UNIVERSITY OF BEIRA INTERIOR
Engineering

Usage of KINECT to detect walking problems of elder people

Pedro Alexandre Lopes de Jesus

Dissertion for obtaining the degree of Master of Science in
Computer Science and Engineering
(2nd Cycle of Studies)

Advisor
Prof. Dr. Nuno Garcia
Co-Advisor
Prof. Dr. Nuno Pombo

Covilhã, October 2017

Dissertation prepared at University of Beira Interior and at ALLab, submitted for defense in a public examination session at University of Beira Interior. Work partially financed by the Portuguese Science and Technology Foundation within Strategic Project PEst-OE/EEI/LA0008/2013 and UID/EEA/50008/2013 Project.

Acknowledgements

First of all, I would like to dedicate this thesis to my parents for their continuous support and strength, even when I was thinking of giving up they encouraged and cheered to never give up and continue until the end.

To my parents, even when I was about to give up they always cheered me on and encouraged me to finish this one last effort on achieving a higher academic degree.

To my brother, for all the jokes and funny moments when I was developing the application and he could find the weirdest details to make me laugh and motivate me in correcting them.

To my love Danissa, presently or by the distance, you always gave me the strength to endure the hardest and most monotonous parts of my dissertation, for all your help on gathering articles and information about work done in this area from different countries in order to have a more global approach.

To my Professors, Nuno Garcia and Nuno Pombo, for the scientific support, the teaching skills and human character.

To my colleagues in the ALLab of the Universidade da Beira Interior, for all their help and good working environment.

To all the unconditional support from my friends, suggestions and advices, laughs and distractions, which provided me with an awesome environment and desire to always move forward and conclude my thesis.

Finally, I would also like to acknowledge the contribution of the COST Action IC1303 – AAPELE – Architectures, Algorithms and Protocols for Enhanced Living Environments.

Resumo

A dissertação aborda o problema da análise e deteção de défice de mobilidade em pessoas idosas, com vista a permitir uma intervenção rápida por parte de pessoas especializadas e capazes de ajudar a melhorar a qualidade de movimentação.

Por forma a alcançar uma solução ao problema referido, foi utilizado o dispositivo de deteção de movimentos e gestos Kinect, desenvolvido pela empresa Microsoft, que permitiu a gravação de todo os movimentos realizados por um conjunto de pessoas idosas selecionadas aleatoriamente, para efeitos de obtenção de dados que nos permitam a posterior análise e classificação das capacidades motoras de cada indivíduo.

Adicionalmente foi necessário a criação de uma aplicação, capaz de extrair informações relevantes dos vídeos gerados pelo Kinect, tratar essas informações de forma a ser possível realizar a classificação dos movimentos dos indivíduos.

Assim, a aplicação desenvolvida subdivide-se em três etapas principais:

- A obtenção das coordenadas XYZ, para um conjunto relevante de ossos do esqueleto do indivíduo, em todas as frames da gravação e a duração total do movimento em si;

- Tratamento dos dados extraídos e normalização dos mesmos, por forma a serem utilizados no classificador;

- Criação de um classificador através de redes neuronais, capaz de utilizar os dados normalizados e classificar o movimento do idoso, de acordo com a qualidade do mesmo (existência de um défice de mobilidade ou não).

Na dissertação será descrito todo o processo de desenvolvimento, desde a estruturação da solução proposta até às classes desenvolvidas em código.

Resumo Alargado

Sem necessidade de recuar muito no tempo é possível constatar que a evolução do conhecimento humano, aumentou exponencialmente ao longo do último século e que esta tendência tem tudo para continuar no século que decorre.

Uma das grandes promessas de interligação da tecnologia é definida como a Internet of Things (IoT) que representa a ligação entre todos os dispositivos capazes de receber, transmitir e compartilhar dados e informações, não só dispositivos como computadores, smartphones e tablets, mas inclusive carros, eletrodomésticos, monitores de frequência cardíaca entre outras inúmeras possibilidades, tudo graças à combinação de processadores cada vez mais rápidos e pequenos e principalmente das tecnologias sem fios, que facilitam as comunicações a níveis excecionais.

Mas como é que a Internet of Things está a revolucionar a saúde?

A resposta a esta questão é muito simples. Pensando no exemplo de um hospital é imprescindível manter o controlo a tempo inteiro dos sinais vitais dos pacientes numa enfermaria, máquinas que mantém seres humanos vivos têm que se adaptar em tempo real às necessidades do mesmo, sem necessitar de intervenção por parte dos médicos e enfermeiros e/ou assistência remota por parte de um médico numa intervenção cirúrgica.

Já num contexto doméstico é de extremo valor um paciente ter sensores em sua casa que permitam a deteção de qualquer problema, permitindo atempadamente auxílio, dando como exemplo o caso de um ataque cardíaco, problemas respiratórios, queda violenta. Estes são apenas alguns exemplos do quão importante e revolucionária tem sido a Internet of Things e as vastas possibilidades que estão ainda por descobrir/aplicar.

Neste caso em particular, apesar de existir o potencial de interligar o nosso trabalho com múltiplos dispositivos, tal não foi considerado como sendo um objetivo pertinente durante o desenvolvimento do projecto, no entanto é sem dúvida algo a ser considerado como trabalho futuro, em que por exemplo se poderia desenvolver um servidor que interligasse os vários dispositivos involucrados e facultasse informações on-demand sempre que solicitado.

Esta dissertação tem como objetivo aplicar todos estes avanços e inovações, com vista a desenvolver um mecanismo capaz de captar/monitorar pessoas incluídas numa faixa etária, onde os problemas têm maior nível de incidência devido a limitações físicas, conseguindo para tal identificar pacientes que apresentem uma capacidade motora reduzia, para que sejam aplicados tratamentos e planos de melhoria atempadamente, evitando assim um maior desconforto, proporcionando-lhes uma maior qualidade de vida.

Para atingir este objetivo foi necessária a deslocação a lares de idosos locais (Caria e Ferro), solicitando a sua cooperação, obtendo assim um conjunto de dados reais e fidedignos, referentes a situações reais e pessoas que podem ou não apresentar problemas de locomoção.

Solicitou-se aos pacientes que se sentassem numa cadeira, se levantassem e caminhassem cerca de 3 metros, em direção ao dispositivo de gravação, de seguida regressassem e voltassem a sentar

na cadeira, num total de 3 trajetos por pessoa.

