentation", the gesture's identification in the continuous movement (separating a discrete time-series of one gesture from the stream of continuous input data) is actually a classification task to one of two categories – "gesture" or "non-gesture" (a.k.a. "null gesture"). An algorithm, responsible for gesture segmentation, could also do the classification task if it recognizes the pattern of a discrete gesture within a continuous stream based on similarity to one of the known gestures (Bettens & Todoroff, 2009).

Classification algorithms in interactive dance systems must accurately classify gestures meeting the following requirements: (1) it should suffice a small training dataset to learn gestures, (2) it should rapidly classify gestures, and (3) it should be able to integrate adaptive features.

(1) *Small training dataset.* Some recognition systems (e.g. object recognition in images) use databases with thousands of samples of previously recorded and labelled gestures to train a classification model (Licsár & Szirányi, 2005). There are attempts to create such databases for recognition of standard gestures, e.g. sign language (Martínez, Wilbur, Shay, & Kak, 2002). Because there is no standard database of dance movements, dancers must to train the model with their own samples. For the convenience of use it should take a reasonable amount of time and number of gesture repetitions to prepare the system.

(2) *Rapid classification*. Training a model requires some processing power and time and can be done offline, once the training dataset is acquired. But during real-time live performance a new gesture must be classified immediately, because it has to trigger an audio track at the right moment (Bettens & Todoroff, 2009). A sparse model (which does little computation to do

the classification) would give freedom to use the system on computers with a wider range of processing power.

(3) *Adaptive feature*. When the initial training dataset is very small, an adaptive feature would be beneficial for a classification model (Licsár & Szirányi, 2005). An adaptive model is able to "learn by doing" and improve accuracy over time of use. It can be achieved by including correctly classified gestures as additional training examples and periodically retraining the model (Licsár & Szirányi, 2005). Adaptive feature requires not only classification but also for the model's training to take a short period of time.

This thesis discusses four machine learning algorithms often used to recognize gestures: (1) Dynamic Time Warping, (2) Hidden Markov Models, (3) Support Vector Machines, and (4) Artificial Neural Networks.

2.4.1. Dynamic Time Warping

The most straight-forward recognition of a gesture is comparing its trace with traces of known gestures and then selecting a label with the most similar trace. A gesture's trace can be represented by a time-series (multidimensional data matrix where one dimension is time) with 3D coordinates of relevant body parts changing over a period of time (Kratz & Rohs, 2010). Comparison of traces can be achieved with the Dynamic Time Warping (DTW) algorithm, which finds the shortest warping path (i.e., the minimum total distance of all data points) between two vectors (Akl, Feng, & Valaee, 2011; e.g. Figure 2.11).

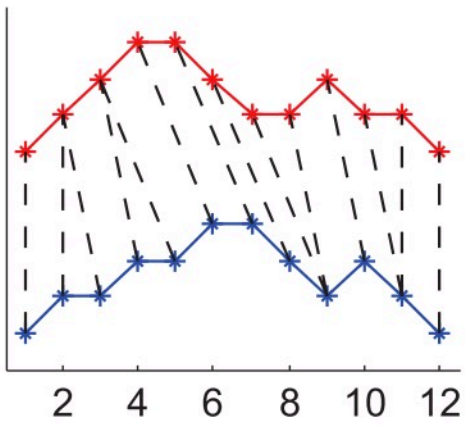

Figure 2.11. Mapping between two time-series based on the DTW algorithm.
Adapted from *Real-time DTW-based gesture recognition external object for Max/MSP and PureData*, by F. Bettens & T. Todoroff, 2009. Copyright by F. Bettens & T. Todoroff.

Computationally, it would be very expensive to calculate distances of every data point of one vector to every data point of the second vector and find the smallest distances of all possible combinations. Instead the algorithm relies on the assumptions that vectors consist of sequential data (every data point leads to the next data point and cannot lead to the previous), that the warping path is continuous and one-directional (similar data segments can be shifted in time but are arranged in the same order) (Gillian, 2011). The algorithm exploits these assumptions of a dynamic programming approach: the minimum distance between data points of two vectors is a warping path (the sum of minimum distances) of up until this pair plus a smallest distance to the next pair (Bettens & Todoroff, 2009). Equations of DTW algorithm are presented and explained in the *Appendix A*.

The algorithm is frequently used in data sequence recognition (e.g. speech or writing), because it can find similar segments of data points when their location is shifted, in-between data segments are not similar and vectors have different length (Boukir & Chenevière, 2004).

If data points have more than one dimension (e.g. body part's 3D coordinates changing in time), the *Euclidean* or similar metric should be applied to compute the sum of distances (Li, Zhai, Zheng, & Prabhakaran, 2004). If dimensions have different data ranges, they should be normalized before computing the Euclidean distance (Bettens & Todoroff, 2009). In order to reduce the amount of computation and overfitting, data should be down-sampled (Gillian, 2011).

Training of the DTW model is a process where a template is found for every gesture. That is, a time-series example which has the smallest warping path to all other examples in the training data (Gillian, 2011).

The model classifies an unknown time-series by comparing its distances to each template and selecting the class with the smallest distance, if it is smaller than a threshold (Gillian, 2011). If all distances are larger than the threshold, the gesture is considered as unrecognized and classified as a non-gesture ("zero class").

Algorithm's *strength*: as every gesture is represented by a separate independent template, it is convenient to remove or add a class without retraining the whole model (Gillian, 2011).

Algorithm's *weakness*: the algorithm heavily depends on finding the "best" example in the training dataset as a template for every gesture, therefore it can perform poorly if all training examples are slightly different (Gillian, 2011).

2.4.2. Hidden Markov Models

Another way to approach the gesture recognition problem is to look for a "hidden" pattern that groups examples of one gesture to one class. When a performer is repeating the same gesture, repetitions may vary, but the intended gesture is the same. Unfortunately, the intended gesture cannot be observed, it can only be inferred based on observable repetitions.

Hidden Markov Model (HMM) is a set of interconnected unobservable "hidden" states which emit observations (Lee & Kim, 1999). Every hidden state can either remain or change to another state and can emit any of the observations from the set (Lee & Kim, 1999).

The simplest way to model sequential data is a first-order left-right *Markov chain* in which every state can transition only into one next state and cannot transition back (Mitra & Acharya, 2007). The model relies on the assumption, that every state only depends on the previous state and every observation only depends on emitting states. Therefore, the structure of hidden states can be inferred based on known observations using computationally efficient dynamic programming (Kahol et al., 2004).

A classic HMM can accept only one-dimensional values as observations, therefore pre-processing of the time-series is needed. One of the methods is *Vector Quantization*, where all data points (3D coordinates of relevant body parts) are grouped to predefined number of clusters, then the IDs of these clusters are used as observation values. *EM-based Gaussian Mixture Model* (Yang et al., 2006), *Symbolic Aggregate Approximation* with *K-Means* and *K-Nearest Neighbour* algorithms (Gillian, 2011) are used to cluster the data points of feature vectors.

In classification problems, every gesture class is represented by a separate HMM, where the model's observations are pre-processed time-series of a gesture (e.g. Figure 2.12). A number of hidden states and observations for a model must be predefined and the model trained – the model's parameters computed using *Baum-Welch* algorithm (Schlömer et al., 2008). Model's parameters are initial distribution of the states, the transition probabilities between states and the emission probabilities of the state to observations (Fahn & Chu, 2011).

Once HMMs for all gestures are trained, an unknown gesture is classified feeding its pre-processed time-series as observations to every model and comparing the probabilities that these observations could be emitted by every model using the *Forward-Backward* algorithm (Fahn & Chu, 2011). If the probability is larger than a threshold, the class with the highest probability is selected, otherwise the time-series is classified as a non-gesture (Gillian, 2011).

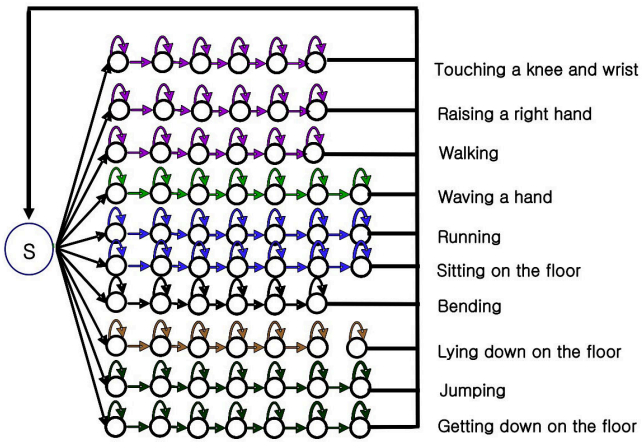

Figure 2.12. First-order left-right HMMs to recognize 10 gestures.
Adapted from *Human-robot interaction by whole body gesture spotting and recognition*, by H. D. Yang et al., 2006. Copyright by IEEE.

Algorithm's *strengths*: thanks to the dynamic programming calculation of probability in HMM is not computationally expensive; every gesture is represented by a separate independent model; therefore, it is convenient to remove or add a class without retraining the whole model (Gillian, 2011).

Algorithm's *weaknesses*: the model needs to set up a lot of parameters – number of states, observations, state relationships, initial states, thresholds; HMMs are limited when multidimensional data has to be classified, because during Vector Quantization important features of the data may be lost (Gillian, 2011).

2.4.3. Support Vector Machines

Previous DTW and HMM algorithms have a separate independent model for every learned gesture and a new gesture is classified finding the best-fitting model. This approach is prone to errors when several models return a high probability of likelihood because differences between classes are not emphasized.

Another approach is to consider all gestures in one model and find key elements which discern one class from another. The Support Vector Machines (SVM) algorithm does exactly that: in its simplest form, when the training data is 2-dimensional data points of two known classes, it draws the optimal line (called "hyperplane") where the distance (called "margin") between nearest data points of the opposite classes (called "support vectors") is largest (Figure 2.13).

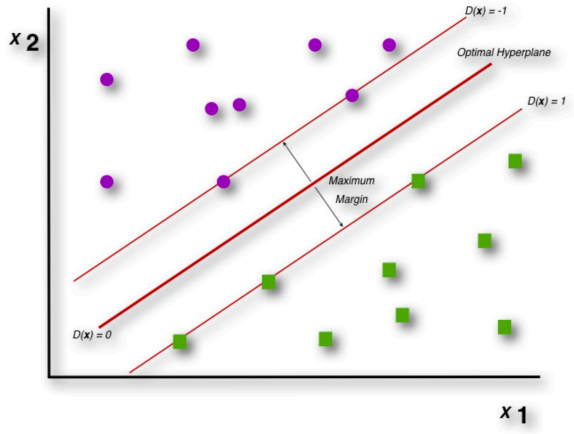

Figure 2.13. Hyperplane, maximum margin and support vectors of the SVM algorithm. Adapted from *Gesture Recognition for Musician Computer Interaction*, by N. Gillian, 2011. Copyright by N. Gillian.

Once the model is trained (i.e. the hyperplane and support vectors are found), a new data point can be assigned to one of the classes depending on which side of the hyperplane it is (Patsadu et al., 2012).

If a straight line cannot separate the classes, data points can be mapped to a higher dimensional space using *non-linear kernels* (it's called "kernel trick"), where the hyperplane is optimal (Gillian, 2011). The algorithm is also able to find a hyperplane when the original data is multidimensional (Kotha, Pinjala, Kasoju, & Pothineni, 2015).

The classic SVM divides only two classes and returns only a discrete value indicating the likely class, but it can be extended to divide multiple classes and return a probability of classification of new data (Gillian, 2011). When the model is extended to provide probability along with the inferred class, a threshold can be applied: if the probability is lower than a threshold, the sample is classified as a non-gesture (Gillian, 2011).

The SVM algorithm requires the input data to have a predefined and fixed length, therefore the time-series needs to be pre-processed by extracting its features (Gillian, 2011). One method of feature extraction is to divide time-series of every gesture trace to a number of equal segments, compute key *distribution measures* (e.g. mean, standard deviation, Euclidean norm, root-mean-square) for each dimension of a segment, then arrange the values of all measures to one vector (Gillian, 2011). For example, if the time-series with 3D coordinates is divided into 10 segments and 4 measures are computed, the input sample is a vector of 120 values.

Another method of feature extraction is to convert each dimension of the whole time-series to its frequencies (e.g. using *Fast Fourier Transform* algorithm), find key *frequency*

measures (e.g. index, amplitude and phase value of maximum frequency, Euclidean norm of phase) for each dimension, then arrange the values to one list (Gillian, 2011). For example, if 6 frequency measures for each of 3 dimensions of the time-series are extracted, the input vector has the length of 18.

Both methods to extract features can be combined: distribution and frequency measures merged to one vector, maintaining the order of features for both the training and the prediction data.

Algorithm's *strengths*: it is able to classify high-dimensional vectors (i.e. long lists of feature values); a small number of examples are needed to train the model and a small amount of computations is needed to classify a new gesture (Gillian, 2011).

Algorithm's *weaknesses*: its ability to handle long vectors of data comes with the restriction to use a fixed number of features; sequential relationship of data points in the time-series is lost during feature extraction; feature extraction relies on the ability to explicitly select relevant features: if irrelevant or insufficient number of features are chosen, the model will perform poorly; if a new gesture has to be added or an existing one removed, the whole model for all gestures has to be retrained (Gillian, 2011).

2.4.4. Artificial Neural Network

The described SVM algorithm has one model to separate gestures based on differences between its features by distance or frequency measures. Direct differences of features may be insufficient to correctly recognize a gesture.

The Artificial Neural Network (ANN) algorithm has not only just one model for all gestures, but also a "hidden" computation stage which captures higher-level dependencies between the input data and classes (Schmidhuber, 2014). The ANN model consists of an input layer, the hidden layers and an output layer, and each layer has a fixed number of nodes interconnected with adjustable weights (Dey, Mohanty, & Chugh, 2012). A simple model is *static* (all input data are fed to the network at once), *feed-forward* (data processing goes only to one direction from one layer to the next) and has *one hidden layer* (Bishop, 1995).

In the classification problem, nodes in the input layer are input data points, nodes in a hidden layer can be viewed as clusters (or unlabelled sub-classes) and their weights as probabilities, that each input data point belongs to one of the clusters (Serrano, 2016). Nodes in the output layer are classes and their weights are probabilities that each cluster (a hidden node) belongs to one of the classes (Figure 2.14).

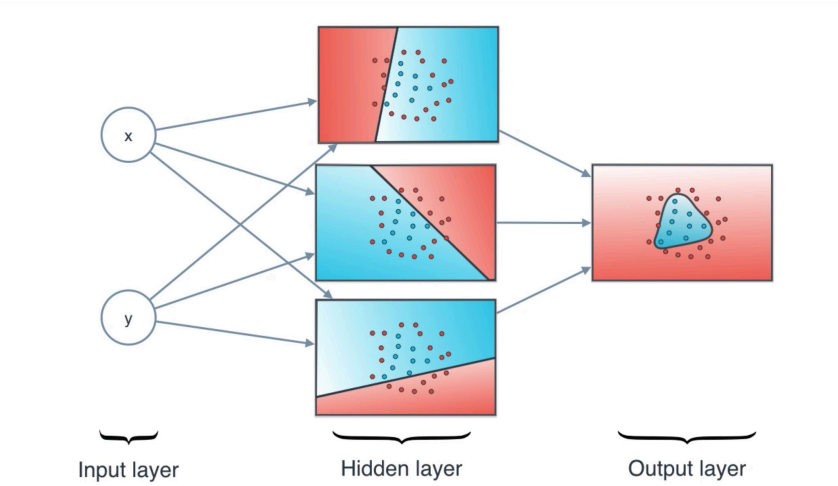

Figure 2.14. ANN layers, where the hidden layer divides data points to clusters. Screenshot of the video *A friendly introduction to Deep Learning and Neural Networks*, by L. Serrano, 2016. Copyright by L. Serrano.

ANN model's training is a process to find optimal weights – probabilities which, based on input data (input layer), correctly predict clusters (hidden layer) and eventually a class (output layer). Optimal weights can be found using *Backpropagation* algorithm with *Stochastic Gradient Descent*: the algorithm gradually updates weights starting from the output layer in order to minimize the classification error (LeCun et al., 2015).

Finding the best weights for all nodes takes a huge number of iterations and is computationally very expensive, therefore the number of input nodes (amount of input values) should be limited (Gillian, 2011). Input data should also be pre-processed to eliminate noise and avoid overfitting (Gillian, 2011). In a similar fashion to the SVM algorithm, features like *distribution* and *frequency* measurements of time-series, *velocity* and/or *acceleration* of skeleton joints could be extracted from the raw data, arranged to a list of values matching the number of input nodes (called "feature vector") and fed to the ANN model (Gillian, 2011).

Once the model is trained, a new gesture is classified by extracting the same features from the time-series, applying the optimal weights on each layer, and selecting the class of an output node with the highest value (Gillian, 2011). If the highest value is smaller than a threshold, the time-series is classified as a non-gesture (Gillian, 2011).

Static ANN model does not consider sequential nature of time-series. *Time Delay Neural Networks* (TDNN), in contrast, are used to take into account time in pattern-over-time recognition like speech, writing and movement (Peddinti, Povey, & Khudanpur, 2015). In TDNN, each node accepts values only from a subset of nodes ("sliding window") instead of all nodes in the previous layer (Peddinti et al., 2015). This method not only encourages the model to cluster neighbour data points in each layer, but also reduces the amount of required

computations (Peddinti et al., 2015). Input for TDNN should be features which preserve time as one of the factors (i.e. frequency measures of the whole time-series are not suitable).

Another type of ANN where time is viewed as an internal mechanism are *Recurrent Neural Networks* (RNN). In RNN, input data is processed in time-steps: a subset of input data ("sliding window") is fed to a RNN model through input nodes and hidden nodes additionally receive weighted values of "context" nodes, which are hidden node values of the previous time-step (Kouchi & Taguchi, 1991). RNNs are widely used in deep learning systems, but are computationally very expensive because of its recurrent mechanism and take too long to infer for real-time recognition systems (Schmidhuber, 2014).

Algorithm's ANN *strengths*: it is able to generalize and classify complicated data features thanks to hidden layers; classification of new data requires little computation, once the model is trained (Gillian, 2011).

Algorithm's ANN *weaknesses*: it requires a complex pre-processing to prepare feature vector for input; model training requires a lot of computational power and is not reliable because the learning algorithm can stop adjusting when locally optimal weights are found although globally better parameters are possible; the entire model has to be retrained when the number of gesture classes changes (Gillian, 2011).

For the system of this thesis, the DTW algorithm was chosen to classify gestures because it does not require a large dataset and many iterations to train the model, it rapidly classifies new gestures, it is able to handle multidimensional time-series data of varying length, it does not need feature extraction, and finally, adaptive feature can be implemented thanks to the model's short retraining time.

2.5. Post-processing

Once a gesture is classified, it has to trigger the playback of an audio track, sample or note in order to transform dance movement to music.

A convenient way to connect gestures to musical elements is through a standard communication protocol like *Musical Instrument Digital Interface* (MIDI) or *Open Sound Control* (OSC). MIDI and OSC are widely used to interconnect musical instruments, audio mixers, light and video installations (Jensenius, Godøy, & Wanderley, 2005). If a recognition system is able to emit the standard signal, any supporting audio platform can be chosen to implement the last processing stage "Output", discussed in the next chapter.

There are several software environments which are not only able to emit MIDI or OSC signals, but also serve as a framework to build the whole gesture recognition system: accept data from input sensors, extract features, segment gestures and use machine learning algorithms to classify them (Gillian, 2011). These software environments usually have graphical user interfaces and visual programming languages which allow to build a system without coding in traditional programming languages. Environments include various libraries, plugins and toolboxes that meet different needs of various signal processing tasks and human-computer interfaces.

Commercial (1) Matlab/Simulink and (2) Max/MSP as well as open-source (3) Pure Data and (4) EyesWeb software programs were considered to provide a framework for the recognition system of this thesis.

2.5.1. Matlab/Simulink

Matlab is a computation environment with its proprietary programming language for analysing and visualizing multi-dimensional data, developed and sold by the company MathWorks (MathWorks, 2018b). The environment is extendable by in-house modules called toolboxes – a set of Matlab functions developed for a specific purpose. For example, *Image Acquisition Toolbox* is able to receive image, depth and skeleton tracking data from Microsoft Kinect device; *Computer Vision System Toolbox* provides algorithms for object detection and feature extraction; *Machine Learning Toolbox* has implementations of SVM, HMM, k-Means, k-Nearest Neighbour algorithms; *Neural Network Toolbox* has implementations of various types of ANN; *Audio System Toolbox* enables to output MIDI signals (MathWorks, 2018a).

Matlab's additional package *Simulink* provides a graphical user interface, which allows to design and evaluate dynamic systems (Figure 2.15).

In Simulink, computation steps are visualized as blocks, which are connected by arrows indicating data flow. Every block is a piece of Matlab code, function or toolbox with fields to enter necessary parameters.

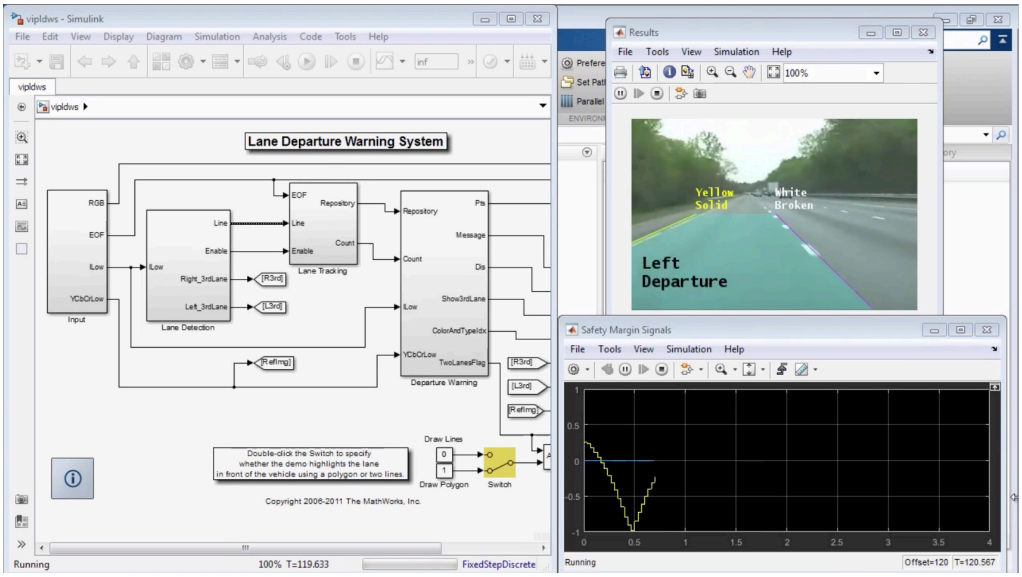

Figure 2.15. Matlab/Simulink user interface, an example project which detects lane markings in a video stream. Screenshot of the video *Computer Vision System Toolbox – MATLAB & Simulink*, by MathWorks, 2018. Copyright by MathWorks.

2.5.2. Max/MSP

Max is a signal processing environment and visual programming language designed specifically for music and multimedia purposes, it is developed and sold by the company Cycling ’74 (Cycling ’74, 2018). Max has a graphical user interface, where signal processing steps are visualized by blocks (called objects) connected with lines indicating data flow. Blocks can be integrated Max functions, JavaScript code or external modules.