A captura dos movimentos foi levada a cabo, utilizando as funcionalidades do dispositivo KINECT, desenvolvido pela empresa Microsoft que possui as seguintes funcionalidades fundamentais: câmara RGB (Red, Green, Blue), sensor de profundidade por infravermelhos que permite todo o mapeamento 3D do ambiente circundante e a deteção de 48 pontos de articulação do corpo humano.

Fazendo uso destas capacidades inerentes ao dispositivo KINECT, foi possível a identificação das articulações dos pacientes nos vários momentos do seu movimento, obtendo a sua localização espacial em coordenadas XYZ, o que permitiu efetuar alguns cálculos de velocidade e aceleração. No entanto estes dados por si só não servem, ou melhor têm que ser transformados, adaptados e normalizados, de modo a ser possível a sua inserção num classificador de redes neuronais. Este classificador como qualquer outro do género, segue o procedimento de treino da rede neuronal e posterior utilização para realização da classificação que se pretende. Neste caso em particular foi utilizado um estilo de aprendizagem supervisionada uma vez que para cada conjunto de valores de entrada é facultado o respetivo valor de saída associada.

Será utilizado um certo número das gravações realizadas para treinar a rede, que incluíam não só casos de pacientes com défice de mobilidade, mas também outros em que o nível de mobilidade esteja dentro dos valores aceitáveis. Sendo assim é extremamente importante fazer uma boa gestão dos recursos disponíveis, uma vez que o número de pacientes e gravações realizadas é reduzido.

Como objetivo final pretende-se que todo este processo de recolha e tratamento de dados dos pacientes, assim como a respetiva classificação dos mesmos, em relação aos níveis de mobilidade dos vários pacientes, esteja de acordo com estudos realizados previamente sob os mesmos indivíduos, mas sem utilização de qualquer mecanismo ou dispositivo. Sendo que em caso afirmativo temos então um mecanismo capaz de prestar um melhor auxilio e controlo acerca das condições de saúde, orientadas à mobilidade dos pacientes, possibilitando um aumento da sua condição de vida.

Abstract

The dissertation addresses the problem in analyzing and detecting the lack of mobility in the elder population, while enabling a rapid intervention by specialized and able people that can help improve their movement quality.

In order to achieve a solution to the problem above, we used the motion detection and gestures device named Kinect, developed by Microsoft, which allowed the recording of all the movements performed by a set of randomly selected people, for the purpose of obtaining data that will allow us to further analysis and classification of motor skills on every person.

Additionally, it was necessary to create an application that can extract relevant information generated by the video Kinect and treat this information in order to carry out the person movements classification.

Thus, the application is divided into three main steps:

-Obtaining the XYZ coordinates, for a relevant set of bones of the person skeleton, in all the recorded frames as well as the total duration of the movement itself;

-Extracted data treatment and standardization, so it can later be used in the classifier;

-Creation of a classifier using neural networks methodology, which uses the standardized data in order to classify the person movement, according to its quality (existence of a mobility deficit or not).

Throughout the dissertation will be described every step of the development process, since the proposed solution design until the code of the developed classes.

Keywords

Movement feature selection, movement analysis, neural network, Kinect, J2K, skeleton feature extraction, data normalization

Contents

List of Figures

List of Tables

List of Acronyms

UBI	Universidade da Beira Interior
WWW	World Wide Web
IoT	Internet of Things
WHO	World Health Organization
OOTB	Out of the box
VR	Virtual Reality
XYZ	3D Coordinates XYZ
RGB	Red, Green and Blue
3D	Three-dimensional
SDK	Software Development Kit
J2K	Java to Kinect Library
NN	Neural Network
MSE	Mean Square Error

Chapter 1

Introduction

1.1 Motivation and Scope

A technology wave invaded our lives and it is safe to say we can no longer live without it. In spite of so many technology improvements a new concept arose, the "Internet of Things" (IoT).

The Internet of Things represents a technological evolution in which objects, devices, equipment are connected to each other through the Internet allowing instant access to remote information. From automobiles to appliances it is possible to connect any object to the Internet.

One of the main components of the IoT is using Off the Shelf devices, which includes the versatile Kinect, that can be used for recreational purposes, but also integrates technology that may have health-related purposes. For this reason, and doing a fusion of the fields, the motivation for this work was precisely to use the Kinect in order to investigate the determination of the mobility degree in elderly.

In the Health Care environment, it is known that most of the laboratory exams depend on technology, such as X-Ray, Tomography, Magnetic Resonance which are used on a regular basis on hospitals for a long time. Furthermore, there other technologies associated with the Internet of Things which have been really welcomed into the Health Care environment.

Recently, applications directed to smartphones have eased the access from healthcare professionals to important details and information, such as laboratory reference values, vaccines, medicine packages and receipts, therapy conducts, and much more.

During surgeries, the patients are monitored through devices each day more sophisticated which shown very precise and valuable data to the medical team.

Robotic surgery has also evolved at a fast pace as well, allowing surgical procedures to be more effective and reducing possible risks.

All this, leads any human being with a capable mind and attitude to try and invest his time into the Health Care environment, allowing himself not only to investigate and develop skills that not only help him enhance his personal knowledge but at the same time creating technology and applications that can help other people lives by giving them better support and life quality improvements.

1.2 Problem Statement and Objectives

The determination of the mobility degree in an elderly person is of the utmost importance as it can serve to validate therapies and well-being, contributing to the enhancement of the person's active aging. Therefore, the design of new methodologies to address this problem is needed, as these may bring innovations and thus contributing to the overall increase in the health of the elderly.

Our goal with this thesis is to be able to read data generated by a Kinect device that represents the motion of several elderly people and being able to judge and classify that person as having walking difficulties or not.

The person's motion is taken right from the moment they get up from a chair, walk a few meters in the direction of the Kinect device, and go back to sit on the chair again thus ending the captured movement.

This data comes in the form of a large set of movie frames taken by the Kinect device. Then it is required to manipulate the data and extract valuable information such as the total number of frames, the total movie and frame time interval respectively, detect and store the human bones and joints locations in every frame.

With this information, we can then calculate the person position during his entire movement (getting up from the chair, walking towards the device, walking back to the chair or sitting on the chair), and calculate his speed and acceleration.

Using the this generated information we can then achieve our real objective which is to classify the person's movement.

1.3 Adopted Approach for Solving the Problem

In order to resolve the problem stated above it is necessary to talk about the technologies and theoretical information that helped us in the entire development process of our application:

The first and most important one is the Kinect [1] device originally codenamed Project Natal during its development phase. It is a motion sensing input device created by Microsoft for Xbox 360 and Xbox One video game consoles and Microsoft Windows PCs. Based around a webcam-style add-on peripheral, it enables users to control and interact with their console/computer without the need for a game controller, through a natural user interface using gestures and spoken commands. Until now there have been two generations of the Kinect device, the first-generation Kinect was first introduced in November 2010. A version for Microsoft Windows was released on February 1, 2012. And the second-generation Kinect was released with the Xbox One platform starting in 2013. Microsoft released the first Beta of the Kinect software development kit for Windows 7 on June 16, 2011. This SDK was meant to allow developers to write Kinecting apps in C++/CLI, C Sharp, or Visual Basic.NET. On our thesis, we used an adaptation of this framework into the JAVA language which eased all of our code development.

In terms of theoretical information, we had to search the Internet for previous work on working with the Kinect device and how to manipulate the data it generates. For this part, we used the Java for Kinect library [2] provided as an open-source project. In terms of information on how to create and implement a Neural Network [3] application we had to do some further research around the subject but plenty of information was found which made it easier to adjust and adapt to our needs.

It is also important to refer the chosen JAVA programming IDE, IntelliJ. Which provided with all the tools and required classes to implement our code, in order to create an interface that could provide the future users of the application with an easier way to access all of its functionalities and also simplifying the way they could perceive the final results.

It is also extremely important to mention one of the tools that were indispensable to the whole work, which is the movie DATASET of every person containing all the graphical representation of the person movement, without them, it would be impossible to do our thesis. The analysis of each individual person DATASET became possible by only having in mind the bones and joints 3D coordinates through time. With this information, we were to identify the exact person position as well as calculate his velocity and acceleration through the entire movement thus leading to a better final classification.

1.4 Main Contributions

The development process of our application can be divided into 5 major steps:

The first one being the manipulation and extraction of all the relevant information from the provided DATASET videos that were generated by the Kinect device when storing each person's movements. Here we used the J4K Java Library in order to read the data from the Kinect generated files so we could identify the total number of frames, the individual frame time, detect the bone and joint structure of the person for every each frame detailing the XYZ coordinates, and store all the gathered information in a new file. This procedure had to be done for each and every provided Kinect video.

The second one being is to read the files containing the gathered information in order to identify the person‘s position during his entire movement using the coordinates XYZ into one of the following situations: getting up from the chair; walking towards the device; walking towards the chair; sitting on the chair. Beyond that, the person‘s average speed and acceleration per frame is calculated and used to detect any anomalies or big oscillations during the entire movement.

The third one is the creation of the Neural Network, setting the number of input and output neurons, the number of hidden layers to be used and the activation function to be used. When all this is done we proceed to the network training phase using some percentage of the input data generated in step two.

The fourth one is the Neural Network classification phase where we use the remaining input data generated in step two to classify the persons in to having a movement disorder or not.

The fifth and last one is where we compare the classification output obtained from step four with the correct classification that has been given to us by a previous study on the same person using methods other than Kinect. in order to verify if our solution is fit for our primary goals and objectives.

1.5 Dissertation Overview

The body of this dissertation is constituted by four main chapters, preceded and succeeded by the Introduction and Conclusions and Future Work, respectively. There is an Appendix at the end of this dissertation. The compilation of the bibliographic references used along this work is included after Chapter 6. The contents of each one of the chapters of this dissertation can be summarized as follows:

Chapter 1 elucidates the context for the subject on which this dissertation is going to elaborate on, identifying the main objectives and the problem to be solved. In this chapter, it is also included the description of the main contributions resulting from this dissertation.

Chapter 2 provides an overview of the current state of the art regarding the subjects of interest of this dissertation. The two main subjects of interest reviewed in this chapter were Health Care problems, algorithms and techniques, and the comparison between the goals of our application and other solutions for similar problems that have been addressed by other persons in previous works, but also how can the data recorded by the Kinect device can be interpreted and manipulated according to our needs, in order to satisfactorily classify elderly people movements as having a quality deficit or not.

Chapter 3 is the core of this dissertation, doing a precise a description of how our application was created and all of its functionalities, together with this it will be discussed the development of our Kinect movies manipulation and data extraction algorithms, as well as our Neural Network implementation and how it uses the given data to produce the desired results.

Chapter 4 presents the whole process of interpreting the various obtained results validating them, or in other words, all the progress since the very first results either being conclusive or not, until the last ones which have attained the correct classification according to the person's movements.

Chapter 5 wraps up the most important conclusions of this dissertation, while also providing some directions for future research and work.

Chapter 2

State of the Art

One of the major problems and challenges the world is facing nowadays is the physical disability [4] concern. It requires rigorous exercises and lots of training from such persons in order to achieve a situation as close to normal life as possible. Recent studies [5] from World Health Organization (WHO) media center that were announced on their website indicated that over a billion people or in other words close to 15 percent of the world's population have some form of disability, which covers any form of impairment, activity and participation limitations.

It is found that physical disability is one of the most neglected social obligations regardless of whether the country is developed, developing or underdeveloped. Such people have the right and need to support in terms of rehabilitation in order to maintain their capacities. The effectiveness of such rehabilitation depends highly on the person's motivation, however, they are often non-linearly related, or in other words, the usual rehabilitation therapy process is generally monotonous which does not help at all towards the patient's motivation. Moreover, disabled people living in rural areas and therefore with less access to health makes the whole process of rehabilitation more demotivating due to inadequate skills and lack knowledge compared to those of hospitals and specialized clinics professionals. Plus the excessive costs of such health services offered by premium hospitals make this health care needs unmet.