Figure 2.16. Max/MSP user interface, an example project recognizes gestures using Kinect device. Screenshot of the video *Pose and Gesture Recognition using Kinect 2 skeleton tracking and Machine Learning Techniques in Max MSP Jitter - Vimeo*, by M. Akten, 2015. Copyright by M. Akten.

Max comes with add-on packages *MSP* and *Jitter*: MSP allows real-time manipulation of digital audio signals and Jitter adds real-time video processing ability (Figure 2.16).

Environment's functionality is extendable by third-party modules called external objects. For example, *Jit.OpenNI* object supports Microsoft Kinect RGB, depth and skeleton input (Phurrough, 2018); *Musical Gesture Toolbox* is a collection of modules made specifically for gesture analysis in video (Jensenius et al., 2005); *Num.DTW* object uses an extended "multi-grid" DTW algorithm to recognize gestures without prior segmentation (Bettens & Todoroff, 2009); *NNLists* object implements feed-forward back-propagation ANN (Robinson, 2018); *HMMM* is an implementation of HMM algorithm (Visell, 2018).

Max/MSP environment has an integrated support of MIDI and OSC signals, which can be received as input or transferred as output.

2.5.3. Pure Data

Pure Data is another visual programming environment for audio processing, created by software engineer Miller Puckette and conceptually similar to his previously co-developed software Max (Puckette, 2018). It is free and open-source and is therefore being maintained and improved by a global community of enthusiasts and researchers.

Pure Data focuses on real-time processing for live music and multimedia performances. Just like Max, the graphical user interface consists of blocks called objects where data processing takes place which are interconnected with lines (Figure 2.17).

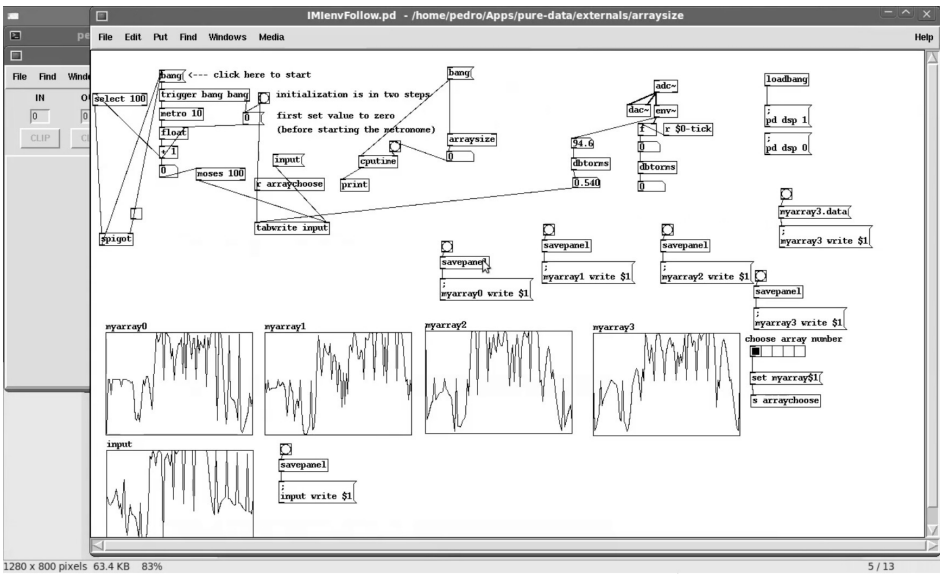

Figure 2.17. Pure Data user interface, an example project classifies gestures using DTW algorithm. Screenshot of the video *Dynamic Time Warping in PureData (alpha) - Vimeo*, by P. Lopes, 2010. Copyright by P. Lopes.

The environment is extendable by third-party modules called externals. As Pure Data is similar and to some degree interoperable with Max/MSP, the open-source community has developed unified externals suitable for both Pure Data and Max. For example, *ML.Lib* is a library of machine learning externals for both platforms, which has some functions for feature extraction and implementations of DTW, HMM, SVM and ANN algorithms (Bullock & Momeni, 2015).

Pure Data natively supports MIDI and OSC protocols for input and output of audio signals.

2.5.4. EyesWeb

EyesWeb is a visual programming environment to develop real-time multimodal systems and interfaces, it is created and maintained by the international research centre Casa Paganini - InfoMus Lab, it is free and open-source (Infomus, 2018).

Real-time systems are designed using a graphical user interface with blocks where data processing takes place interconnected with lines indicating data flow (Figure 2.18).

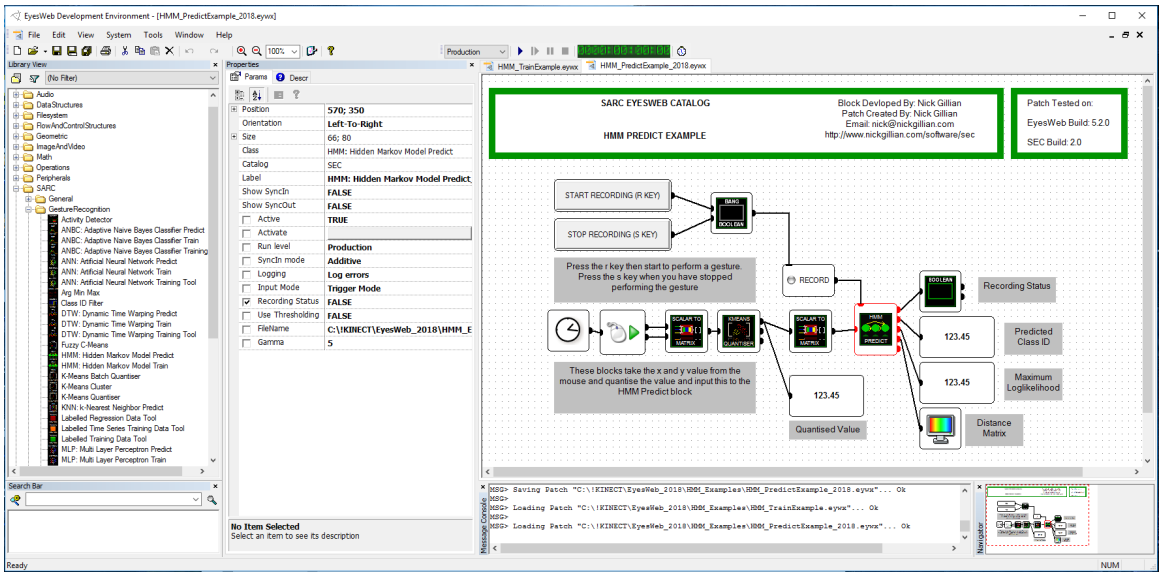

Figure 2.18. EyesWeb user interface, an example project classifies gestures using HMM algorithm. Screenshot of the project *HMM Predict Example*, by N. Gillian, 2011. Copyright by N. Gillian.

The environment natively supports wide number of motion capture devices including Microsoft Kinect, MIDI and OSC standards as input and output signals, includes in-house and third-party libraries for feature extraction and classification. For example, *SARC EyesWeb Catalog* (SEC) is a collection of blocks designed specifically for real-time gesture recognition

in musical-computer interfaces (Gillian, Knapp, & Modhrain, 2009). SEC has implementations of k-Means, k-Nearest Neighbour, Principal Component Analysis algorithms for feature extraction, of DTW, HMM, SVM, ANN algorithms for classification, mathematical functions for multi-dimensional data processing, as well as various blocks to structure and visualize data flow.

Matlab/Simulink framework is slow and more suitable for off-line simulation and analysis. Max/MSP and Pure Data are made for audio signal manipulation rather than gesture recognition.

EyesWeb environment with SEC catalogue was chosen for the project of this thesis as it perfectly fits all needs. EyesWeb is open-source and is in active development for the last 20 years – last version 5.7 was released in January 2017 and the next regular workshop is scheduled in July 2018 (Infomus, 2018).

2.6. Output

In the last processing stage MIDI or OSC signals sent by the "gesture engine" have to be received by "music engine". Receiver of the signal should facilitate a dancer to create harmonic music: it should have a prearranged set of audio tracks, loops or samples that sound well when combined. Also, it should assist the performer to play triggered tracks at the right time (i.e., in synchronization with song's tempo and bar measure).

There are many software packages known as *Digital Audio Workstations* (DAW) and *MIDI sequencers* that can function as the "music engine". For the project of this thesis are considered commercial products (1) Ableton Live and (2) Bitwig Studio as well as free open-source project (3) LMMS.

2.6.1. Ableton Live

Ableton Live is a DAW for music creation and live performances, developed and sold by the company Ableton AG (Ableton, 2018). Its graphical user interface has two views: in *Arrangement View* virtual music instruments are arranged in a timeline, in *Session View* multiple melodies or rhythms are selected for every instrument (Figure 2.19).

Melodies and rhythms (called "clips") are chosen from an internal sound library, imported from external sources, recorded or generated with a built-in MIDI controller. There is

a large list of built-in audio effects to modify sound clips and overall live set. Tempo and bar measurement are defined for the whole arrangement of music elements.

Figure 2.19. Ableton Live user interface, Session View of an example project. Screenshot of the project *Live 9 Demo*, by Ableton, 2018. Copyright by Ableton.

At a live performance using Session View, pre-arranged music elements are triggered by MIDI signals (OSC signals are supported only with a third-party plugin) mapped to melodies and rhythms. When a music clip in Session View is toggled, it starts playing only when a new bar begins and plays in loops until it is toggled again or another element of the same instrument is triggered.

2.6.2. Bitwig Studio

Bitwig Studio is a relatively new DAW developed by the company Bitwig GmbH, founded by ex-Ableton engineers (Bitwig, 2018). Just like Ableton, its graphical interface has *Arrange* view and *Mix* view: in Arrange view, music instruments (called “track”) are arranged in a timeline, in the Mix view, every track has a stack of music elements (called “clips”) like melodies and beats (Figure 2.20).

Every clip can be mapped to a MIDI value and triggered by an external MIDI controller which sends this value. Clips are adjusted (stretched on shrank) to match tempo and bar measure selected for the whole composition. When a clip is triggered, it starts to play in the beginning of next bar and plays in a loop until it’s triggered again or another clip of the same track is triggered.

Figure 2.20. Bitwig Studio user interface, Mix view of an example project. Screenshot of a *demo project*, by Bitwig, 2018. Copyright by Bitwig.

Clips can be generated with built-in and third-party plugins, selected from the internal library or imported from external sources. Various audio effects (like delay, reverb, chorus) can be applied to the whole composition, specific instruments or clips, effects and main controls (like volume, gain, pitch) can be controlled by MIDI signals received from an external controller.

2.6.3. LMMS

LMMS is a free and open-source DAW, created by Paul Giblock and Tobias Doerffel and has a large community of contributors to the project development (Giblock & Junghans, 2018). Its user interface has a *Song Editor* window where instruments are arranged in the timeline and a *Beat+Bassline Editor* window where, as the name indicates, clips for Beat/Bassline instrument are stacked (Figure 2.21).

Only Beat/Bassline instrument can have multiple clips stacked for looped playback and a composition can have only one list of Beat/Bassline clips. Every clip can be triggered by an external MIDI input, but it starts to play as soon as it is toggled and does not stop playback of another clip in the stack. Instruments in Song Editor can be toggled by MIDI signal too and they start to play when the next bar begins.

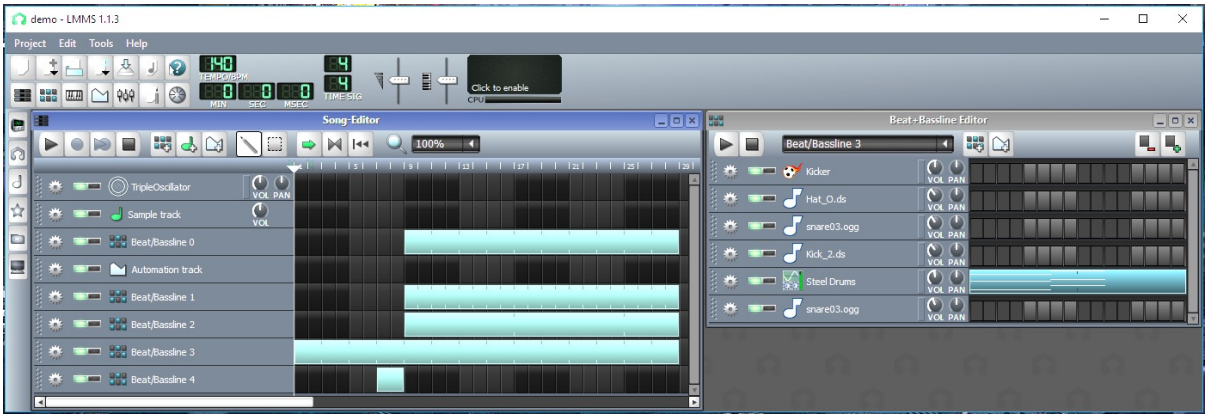

Figure 2.21. LMMS user interface, an example project.
Screenshot of a *demo project*, by LMMS, 2018. Copyright by LMMS.

LMMS is made for music composition and does not have an interface dedicated specifically for live performance, but a project can be tweaked to perform real-time. For example, part of the timeline in Song Edit window can be selected to loop and during its playback, instruments and clips toggled with an external MIDI controller. External controllers must communicate using MIDI signals as LMMS does not support OSC protocol.

LMMS program is an attractive option because it is free and open-source, but is not designed for live performances and does not have features assisting a real-time manipulation of musical elements. Both Ableton Live and Bitwig Studio have required features, therefore can be conveniently used in interactive dance systems.

Ableton Live was chosen for the project of this thesis, because it has very large community of users who have created a lot of online tutorials and Q&A discussions.

2.7. Summary

In order to design a real-time recognition system able to recognize temporal gestures, a developer has to make decisions on 6 stages of the data processing chain. Based on related work in the literature, a few viable options for each stage were discovered.

In the first stage, “Input” could be received from Marker-less Video Cameras, Video Cameras with Markers, Wearable Sensors or Depth Cameras. For the second stage “Feature extraction”, two approaches Body Contour and Body Skeleton were discussed. In the third stage “Segmentation”, extraction of separate gestures from the data stream can be done based on Body Tracking or Music Metrics. The most important stage “Classification” requires the selection of a machine learning algorithm; strengths and weaknesses of four most popular and promising algorithms were presented: Dynamic Time Warping, Hidden Markov Models,

Support Vector Machines and Artificial Neural Networks. Tools for the fifth stage “Post-processing” provide a framework to build the whole recognition system; commercial Matlab/Simulink, Max/MSP and open-source Pure Data, EyesWeb software packages were considered. For the last stage “Output”, a digital audio workstation like commercial Ableton Live, Bitwig Studio and open-source LMMS has to be selected.

Next chapter presents a vision of the gesture recognition system, which would transform dance to music. Then, decisions are made and justified for each processing stage to design such a system.

3. System Design

The system, envisioned for this thesis, has a music composition of three virtual instruments (in DAW called as “tracks”) which have to be controlled by different body parts: a beat controlled by the hips, a baseline by the legs, and a melody by the hands. Every track has a set of musical loops (in DAW called as “clip”) mapped to specific gestures of an appropriate body part. Dance gestures must begin and end along with the music composition’s bar measure.

The system learns gestures before the main performance: for each clip in a track, the performer repeats the same gesture multiple times in synchronization with the clip played in a loop, positional data is recorded, pre-processed, labelled and used to train a classification model. Once the model is trained, the system is ready for a live performance.

At a live performance, the dancer performs trained gestures to play specific clips. When a gesture is recognized, the mapped clip has to be played in a loop as long as the same gesture is repeated. If another gesture is recognized, playback of another clip should replace the first clip. If a performer does not repeat the gesture, playback of the clip should be stopped.

In order to build this kind of gesture recognition system, the author made decisions for every stage of the processing chain: (1) Input, (2) Feature Extraction, (3) Segmentation, (4) Classification, (5) Post-processing, (6) Output.

3.1. Input

As discussed in the chapter 2.1, video cameras with and without markers, wearable sensors and depth cameras were considered to provide input data.

Microsoft Kinect v2 device was chosen as the best option, because it is able to provide depth image, as well as detect human body and extract 3D coordinates of “skeleton joints” (i.e., locations of the main body parts). Kinect v2, released in 2014, uses state-of-the-art time-of-flight technology to compute depth information, and streams 512x424 pixels resolution depth and infrared image at 30 fps as well as 1920x1080 pixels resolution RGB video at the same frame-rate (Lun & Zhao, 2015). It has 70 degrees horizontals and 43 degrees vertical field of view and can sense depth at the range from 0.5m to 8m, as well as detect a human body up to 4.5m (Lun & Zhao, 2015).

System requirements for Kinect v2 device are Windows 8 or later OS, 64-bit (x64) dual-core 3.2 GHz or faster CPU processor, 2GB or more RAM memory and dedicated USB 3.0 bus (Microsoft Corporation, 2018).

3.2. Feature Extraction

As discussed in the chapter 2.2, the amount of input data should be reduced in the way that it keeps the key information to detect a human body and segment it to labelled body parts. Body contour and body skeleton approaches were considered for feature extraction.

Kinect SDK 2.0 was chosen to extract relevant features from the depth image. Version 2.0 can track up to 6 people at the same time and provide coordinates of 25 joints per skeleton (Lun & Zhao, 2015). For each joint, SDK provides absolute (in relation to the camera) and hierarchical (in relation to its parent joint) position and orientation values in x, y and z axis.

For an interactive dance system, only one person needs to be observed and selected joints tracked to trigger virtual instruments:

- to trigger a *beat*: absolute position of left and right *Hips* (position in relation to the sensor); for sophisticated gestures, *Torso* and *Head* could be tracked additionally;
- to trigger a *baseline*: hierarchical position of left and right *Ankles* (position in relation to Torso); for sophisticated gestures, *Knees* and *Feet* could be tracked additionally;
- to trigger a *melody*: hierarchical position of left and right *Hands* (position in relation to Torso); for sophisticated gestures, *Shoulders*, *Elbows* and *Hands* could be tracked additionally.

Orientation data is not necessary for recognition as this information overlaps with position data, it is more useful for visualization of the skeleton.

3.3. Segmentation

As discussed in the chapter 2.3, segmentation based on body tracking or music metrics was considered for the system.

Music metrics was preferred and *bar measure* was chosen as the most suitable approach to split data stream to distinct gestures, because it is quick and robust (i.e., it does not require data processing and always divides data to the same time periods).

Time-series of fixed length is beneficial for time-based classification algorithms like DTW and HMM because the data is less distorted by pre-processing.

The method relies on the requirement that a performer has to match dance gestures with composition's predefined bar measure. This restriction is beneficial for performers too, because it encourages them to repeat gestures with better precision.

3.4. Classification

DTW, HMM, SVM and ANN algorithms, described in the chapter 2.4, were explored and considered for the system.

Dynamic Time Warping was chosen as the most suitable algorithm, because it accepts raw time-series as input data and requires relatively small amount of computation to train the model. Accepting raw time-series allows to skip data pre-processing like Down-sampling, Vector Quantization or Frequency Features Extraction, which makes classification of unknown gesture faster therefore more suitable for real-time recognition. Sparse computation means a short time needed to train the model, which not only makes the system more convenient to use (recording, training and prediction can take place at the same work session), but also enables extension of the system to be adaptive (model can be retrained with additional training examples during prediction session).

In order to reduce noise and amount of computations, every time-series was pre-processed before feeding it to the model for training: data was *down-sampled by factor 6*, which means 60 frames were down-sampled to 10 frames for every sample of a gesture.

Thresholding using a *gamma coefficient* was used to classify unrecognized samples as non-gestures. Gamma greatly affects the accuracy of the model, coefficients 0-6 were tested, it is discussed in the chapter 5.3 “Results”.

3.5. Post-Processing

Matlab/Simulink, Max/MSP, Pure Data and EyesWeb environments, discussed in the chapter 2.5, were considered to transform a predicted class to a sound trigger and build overall “gesture engine”.

EyesWeb framework was selected as the best choice because it is designed for building human-computer interaction systems and has a large library of tools made specifically for gesture analysis and music control. Visual programming language is used in this environment which greatly reduces the need for programming skills and the workload experimenting with different system designs.

It is free, open-source and has been actively used and extended by the community of artists and researchers. One of the most remarkable extensions is *SARC EyesWeb Catalog v2.0* (SEC) – a collection of blocks designed to exploit machine learning algorithms for gesture recognition purposes. Many blocks of this catalogue were used to explore classification

algorithms and build the system. For example, in order to exploit DTW classifier, blocks “DTW Train” and “DTW Predict” were used in the data processing chain.

SEC blocks were developed by Nicholas E. Gillian at Queen’s University Belfast (Gillian, Knapp, & Modhrain, 2009). Equations of DTW algorithm, implemented to SEV blocks by N.E. Gillian, are presented and explained in the *Appendix A*. During development of the system, the author of this thesis contacted N.E. Gillian personally and received valuable consultation regarding usage of his modules.

3.6. Output

As discussed in the chapter 2.6, commercial digital audio workstations Ableton Live, Bitwig Studio, and open-source LMMS were considered to function as a “music engine”. All options are able to accept a standard MIDI or OSC signal and trigger musical instruments based on a gesture-clip mapping, but only the commercial products have features needed for live performances.

Ableton Live was chosen to be used as system’s output, because this software is widely used by musicians for many years, therefore a large knowledge base is available on the internet.

Signal protocol OSC was chosen for communication between “gesture engine” and “music engine”. The system made use of Ableton Live’s Session View where audio clips for every gesture can be stacked and an assistance to play clips in synchronization with composition’s tempo and bar measure.

3.7. Summary

In the vision of the gesture recognition system, virtual instruments are coupled with body parts. The system has to learn gestures of each group of body parts and know which gesture plays which clip in a music composition.

Three main components were selected in the design of the envisioned system: Kinect v2 device, EyesWeb framework and Ableton Live digital audio workstation (Figure 3.1). Data from one component to the next flows over OSC signal.

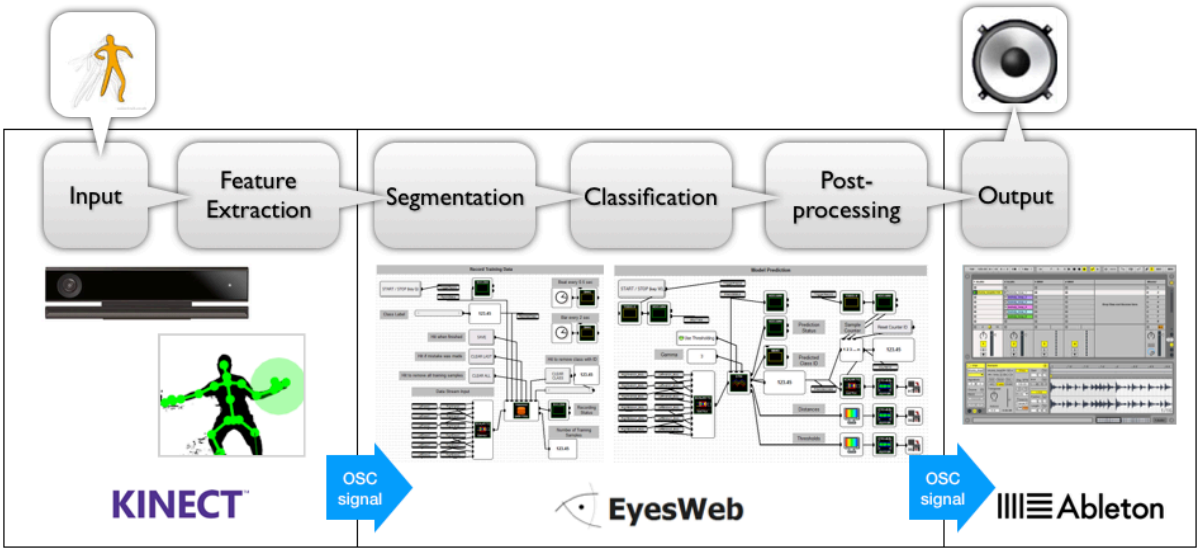

Figure 3.1. Main components chosen to implement the gesture recognition system.