As the human beings grow older they are susceptible to attain motor deficiencies and experience dramatic limitations in performing everyday activities such as walking, bathing, dressing, and eating. Motor deficiencies can be at least partially compensated for or recovered through physical rehabilitation, however, conventional rehabilitation training programs typically involve extensive repetitive range-of-motion and coordination exercises and require professional therapists to supervise the patient's movements and assess the progress. In the last decade, there have been some innovations and successful approaches to making use of Virtual Reality (VR) technology and video games to create a new generation of tools specialized in advanced rehabilitation programs. The main contributions of the VR technology to the physical therapy and rehabilitation area is the ability to use sensing devices capable of capturing and quantitatively assess the movements of patients under treatment, tracking their overall progress more accurately, whereas the video games are capable of motivating and encourage the patients towards their rehabilitation.

Various motion capture system has been developed so far: On one side we have devices capable of a precise tracking of the patient's movements which consist on attaching reflective markers to their bodies and limbs, however, this type of approach is normally cumbersome, uncomfortable and usually inhibits the patient's already limited movement, not to mention their expensive cost; On the other side they are relatively low-cost approach which consists of using game interfaces capable of capturing mechanical motions. Popular motion capture devices include SMART-DX [6], Optoelectronic Plethysmography System [7], OptiTrack [8], Xsens MVN [9], Vicon [10], Autodesk MAYA [11], Microsoft Kinect, and more on the gaming side the Nintendo Wii Remote and PlayStation Move.

On this project, we focused on using the low-cost side of motion capture systems, in particular, the Microsoft Kinect device [1], which is overall inexpensive, easy to set up and can be used in either home or clinical environments. It includes an RGB camera and a depth sensor, which together provide full-body 3D motion capture and joint tracking capabilities without markers or handheld controllers [12]. Moreover, its interface is nonintrusive and requires minimal interaction with the patients themselves.

The software libraries available right out-of-the-box (OOTB) provided by Microsoft Software Development Kit [13] (SDK) as well as other third party middleware implementations and toolkits for those who prefer different programming languages instead of C Plus Plus and/or C Sharp such as JAVA, an excellent example of these is the J4K Java Library [2]. All of them have a huge set of features and capacities which enable the reception of 3D positions of skeletal joints through streams of skeletal frames in real-time, hence Kinect can be used in areas far beyond the scope of video games.

By far, the most interesting data provided by Kinect is the skeletal data provided by the skeletal data streams and by registering an event handler responsible for receiving and processing the various skeletal frames. Each of these frames is capable of containing data for up to 6 users, however only 2 can be fully tracked. The users which are fully tracked, their 3D positions (x, y, z) on a total of 26 joints are reported, including: head, shoulder center, left/right shoulder, left/right elbow, left/right wrist, left/right hand, spine, hip center, left/right hip, left/right knee, left/right ankle and left/right foot. All these Kinect features and technologies enable a large set of applications to be developed far beyond the original gaming scope, covering other areas such as: 3D construction, physical therapy, fall detection, medical operation, education, training, sign language, recognition, retail, natural language interface, robotics control and much more.

A list of examples, previous adaptations, and usages of Kinect, will be presented while referencing their respective papers in the following paragraphs of this chapter, together with a comparison table between all systems similar to Kinect and therefore capable of replacing him:

	Kinect V1	Kinect V2 (Ours)	OptiTrack
Resolution	640 x 480	1920 x 1080	640 x 480
Frame Rate	30 FPS	30 FPS	100 FPS
Sensing Range	1.2 to 3.5m	4.5m	20m
Field of View	Horizontal: 57° Vertical: 43°	Horizontal: 70° Vertical: 60°	360°
Number of Cameras	1	1	Multiple
Markers	No	No	Yes
Number of Joints	20	26	Desired
Cost	Low	Low	High
Environment	Clinic, Home	Clinic, Home	Medical Center
Related to	[14] [15] [16] [17] [18] [19] [20] [21]	[19] [22] [23] [24] [25]	[14]

Table 2.1: Kinect and OptiTrack Specifications

Chien-Yen Chang, Belinda Lange, Mi Zhang, Sebastian Koenig, Phil Requejo, Noom Somboon, Alexander A. Sawchuk, Albert A. Rizzo [14], investigated the Microsoft Kinect capabilities as a robust tool for SCI rehabilitation, experimentally comparing its performance against the high precision OptiTrack optical motion capture system, the table 2.1 lists both systems primary spec-

ifications. They developed a prototype game for SCI rehabilitation and integrated it with their Kinect-based system to quantitatively assess patients' rehabilitation performance and track their progress in a clinic and home environments. According to their research, experimentation, and testing between a low-cost single-camera and high-cost multi-camera system, they concluded that the Kinect Device is a promising VR neurological rehabilitation tool if destined to be used in a clinic and home environments and highly recommendable for movements tasks involved in rehabilitation interventions.

Anil K. Roy, Yash Soni, Sonali Dubey [15], in order to make the unexciting rehabilitation process fun and rewarding they incorporated suitable motion-sensing serious games. Named the application "Kinect-o-Therapy", which consists of a motor rehabilitation tool, that complements the existing system of physical rehabilitation, targeting people suffering from motor disorders caused primarily by cerebral palsy, spinal cord injury, post stroke or hereditary muscle ailments that significantly affect limb movements and hinders a person's range of motion and ability to make coordinated movements. They tried to incorporate the three key elements of effective rehabilitation: Repetition, Motivation, and Feedback, in order to make their product as much effective and complete as possible. In spite of Kinect device being such a very cheap device with low to zero maintenance costs when compared to other clinic and laboratory machinery, they affirmed their system has a high potentiality to be used as a viable replacement for physical rehabilitation centers, moreover it also viable for a home environment as well and therefore the patient retains his privacy while still enhancing his own physical capacities.

Wenbing Zhao, Hai Feng, Roanna Lun, Deborah D. Espy, M. Ann Reinthal [16] created a system that demonstrates the correct way of doing a rehabilitation exercise, having a 3D avatar on one side of the screen based on pre-recorded motion data, and another on the other side of the screen, that reflects the actual patient movement. They also implemented a set of correctness rules for each exercise and assesses the patient's movement in real-time. The assessment results are incorporated in the patient avatar in the form of visual guides to help the patient perform the exercise correctly. The system also records vital data pertinent to the quality and quantity of exercises such as correct iterations as well as detailed motion data for real-time feedback and post-analysis. Concluding that their system can be of practical use for rehabilitation exercises due to being low computationally intensive and having a real-time assessment, plus the definition of correctness rules that serve as the invariance of the exercise independently from the size and form of the users.

Hesham Alabbasi, Alex Gradinaru, Florica Moldoveanu, Alin Moldoveanu [22], presented a two steps application: the first is to record a sequence sports exercises performed by a professional using the Kinect sensor while using the recorded data to generate a 3D avatar character; the second step is to record the imitative actions performed by the trainee in order to make a comparison between them. The overall objective is to make real-time corrections on the screen through messages and body part highlights, helping the user to correct his actions and self-learn the exercises sequence by following the real-time visual feedback. At the end, a summary of the trainee's performance and a general total score is displayed on the screen. They noticed that the scores are much lower as the complexity of the exercise rises. The capture from the Kinect sensor V2 has a real improvement when compared to its previous version providing good support for real-time evaluation of medical rehab or physical training exercises but still requires some improvements, however, their own developed skeletal animation could be improved to attain better results.

Daniel Leightley, Moi Hoon Yap, Jessica Coulson, Yoann Barnouin, Jamie S. McPhee [17], created a dataset of healthcare-related motions, named Kinect 3D Active (K3Da), by capturing balance, walking, sitting and standing poses from a diversified population range of both young and older adults. The motions were based on common clinical assessments used to assess movements in disease and frailty. Their objective is to provide said dataset for others to use as a benchmark and starting point of human movement detection, quantification, and recognition algorithms. Plus, they even released a basic toolset to facilitate novel data capture, viewing and motion analysis. They believe to have created the first dataset for Kinect One that enables benchmarking of healthcare-based applications, methods, and techniques with diverse samples of both young and elderly people. Despite having a large set of data our own application cannot make use of it since our objective is to classify a particular group of people and therefore individual data has to be retrieved, saved, transformed and classified independently of an existing dataset.

Monalisa Pal, Sriparna Saha, Amit Konar [23], recorded the movements from a group of subjects using the Kinect Sensor and later tried to find and identify frames where the angles corresponded to gestures previously checked as disease related by several doctors of renowned medical colleges. They achieved a high success, best showing an accuracy of 94.29 percent, responding in 3.83 milliseconds to a query. Of all the seven similarity measures they used: Euclidean distance (DE), Standardized Euclidean distance (DSE), City-block metric (DCB), Chebychev distance (DCC), Mahalanobis distance (DM), Cosine distance (DCS) and Correlation distance (DCR) they concluded that the one with the City-block metric outperforms the other metrics with respect to almost all the performance measure expect Precision as shown in table 2.2

Method	Recall (%)	Precision (%)	Accuracy (%)	F1 Score (%)	Error Rate (%)	Time (ms)
DE	89.02	92.03	91.43	90.50	8.57	3.86
DSE	92.31	89.02	92.86	90.64	7.14	5.41
DCB	93.21	94.30	94.29	93.75	5.71	3.83
DCC	91.02	91.12	91.43	91.07	8.57	3.86
DM	86.41	89.90	90.00	88.12	10.00	5.20
DCS	79.38	97.02	92.86	87.32	7.14	4.08
DCR	88.39	86.63	94.29	87.50	5.71	4.41

Table 2.2: Distance Metric Comparison

Liang Liu, Sanjay Mehrotra [24], used the Kinect Sensor V2 in order to detect and prevent pneumonia by evaluating patient activities while in a hospital room. Studies indicate that ambulation and/or walking is essential and therefore the patients are encouraged to walk as early as possible, and this is where the Kinect Sensor comes into hand by controlling and monitoring the patients in real time without any concerns or preoccupations on their side. They demonstrated that the walk detector performance matches well with the ground truth observed on the recorded depth images being a tool capable of evaluating a patient walk with an acceptable accuracy making it viable for hospital room environments. Moreover, it is noninvasive, contact-free, low-cost, very easy maintenance and protects the patient's privacy since it does not require color information.

Pavia Bera, Reshma Kar, Amit Konar [18], in order to recognize problems related to specific lower right side joints of the human body (ankle, knee, hip, and forefoot), used the Microsoft Kinect device together with a gait analysis algorithm, which allowed them to identify patterns associated

with pain. They initially recorded human body gait patterns and processed this information for data enhancement, retaining only the essential information. Then, when capturing the gait of a person that suffered from a joint problem they try to match it with the previously recorded ones thus being able to classify it into one of the four considered classes. They demonstrated experimentally that their scheme of data enhancement increases recognition rate of the classifier used as compared to a traditional feature selection method and obtaining the results displayed in the table 2.3

Body Joint	Classification Accuracy with Traditional Features (%)	Classification Accuracy with Proposed Data Enhancement (%)
Right Foot	57.99	77.84
Right Hip	88.77	90.12
Right Ankle	63.22	86.24
Right Knee	78.99	93.44
Average	72.24	86.91

Table 2.3: Classification Accuracy Comparison

Roanna Lun, Connor Gordon, Wenbing Zhao [19], focused on tracking the activities of daily living (ADL), making use of multiple Microsoft Kinect sensors (chosen by it's low-cost and excellent open source programming support) linked to a dedicated server. Pairing it with smart watches equipped with accelerometers and/or gyroscopes and magnetometers for the inertial sensing modality. Concluding that both devices together provide an excellent user identification while delivering real time feedbacks to the users. Plus, they designed their system in order to communicate over the network with other systems and mobile/wearable devices making it possible to continuously track each individual's ADL in the indoor environment as well as outdoor. In resume, they managed to create a system that integrates computer vision-based server, image sensors, mobile smartphones, wearable devices and a cloud web server for motion tracking, however, they admitted as future work that their solution could be further improved to enable the communication between multiple Kinect servers federating them together to cover a large area and/or multiple rooms.

D. González Ortega, F.J. Díaz Pernas, M. Martínez Zarzuela, M. Antón Rodríguez [20], presented a 3D vision-based marker-free system capable of monitoring the motion of multiple human body parts for post assessment and rehabilitation of body scheme dysfunctions and left-right confusion using the Microsoft Kinect device. Their aim was to monitor and extract the achievement level, calculate temporal parameters such as reaction time, fulfillment time, or failure time, as well as track the trajectories of human body parts in psychomotor exercises. Since the Kinect device does not any illumination to extract the depth information and only needs some amount of illumination for the AdaBoost face detector, this characteristic was crucial to their work allowing them to achieve a more robust and satisfactory performance on the cognitive rehabilitation exercises monitoring compared to a 2D system. The Kinect device was proved to be very useful in developing applications in many fields such as rehabilitation, easy to use and with optimal performance. being quite capable of human limb monitoring. Their system was evaluated with 15 users, achieving a successful monitoring percentage of 96.28%.