In the first stage "Input", depth camera Kinect v2 was picked up to provide the stream of data. For the second stage "Feature extraction", Body Skeleton approach was adopted exploiting Kinect SDK's ability to provide 3D coordinates of 25 joints of dancer's body. In the third stage "Segmentation", data stream was split to separate gestures based on Music Metrics, specifically – the length of song's bar measure. For the crucial stage "Classification", Dynamic Time Warping algorithm was chosen to classify gestures. EyesWeb framework not only served as a tool for the fifth stage "Post-processing" but also provided an environment where the whole recognition system was built. The last stage "Output" employed the digital audio workstation Ableton Live, which is able to manage the playback of music clips to create a harmonic composition.

The next chapter presents the implementation of system's prototype which is able to recognize one group of body parts and play clips of one virtual instrument. Setup of the hardware and the software is described, and the system's modules responsible for various tasks are presented. Finally, the procedure for a user and the workflow of the system is explained. There is also a description of how the system's model is trained to recognize specific gestures and the prediction of new gestures is executed during a live performance.

4. System Implementation

A proof-of-concept system was built to track *one group* of skeleton joints and play clips in *one audio track* based on recognized gestures: *hands* and *elbows* tracked to play *melody* clips. If this kind of system is able to provide sufficient results, it can be extended in future projects to track multiple body parts mapped with multiple virtual instruments.

4.1. Sensor Setup

Kinect v2 device was connected via USB 3.0 to a desktop computer with 64-bit Windows 10 operation system. The computer had Intel i5-6600 processor, Nvidia GTX1060 graphical card and 8GB of RAM. A 1080p projector was connected to provide visual feedback on 72" screen.

Kinect sensor was sending the stream of Skeleton data as well as RGB and depth image to *Kinect SDK 2.0*. Skeleton data from SDK was broadcasted via OSC protocol using the open-source utility *Kinect2share* developed by Ryan Webber (Webber, 2018). Kinect2share utility's parameters are described in the *Appendix B*.

OSC signal could be observed using the open-source utility *OSC Data Monitor* developed by Kasper Kamperman (Kamperman, 2018; Figure 4.1).

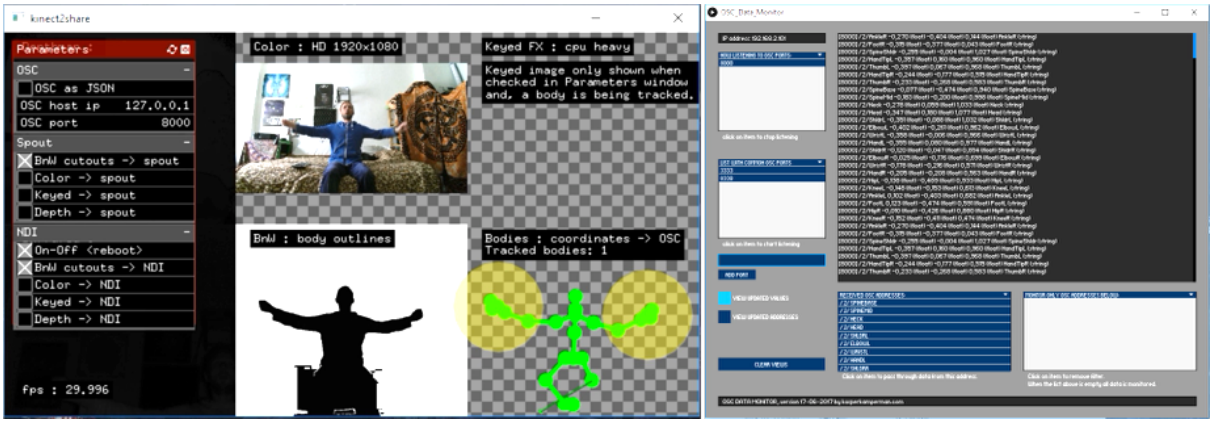

Figure 4.1. User interfaces of Kinect2share (left) and OSC Data Monitor (right).

Broadcasted OSC was received in EyesWeb environment for further processing. "Gesture engine" was built using *EyesWeb v5.2.1* visual programming language and divided to two projects, Training Patch and Prediction Patch.

4.2. Training Patch

The first project in EyesWeb was created to train DTW model to classify gestures. It consists of several modules. All parameters of the Training Patch are described in the *Appendix C*.

Module *"Input from Kinect"* receives OSC signal broadcasted from Kinect2share and separates it to 12 streams of data values: x, y, z coordinates of 4 skeleton joints (i.e., left and right hands and both elbows) (Figure 4.2).

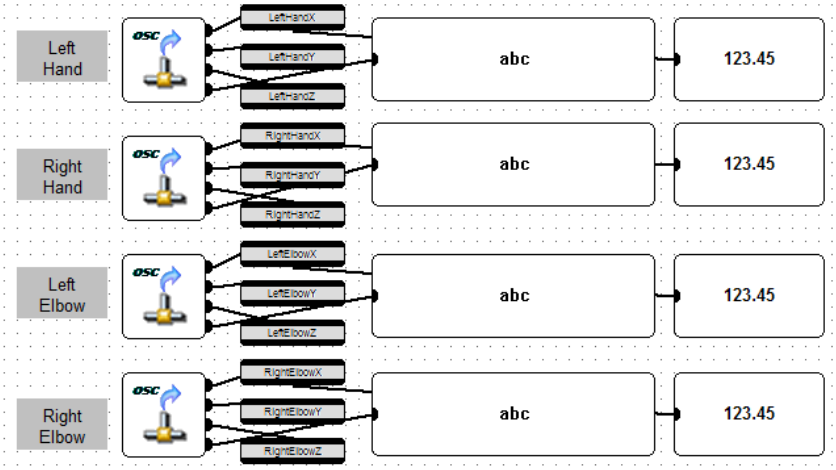

Figure 4.2. EyesWeb module "Input from Kinect" in the Training Patch.

Module *"Record Training Data"* combines 12 streams of data values to a matrix, divides it to time-series and saves it with manually selected class ID (Figure 4.3). Status indicators provide visual feedback when data is recorded, visual metronome indicating every beat and bar, number of classes and recorded time-series. Visual metronome is controlled by periodic timers set to 0.5 second for every beat and 2 seconds for every bar. Recording session and saving of recorded time-series is controlled with Start, Stop, Save and Clear buttons.

Module *"Timer for Recording"* is responsible for dividing the stream of data to time-series of equal length to be used as training examples (Figure 4.4). Periodic timer set to 2 seconds with 50ms delay sends a signal to *start* recording data, another timer of 2 seconds sends periodic signals to *stop* recording data, which results in 1 sec 950 ms time-series. 50 ms "break" is needed for the module "Record Training Data" to add last time-series to training examples.

Module *"Music Clip"* sends OSC signal to Ableton Live to control, which audio track and music clip has to be played (Figure 4.5).

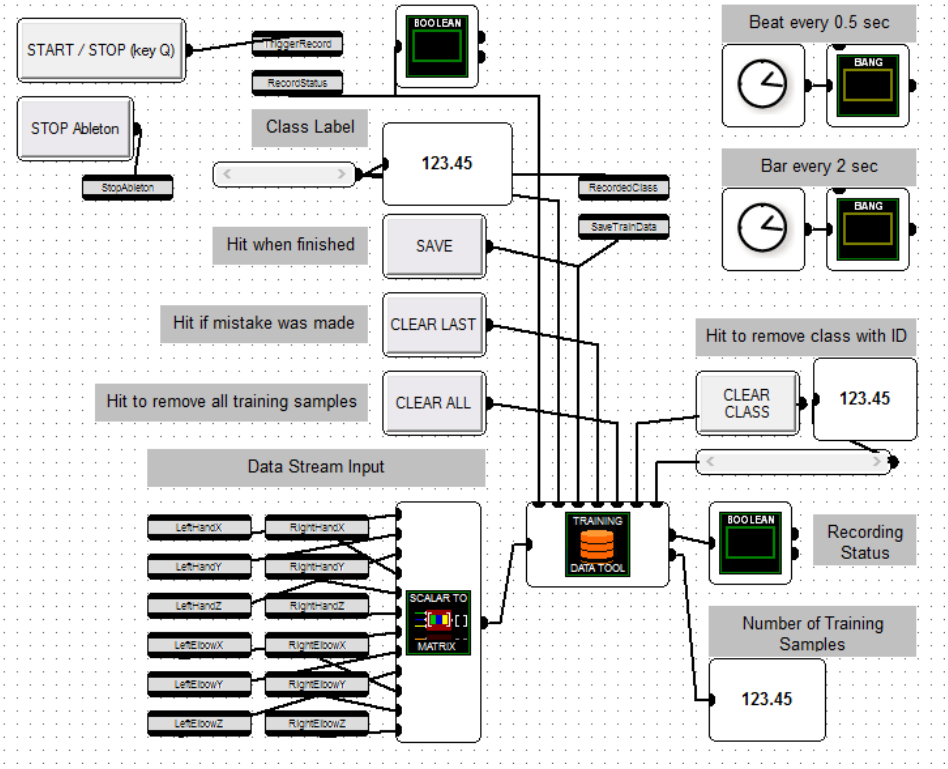

Figure 4.3. EyesWeb module “Record Training Data” in the Training Patch.

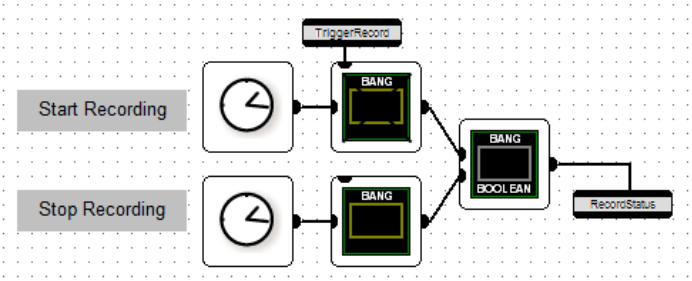

Figure 4.4. EyesWeb module “Timer for Recording” in the Training Patch.

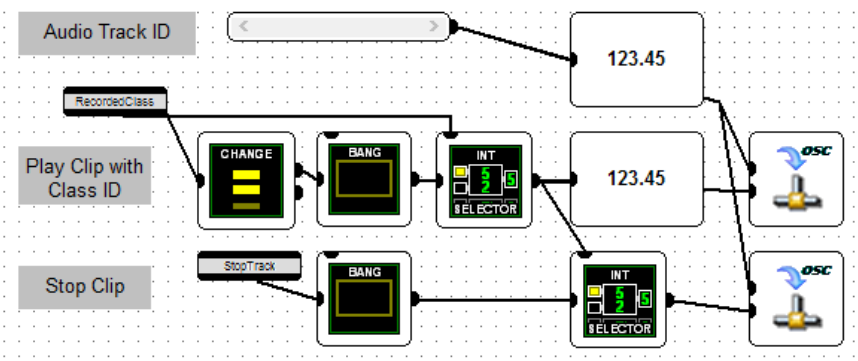

Figure 4.5. EyesWeb module “Music Clip” in the Training Patch.

Here the audio track “melody” (track ID = 2) has to be manually selected. When recording is started in “Record Training Data”, the module automatically broadcasts *class ID* via OSC to start the playback of a music clip with the same *clip ID* in Ableton Live.

Module *“Ableton Control”* automatically sends a OSC signal for Ableton Live to run the music composition, when the EyesWeb project starts running (Figure 4.6). When Ableton Live receives the signal, the audio track “Drums & Percussion” starts to play immediately indicating that Ableton Live is ready to receive further OSC signals.

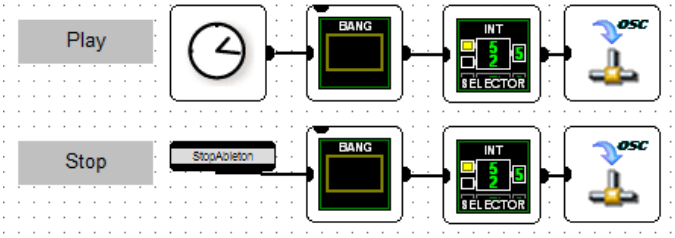

Figure 4.6. EyesWeb module “Ableton Control” in the Training Patch.

Finally, module *“Model Training”* is responsible for training the model with DTW algorithm (Figure 4.7). Training is triggered with “Train” button and training status displayed with an indicator.

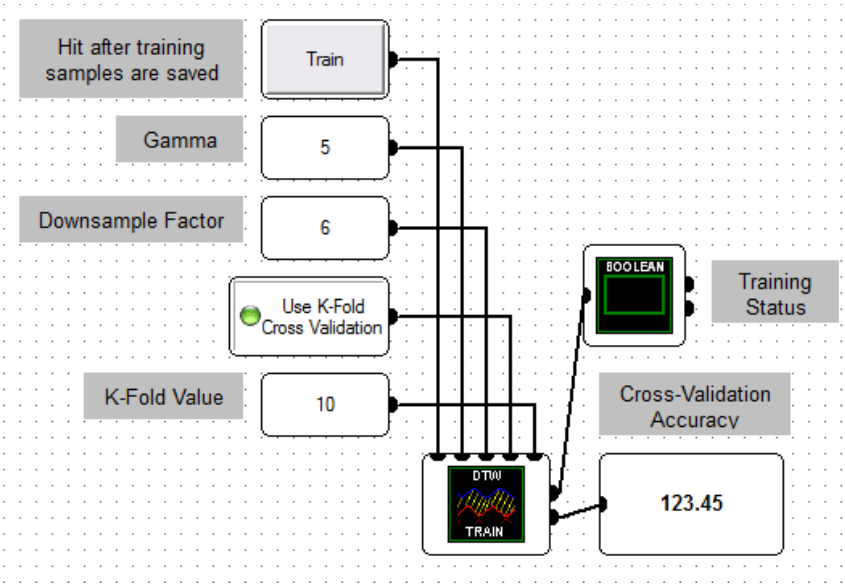

Figure 4.7. EyesWeb module “Model Training” in the Training Patch.

During training of the model, DTW algorithm finds the best template for each gesture. Once the model is trained, the model is saved in a file and 10-fold cross-validation accuracy value is displayed.

4.3. Prediction Patch

The second EyesWeb project classifies gestures based on the trained model and adapts the model based on recognized gestures. It consists of several modules. All parameters of the Prediction Patch are described in the *Appendix D*.

Just like in the first project, module *"Input from Kinect"* (Figure 4.2) receives OSC signal from Kinect2share and separates it to 12 streams of data values; module *"Ableton Control"* (Figure 4.6) automatically sends a OSC signal for Ableton Live to run the composition and indicate that "the music engine" is ready to receive further OSC commands.

Module *"Model Prediction"* is the place where gesture classification takes place (Figure 4.8). The module merges 12 data streams to one matrix and sends it to the block "DTW Predict," which splits the stream to time-series of equal length. The block then uses previously saved DTW model to compare each time-series to model's templates and select the class ID of a template with the smallest Euclidean distance. If the distance of the closest template is still larger than its threshold, the block predicts that time-series is a non-gesture (i.e. class ID = 0).

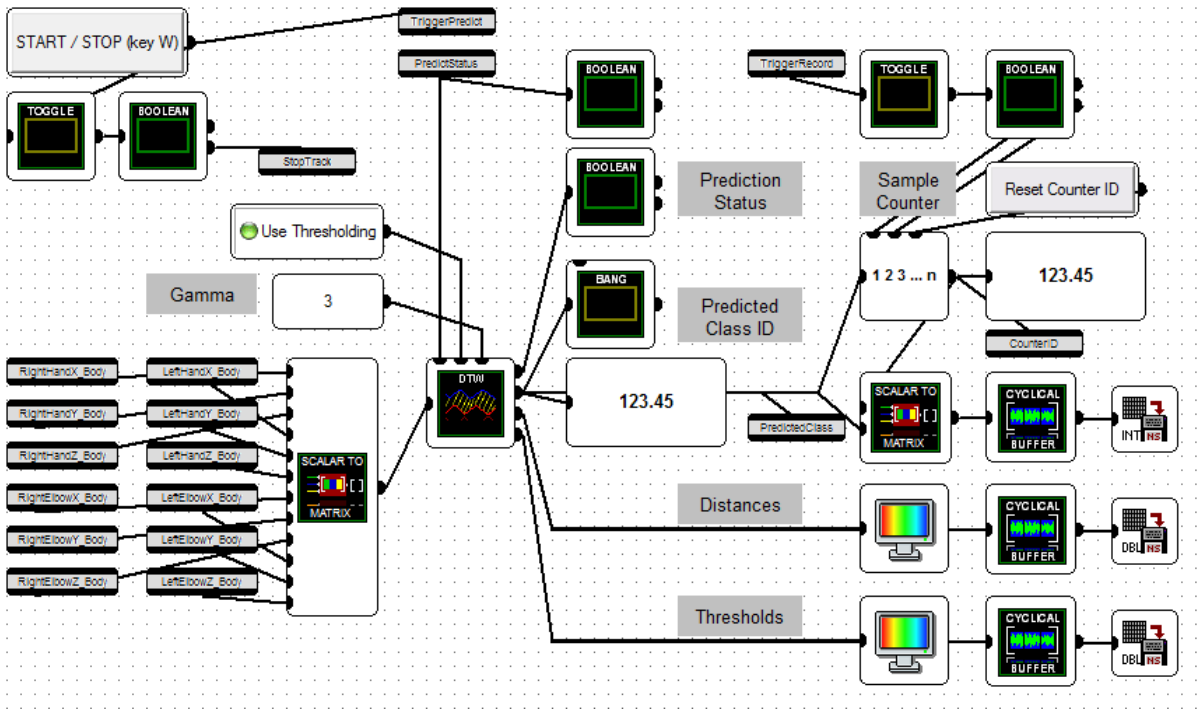

Figure 4.8. EyesWeb module "Model Prediction" in the Prediction Patch.

The module displays predicted class ID and additional information "Distances" and "Thresholds" indicating the confidence of prediction. Predicted class IDs, distances and

thresholds are saved to files for analysis. Visual feedback elements “Bang” and “Boolean” show the status of prediction.

Module *“Timer for Prediction”* controls time periods, when exactly the data stream has to be split to time-series (Figure 4.9). It is similar to “Timer for Recording” module in the first project, but was extended to suppress periodical STOP signals. If these signals are not suppressed and “DTW Predict” block receives it, it continues to emit the last predicted class ID even when prediction process is paused (bug workaround).

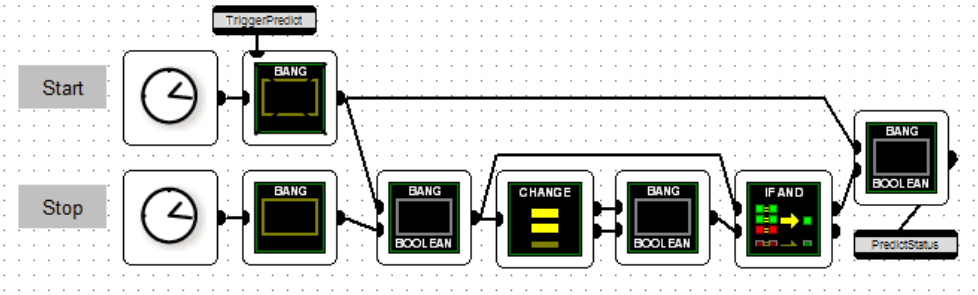

Figure 4.9. EyesWeb module “Timer for Prediction” in the Prediction Patch.

Predicted class ID is sent to the module *“Music Clip”*, which is responsible for sending OSC signals to Ableton Live (Figure 4.10). Here the audio track “melody” (track ID = 2) has to be manually selected. Once the module receives class ID of a recognized gesture, it sends a OSC command along with track ID and clip ID to play it in Ableton Live. If the model recognizes a repeated gesture and sends the same as previous class ID, OSC command is suppressed and Ableton Live continues to play the same clip. If the algorithm does not recognize a gesture and sends the class ID = 0, the module sends OSC command to play clip ID = 0 (an empty slot in Ableton Live) which actually stops playback of any clips at the end of the bar.

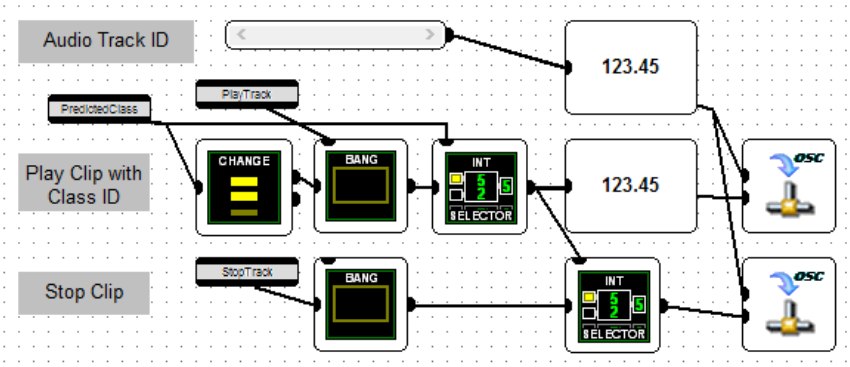

Figure 4.10. EyesWeb module “Music Clip” in the Prediction Patch.

Module *"Record Prediction Data"* runs in parallel with "Model Prediction" module and records time-series of last gestures (Figure 4.11). Differently than the module "Record Training Data" in the first project, class ID of recorded time-series is not set manually but received from the previous module "Model Prediction". Number of recorded time-series is sent to the next module "Sequence for Adaptive Training"; when a command from this module is received, all data is deleted and recording is restarted.

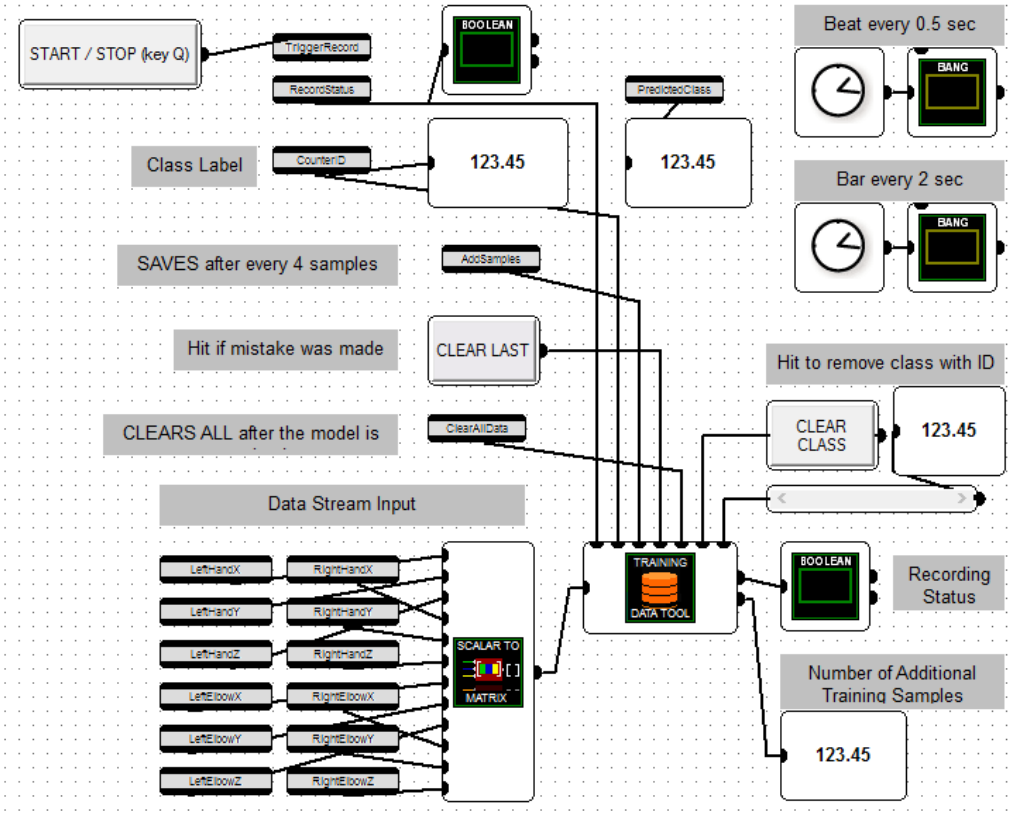

Figure 4.11. EyesWeb module "Record Prediction Data" in the Prediction Patch.

Module *"Sequence for Adaptive Training"* watches the amount of recorded time series and, when the number reaches 4, a sequence of events is initiated (Figure 4.12): (1) the command to save the training data to a file is sent to the previous module "Record Prediction Data"; (2) Python script, which merges prediction data file with the original training data file, is executed; (3) the command to clear all data is sent to the module "Record Prediction Data"; (4) the command to retrain the model is sent to the next module "Model Training".

Module *"Model Training"* uses merged training dataset to retrain the model with DTW algorithm (Figure 4.13). It has the same structure and parameters as module "Model Training" in Training Patch. Once the model is trained, it is saved to the file and immediately used by the

module “Model Prediction” to classify next gestures. Parameter “Use K-Fold Cross Validation” was set to FALSE in order to minimize data processing time.

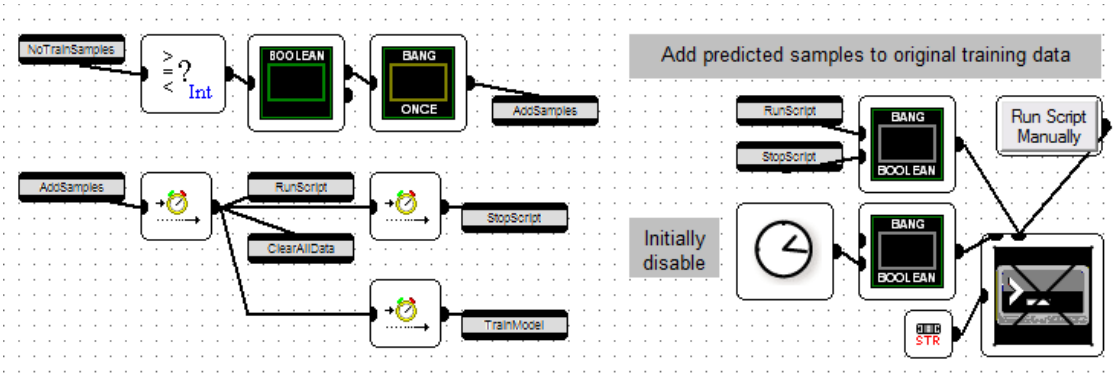

Figure 4.12. EyesWeb module “Sequence for Adaptive Training” in the Prediction Patch.

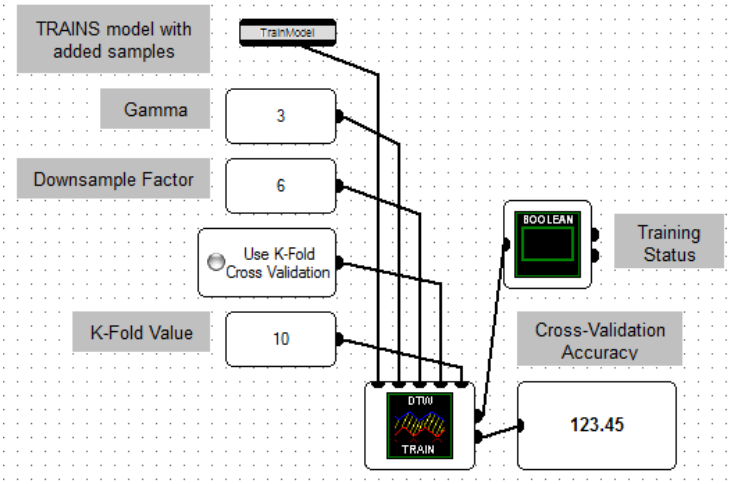

Figure 4.13. EyesWeb module “Model Training” in the Prediction Patch.

4.4. Composition Setup

A live-set project was created in *Ableton Live v9.0.1* with two audio tracks: (1) drums & percussion, (2) melody (Figure 4.14). In Session View, the first track “Drums & Percussion” has one clip, it will play in a loop immediately when playback of the whole live set is initiated via OSC protocol. It indicates that Ableton is receiving OSC signals.

The second track “Melody” (track ID = 2) has 6 clips of distinct melodies with clip IDs from 1 to 6. Clips are stopped by default and start to play only when OSC signal with clip ID is received. Clip ID with value 0 is an empty slot, which is triggered to stop the playback of any clip.

Figure 4.14. System's music composition in Ableton Live, Session View.

In order to enable OSC interface for Ableton Live, the open-source utility *LiveOSC* developed by Stu Fisher was used (Fisher, 2018). LiveOSC utility's parameters are described in the *Appendix B*.

Clips for audio tracks were downloaded from the website *Looperman.com*, where a community of musicians share their royalty free loops, samples and sounds (Looperman, 2018).

Composition's tempo was set to 120 beats per minute (BPM), time signature to 4/4. This means that a beat loops every 0.5 seconds, one bar consists of *4 beats* and lasts *2 seconds*. As a gesture must match the length of a bar, gesture's time-series should not be longer than 2 seconds.

To be able to hear the playback of music composition, speakers have to be connected to the operating computer. The author used a Sony amplifier and a pair of Nubert stereo speakers.

4.5. Workflow

In order to use the gesture recognition system for a live performance, the user has to set it up first:

1) Connect and power up *Kinect* device.
2) Open *Kinect2share* utility, make sure that the utility is receiving the stream of Skeleton 3D coordinates (it should display skeleton's image in the user interface).

3) Open *OSC Data Monitor*, make sure that Kinect2share is sending the coordinates over OSC (the monitor should display the updating list of OSC data); *close* OSC Data Monitor to release the listening port.
4) Open the music composition in *Ableton Live* with integrated utility *LiveOSC*.
5) Open *EyesWeb* projects *Training Patch* and *Prediction Patch*.
6) Arrange EyesWeb, Kinect2share and Ableton Live windows in the display so that relevant information is visible. Main user's control interface is the EyesWeb projects, Kinect2share and Ableton Live windows are needed only to monitor that they are operating as expected.

Once all programs and utilities are opened and arranged, the user has to record training data in EyesWeb's Training Patch:

1) Run the *Training Patch*, Ableton Live should automatically start playing the first track "Drums & Percussion".
2) Select *class ID* for recorded gesture, start with class ID = 1.
3) Click the button START/STOP in the module *"Record Training Data"* to start recording the data. The best time to click the button is in the beginning of a new bar (which has a length of 2 sec), it gives some time to prepare for the first gesture.
4) Execute a dance gesture when Ableton Live starts playing the clip with ID matching selected class ID, make sure it fits within one bar (2 sec).
5) Repeat the gesture listening to the rhythm of the composition and observing the visual indicators of beats and bars, try to make gestures as temporally and spatially similar as possible. Observe the number of training samples displayed in the module.
6) Once enough samples are recorded, click the button START/STOP again to stop recording. If needed, click the button CLEAR LAST to remove the last recorded sample.
7) Repeat steps 2-6 until all samples for all gestures are recorded, incrementing the *class ID* value. If needed, click the button CLEAR CLASS to remove a faulty class or CLEAR ALL to remove all samples of all classes and restart recording.
8) When all samples of all gestures with different class IDs are recorded, click SAVE to save recorded time-series to a file.
9) Do not stop *Training Patch*, keep it running for the next procedure.

Figure 4.15 visualises the data workflow of the described procedure and the result should be a DAT file with the training dataset saved to a predefined location.

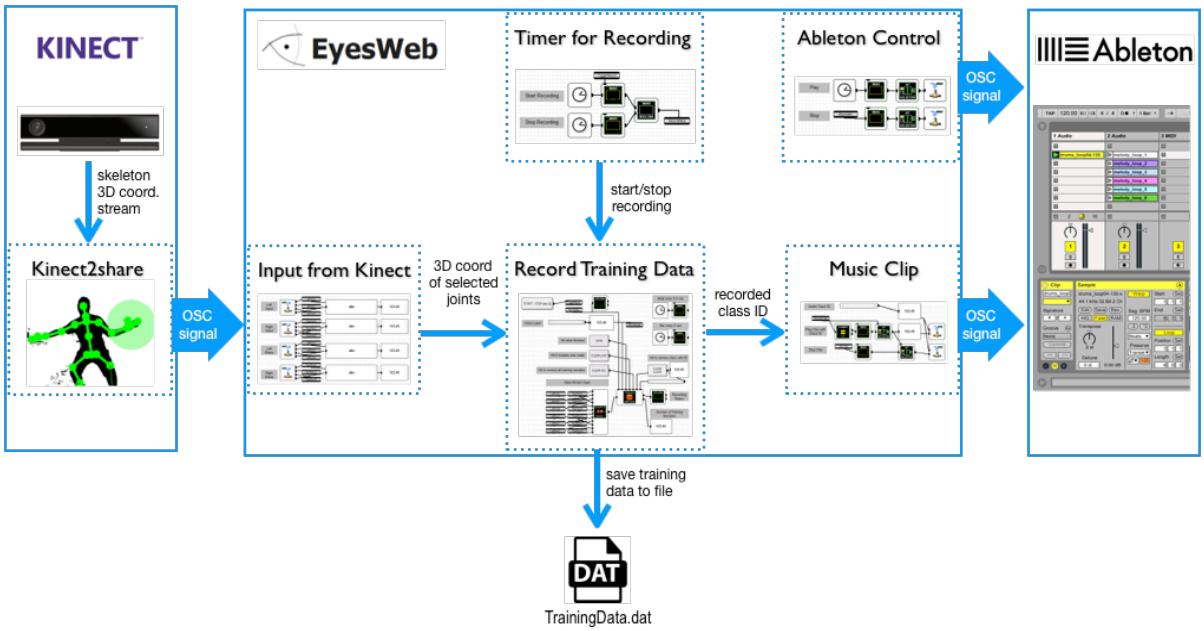

Figure 4.15. Workflow of the system during recording of the training data.

The next procedure is short and simple: to train the initial model based on recorded training data:

1) While *Training Patch* still running, click the button START/STOP in the module *"Model Training"* to train the model. The module should be predefined to use the training dataset from the file, saved in the previous procedure.
2) Stop the *Training Patch*.

Figure 4.16 visualises the data workflow of the described procedure, which result should be a DAT file with the trained model, saved to a predefined location.

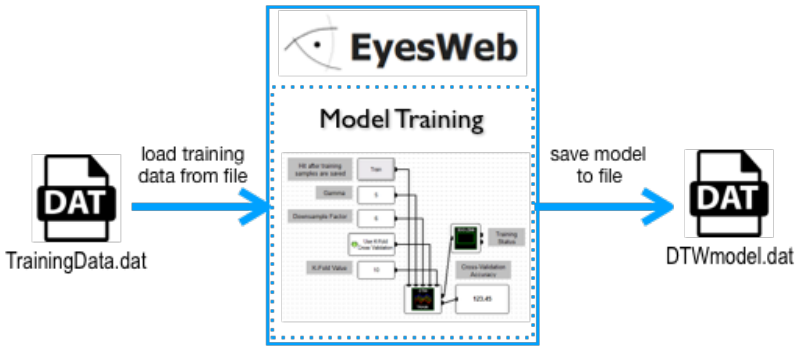

Figure 4.16. Workflow of the system during training of the DTW model.

The last procedure should be followed in EyesWeb's Prediction Patch during live performance to create music executing previously trained dance gestures:

1) Run the *Prediction Patch*, Ableton Live should automatically start playing the first track "Drums & Percussion".
2) Click the button START/STOP in the module *"Model Prediction"* to start prediction of new gestures. The best time to click the button is in the beginning of a new bar (which has a length of 2 sec), it gives some time to prepare for the first gesture.
3) Execute any of the previously trained dance gestures, make sure it fits within one bar (2 sec). At the end of the bar Ableton Live should start playing the clip with ID matching predicted class ID.
4) Repeat the gesture as many times as preferred, listening to the rhythm of the composition and observing the visual indicators of beats and bars, try to make gestures as temporally and spatially similar as possible.
5) When the live performance is over, click the button START/STOP again to stop the prediction.
6) Stop *Training Patch*, if no more dance will be performed.

Figure 4.17 visualises the data workflow of the described procedure, which result should be a music composition performed live.

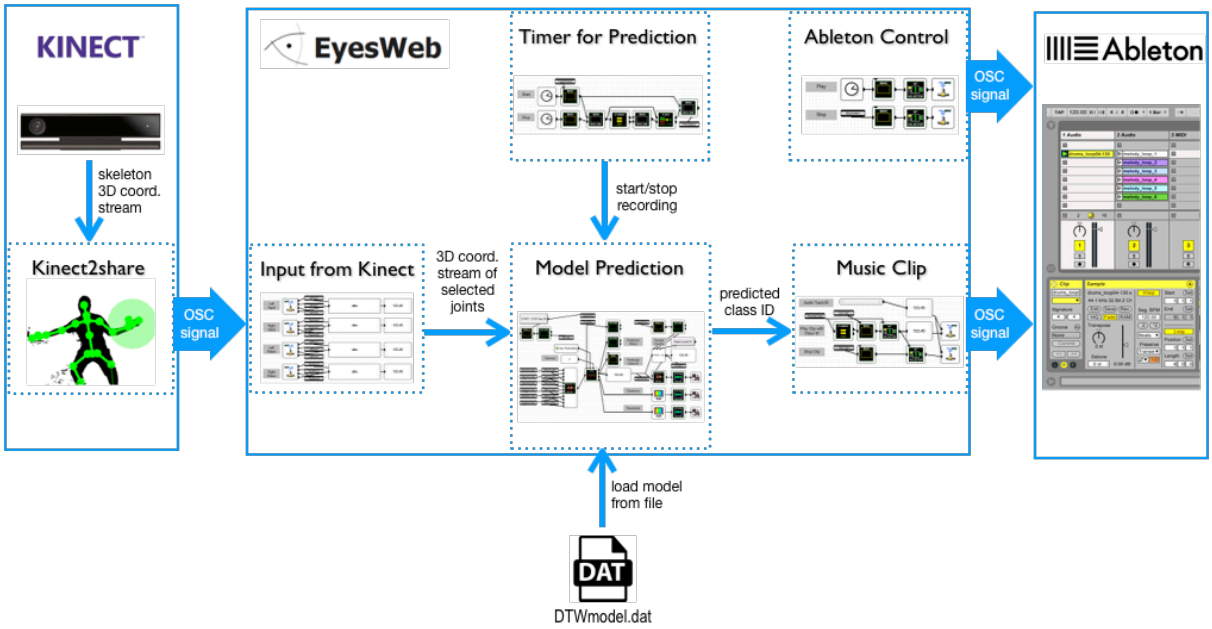

Figure 4.17. Workflow of the system during prediction of the new samples.

During live performance of the last procedure the gesture recognition system exploits adaptive feature in order to improve prediction, when the model is initially trained with a small number of training samples. Figure 4.18 visualises the data workflow of the adaptive feature, no additional actions are required from the user. If needed, click the button START/STOP in the module "Record Prediction Data" to manually turn off the adaptive feature.

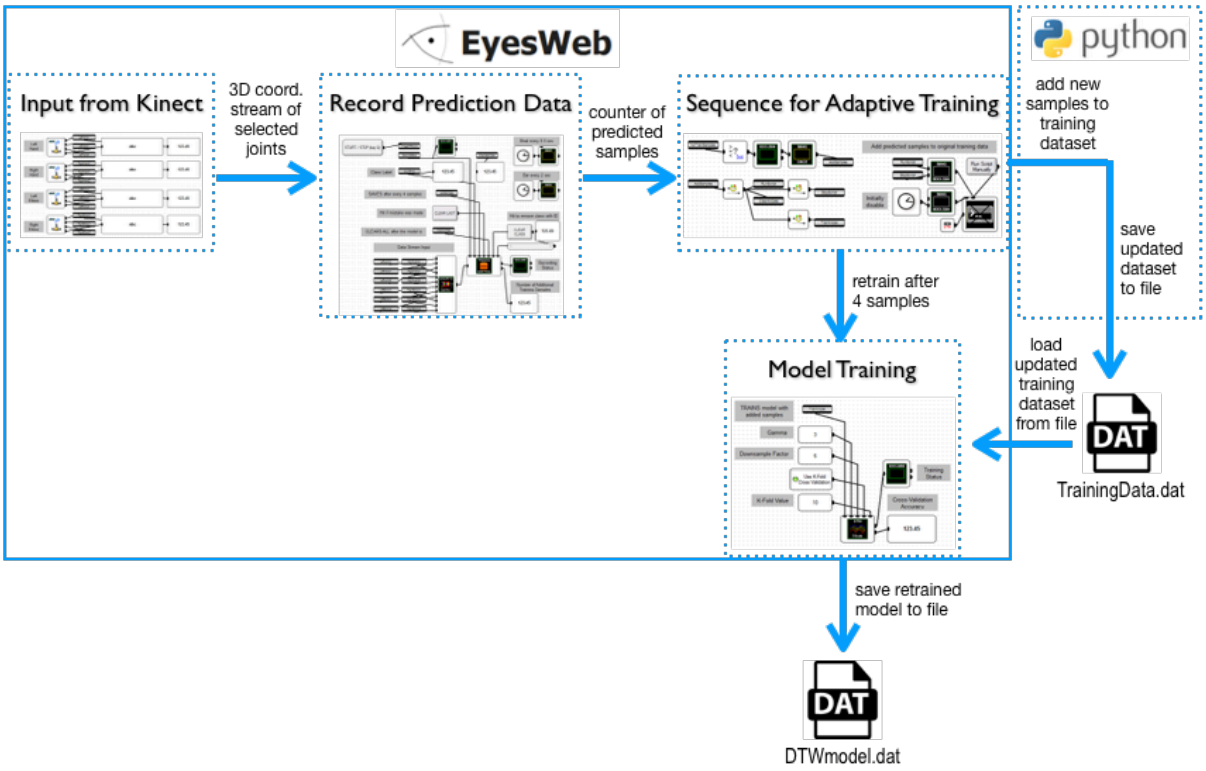

Figure 4.18. Workflow of the system during prediction with the adaptive feature.

4.6. Summary

This chapter has revealed the contents of created system's prototype, able to recognize gestures of one group of body parts and play mapped music clips. The system requires a setup of Kinect v2 device with installed Kinect SDK and Kinect2share utility to extract Body Skeleton data and send it to EyesWeb over OSC signal. In order to record training data and train DTW model, the Training Patch project was created in EyesWeb environment, which consists of six modules made of blocks and interconnecting pins. The second EyesWeb project, the Prediction Patch, has seven modules, responsible for classification of new gestures, adaptation of the model and sending OSC commands to Ableton Live. The modules use initially trained DTW model to predict a class for new gestures and periodically retrain the model adding

predicted samples to the training dataset. OSC signal contains predicted class ID, which serves as clip ID in a music composition. Finally, the music composition was set up in Ableton Live, which was receiving OSC signals from the EyesWeb project and arranging the playback of music clips accordingly.

Procedures for a user explained, how to use the system step-by-step. Workflow diagrams visualised the data flow between the main components and the modules within the framework of the system.

The following chapter presents some results of system's evaluation. Performance of system under various conditions is discussed in an attempt to reveal its strengths and weaknesses.

5. Evaluation

The prototype of the designed recognition system was successfully created and tested in a live performance. The system was able to record training data, train a DTW model and then use this model to recognize dancing gestures. Recognition triggered the playback of audio clips, based on the value of recognized class. The system was also able to retrain the model using recognized gestures so as to exhibit its adaptive property.

In order to evaluate the system's ability to recognize gestures at various conditions, a collection of gestures had to be made available offline (i.e., without the real-time stream of data from Kinect device). Gestures were recorded as the data stream to a proprietary EyesWeb data file, using module "Input to File" (Figure 5.1). All parameters of the modules for recording and reproduction of the data stream are described in the *Appendix E*.

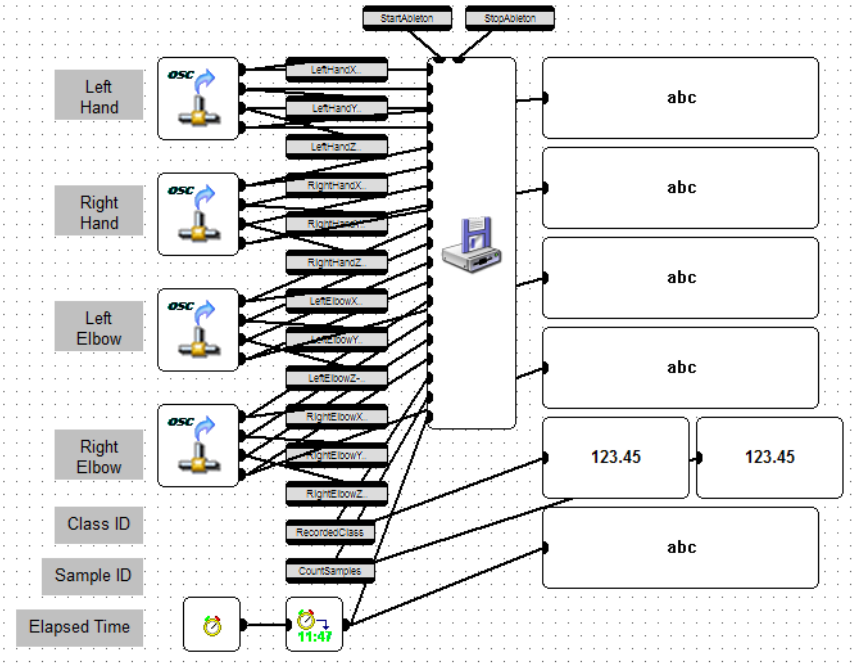

Figure 5.1. EyesWeb module "Input to File".

Once data stream was saved to a file, it could be reproduced from the file multiple times imitating identical real-time live performances using the module "Input from File" (Figure 5.2).

Module "Sequence for Input to & from File" was responsible for setting the class ID and sample ID for each gesture and feedback visualisation of recording session (Figure 5.3).