Yao-Jen Chang, Shu-Fang Chen, Jun-Da Huang [21], developed a Kinect based application to assist therapists in rehabilitating school students with motor disabilities, which detects the stu-

dent's joint position and uses the recorded data to determine whether the student's movements have reached the rehabilitation standard and/or if the number of exercises completed successfully in a therapy session is good enough. Real time information is given to the student's so they know if they are doing fine, together with an interactive interface and video-audio feedback enhancing their motivation, interest, and perseverance to maintain their rehabilitation exercises. All the recorded data by the system is then handled automatically to therapists so they can assess and review the student's rehabilitation progress quickly. They presented results that were applied to two students with a high which shown a high success rate as described in the table 2.4 and therefore demonstrating the viability of the Kinect technology on physical rehabilitation.

Days	Phase	Daily Average Correct Movements
01-05	Non-Kinect	42
06-16	Kinect	136
17-22	Non-Kinect	44
23-34	Kinect	173

Table 2.4: Application Results

César Bernal Bravo, Juan Jesus Ojeda-Castelo, Jose Antonio Piedra-Fernandez [25], made use of the Kinect full body gesture recognition capacity in order to help students with cognitive disabilities to improve their movements. They adapted their application to three different types of exercises: music, dance, and painting. In the music section, the students have to play a musical instrument and try to achieve specific gestures, and whenever successful they receive a music note feedback. For the dance section, they will have to follow a 3D avatar and follow its lead, accompanied by an acoustic music to encourage the student to move, if they fail the dance restarts. Lastly, the painting exercise where they have to follow painting pattern, controlling the color as well as the thickness of the stroke, plus, they also allowed this exercise to be action free letting the students use their own creativity. Comparing the usage and absence of the Kinect on the same exercises, they concluded that the students pay way more attention and focus when the Kinect was present which proved it to be a very useful tool for physical rehabilitation purposes.

Chapter 3

Development

The main objective of this dissertation is to use the capabilities of the Microsoft Windows Kinect device in order to record the movement of people with walking disabilities, more specifically elderly people, with post interpretation, analysis, and manipulation of the recorded data in a way that allow us to identify walking quality deficit and difficulties.

The entire development phase can be subdivided into four iterative phases which will be presented and explained thoroughly in their respective order, accompanied with image examples and code parts from our application: Video capturing using the Kinect device; Application organization and dataset preparation; Video data analysis and manipulation; Data classification.

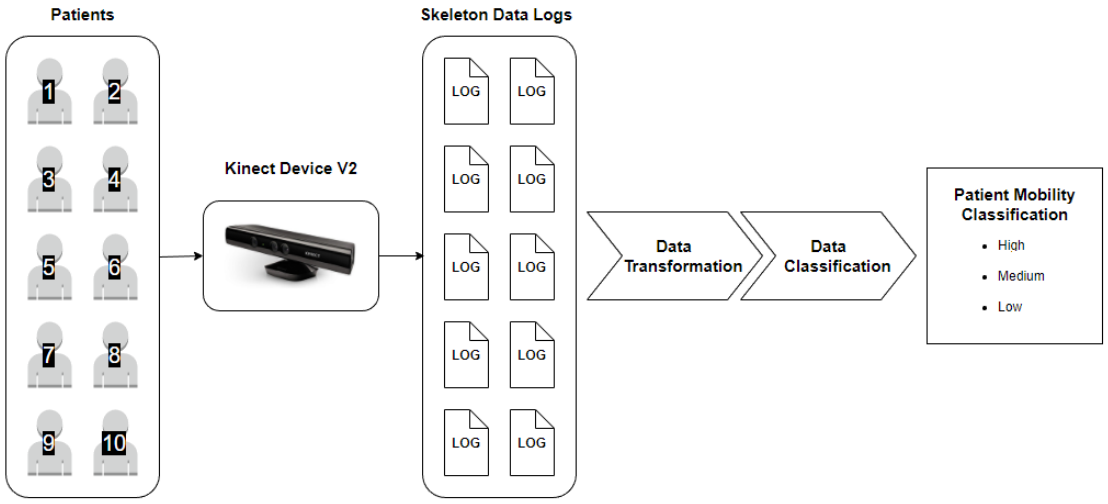

Figure 3.1: Application Architecture

3.1 Application Workflow

Before going deep into detail about what each of the implemented classes and their respective methods do it is of the utmost importance to explain the overall workflow of our programs. The diagram 3.1 already provides a high view of how we achieved our main objective however how everything connects together has to be described as well.

It all starts with the provided Kinect videos which contain all the necessary information about the patients' movements, however, they had to be analyzed so that pertinent and relevant information could be extracted from them, more specifically the skeleton and joints of the human body structure. The method "onSkeletonFrameEvent" from the class "Kinect" goes frame by frame detecting the skeleton and identifying each of the 25 joints coordinates (still only 20 can be detected in our case due to movements particularities), which are instantly saved in a log file together with the videos total time. This log files had to be stored locally accordingly to their dataset.

Having the datasets properly stored the developed application can be launched by running the main method on the "Application" class. Firstly it will try to load the training dataset inputs and outputs into memory and then the classification dataset. This is defined in the "ManageData" class which has the list of every log file and its respective information separated by their respective dataset. In order to fetch the data stored in each of the log files, plus handling the data normalization and transformation, the class "CalculateInput" was built with the sole purpose of setting the correct order of work for each of the log files, by calling respectively the "ReadTextFile", "NormalizeData" and "TransformData" in this same order.

Only when both datasets have been loaded and processed the neural network can be initiated and set to do fulfill its purpose. The class "Classification" is the one that orchestrates every procedure of the neural network, having the necessary methods which firstly initiate the "Network" class and then use its public methods to train it and classify the desired dataset to obtain a group of outputs which should be according to what we expected if everything went as expected.