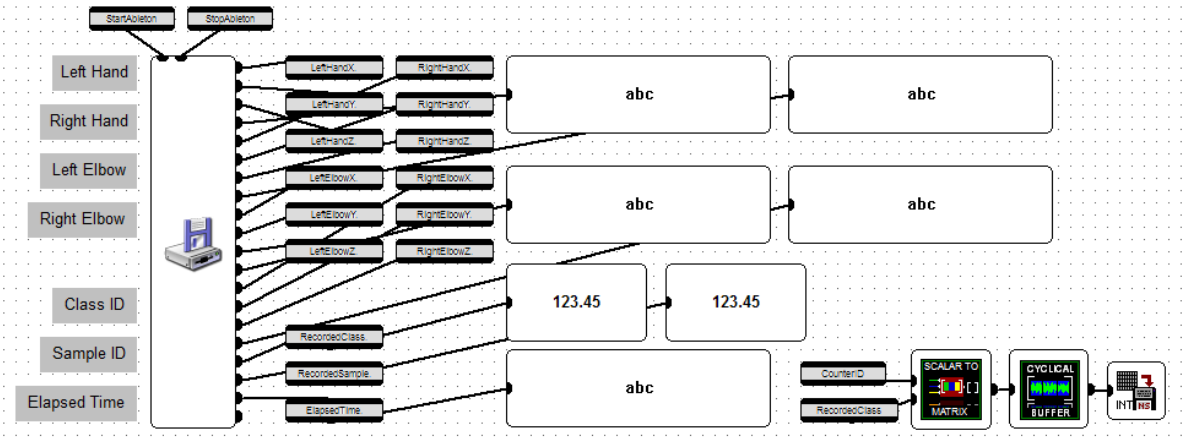

Figure 5.2. EyesWeb module "Input from File".

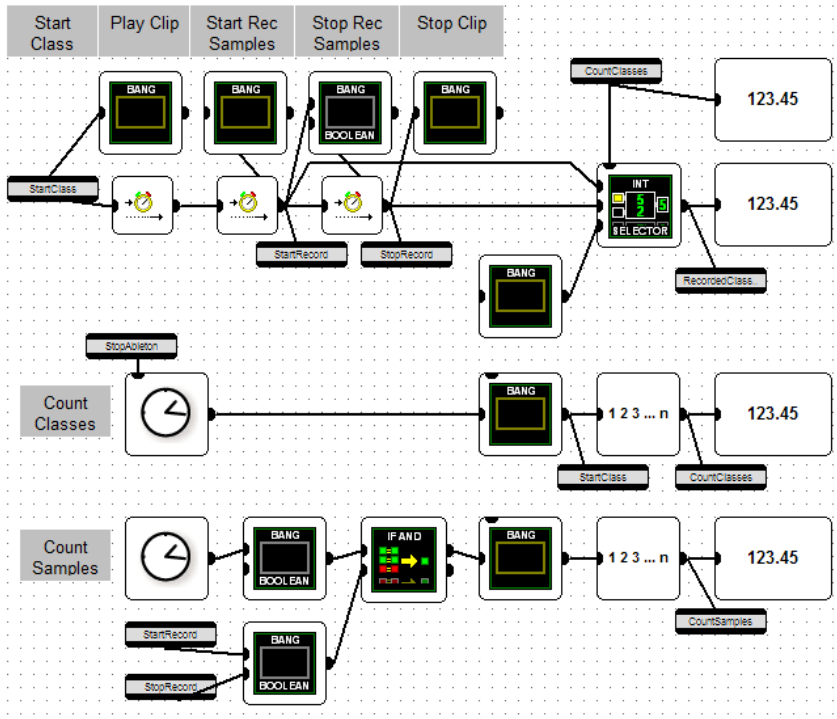

Figure 5.3. EyesWeb module "Sequence for Input to & from File".

5.1. Experimental Setup

As the system is made for intrapersonal use (a user trains the model with his/her own gestures and then uses it for recognition during his/her live performance), evaluation of the system is based on the data from one participant. The author of this thesis performed hand gestures sitting on a couch in front of Kinect device.

Distance between the couch and the device was 130 cm, the height of the couch was 50 cm, and the height of the platform for Kinect device was 70 cm. Projector's screen above the

device was displaying the interface of the system in EyesWeb environment and controlled by a wireless mouse from the couch.

5.2. Methods

A collection of generic hand gestures was recorded as data streams from Kinect device and saved to a file using a EyesWeb project. 12 data streams were recorded at ~30 fps: x, y and z coordinates of both hands and elbows, extracted from the signal of Kinect skeleton joints.

6 gestures, each lasting 2 seconds and matching the bar of the music composition, were repeated 32 times. With 8 seconds breaks before every new gesture, the whole session took 7 min 28 sec.

Three relatively similar gestures and three distinct gestures were selected (Figure 5.4):

1) drawing a *circle*, left hand in clockwise and right hand in counter clockwise, starting with the hands up;
2) drawing a *square*, left hand in clockwise and right hand in counter clockwise, starting with the hands up;
3) drawing a *triangle*, left hand in clockwise and right hand in counter clockwise, starting with the hands up;
4) drawing *two spirals*, starting with both hands on the left, making a spiral down and moving to the right, making a spiral down and returning to the left;
5) two *"chicken dance"* moves, holding arms horizontally and moving elbows up and down, starting with both elbows up;
6) *hands-up* and *three claps*, starting with the hands up.

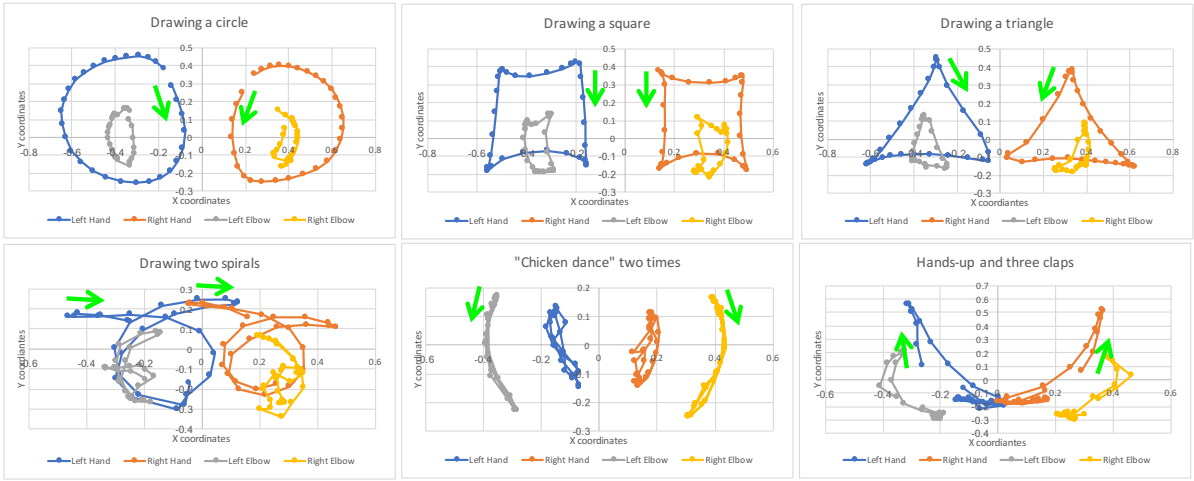

Figure 5.4. A sample of each gesture, plotted only horizontal and vertical coordinates. Green arrows indicate the beginning of a gesture.

5.2.4. Datasets

The session was recorded two times, first to use as a dataset for training the model, and second as a dataset for cross-validation. Each dataset had 6 classes in the same order with 32 sequential samples for each class. Breaks of 8 seconds between classes were ignored in the analysis.

For model's training and prediction, recorded data stream was reproduced from a file. Modules *"Record Training Data"* and *"Model Prediction"* segmented the data stream to samples of ~60 frames (30 fps x 2 sec) each with 12 values (3D coordinates x 4 joints), module *"Model Training"* down-sampled it by factor 6 to 10 frames (60 frames / 6).

5.2.5. Measurements

The system was tested and predictions analysed in attempt to answer these questions:

1) How does *the threshold* of classification (gamma coefficient in DTW model) influence the system's performance?
2) How does *the number of training samples* (used to train DTW model) influence the system's performance?
3) How does *the number of learned gestures* (known classes in DTW model) the influence system's performance?
4) How does *the adaptive feature* change the system's performance?

Measurements of Signal Detection Theory (Burgoon et al., 2005; Table 1) were used in both fitting and cross-validation analysis:

- *Hits* = Number of samples correctly classified as the gesture X
- *Misses* = Number of samples unrecognized as the gesture X
- *False Alarms (FA)* = Number of samples incorrectly classified as the gesture X
- *Correct Rejections (CR)* = Number of samples correctly unrecognized as the gesture X
- *Accuracy* = (Hits + CR) / Total number of samples

In order to reveal, what affects prediction's accuracy, additional measurements were calculated (Stiehl & Breazeal, 2005):

- *Positive Predictive Value (PPV)* = Hits / (Hits + FA)
- *Negative Predictive Value (NPV)* = CR / (CR + Misses)
- *Sensitivity* = Hits / (Hits + Misses)
- *Specificity* = CR / (CR + FA)

Table 1.
Contingency table of possible judgements based on Signal Detection Theory.

		PREDICTION	
		Signal	**Noise**
FACT	**Signal**	Hit	Miss
	Noise	False Alarm	Correct Rejection

Note. Adapted from *An Approach for Intent Identification by Building on Deception Detection,* by J. Burgoon et al., 2005. Copyright by IEEE.

Non-gestures (unrecognized samples which the model assigned to "zero class") were analysed using the same measurements, e.g. *Hits* is number of samples correctly classified as non-gestures, *Correct Rejections* is the number of samples correctly recognized as one of known gestures.

5.3. Results

This chapter spotlights notable discoveries of the system evaluation, visualising some results using the charts. All results of the evaluation can be found in Appendix F.

5.3.1. Threshold

In order to give the possibility for DTW algorithm to reject unfamiliar gestures (classify unrecognized gestures to "zero class"), the model has to be trained with predefined threshold (i.e., the gamma coefficient). If the threshold is switched off, any gesture will be assigned to one of known classes, even if all known classes are very different. The smaller is the coefficient, the less deviation from known classes is "allowed" by the model (Figure 5.5).

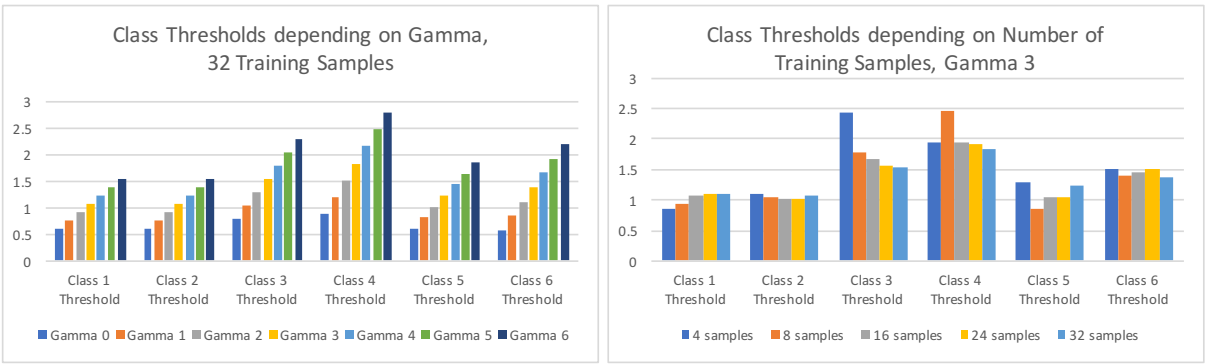

Figure 5.5. Threshold for each class in the trained model depending on the *gamma* coefficient (left) or the number of training samples (right).
Sources: Table 3 (left) and Table 4 (right) in the Appendix F.

To answer the question about how the gamma coefficient influences system's performance, models with *gamma from 0 to 6* were trained with *all 32 samples* for each of *6 classes* in the first dataset. Fitting predictions using the same dataset revealed that accuracy gets better when gamma is increased until it reaches 3, further increments of gamma do not improve the measurements. The same pattern was observed when model was trained to recognize only the first 3 classes (drawing a circle, a square and a triangle); the remaining 3 classes were not included in the training dataset, but were present in the prediction dataset (Figure 5.6).

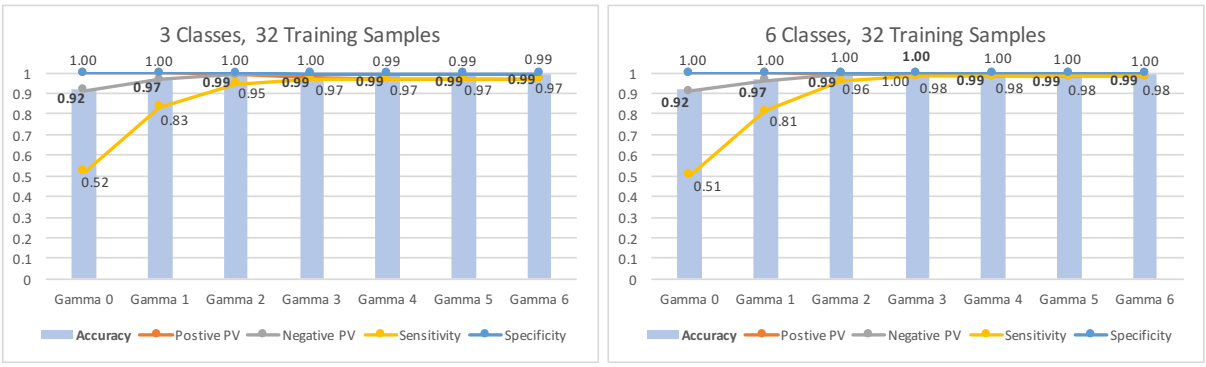

Figure 5.6. Fitting prediction depending on the *gamma* coefficient when training dataset included only the first 3 classes (left) or all 6 classes (right).
Note. Positive predictive values are nearly identical to Specificity (gamma 0-2) and Sensitivity (gamma 3-6). *Sources*: Table 5 (left) and Table 6 (right) in the Appendix F.

0-6 gamma coefficient were tested with various numbers of samples – *4, 8 ,16 and 24 samples* for each class. While gamma coefficient did not have much influence when all 6 classes where known training the model, it was not the case with less known classes. When 2-4 classes are trained and the rest of the samples are unknown to the model, it's fitting prediction peaks at gamma 2-3, then gets worse with larger gamma coefficients (Figure 5.7).

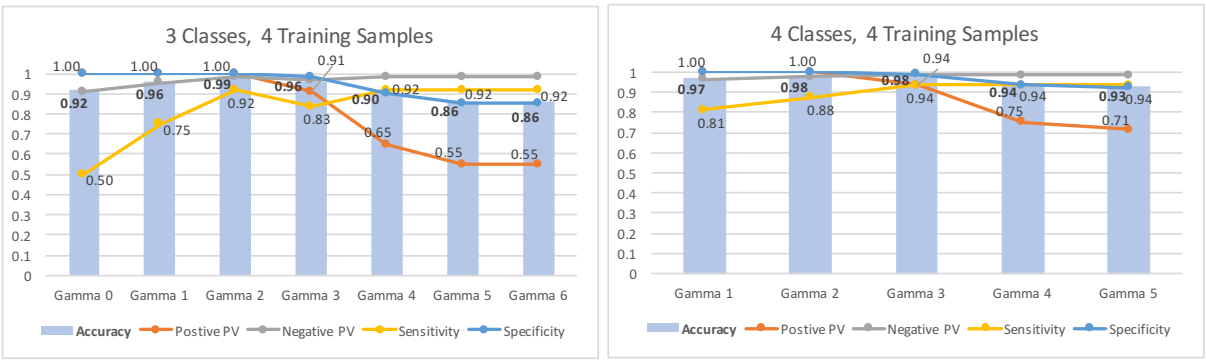

Figure 5.7. Fitting prediction depending on the *gamma* coefficient, when training dataset included only 4 samples of the first 3 classes (left) or 4 classes (right).
Sources: Table 9 (left) and Table 10 (right) in the Appendix F.

While trained with the smallest training dataset of 4 samples the model predicts best with gamma 2, it was discovered that in most other cases gamma 3 provides the highest accuracy. Gamma 3 was chosen to be used for the rest of analysis, in both model's training and prediction.

Cross-validation with the second dataset of samples, which were not used for model's training, revealed that prediction accuracy remains very high with chosen gamma 3 (Table 2).

Table 2.
Cross-validation prediction with gamma 3 depending on the number of training samples, 6 classes.

6 classes	**Hits**	**Misses**	**FA**	**CR**	**Total**	**Accur.**	**PPV**	**NPV**	**Sens.**	**Spec.**
4 samples	157	35	35	1117	1344	0.95	0.82	0.97	0.82	0.97
8 samples	149	43	43	1109	1344	0.94	0.78	0.96	0.78	0.96
16 samples	170	22	22	1130	1344	0.97	0.89	0.98	0.89	0.98
24 samples	183	9	9	1143	1344	0.99	0.95	0.99	0.95	0.99
32 samples	187	5	5	1147	1344	0.99	0.97	1.00	0.97	1.00

5.3.2. Number of Samples

When many time-series are available in the training dataset, it allows DTW model to learn gestures based on the most consistent samples, ignoring samples with higher variation, which improves model's prediction. Too many samples can also cause worse prediction, because it can include more outliers which the algorithm has to consider.

DTW models were trained with *4, 8, 16, 24 and 32 samples* for each of *6 classes*, comparing its fitting prediction. Based on previous analysis, *gamma 3* coefficient was chosen as a threshold to reject unknown gestures.

Fitting prediction measurements show, that DTW algorithm performs worst with the smallest training dataset - 4 samples for each class. When the dataset is increased, accuracy and especially sensitivity improves noticeably. With the largest available number of 32 samples, sensitivity drops again, probably due to the model's overfitting (Figure 5.8).

Cross-validation with the second dataset shows that accuracy is low when a small number of 4-8 samples was used to train the model to recognize *6 classes*. Peculiar case of the 8-training-samples model, which predicted worse than the 4-training-samples model in cross-validation, alerts that 8 samples are not enough for the model to ignore outliers deteriorating its performance. Cross-validation of models, which were trained to recognize only *3 classes*,

shows more consistent results with correct predictions increasing when more samples were used for training (Figure 5.9).

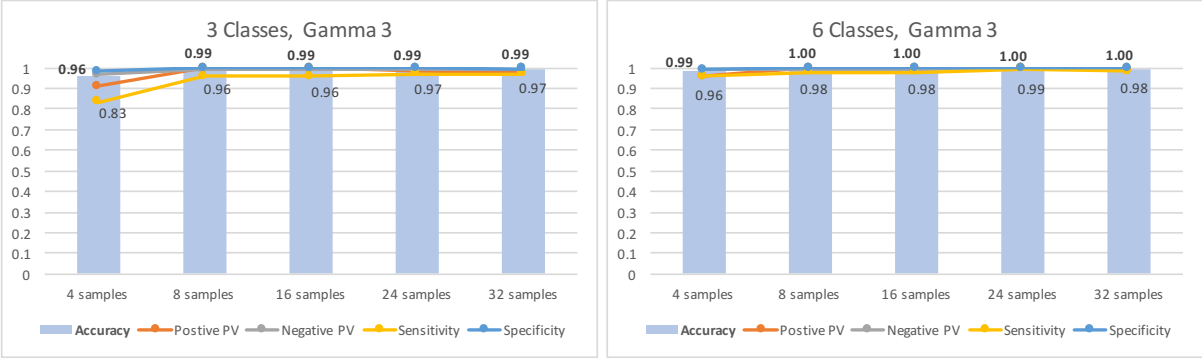

Figure 5.8. Fitting prediction with gamma 3 depending on the *number of training samples* when training dataset included only the first 3 classes (left) or all 6 classes (right).
Sources: Table 14 (left) and Table 15 (right) in the Appendix F.

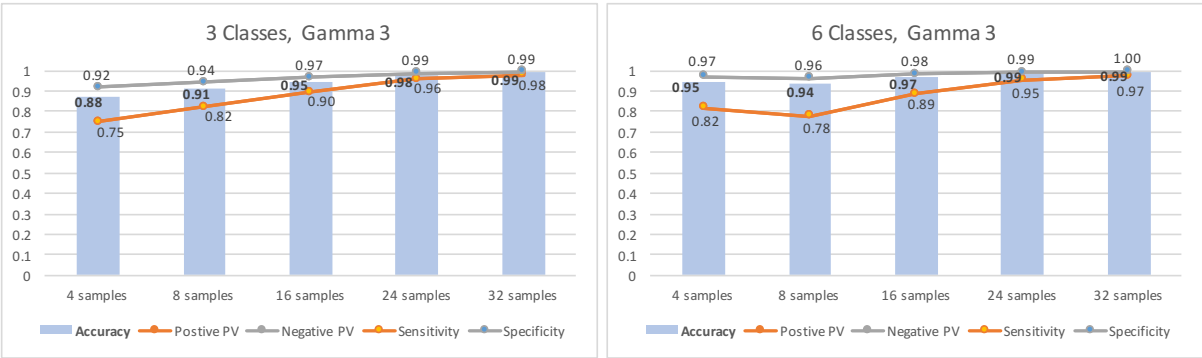

Figure 5.9. Cross-validation prediction with gamma 3 depending on the *number of training samples* when training dataset included only the first 3 classes (left) or all 6 classes (right).
Note. Measurements with the same line colour have the same values.
Sources: Table 16 (left) and Table 17 (right) in the Appendix F.

5.3.3. Number of Classes

If the model is trained to recognize all 6 gestures and only one of these gestures have to be classified during prediction, the system performs incredibly well, reaching 99% accuracy. However, if the system "knows" only some gestures and unknown gestures should be classified as "zero class", the system may try to assign a learned class to any new gesture.

To observe the influence of the number of learned classes, DTW models were trained with the datasets including *1, 2, 3, 4, 5* and all *6 classes*. Only *4 training samples* for each class were given to the system, since the previous analysis showed that this is when the model is most prone to errors. Based on previous analysis, *gamma 3* coefficient was chosen as a threshold to reject unknown gestures.

Measurements showed, that prediction improves with the number of known classes. Sensitivity is the most dependent on this variable, with its lowest when the model is train to recognize 2, 3 or 5 classes.

Cross-validation with the second dataset mostly confirms the assumption that the number of learned classes correlates with accuracy of predictions, with the exception when the model "knows" 3 classes (Figure 5.10).

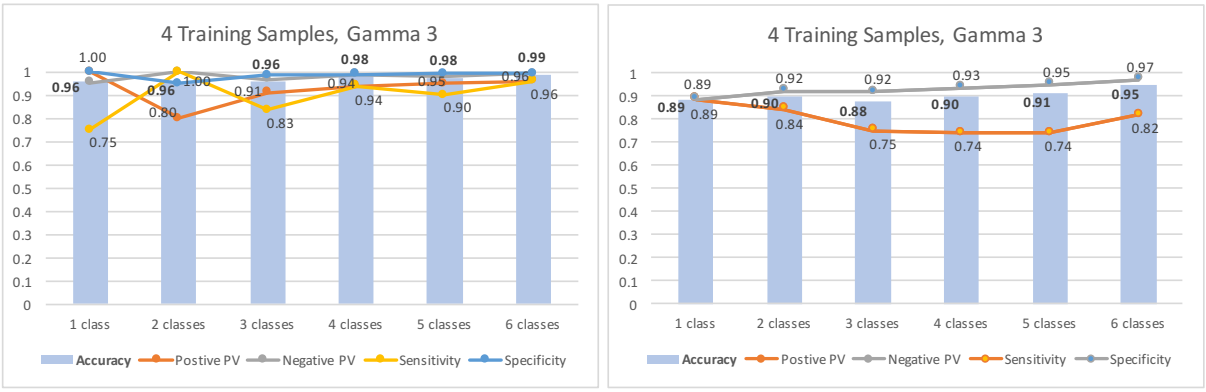

Figure 5.10. *Fitting* (left) and *Cross-validation* (right) prediction with gamma 3 depending on the *number of classes* in training dataset.
Note. Measurements with the same line colour have the same values.
Sources: Table 18 (left) and Table 19 (right) in the Appendix F.

5.3.4. Adaptive Feature

The system was able to adapt its model during prediction process, periodically including additional samples to the training dataset and retraining the model.

Adaptive feature was tested using the second dataset used for *cross-validation*, which had *32 samples* for each of *6 classes*, 192 samples in total. The model was set to retrain after *every 4th* classified sample, adding recognized samples to the training dataset, which means that the model retrained 48 times during the data stream of 192 samples. Only recognized samples (classified as class 1-6) were reused, ignoring unrecognized samples which got classified as "zero class".

For example, if the model was initially trained with *4 samples* for each of *6 classes*, totally 24 samples in the initial training dataset, the model was retrained 48 times during prediction process, each time adding 4 newly classified samples to the training dataset. After the model was retrained the last time, the training dataset grew from 24 to 181 samples in total. 35 samples were classified as "zero class" and were not included to the retraining dataset.

Adaptive feature is supposed to benefit in cases, when the model was initially trained with very small number of samples. However, system's adaptive feature may not necessarily improve model's performance, because it is not able to validate if additional samples were classified correctly.

To investigate, if adaptive feature improves system's performance, it was tested on the models initially trained with *4, 8, 16, 24* and *32 samples*.

According to *cross-validation* of the models, which were trained to recognize all *6 classes*, adaptive feature was beneficial only for the model that was initially trained with the smallest number of 4 samples. Adaptive models initially trained with 8-24 samples performed worse, 32-training-samples model's performance did not change. Overall adaptive feature did not have a large impact on the system's performance, altering its prediction accuracy from -0.1 to +0.3 percentage points (Figure 5.11).

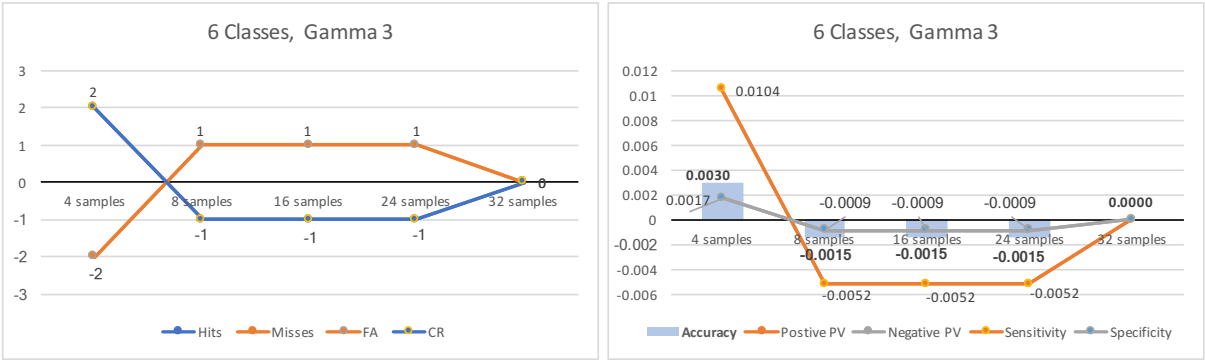

Figure 5.11. Difference of cross-validation *with* the adaptive feature compared to cross-validation *without* the adaptive feature depending on the number of training samples, when all 6 classes were included in the training dataset.
Note. Measurements with the same line colour have the same values.
Source: Table 21 in the Appendix F.

Adaptive feature slightly differently affected models, which were trained to recognize only the first *3 classes*, other gestures to be categorized as "zero classes". During cross-validation 4-training-samples model's performance did not change, 8-training-samples model performed better, other models predicted slightly worse than without adaptive feature. With 3 learned classes, adaptive feature had a larger impact on system's performance, altering its prediction accuracy from -0.5 to +1.6 percentage points (Figure 5.12).

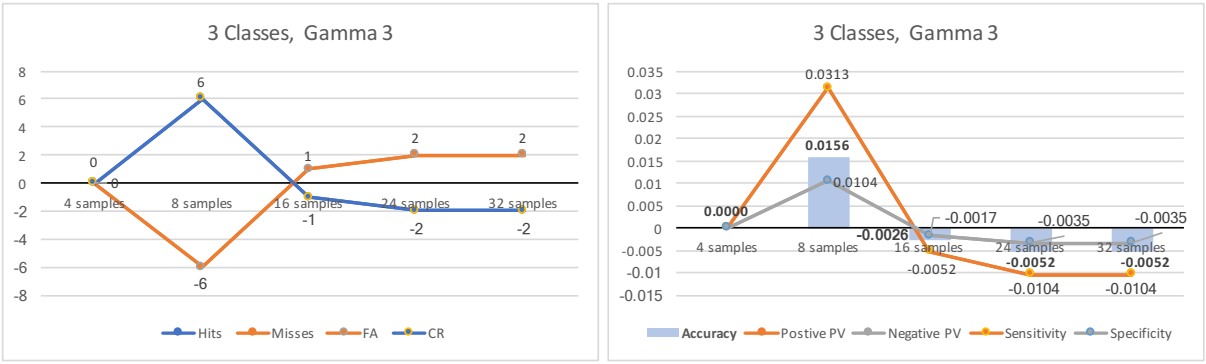

Figure 5.12. Difference of cross-validations *with* the adaptive feature compared to cross-validation *without* the adaptive feature depending on the number of training samples, when only 3 classes were included in the training dataset.
Note. Measurements with the same line colour have the same values.
Source: Table 23 in the Appendix F.

5.4. Summary

This chapter presented some results of the system's testing. Six gestures were performed by the author of this thesis, each repeated 32 times. Two performance sessions were recorded, first was used as a dataset for model's training, and the second for cross-validation. Tests attempted to measure, how the threshold (gamma coefficient in DTW model), the number of training samples, the number of learned gestures and the adaptive feature influence system's performance. Measurements based on Signal Detection Theory were calculated in both fitting and cross-validation analysis.

Results disclosed a very high prediction accuracy of the system: in most of the cases it was over 90%. Analysis showed that the system performs best when all gestures performed during prediction were included in the training dataset and when each gesture had at least 16 training samples.

Threshold has a considerable impact on the system's ability to correctly classify gestures, when the training dataset is very small (e.g. 4 samples for each gesture) and/or not all gestures performed during prediction were included in the training dataset.

The number of training samples did not affect system's performance much, if all gestures performed during predictions were included in the model's training (i.e. there are no non-gestures). The system's prediction accuracy decreased slightly when a very small dataset is used to train the model.

The number of learned gestures had a higher influence: if there were gestures during prediction which had not been trained during learning, the model incorrectly classified some of them to one of known classes. In this "vulnerable" situation, the performance highly depends

on which gestures were included to the training dataset and how similar are unknown gestures to the learned ones.

The adaptive feature did not meet expectations: only with the smallest initial training dataset (4 samples for each gesture) did it slightly increase the prediction accuracy. In all other conditions, it did not have any effect or even lowered the performance.

The last chapter discusses the results of system's evaluation, comments the decisions made for the design of the system's processing chain based on its practical implementation and some ideas for the system's future development.

6. Conclusions & Implications

The objective of this thesis was successfully achieved: a gesture recognition system, able to recognize dance movements and transform it to music, was designed and a working prototype was implemented and tested. Results showed that a machine learning algorithm could be used for a flexible gesture recognition. Evaluation results of the system's prediction abilities revealed its strengths and weaknesses. Even trained with as small dataset as 4 samples it can achieve over 90% accuracy. The system performed especially well, when all 6 gestures were learned during training and only these 6 gestures were executed for prediction. It had trouble classifying new gestures as non-gestures if these gestures were not learned in training. However, this should not be a big issue for choreographed dance performances where all gestures are carefully planned and scheduled.

Decision to choose the *Kinect v2* device for input did not disappoint because it delivered an impressively high-precision depth image thanks to its TOF technology. The device was well integrated with *Kinect SDK 2.0* which did a lot of pre-processing of the raw signal to remove noise, exclude the background, track the body, infer the position of occluded parts of the body and finally provide a stable stream of Body Skeleton data that was easy to handle in the workflow. The only drawback is an unclear future of the device's support due to Microsoft's discontinuation of its production. It is unclear whether Kinect SDK will be compatible with new Windows OS versions or other new software.

Segmentation of the data stream exploiting *music metrics* was an easy-to-implement solution but it had the drawback of making the system prone to errors: it was difficult for an unexperienced user to fit every gesture within the bar measure of 2 sec. When a gesture was executed too slowly or too quickly, the sample was segmented incorrectly (e.g. with a chunk of previous gesture in the beginning and/or a chunk of next gesture at the end) leading to the system confusing it with other classes. Nevertheless, this should not be an issue for a highly-skilled dancer who would be able to repeat gestures exactly as required. One solution to make a less penalizing system would be to split the data stream into several overlapping segments of various length and consider them all as possible gestures.

The *Dynamic Time Warping* algorithm was impressively robust and sparse: it was able to handle almost raw time-series (only down-sampling was applied to reduce the number of data points in time-series) and train a model in the blink of an eye. It was particularly useful in implementing the adaptive feature where the model had to be regularly retrained during the process of prediction. DTW algorithm's weakness showed up when the model was not trained

to recognize all gestures presented during prediction, as already discussed in the beginning of this chapter. The author believes that it is the Achilles heel of all algorithms under such conditions. To avoid this problem at a live performance, if there are gestures choreographed which the system should ignore, they should be included in the training dataset as separate classes, which would then be recognized during prediction but would not trigger any music clips.

The *EyesWeb* framework greatly facilitated prototyping of the system and the exploring of various classification algorithms and workflow solutions. Graphical user interface with drag-and-drop blocks and click-and-click pins made the development quick and easy while at the same time serving as a visualisation of the data flow which greatly supported the creative process. However, convenience of the blocks came at a price: because blocks were "black boxes" with restricted inputs, outputs and limited parameters, it was sometimes difficult to find the right blocks for a specific task and to integrate them to the system or understand why they function differently than expected. As the framework is open-source, it is possible to create, modify or extend the blocks, or to include custom-made Python scripts to serve specific purposes (as it was done to implement adaptive feature in the system), but this requires advanced programming skills.

Ableton Live served well an output of the system. It made a crucial job to play the right clips at a right time to create a harmonic music composition. The only issue was its lack of official support of OSC protocol: third-party utility LiveOSC had to be used, which required a specific version of Python, which was not supported by the most recent versions of Ableton Live, therefore an older version had to be used. For a more reliable and future-proof system, MIDI protocol should be used for communication between EyesWeb and Ableton Live.

The *Adaptive feature*, although successfully implemented and integrated to the system, did not meet expectations: it only made small improvements in prediction when the model was initially trained with very small number of samples and had no effect even slightly decreased the accuracy when trained with a larger initial training dataset. The feature negatively affected the model's performance because both correctly and incorrectly classified samples were added to the training dataset. A simple fix would be to have a separate, higher threshold for the new samples: it would reject the samples which were on the borderline and add only those samples that were classified with a high confidence. This solution would probably prevent the model from outliers but also slow down the improvement of the model during prediction session. More sophisticated options should be explored.

The implementation process provided some ideas for future development about how the dance recognition system could be expanded to deliver more possibilities in music creation. For example, a classifier could emit not only a class ID of an unknown gesture, but also a "confidence coefficient" (probability of the class vs other classes or vs threshold), which could be used as audio-visual feedback together with the musical output, e.g. the higher is classification confidence, the louder is a certain audio and/or the brighter is a visualization. Audio-visual feedback could indicate during the learning stage, how similar are observed repeating gestures, which would be fed to the model as training examples. The system could send some "context" data along with classification predictions, which would be used to enhance overall music creation and control. For example, continuous streamed performer's absolute position in the stage could control the main volume, overall speed of performer's movements could change track's tempo.

For the author of this thesis, music creation using gestures was an exciting and empowering experience. Even the long and tiring process of recording of the training data felt as a form of meditation, because repetition of gestures in synchronization with music rhythm required a high level of embodied cognition: a sustained awareness of his own body, a constant focus on audio-visual feedback. This experience implied that further advancements in machine learning and human-computer interfaces will not only enhance two-way interaction of dance and music, but also build closer relationship of body and mind.

Bibliography

Abdelnasser, H., Youssef, M., & Harras, K. A. (2015). WiGest: A ubiquitous WiFi-based gesture recognition system. In *2015 IEEE Conference on Computer Communications (INFOCOM)* (Vol. 26, pp. 1472–1480). IEEE. https://doi.org/10.1109/INFOCOM.2015.7218525

Ableton. (2018). Ableton Live 10 – Ableton. Retrieved from https://www.ableton.com/en/live/

Aigner, F. (2011). Kamera statt Maus und Joystick - Technische Universität Wien. Retrieved from https://www.tuwien.ac.at/aktuelles/news_detail/article/7110/

Akl, A., Feng, C., & Valaee, S. (2011). A novel accelerometer-based gesture recognition system. *IEEE Transactions on Signal Processing*, *59*(12), 6197–6205. https://doi.org/10.1109/TSP.2011.2165707

Akten, M. (2015). Pose and gesture recognition using Kinect 2 skeleton tracking and various machine learning techniques in Max MSP Jitter - Vimeo. Retrieved from https://vimeo.com/122166652

Al-Ali, S., Milanova, M., Al-Rizzo, H., & Fox, V. L. (2015). Human action recognition: contour-based and silhouette-based approaches. In *Intelligent Systems Reference Library* (Vol. 75, pp. 11–47). Springer, Cham. https://doi.org/10.1007/978-3-319-11430-9_2

Aylward, R., & Paradiso, J. A. (2006). Sensemble: A wireless, compact, multi-user sensor system for interactive dance. In *the Conference on New Interfaces for Musical Expression* (pp. 134–139). Retrieved from https://dl.acm.org/citation.cfm?id=1142248

Bahn, C., Hahn, T., & Trueman, D. (2001). Physicality and feedback: a focus on the body in the performance of electronic music. In *the International Computer Music Conference*, (vol. 2, pp. 44–51). Retrieved from http://www.cogsci.rpi.edu/public_html/bahnc2/practicum/readings/physicality_feedback.pdf

Bettens, F., & Todoroff, T. (2009). Real-time DTW-based gesture recognition external object for Max/MSP and PureData. In *The 6th Sound and Music Computing Conference* (pp. 23–25). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.654.2131&rep=rep1&type=pdf#page=41

Bevilacqua, F., Schnell, N., & Alaoui, S. F. (2011). Gesture capture: paradigms in interactive music / dance Systems. In *Emerging bodies: the performance of worldmaking in dance and choreography* (pp. 183–193). Retrieved from http://architexte.ircam.fr/textes/Bevilacqua11a/index.pdf

Beyl, T., Nicolai, P., Raczkowsky, J., Wörn, H., Comparetti, M. D., & De Momi, E. (2013). Multi Kinect people detection for intuitive and safe human robot cooperation in the

operating room. In *The 16th International Conference on Advanced Robotics, ICAR 2013* (pp. 1-6). https://doi.org/10.1109/ICAR.2013.6766594

Bhagavatula, C., Ur, B., Iacovino, K., Kywe, S. M., Cranor, L. F., & Savvides, M. (2015). Biometric authentication on iPhone and Android: usability, perceptions, and influences on adoption. In *Workshop on Usable Security*. https://doi.org/10.14722/usec.2015.23003

Bishop, C. M. (1995). Neural networks for pattern recognition. *Journal of the American Statistical Association*, *92*, 482–482. https://doi.org/10.2307/2965437

Biswas, K. K., & Basu, S. K. (2011). Gesture recognition using Microsoft Kinect. In *the 5th International Conference on Automation, Robotics and Applications* (pp. 100–103). https://doi.org/10.1109/ICARA.2011.6144864

Bitwig. (2018). Bitwig Studio – Bitwig. Retrieved from https://www.bitwig.com/en/bitwig-studio.html

Boukir, S., & Chenevière, F. (2004). Compression and recognition of dance gestures using a deformable model. *Pattern Analysis and Applications*, *7*(3), 308–316. https://doi.org/10.1007/s10044-004-0228-z

Bullock, J., & Momeni, A. (2015). ML.Lib : Robust, cross-platform, open-source machine learning for Max and Pure Data. In *the International Conference on New Interfaces for Musical Expression*, (pp. 265–270). Retrieved from http://nime2015.lsu.edu/proceedings/201/0201-paper.pdf

Burger, B., Thompson, M. R., Luck, G., Saarikallio, S., & Toiviainen, P. (2013). Influences of rhythm- and timbre-related musical features on characteristics of music-induced movement. *Frontiers in Psychology*, *4*, 183–183. https://doi.org/10.3389/fpsyg.2013.00183

Burgoon, J., Adkins, M., Kruse, J., Jensen, M. L., Meservy, T., Twitchell, D. P., … Younger, R. E. (2005). An approach for intent identification by building on deception detection. In *the 38th Annual Hawaii International Conference on System Sciences*. IEEE. https://doi.org/10.1109/HICSS.2005.78

Camurri, A., Lagerlöf, I., & Volpe, G. (2003). Recognizing emotion from dance movement: comparison of spectator recognition and automated techniques. *International Journal of Human Computer Studies*, *59*(1–2), 213–225. https://doi.org/10.1016/S1071-5819(03)00050-8

Camurri, A., Mazzarino, B., Ricchetti, M., Timmers, R., & Volpe, G. (2004). Multimodal analysis of expressive gesture in music and dance performances. *Springer*, 20–39. https://doi.org/10.1007/978-3-540-24598-8_3

Castellano, G., Bresin, R., Camurri, A., & Volpe, G. (2007). Expressive control of music and visual media by full-body movement. *New Interfaces for Musical Expression*, 390–390. https://doi.org/10.1145/1279740.1279829

Chaaraoui, A. A., Padilla-López, J. R., & Flórez-Revuelta, F. (2013). Fusion of skeletal and silhouette-based features for human action recognition with RGB-D devices. In *the*

IEEE International Conference on Computer Vision (pp. 91–97). https://doi.org/10.1109/ICCVW.2013.19

Chiu, W.-C., Blanke, U., & Fritz, M. (2011). Improving the Kinect by cross-modal stereo. In *the British Machine Vision Conference*, (pp. 116.1-116.10). https://doi.org/10.5244/C.25.116

Cycling ’74. (2018). Max software tools for media – Cycling ’74. Retrieved from https://cycling74.com/products/max/

Desmet, F., Nijs, L., Demey, M., Lesaffre, M., Martens, J. P., & Leman, M. (2012). Assessing a clarinet player’s performer gestures in relation to locally intended musical targets. *Journal of New Music Research*, *41*(1), 31–48. https://doi.org/10.1080/09298215.2011.649769

Dey, N. S., Mohanty, R., & Chugh, K. L. (2012). Speech and speaker recognition system using Artificial Neural Networks and Hidden Markov Model. In *the International Conference on Communication Systems and Network Technologies* (pp. 311–315). https://doi.org/10.1109/CSNT.2012.221

Eisenstein, J., Ghandeharizadeh, S., Golubchik, L., Shahabi, C., Yan, D., & Zimmermann, R. (2003). Device independence and extensibility in gesture recognition. In *the IEEE Virtual Reality* (pp. 207–214). https://doi.org/10.1109/VR.2003.1191141

Fahn, C.-S., & Chu, K.-Y. (2011). Hidden-Markov-Model-based hand gesture recognition techniques used for a human-robot interaction system. *Human-Computer Interaction. Interaction Techniques and Environments*, 248–258. https://doi.org/10.1007/978-3-642-21605-3_28

Fernández-Baena, A., Susín, A., & Lligadas, X. (2012). Biomechanical validation of upper-body and lower-body joint movements of Kinect motion capture data for rehabilitation treatments. In *the 4th International Conference on Intelligent Networking and Collaborative Systems, INCoS* (pp. 656–661). https://doi.org/10.1109/iNCoS.2012.66

Fisher, S. (2018). Complete control of Ableton Live using OSC – LiveOSC. Retrieved from https://livecontrol.q3f.org/ableton-liveapi/liveosc/

Gavrila, D. (1999). The visual analysis of human movement: a survey. *Computer Vision and Image Understanding*, *73*(1), 82–98. https://doi.org/10.1006/cviu.1998.0716

Giblock, P., & Junghans, T. (2018). Free, open source, multiplatform digital audio workstation – LMMS. Retrieved from https://lmms.io/

Gillian, N. E. (2011). *Gesture recognition for musician computer interaction* (PhD thesis). School of Music and Sonic Arts, Queen’s University Belfast. Retrieved from http://www.nickgillian.com/papers/NicholasGillianThesisElectronic.pdf

Gillian, N.E., Knapp, R. B., & Modhrain, S. O. (2009). The SARC EyesWeb Catalog: a pattern recognition toolbox for musician-computer interaction. In *the International Conference on New Interfaces for Musical Expression*, (pp. 60–61). Retrieved from

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.516.9846&rep=rep1&type=pdf

Gurwin, G. (2017). Once the future of gaming, Microsoft's Kinect has been discontinued. Retrieved from https://www.digitaltrends.com/gaming/kinect-for-xbox-one-discontinued/

Han, J., Shao, L., Xu, D., & Shotton, J. (2013). Enhanced computer vision with Microsoft Kinect sensor: a review. *IEEE Transactions on Cybernetics*, *43*(5), 1318–1334. https://doi.org/10.1109/TCYB.2013.2265378

Hauert, S. (2017). Eight ways intelligent machines are already in your life - BBC News. Retrieved from http://www.bbc.com/news/uk-39657382

Hernandez-Rebollar, J. L., Kyriakopoulos, N., & Lindeman, R. W. (2004). A New Instrumented Approach For Translating American Sign Language Into Sound And Text. In *the 6th IEEE International Conference on Automatic Face and Gesture Recognition* (pp. 547-552). https://doi.org/https://doi.org/10.1109/AFGR.2004.1301590

Hollander, R. (2017). Voice assistant usage remains low - Business Insider. Retrieved from http://www.businessinsider.de/voice-assistant-usage-remains-low-2017-12?r=US&IR=T

Hoy, M. B. (2018). Alexa, Siri, Cortana, and more: an introduction to voice assistants. *Medical Reference Services Quarterly*, *37*(1), 81–88. https://doi.org/10.1080/02763869.2018.1404391

Infomus. (2018). EyesWeb - InfoMus. Retrieved from http://www.infomus.org/eyesweb_eng.php

Jensenius, A. R. (2006). Using motiongrams in the study of musical gestures. In *the International Computer Music Conference* (pp. 499–502). https://doi.org/10.13140/2.1.1895.7124

Jensenius, A. R., Godøy, R. I., & Wanderley, M. M. (2005). Developing tools for studying musical gestures within the Max/MSP/Jitter environment. In *the International Computer Music Conference*, (Vol. 3, pp. 3–6). Retrieved from https://www.duo.uio.no/bitstream/handle/10852/26907/1/Jensenius_2005.pdf

Jeong, Y. J., Hong, S. C., Myeong, S. L., Park, M. C., Kim, Y. K., & Suh, C. M. (2005). Dance movement therapy improves emotional responses and modulates neurohormones in adolescents with mild depression. *International Journal of Neuroscience*, *115*(12), 1711–1720. https://doi.org/10.1080/00207450590958574