3.2 Video capturing using the Kinect device

Two local elderly people houses were contacted, asking them for support on recording video examples from voluntary persons willing to help us with our investigations and work. Every one of the persons involved was asked to do a particular set of five movements three times with intervals between them:

- At the beginning of the movement, the patient is sitting on a chair, located approximately three to four meters away from the Kinect device, figure 3.2;
- Then they have to get up from the chair and walk towards the Kinect device until reaching approximately one meter away from it, figure 3.3;
- Next, they will turn back around towards the chair, figure 3.4;
- Walking back towards the chair, figure 3.5;
- Lastly, sitting back on the chair, thus ending the movement, figure 3.6.

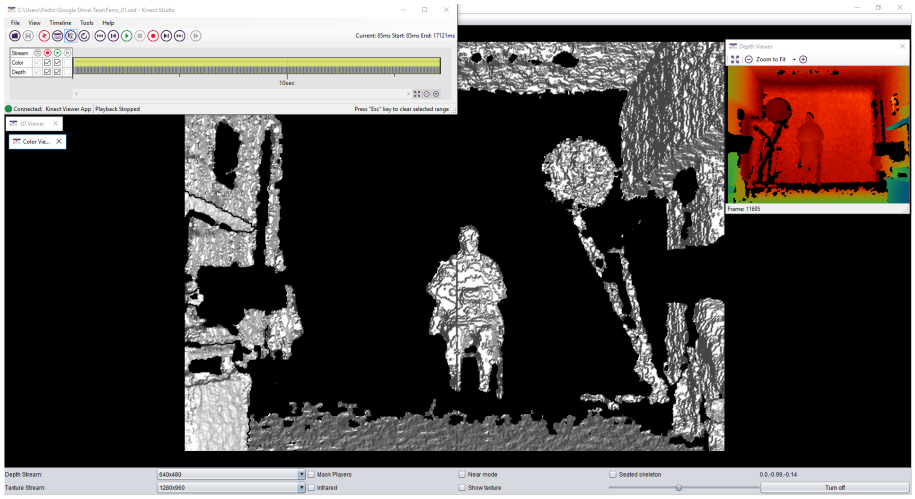

Figure 3.2: The patient is sitting on the chair

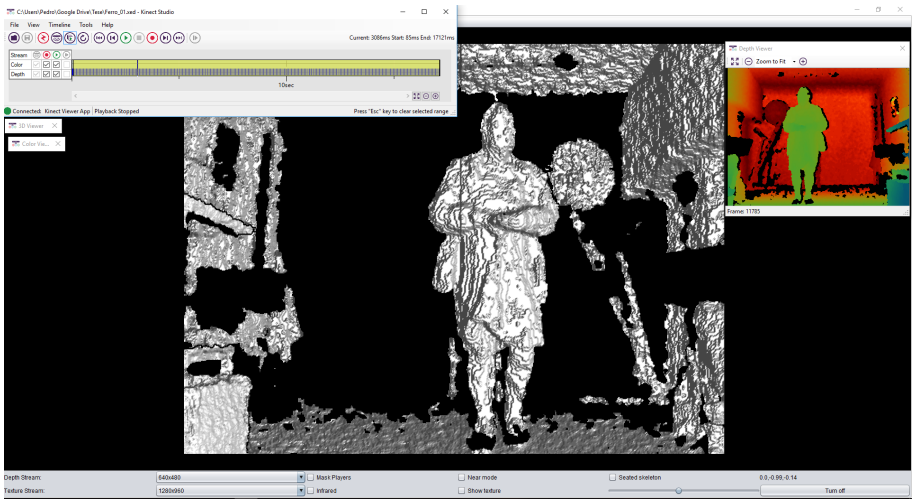

Figure 3.3: Gets up from the chair and walk towards the Kinect device

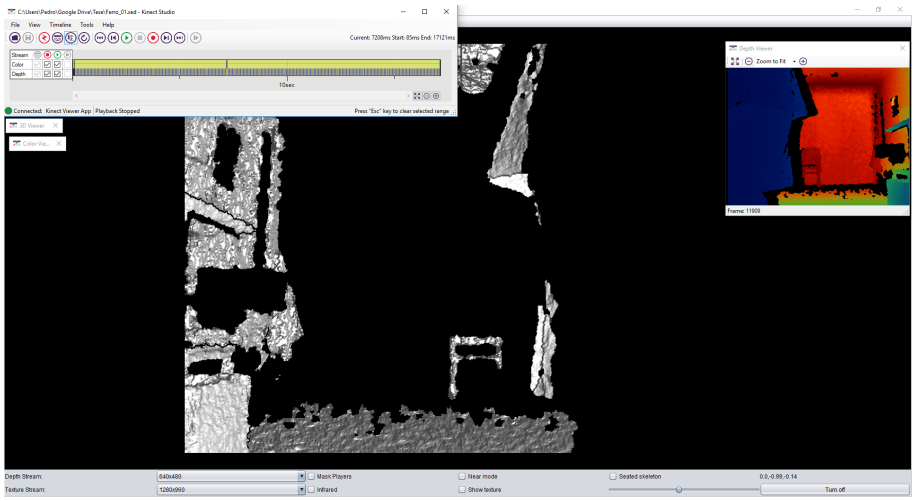

Figure 3.4: Turn back around towards the chair

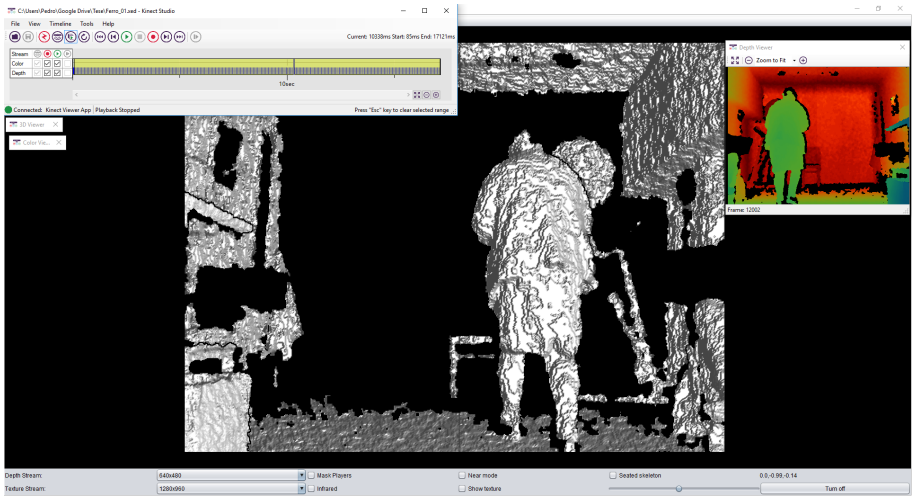

Figure 3.5: Walking back towards the chair

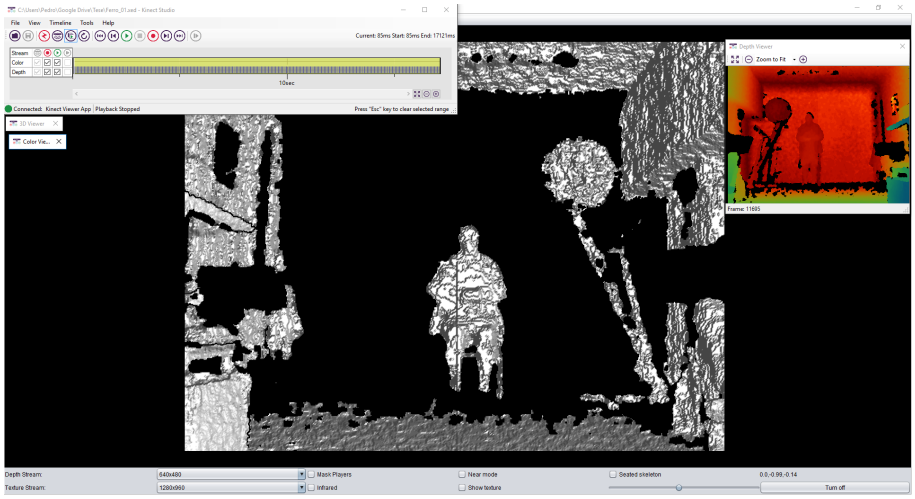

Figure 3.6: Sitting back on the chair