Junker, H., Amft, O., Lukowicz, P., & Tröster, G. (2008). Gesture spotting with body-worn inertial sensors to detect user activities. *Pattern Recognition*, *41*(6), 2010–2024. https://doi.org/10.1016/j.patcog.2007.11.016

Kahol, K., Tripathi, P., & Panchanathan, S. (2004). Automated gesture segmentation from dance sequences. In *–the 6th IEEE International Conference on Automatic Face and Gesture Recognition* (pp. 883–888). https://doi.org/10.1109/AFGR.2004.1301645

Kamperman, K. (2018). OSC Data Monitor. Retrieved from https://www.kasperkamperman.com/blog/osc-datamonitor/

Kiran, M., Chan, C. S., Lai, W. K., Ali, K. K. H., & Khalifa, O. (1996). A comparison of posture recognition using supervised and unsupervised learning algorithms. In *the 24th European Conference on Modelling and Simulation* (Vol. 2). Retrieved from http://www.scs-europe.net/conf/ecms2010/2010%20accepted%20papers/is_ECMS2010_0029.pdf

Kotha, S. K., Pinjala, J., Kasoju, K., & Pothineni, M. (2015). Gesture recognition system. *International Journal of Research in Engineering and Technology*, *04*(05), 99–104. Retrieved from http://esatjournals.net/ijret/2015v04/i05/IJRET20150405019.pdf

Kouchi, M., & Taguchi, H. (1991). Gesture recognition using Recurrent Neural Networks. In *the SIGCHI Conference on Human Factors in Computing Systems* (pp. 237–242). https://doi.org/10.1145/108844.108900

Kratz, S., & Rohs, M. (2010). A $3 gesture recognizer. In *the 15th International Conference on Intelligent User Interfaces, IUI '10* (pp. 341–344). https://doi.org/10.1145/1719970.1720026

Kurakin, A., Zhang, Z., & Liu, Z. (2012). A real-time system for dynamic hand gesture recognition with a depth sensor. In *the 20th European Signal Processing Conference* (pp. 1975–1979). Retrieved from https://pdfs.semanticscholar.org/21f8/bc079d10884dd7add65bd3350e45b5fa31b1.pdf

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lee, H.-K. & Kim, J. H. (1999). An HMM-based threshold model approach for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *21*(10), 961–973. https://doi.org/10.1109/34.799904

Lee, S. H., Sohn, M. K., Kim, D. J., Kim, B., & Kim, H. (2013). Smart TV interaction system using face and hand gesture recognition. In *Digest of Technical Papers - IEEE International Conference on Consumer Electronics* (pp. 173–174). IEEE. https://doi.org/10.1109/ICCE.2013.6486845

Leman, M. (2012). Musical gestures and embodied cognition. In *Actes Des Journées d'Informatique Musicale, JIM 2012* (pp. 5–7). Retrieved from https://biblio.ugent.be/publication/2999983/file/2999985.pdf

Li, C., & Prabhakaran, B. (2005). A similarity measure for motion stream segmentation and recognition. In *the 6th International Workshop on Multimedia Data Mining, MDM '05* (pp. 89–94). https://doi.org/10.1145/1133890.1133901

Li, C., Zhai, P., Zheng, S.-Q., & Prabhakaran, B. (2004). Segmentation and recognition of multi-attribute motion sequences. In *the 12th Annual ACM International Conference on Multimedia - MULTIMEDIA '04* (pp. 836-843). https://doi.org/10.1145/1027527.1027721

Licsár, A., & Szirányi, T. (2005). User-adaptive hand gesture recognition system with interactive training. *Image and Vision Computing*, *23*(12), 1102–1114. https://doi.org/10.1016/j.imavis.2005.07.016

Liu, J., & Kavakli, M. (2010). Hand gesture recognition based on segmented singular value decomposition. In *Knowledge-Based and Intelligent Information and Engineering Systems. KES 2010. Lecture Notes in Computer Science* (Vol. 6277, pp. 214–223). https://doi.org/10.1007/978-3-642-15390-7_22

Looperman. (2018). Free, loops, samples, acapellas, vocals – royalty-free music. Retrieved from https://www.looperman.com/

Lopes, P. (2010). Dynamic Time Warping in PureData (alpha) - Vimeo. Retrieved from https://vimeo.com/11792446

Lun, R., & Zhao, W. (2015). A survey of applications and human motion recognition with Microsoft Kinect. *International Journal of Pattern Recognition and Artificial Intelligence*, *29*(05), 1555008–1555008. https://doi.org/10.1142/S0218001415550083

Maiberg, E. (2016). The best Kinect game is the last Kinect game - Vice. Retrieved from https://motherboard.vice.com/en_us/article/3davxk/the-best-kinect-game-is-the-last-kinect-game

Malassiotis, S., Aifanti, N., & Strintzis, M. G. (2002). A gesture recognition system using 3D data. In *the 1st International Symposium On 3D Data Processing Visualization and Transmission* (pp. 190–193). https://doi.org/10.1109/TDPVT.2002.1024061

Martínez, A. M., Wilbur, R. B., Shay, R., & Kak, A. C. (2002). Purdue RVL-SLLL ASL database for automatic recognition of American sign language. In *the 4th IEEE International Conference on Multimodal Interfaces, ICMI 2002* (pp. 167–172). IEEE Comput. Soc. https://doi.org/10.1109/ICMI.2002.1166987

MathWorks. (2018a). Computer Vision System Toolbox – MATLAB & Simulink. Retrieved from https://www.mathworks.com/products/computer-vision.html

MathWorks. (2018b). MATLAB - MathWorks Products. Retrieved from https://www.mathworks.com/products/matlab.html

Microsoft Corporation. (2014). Human interface guidelines for Kinect v2.0. Retrieved from https://social.msdn.microsoft.com/Forums/en-US/b7a2d86c-ea2f-4ffc-89e5-01652d257788/human-interface-guidelines-for-kinect-20?forum=kinectv2sdk

Microsoft Corporation. (2018). Set up Kinect for Windows v2 or an Xbox Kinect sensor with Kinect adapter for Windows. Retrieved from https://support.xbox.com/en-US/xbox-on-windows/accessories/kinect-for-windows-v2-setup

Miranda, L., Vieira, T., Martinez, D., Lewiner, T., Vieira, A. W., & Campos, M. F. M. (2012). Real-time gesture recognition from depth data through key poses learning and decision forests. In *the 25th SIBGRAPI Conference on Graphics, Patterns and Images* (pp. 268–275). IEEE. https://doi.org/10.1109/SIBGRAPI.2012.44

Mitra, S., & Acharya, T. (2007). Gesture recognition: A survey. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, *37*(3), 311–324. https://doi.org/10.1109/TSMCC.2007.893280

Moore, G. A. (2014). *Crossing the chasm: marketing and selling high-tech products to mainstream customers*. *New York HarperBusiness*. Retrieved from http://cds.cern.ch/record/2032197

Naveda, L., & Leman, M. (2008). Representation of samba dance gestures, using a multi-modal analysis approach. In *the 5th International Conference on Enactive Interfaces* (pp. 68–74). https://doi.org/1854/LU-503783

Naveda, L., & Leman, M. (2009). A cross-modal heuristic for periodic pattern analysis of samba music and dance. *Journal of New Music Research*, *38*(3), 255–283. https://doi.org/10.1080/09298210903105432

Naveda, L., & Leman, M. (2010). The spatiotemporal representation of dance and music gestures using topological gesture analysis (TGA). *Music Perception*, *28*(1), 93–111. https://doi.org/10.1525/mp.2010.28.1.93

Noonan, P. J., Howard, J., Hallett, W. A., & Gunn, R. N. (2015). Repurposing the Microsoft Kinect for Windows v2 for external head motion tracking for brain PET. *Physics in Medicine and Biology*, *60*(22), 8753–8766. https://doi.org/10.1088/0031-9155/60/22/8753

Patsadu, O., Nukoolkit, C., & Watanapa, B. (2012). Human gesture recognition using Kinect camera. In *the 9th International Joint Conference on Computer Science and Software Engineering*, *JCSSE 2012* (pp. 28–32). https://doi.org/10.1109/JCSSE.2012.6261920

Peddinti, V., Povey, D., & Khudanpur, S. (2015). A time delay neural network architecture for efficient modeling of long temporal contexts. In *the 16th Annual Conference of the International Speech Communication Association* (pp. 3214–3218). Retrieved from https://www.isca-speech.org/archive/interspeech_2015/papers/i15_3214.pdf

Phurrough, D. (2018). Jit.OpenNI - Max Objects Database. Retrieved from http://www.maxobjects.com/?v=objects&id_objet=4732&requested=Jit.OpenNI&operateur=AND&id_plateforme=0&id_format=0

Pu, Q., Gupta, S., Gollakota, S., & Patel, S. (2013). Whole-home gesture recognition using wireless signals. In *the 19th Annual International Conference on Mobile Computing & Networking - MobiCom '13* (pp. 27–27). https://doi.org/10.1145/2500423.2500436

Puckette, M. (2018). PD community site - Pure Data. Retrieved from https://puredata.info/

Pun, J. C.-H. (2006). *Gesture recognition with application in music arrangement gesture recognition with application in music arrangement* (Master's thesis). University of Pretoria. Retrieved from https://repository.up.ac.za/handle/2263/29240

Randewich, N. (2017). Tesla becomes most valuable U.S. car maker, edges out GM. Retrieved from https://www.reuters.com/article/us-usa-stocks-tesla/tesla-becomes-most-valuable-u-s-car-maker-edges-out-gm-idUSKBN17C1XF

Raptis, M., Kirovski, D., & Hoppe, H. (2011). Real-time classification of dance gestures from skeleton animation. In *the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '11* (pp. 147–157). https://doi.org/10.1145/2019406.2019426

Rautaray, S. S. (2012). Real-time hand gesture recognition system for dynamic applications. *International Journal of UbiComp*, *3*(1), 21–31. https://doi.org/10.5121/iju.2012.3103

Ren, Z., Yuan, J., & Zhang, Z. (2011). Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera. In *the 19th ACM international conference on Multimedia* (pp. 1093–1096). ACM Press. https://doi.org/10.1145/2072298.2071946

Reynolds, M. (2017). DeepMind's AI beats world's best Go player in latest face-off - New Scientist. Retrieved from https://www.newscientist.com/article/2132086-deepminds-ai-beats-worlds-best-go-player-in-latest-face-off/

Robinson, M. (2018). nnLists - Max objects database. Retrieved from http://www.maxobjects.com/?v=objects&id_objet=4302

Schlömer, T., Poppinga, B., Henze, N., & Boll, S. (2008). Gesture recognition with a Wii controller. In *the 2nd International Conference on Tangible and Embedded Interaction TEI 08*, (pp. 11–14). https://doi.org/10.1145/1347390.1347395

Schmidhuber, J. (2015). Deep learning in neural networks: an overview. *Neural Networks*, *61*, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

Schönauer, C., & Kaufmann, H. (2013). Wide area motion tracking using consumer hardware. *The International Journal of Virtual Reality*, *12*(1), 1–9. Retrieved from http://publik.tuwien.ac.at/files/PubDat_219247.pdf

Serrano, L. (2016). A friendly introduction to deep learning and neural networks - YouTube. Retrieved from https://youtu.be/BR9h47Jtqyw

Song, Y., Demirdjian, D., & Davis, R. (2012). Continuous body and hand gesture recognition for natural human-computer interaction. *ACM Transactions on Interactive Intelligent Systems*, *2*(1), 1–28. https://doi.org/10.1145/2133366.2133371

Springmann, A. (2010). "Minority Report" meets Kinect thanks to MIT – PCWorld. Retrieved from https://www.pcworld.com/article/213126/minority_report_meets_kinect_thanks_to_mit.html

Stiehl, W. D., & Breazeal, C. (2005). Affective touch for robotic companions. In *Affective Computing and Intelligent Interaction* (pp. 747–754). Springer, Berlin, Heidelberg. https://doi.org/10.1007/11573548_96

Theis, T. N., & Wong, H.-S. P. (2017). The end of Moore's law: a new beginning for information technology. *Computing in Science & Engineering*, *19*(2), 41–50. https://doi.org/10.1109/MCSE.2017.29

Thier, D. (2018). Microsoft's Xbox Kinect is now really, truly dead. Retrieved from https://www.forbes.com/sites/davidthier/2018/01/04/microsofts-xbox-kinect-is-now-really-truly-dead/#78f987261195

Tran, C., & Trivedi, M. M. (2012). 3-D posture and gesture recognition for interactivity in smart spaces. *IEEE Transactions on Industrial Informatics*, *8*(1), 178–187. https://doi.org/10.1109/TII.2011.2172450

Visell, Y. (2018). HMMM - Max objects database. Retrieved from http://www.maxobjects.com/?v=objects&id_objet=3295&requested=HMMM&operateur=AND&id_plateforme=0&id_format=0

Wang, Q., Kurillo, G., Ofli, F., & Bajcsy, R. (2015). Evaluation of pose tracking accuracy in the first and second generations of Microsoft Kinect. In *2015 International Conference on Healthcare Informatics* (pp. 380–389). IEEE. https://doi.org/10.1109/ICHI.2015.54

Wang, T.-S., Shum, H.-Y., Xu, Y.-Q., & Zheng, N.-N. (2001). Unsupervised analysis of human gestures. In *PCM 2001: Advances in Multimedia Information Processing* (pp. 174–181). Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45453-5_23

Wasenmüller, O., & Stricker, D. (2017). Comparison of Kinect v1 and v2 depth images in terms of accuracy and precision. In *Computer Vision – ACCV 2016 Workshops* (Vol. 10117, pp. 34–45). Springer, Cham. https://doi.org/10.1007/978-3-319-54427-4_3

Webber, R. (2018). Kinect2share - GitHub. Retrieved from https://github.com/rwebber/kinect2share

Weiss, A., Hirshberg, D., & Black, M. J. (2011). Home 3D body scans from noisy image and range data. In *the International Conference on Computer Vision* (pp. 1951–1958). IEEE. https://doi.org/10.1109/ICCV.2011.6126465

Yang, H. D., Park, A. Y., & Lee, S. W. (2006). Human-robot interaction by whole body gesture spotting and recognition. In *the International Conference on Pattern Recognition* (Vol. 4, pp. 774–777). https://doi.org/10.1109/ICPR.2006.642

Zhang, Z. (2012). Microsoft Kinect sensor and its effect. *IEEE Multimedia*, *19*(2), 4–10. https://doi.org/10.1109/MMUL.2012.24

Zhou, R., Jingjing, M., & Junsong, Y. (2011). Depth camera based hand gesture recognition and its applications in human-computer-interaction. In *the 8th International Conference on Information, Communications & Signal Processing* (Vol. 5, pp.1-5). https://doi.org/10.1109/icics.2011.6173545

Appendix A.

Equations of the DTW Algorithm

The equations and explanations are based on the work by Nicholas Gillian, who has implemented the DTW algorithm as blocks in EyesWeb environment (Gillian, 2011).

Training the model is finding a template for each gesture in the training dataset – one sample in the class that gives the minimum normalized total warping distance when matched against the other training samples in that class:

$$\phi_g = \arg\min_i \quad \frac{1}{M_g - 1} \sum_{j=1}^{M_g} \mathbf{1}\{\text{ND-DTW}(\mathbf{X}_i, \mathbf{Y}_j)\}$$
$$1 \leq i \leq M_g$$

where:

G – the number of gestures in the training dataset.
φ_g – the N-dimensional template for the g^{th} gesture.
N – the number of data points in a sample or template (N-dimensional vector).
M_g – the number of training samples for the g^{th} gesture.
$1\{...\}$ – the indicator bracket, giving 1 when i ≠ j or 0 otherwise.
X_i and Y_j – i^{th} and j^{th} training samples for the g^{th} gesture
ND-DTW (X_i, Y_j) – the extension of the standard DTW algorithm to N-dimensions

ND-DTW function finds the warping path that minimizes the total normalised warping cost:

$$\text{ND-DTW}(\mathbf{X}, \mathbf{Y}) = \quad \min \quad \frac{1}{|\mathbf{w}|} \sum_{k=1}^{|\mathbf{w}|} DIST(\mathbf{w}_{k_i}, \mathbf{w}_{k_j})$$
$$DIST(i,j) = \sqrt{\sum_{n=1}^{N} (i_n - j_n)^2}$$

where:

X and Y – training samples in the form of $X = \{x_1, x_2, ..., x_N\}$ and $Y = \{y_1, y_2, ..., y_N\}$.
w_k – warping path of k^{th} pair of x_i and y_j .
$|w|$ – length of constructed warping path which has to be:

$$\max\{|\mathbf{x}|, |\mathbf{y}|\} \leq |\mathbf{w}| < |\mathbf{x}| + |\mathbf{y}|$$

$\frac{1}{|w|}$ – normalisation factor to allow comparison of warping paths of varying lengths
DIST(w_{ki} ,w_{kj}) –distance function of the warping path w_k.
DIST(i, j) – Euclidean distance between data point i in sample X and data point j in sample Y.

The minimum total warping path is found filling a cost matrix C of $|x|$ by $|y|$ dimensions, where the value of each matrix cell is given by:

$$\mathbf{C}_{(i,j)} = DIST(i,j) + \min\{\mathbf{C}_{(i-1,j)}, \mathbf{C}_{(i,j-1)}, \mathbf{C}_{(i-1,j-1)}\}$$

where:

$C_{(i,j)}$ – cell value that represents the accumulated minimum warping cost so far of data point *i* in sample *X* and data point *j* in sample *Y*.

min{...} – minimum accumulated distance from the three previous cells that neighbour the cell *i,j* (the cell above it, to its left and at its diagonal).

Once the cost matrix C is filled, the minimum possible warping path is calculated by navigating through it in reverse order from the cell $C_{(|x|,|y|)}$ to the cell $C_{(1,1)}$. At each step, the neighbour cell (to the left, above or diagonally) with the minimum value is selected and the previous three-cell search is repeated.

A *classification threshold* is calculated for each template in the trained model, it is controlled by the user with *gamma coefficient*:

$$\tau_g = \mu_g + (\sigma_g \gamma)$$

$$\mu_g = \frac{1}{M_g - 1} \sum_{i=1}^{M_g} \mathbf{1}\{\text{ND-DTW}(\phi_g, \mathbf{X}_i)\}$$

$$\sigma_g = \sqrt{\frac{1}{M_g - 2} \sum_{i=1}^{M_g} \mathbf{1}\{(\text{ND-DTW}(\phi_g, \mathbf{X}_i) - \mu_g)^2\}}$$

where:

μ_g - mean of total normalised warping distances between g^{th} template and X_i training sample.

σ_g – standard deviation of the normalised total warping distance.

γ – gamma coefficient that controls the number of standard deviations, manually defined by the user.

1{...} – indicator bracket, giving 1 when i ≠ the index of the training sample that gave the minimum normalised total warping distance when matched against the other M_{g-1} training samples in that class (i.e. the template) or 0 otherwise.

In the *prediction phase*, classification of a new sample X is finding the template in the model, which has the smallest total normalized warping distance:

$$c = \arg\min_{g} \quad \text{ND-DTW}(\phi_g, \mathbf{X}) \qquad 1 \leq g \leq G$$

where:

c – classification index representing g^{th} gesture with the minimum total warping distance.

Classification threshold is used to classify a new gesture as non-gesture (c = 0), if it is not similar to any learned gesture:

$$\hat{c} = \begin{cases} c & if(d \leq \tau_g) \\ 0 & \text{otherwise} \end{cases}$$

where:

d – total normalised warping distance between the g^{th} template and a new gesture X.
τ^g – classification threshold for the g^{th} template.

Appendix B.

Parameters of the Sensor Setup and the Composition Setup

Kinect2share utility (Figure 4.1), responsible for broadcasting sensor's data from Kinect SDK to EyesWeb environment, had these parameters:

- OSC host IP: 125.0.0.1
- OSC port: 8000

Ableton Live program with *LiveOSC* plugin (Figure 4.14), responsible for receiving OSC commands from EyesWeb environment, had these parameters:

- Control Surface: LiveOSC
- Input: none
- Output: Microsoft GS Wavetable Synth

LiveOSC accepts packets on port 9000.

Appendix C.

Parameters of the Training Patch

Module *"Input from Kinect"* (Figure 4.2) had these parameters:

Blocks "OSC Server":

- Port: 8000
- Number of outputs: 4
- Type of output 1: Double
- Type of output 2: Double
- Type of output 3: Double
- Type of output 4: String

Block "OSC Server" for "Left Hand" additionally had:

- Address Pattern: /0/HandL

Block "OSC Server" for "Right Hand" additionally had:

- Address Pattern: /0/HandR

Block "OSC Server" for "Left Elbow" additionally had:

- Address Pattern: /0/ElbowL

Block "OSC Server" for "Right Elbow" additionally had:

- Address Pattern: /0/ElbowR

Module *"Record Training Data"* (Figure 4.3) had these parameters:

Block "Scalar to Matrix":

- Number of Inputs: 12
- Output Mode: Row Vector [1 x N]

Block "Labelled Time Series Training Data Tool":

- ☑* Record: FALSE
- ☑* Class Label: 100
- ☑* Save
- ☑* Clear Last
- ☑* Clear All
- ☑* Clear Class
- ☑* Clear Class Value ID: 100
- File Name: C:\!KINECT\Data\TimeSeries.dat
- Recording Mode: Overwrite Mode

* Selected checkbox means that parameter's value can be received or changed by another block, connected to a pin on top of the block.

Block “Clock Generator” for “Beat every 0.5 sec”:

- Periodic: TRUE
- Period: 00:00:00:500
- Phase: 00:00:00:000

Block “Clock Generator” for “Bar every 2 sec”:

- Periodic: TRUE
- Period: 00:00:02:000
- Phase: 00:00:00:000

Module *“Timer for Recording”* (Figure 4.4) had these parameters:

Block “Clock Generator” for “Start Recording”:

- Periodic: TRUE
- Period: 00:00:02:000
- Phase: 00:00:00:050

Block “Bang” for “Start Recording”:

- Active: FALSE
- ☑ Activate

Block “Clock Generator” for “Stop Recording”:

- Periodic: TRUE
- Period: 00:00:02:000
- Phase: 00:00:00:000

Module *“Music Clip”* (Figure 4.5) had these parameters:

Block “Int Generator” for “Audio Track ID”:

- Value: 2
- Continuous output: FALSE

Blocks “Int Selector”:

- Num Inputs: 1
- ☑ Value Pin for Input: 0

Blocks “OSC Client”:

- Host/IP: 127.0.0.1
- Port: 9000
- Number of Inputs: 2
- Type of Input 1: Integer
- Type of Input 2: Integer

Block “OSC Client” for “Play Clip with Class ID” additionally had:

- Address Pattern: /live/play/clip

Block “OSC Client” for “Stop Clip” additionally had:

- Address Pattern: /live/stop/clip

Module *"Ableton Control"* (Figure 4.6) had these parameters:

Block "Clock Generator":

- Periodic: FALSE
- Absolute Time: TRUE
- Absolute Time: 00:00:02:000

Blocks "Int Selector":

- Num Inputs: 1
- Value Pin for Input: 1

Blocks "OSC Client":

- Host/IP: 127.0.0.1
- Port: 9000
- Number of Inputs: 1
- Type of Input: Integer

Block "OSC Client" for "Play" additionally had:

- Address Pattern: /live/play

Block "OSC Client" for "Stop" additionally had:

- Address Pattern: /live/stop

Module *"Model Training"* (Figure 4.7) had these parameters:

Block "DTW Train":

- ☑ Train
- File Format: Labelled Time Series Training Data
- Distance Method: Euclidean
- Pre-processing Method: None
- ☑ Gamma: 3
- ☑ Downsample Factor: 6
- ☑ Cross-Validation: TRUE
- ☑ K-Fold Value: 10
- Training Data File Name: C:\!KINECT\Data\TimeSeries.dat
- Model File Name: C:\!KINECT\Data\DTWmodel.dat

Appendix D.

Parameters of the Prediction Patch

Module *"Model Prediction"* (Figure 4.8) had these parameters:

Block "Scalar to Matrix" for data input:

- Number of Inputs: 12
- Output Mode: Row Vector [1 x N]

Block "DTW Predict":

- Input Mode: Trigger Mode
- ☑ Recording Status: FALSE
- ☑ Use Thresholding: TRUE
- Fle Name: C:\!KINECT\Data\DTWmodel.dat
- ☑ Gamma: 2

Block "Counter" for "Sample Counter":

- Type: Integer
- Step: 1
- Begin: 1000
- End: 1000000
- Custom reset value: 1
- ☑ Start
- ☑ Stop
- ☑ Reset
- Start Mode: Manual
- Reset Mode: Begin

Block "Scalar to Matrix" for "Predicted Class ID":

- Number of Inputs: 2
- Output Mode: Row Vector [1 x N]

Blocks "Matrix Display":

- Double buffering: FALSE
- Min Char: 3
- Max Char: 9
- Decimal Digits: 2
- Window Rect X: 1200
- Window Rect Width: 400
- Window Rect Height: 100
- Docked: FALSE
- Fullscreen: FALSE

Block "Matrix Display" for "Distances":

- Window Title: Distances
- Window Rect Y: 800

Block "Matrix Display" for "Thresholds":

- Window Title: Thresholds
- Window Rect Y: 900

Blocks “Cyclical Buffer”:

- IO Type: Double Matrix
- Buffer Size: 100
- Hop Size: 1
- Wait for Fill: FALSE

Block “Write Matrix to File - Int” for “Prediced Class ID”:

- File Name: C:\!KINECT\Data\PredictedClassIDs.txt

Block “Write Matrix to File - Double” for “Distances”:

- File Name: C:\!KINECT\Data\PredictionDistances.txt

Block “Write Matrix to File - Double” for “Thresholds”:

- File Name: C:\!KINECT\Data\PredictionThresholds.txt

Module *“Timer for Prediction”* (Figure 4.9) had these parameters:

Block “Clock Generator” for “Start”:

- Periodic: TRUE
- Period: 00:00:02:000
- Phase: 00:00:00:050

Block “Bang” for “Start”:

- Active: FALSE
- ☑ Activate

Block “Clock Generator” for “Stop”:

- Periodic: TRUE
- Period: 00:00:02:000
- Phase: 00:00:00:000

Block “If And”:

- Num Inputs: 2
- Percentage: 100
- Status Pin For Input 1: FALSE
- Status Pin For Input 2: TRUE

Module *“Music Clip”* (Figure 4.10) had these parameters:

Block “Int Generator” for “Audio Track ID”:

- Value: 2
- Continuous output: FALSE

Blocks “Int Selector”:

- Num Inputs: 1
- ☑ Value Pin for Input: 0

Blocks “OSC Client”:

- Host/IP: 127.0.0.1

- Port: 9000
- Number of Inputs: 2
- Type of Input 1: Integer
- Type of Input 2: Integer

Block “OSC Client” for “Play Clip with Class ID” additionally had:

- Address Pattern: /live/play/clip

Block “OSC Client” for “Stop Clip” additionally had:

- Address Pattern: /live/stop/clip

Module *“Record Prediction Data”* (Figure 4.11) had these parameters:

Block “Scalar to Matrix”:

- Number of Inputs: 12
- Output Mode: Row Vector [1 x N]

Block “Labelled Time Series Training Data Tool”:

- ☑ Record: FALSE
- ☑ Class Label: 100
- ☑ Save
- ☑ Clear Last
- ☑ Clear All
- ☑ Clear Class
- ☑ Clear Class Value ID: 100
- File Name: C:\!KINECT\Data\PredictedTimeSeries.dat
- Recording Mode: Overwrite Mode

Block “Clock Generator” for “Beat every 0.5 sec”:

- Periodic: TRUE
- Period: 00:00:00:500
- Phase: 00:00:00:000

Block “Clock Generator” for “Bar every 2 sec”:

- Periodic: TRUE
- Period: 00:00:02:000
- Phase: 00:00:00:000

Module *“Sequence for Adaptive Training”* (Figure 4.12) had these parameters:

Block “Compare With Value - Int”:

- Operation Type: Equal To (=)
- Value: 4

Block “Delay” gets the signal from the patch pin “AddSamples”:

- Delay: 00:00:00:050

Block “Delay” gets the signal from the first “Delay” and sends it to the patch pin “StopScript”:

- Delay: 00:00:00:100

Block “Delay” gets the signal from the first “Delay” and sends it to the patch pin “TrainModel”:

- Delay: 00:00:00:500

Block “Clock Generator”:

- Patch Start: TRUE
- Periodic: FALSE
- Absolute Time: FALSE

Block “String Generator”:

- Value: C:\Python27\python.exe C:\!KINECT\Python\addPredictedSamples.py
- Continuous Output: TRUE

Block “Spawn Command”:

- ☑ Active: FALSE
- ☑ Activate
- Allow Duplicates: TRUE

Module *“Model Training”* (Figure 4.13) had these parameters:

Block “DTW Train”:

- ☑ Train
- File Format: Labelled Time Series Training Data
- Distance Method: Euclidean
- Pre-processing Method: None
- ☑ Gamma: 3
- ☑ Downsample Factor: 6
- ☑ Cross-Validation: FALSE
- Training Data File Name: C:\!KINECT\Data\TimeSeries.dat
- Model File Name: C:\!KINECT\Data\DTWmodel.dat

Appendix E.

Parameters of Recording and Reproduction of the Data Stream

Module *"Input to File"* (Figure 5.1) had these parameters:

Blocks "OSC Server":

- Port: 8000
- Number of outputs: 4
- Type of output 1: Double
- Type of output 2: Double
- Type of output 3: Double
- Type of output 4: String

Block "OSC Server" for "Left Hand" additionally had:

- Address Pattern: /0/HandL

Block "OSC Server" for "Right Hand" additionally had:

- Address Pattern: /0/HandR

Block "OSC Server" for "Left Elbow" additionally had:

- Address Pattern: /0/ElbowL

Block "OSC Server" for "Right Elbow" additionally had:

- Address Pattern: /0/ElbowR

Block "Write to File":

- Name File: C:\!KINECT\Data\Kinect_toFile.ebf
- File Mode: OVERWRITE
- N Input: 19
- ☑ Record
- ☑ Stop

Module *"Input from File"* (Figure 5.2) had these parameters:

Block "Read from File":

- Name File: C:\!KINECT\Data\Kinect_toFile.ebf
- ☑ Play
- ☑ Stop

Block "Scalar to Matrix":

- Number of Inputs: 2
- Output Mode: Row Vector [1 x N]

Block "Cyclical Buffer":

- IO Type: Double Matrix
- Buffer Size: 300
- Hop Size: 1
- Wait for Fill: FALSE

Block “Write Matrix To File - Int”:

- File Name: C:\!KINECT\Data\RecordedClassIDs_fromFile.txt

Module *“Sequence for Input to & from File”* (Figure 5.3) had these parameters:

Block “Delay” in the column Play Clip:

- Delay: 00:00:07:500

Block “Delay” in the column Start Rec Samples:

- Delay: 00:00:00:500

Block “Delay” in the column Stop Rec Samples:

- Delay: 00:01:03:900

Block “Int Selector”:

- NumInputs: 3
- ☑ Value Pin for Input 1: 2
- Value Pin for Input 2: 0
- Value Pin for Input 3: 0

Block “Clock Generator” for the Count Classes:

- Periodic: TRUE
- Period: 00:01:12:000
- Phase: 00:00:00:000
- Absolute time: TRUE
- Absolute time: 00:00:02:000

Blocks “Counter”:

- Type: Integer
- Step: 1
- Begin: 0
- End: 1000

Block “Clock Generator” for the Count Samples:

- Periodic: TRUE
- Period: 00:00:02:000
- Phase: 00:00:00:000
- Absolute time: FALSE

Block “If And”:

- NumInputs: 2
- Percentage: 100
- Status Pin for Input 1: TRUE
- Status Pin for Input 2: TRUE

Appendix F.

Results of Evaluation

Table 3.
Thresholds of the classes depending on gamma coefficient, 32 training samples for each class.

32 samples	**Class 1 Threshold**	**Class 2 Threshold**	**Class 3 Threshold**	**Class 4 Threshold**	**Class 5 Threshold**	**Class 6 Threshold**
Gamma 0	0.621257	0.604583	0.800298	0.872311	0.603913	0.588843
Gamma 1	0.776884	0.758091	1.047386	1.193046	0.812408	0.854391
Gamma 2	0.932511	0.911599	1.294474	1.513781	1.020903	1.119939
Gamma 3	1.088138	1.065107	1.541562	1.834516	1.229398	1.385487
Gamma 4	1.243765	1.218615	1.78865	2.155251	1.437893	1.651035
Gamma 5	1.399392	1.372123	2.035738	2.475986	1.646388	1.916583
Gamma 6	1.555019	1.525631	2.282826	2.796721	1.854883	2.182131

Table 4.
Thresholds of the classes depending on the number of training samples, gamma 3.

Gamma 3	**Class 1 Threshold**	**Class 2 Threshold**	**Class 3 Threshold**	**Class 4 Threshold**	**Class 5 Threshold**	**Class 6 Threshold**
4 samples	0.8437115	1.091806	2.432071	1.947599	1.280601	1.520592
8 samples	0.941805	1.038489	1.790802	2.453806	0.853342	1.38806
16 samples	1.071221	1.011873	1.676632	1.931758	1.044471	1.447276
24 samples	1.108665	1.021059	1.564581	1.916805	1.054769	1.519906
32 samples	1.088138	1.065107	1.541562	1.834516	1.229398	1.385487

Table 5.
*Fitting prediction depending on gamma coefficient, **3 classes**, 32 training samples for each class.*

3 classes	**Hits**	**Misses**	**FA**	**CR**	**Total**	**Accur.**	**PPV**	**NPV**	**Sens.**	**Spec.**
Gamma 0	50	46	0	480	576	0.9201	1.0000	0.9125	0.5208	1.0000
Gamma 1	80	16	0	480	576	0.9722	1.0000	0.9677	0.8333	1.0000
Gamma 2	91	5	1	479	576	0.9896	0.9891	0.9897	0.9479	0.9979
Gamma 3	93	3	2	478	576	0.9913	0.9789	0.9938	0.9688	0.9958
Gamma 4	93	3	3	477	576	0.9896	0.9688	0.9938	0.9688	0.9938
Gamma 5	93	3	3	477	576	0.9896	0.9688	0.9938	0.9688	0.9938
Gamma 6	93	3	3	477	576	0.9896	0.9688	0.9938	0.9688	0.9938

Table 6.
*Fitting prediction depending on gamma coefficient, **6 classes**, 32 training samples for each class.*

6 classes	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
Gamma 0	97	95	0	960	1152	0.9175	1.0000	0.9100	0.5052	1.0000
Gamma 1	156	36	0	960	1152	0.9688	1.0000	0.9639	0.8125	1.0000
Gamma 2	184	8	1	959	1152	0.9922	0.9946	0.9917	0.9583	0.9990
Gamma 3	189	3	2	958	1152	0.9957	0.9895	0.9969	0.9844	0.9979
Gamma 4	189	3	3	957	1152	0.9948	0.9844	0.9969	0.9844	0.9969
Gamma 5	189	3	3	957	1152	0.9948	0.9844	0.9969	0.9844	0.9969
Gamma 6	189	3	3	957	1152	0.9948	0.9844	0.9969	0.9844	0.9969

Table 7.
*Fitting prediction depending on gamma coefficient, **1 class**, 4 training samples for this class.*

1 class	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
Gamma 2	3	1	0	20	24	0.9583	1.0000	0.9524	0.7500	1.0000
Gamma 3	3	1	0	20	24	0.9583	1.0000	0.9524	0.7500	1.0000
Gamma 4	4	0	0	20	24	1.0000	1.0000	1.0000	1.0000	1.0000
Gamma 5	4	0	0	20	24	1.0000	1.0000	1.0000	1.0000	1.0000
Gamma 6	4	0	0	20	24	1.0000	1.0000	1.0000	1.0000	1.0000

Table 8.
*Fitting prediction depending on gamma coefficient, **2 classes**, 4 training samples for each class.*

2 classes	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
Gamma 1	7	1	0	40	48	0.9792	1.0000	0.9756	0.8750	1.0000
Gamma 2	7	1	0	40	48	0.9792	1.0000	0.9756	0.8750	1.0000
Gamma 3	8	0	2	38	48	0.9583	0.8000	1.0000	1.0000	0.9500
Gamma 4	8	0	2	38	48	0.9583	0.8000	1.0000	1.0000	0.9500
Gamma 5	8	0	3	37	48	0.9375	0.7273	1.0000	1.0000	0.9250

Table 9.
*Fitting prediction depending on gamma coefficient, **3 classes**, 4 training samples for each class.*

3 classes	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
Gamma 0	6	6	0	60	72	0.9167	1.0000	0.9091	0.5000	1.0000
Gamma 1	9	3	0	60	72	0.9583	1.0000	0.9524	0.7500	1.0000
Gamma 2	11	1	0	60	72	0.9861	1.0000	0.9836	0.9167	1.0000
Gamma 3	10	2	1	59	72	0.9583	0.9091	0.9672	0.8333	0.9833
Gamma 4	11	1	6	54	72	0.9028	0.6471	0.9818	0.9167	0.9000
Gamma 5	11	1	9	51	72	0.8611	0.5500	0.9808	0.9167	0.8500
Gamma 6	11	1	9	51	72	0.8611	0.5500	0.9808	0.9167	0.8500

Table 10.
*Fitting prediction depending on gamma coefficient, **4 classes**, 4 training samples for each class.*

4 classes	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
Gamma 1	13	3	0	80	96	0.9688	1.0000	0.9639	0.8125	1.0000
Gamma 2	14	2	0	80	96	0.9792	1.0000	0.9756	0.8750	1.0000
Gamma 3	15	1	1	79	96	0.9792	0.9375	0.9875	0.9375	0.9875
Gamma 4	15	1	5	75	96	0.9375	0.7500	0.9868	0.9375	0.9375
Gamma 5	15	1	6	74	96	0.9271	0.7143	0.9867	0.9375	0.9250

Table 11.
*Fitting prediction depending on gamma coefficient, **5 classes**, 4 training samples for each class.*

5 classes	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
Gamma 1	18	2	0	100	120	0.9833	1.0000	0.9804	0.9000	1.0000
Gamma 2	19	1	0	100	120	0.9917	1.0000	0.9901	0.9500	1.0000
Gamma 3	18	2	1	99	120	0.9750	0.9474	0.9802	0.9000	0.9900
Gamma 4	19	1	3	97	120	0.9667	0.8636	0.9898	0.9500	0.9700
Gamma 5	19	1	5	95	120	0.9500	0.7917	0.9896	0.9500	0.9500

Table 12.
*Fitting prediction depending on gamma coefficient, **6 classes**, 4 training samples for each class.*

6 classes	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
Gamma 2	22	2	0	120	144	0.9861	1.0000	0.9836	0.9167	1.0000
Gamma 3	23	1	1	119	144	0.9861	0.9583	0.9917	0.9583	0.9917
Gamma 4	23	1	1	119	144	0.9861	0.9583	0.9917	0.9583	0.9917
Gamma 5	23	1	1	119	144	0.9861	0.9583	0.9917	0.9583	0.9917
Gamma 6	23	1	1	119	144	0.9861	0.9583	0.9917	0.9583	0.9917

Table 13.
*Cross-validation prediction with **gamma 3** depending on the number of training samples.*

Gamma 3	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
4 samples	157	35	35	1117	1344	0.95	0.82	0.97	0.82	0.97
8 samples	149	43	43	1109	1344	0.94	0.78	0.96	0.78	0.96
16 samples	170	22	22	1130	1344	0.97	0.89	0.98	0.89	0.98
24 samples	183	9	9	1143	1344	0.99	0.95	0.99	0.95	0.99
32 samples	187	5	5	1147	1344	0.99	0.97	1.00	0.97	1.00

Table 14.
*Fitting prediction depending on the number of training samples, **3 classes**, gamma 3.*

3 classes	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
4 samples	10	2	1	59	72	0.9583	0.9091	0.9672	0.8333	0.9833
8 samples	23	1	0	120	144	0.9931	1.0000	0.9917	0.9583	1.0000
16 samples	46	2	0	240	288	0.9931	1.0000	0.9917	0.9583	1.0000
24 samples	70	2	1	359	432	0.9931	0.9859	0.9945	0.9722	0.9972
32 samples	93	3	2	478	576	0.9913	0.9789	0.9938	0.9688	0.9958

Table 15.
*Fitting prediction depending on the number of training samples, **6 classes**, gamma 3.*

6 classes	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
4 samples	23	1	1	119	144	0.9861	0.9583	0.9917	0.9583	0.9917
8 samples	47	1	0	240	288	0.9965	1.0000	0.9959	0.9792	1.0000
16 samples	94	2	0	480	576	0.9965	1.0000	0.9959	0.9792	1.0000
24 samples	143	1	0	720	864	0.9988	1.0000	0.9986	0.9931	1.0000
32 samples	189	3	2	958	1152	0.9957	0.9895	0.9969	0.9844	0.9979

Table 16.
*Cross-validation prediction depending on the number of training samples, **3 classes**, gamma 3.*

3 classes	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
4 samples	144	48	48	528	768	0.8750	0.7500	0.9167	0.7500	0.9167
8 samples	158	34	34	542	768	0.9115	0.8229	0.9410	0.8229	0.9410
16 samples	172	20	20	556	768	0.9479	0.8958	0.9653	0.8958	0.9653
24 samples	184	8	8	568	768	0.9792	0.9583	0.9861	0.9583	0.9861
32 samples	188	4	4	572	768	0.9896	0.9792	0.9931	0.9792	0.9931

Table 17.
*Cross-validation prediction depending on the number of training samples, **6 classes**, gamma 3.*

6 classes	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
4 samples	157	35	35	1117	1344	0.9479	0.8177	0.9696	0.8177	0.9696
8 samples	149	43	43	1109	1344	0.9360	0.7760	0.9627	0.7760	0.9627
16 samples	170	22	22	1130	1344	0.9673	0.8854	0.9809	0.8854	0.9809
24 samples	183	9	9	1143	1344	0.9866	0.9531	0.9922	0.9531	0.9922
32 samples	187	5	5	1147	1344	0.9926	0.9740	0.9957	0.9740	0.9957

Table 18.
*Fitting prediction depending on the number of classes, **4 training samples** for each class, gamma 3.*

4 samples	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
1 class	3	1	0	20	24	0.9583	1.0000	0.9524	0.7500	1.0000
2 classes	8	0	2	38	48	0.9583	0.8000	1.0000	1.0000	0.9500
3 classes	10	2	1	59	72	0.9583	0.9091	0.9672	0.8333	0.9833
4 classes	15	1	1	79	96	0.9792	0.9375	0.9875	0.9375	0.9875
5 classes	18	2	1	99	120	0.9750	0.9474	0.9802	0.9000	0.9900
6 classes	23	1	1	119	144	0.9861	0.9583	0.9917	0.9583	0.9917

Table 19.
*Cross-validation prediction depending on the number of classes, **4 training samples** for each class, gamma 3.*

4 samples	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
1 class	170	22	22	170	384	0.8854	0.8854	0.8854	0.8854	0.8854
2 classes	162	30	30	354	576	0.8958	0.8438	0.9219	0.8438	0.9219
3 classes	144	48	48	528	768	0.8750	0.7500	0.9167	0.7500	0.9167
4 classes	142	50	50	718	960	0.8958	0.7396	0.9349	0.7396	0.9349
5 classes	142	50	50	910	1152	0.9132	0.7396	0.9479	0.7396	0.9479
6 classes	157	35	35	1117	1344	0.9479	0.8177	0.9696	0.8177	0.9696

Table 20.
*Cross-validation prediction **with adaptive feature** depending on the number of training samples, **6 classes**, gamma 3.*

6 classes	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
4 samples	159	33	33	1119	1344	0.9509	0.8281	0.9714	0.8281	0.9714
8 samples	148	44	44	1108	1344	0.9345	0.7708	0.9618	0.7708	0.9618
16 samples	169	23	23	1129	1344	0.9658	0.8802	0.9800	0.8802	0.9800
24 samples	182	10	10	1142	1344	0.9851	0.9479	0.9913	0.9479	0.9913
32 samples	187	5	5	1147	1344	0.9926	0.9740	0.9957	0.9740	0.9957

Table 21.
*Difference of cross-validation **with adaptive feature** compared to cross-validation **without** adaptive feature, depending on the number of training samples, **6 classes**, gamma 3.*

6 classes	Hits	Misses	FA	CR	Total	Accur.	PPV	NPV	Sens.	Spec.
4 samples	2	-2	-2	2	0	0.0030	0.0104	0.0017	0.0104	0.0017
8 samples	-1	1	1	-1	0	-0.0010	-0.0052	-0.0009	-0.0052	-0.0009
16 samples	-1	1	1	-1	0	-0.0010	-0.0052	-0.0009	-0.0052	-0.0009
24 samples	-1	1	1	-1	0	-0.0010	-0.0052	-0.0009	-0.0052	-0.0009
32 samples	0	0	0	0	0	0.0000	0.0000	0.0000	0.0000	0.0000

Table 22.
*Cross-validation prediction **with adaptive feature** depending on the number of training samples, **3 classes**, gamma 3.*

3 classes	**Hits**	**Misses**	**FA**	**CR**	**Total**	**Accur.**	**PPV**	**NPV**	**Sens.**	**Spec.**
4 samples	144	48	48	528	768	0.8750	0.7500	0.9167	0.7500	0.9167
8 samples	164	28	28	548	768	0.9271	0.8542	0.9514	0.8542	0.9514
16 samples	171	21	21	555	768	0.9453	0.8906	0.9635	0.8906	0.9635
24 samples	182	10	10	566	768	0.9740	0.9479	0.9826	0.9479	0.9826
32 samples	186	6	6	570	768	0.9844	0.9688	0.9896	0.9688	0.9896

Table 23.
*Difference of cross-validation **with adaptive feature** compared to cross-validation **without** adaptive feature, depending on the number of training samples, **3 classes**, gamma 3.*

3 classes	**Hits**	**Misses**	**FA**	**CR**	**Total**	**Accur.**	**PPV**	**NPV**	**Sens.**	**Spec.**
4 samples	0	0	0	0	0	0.0000	0.0000	0.0000	0.0000	0.0000
8 samples	6	-6	-6	6	0	0.0156	0.0313	0.0104	0.0313	0.0104
16 samples	-1	1	1	-1	0	-0.0026	-0.0052	-0.0017	-0.0052	-0.0017
24 samples	-2	2	2	-2	0	-0.0052	-0.0104	-0.0035	-0.0104	-0.0035
32 samples	-2	2	2	-2	0	-0.0052	-0.0104	-0.0035	-0.0104	-0.0035