We counted with the help of twenty individuals, and a total of fifty-eight recorded videos overall. Which we believe to be more than enough to for our thesis objectives and aims. Still, the Kinect device is so much more than a simple recorder, it enables the user to work with the recorded data. Its framework provides with an abundant set of tools that enables video analysis and data extraction. On our case, we focused specifically on its amazing capability to read through the recorded video frames while detecting and identifying 25 different bones and joints for each present individual. Two tasks were considered on this part of our application development. The first one being the joints analysis and identification followed by the second one which consisted on extracting of their corresponding coordinates on a 3D axis XYZ on every video frame.

3.2.1 Joint Analysis and Identification

The video data analysis task translates into developing a way to isolate and identify the patient's joints through all their movement. Having in mind the set of five movements the patient has to follow presented previously, their corresponding skeleton images will be:

- Patient skeleton sitting on the chair, figure 3.7;
- Patient skeleton walking towards the device, figure 3.10;
- Patient skeleton turning around facing the chair, figure 3.9;
- Patient skeleton walking back towards the chair, figure 3.10;
- Patient skeleton sitting back on the chair, figure 3.11.

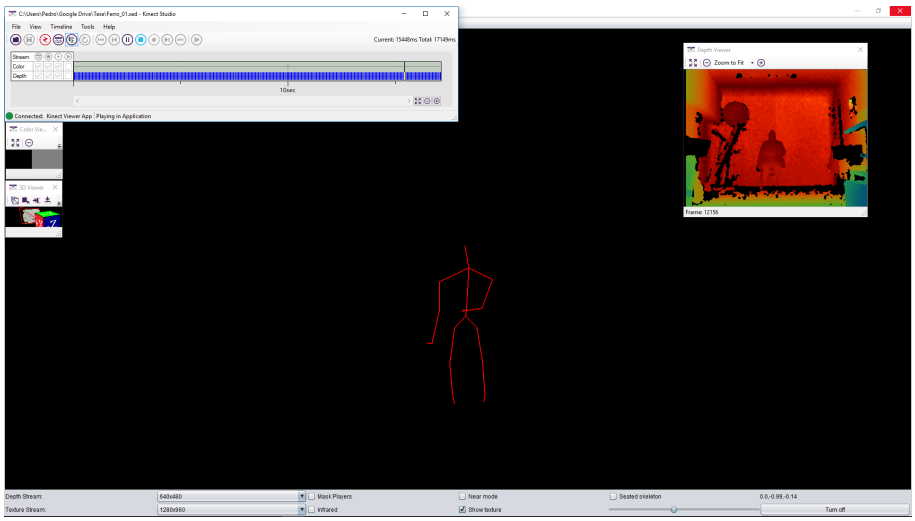

Figure 3.7: Patient skeleton sitting on the chair

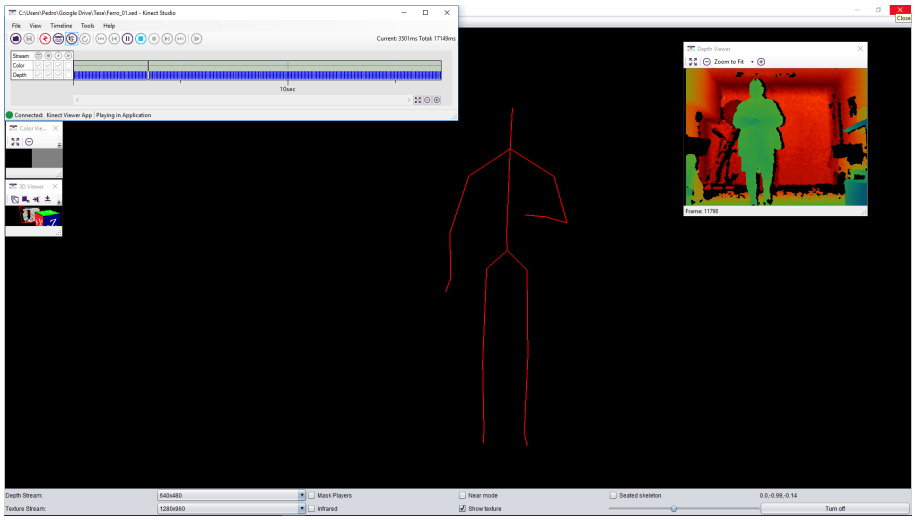

Figure 3.8: Patient skeleton walking towards the device

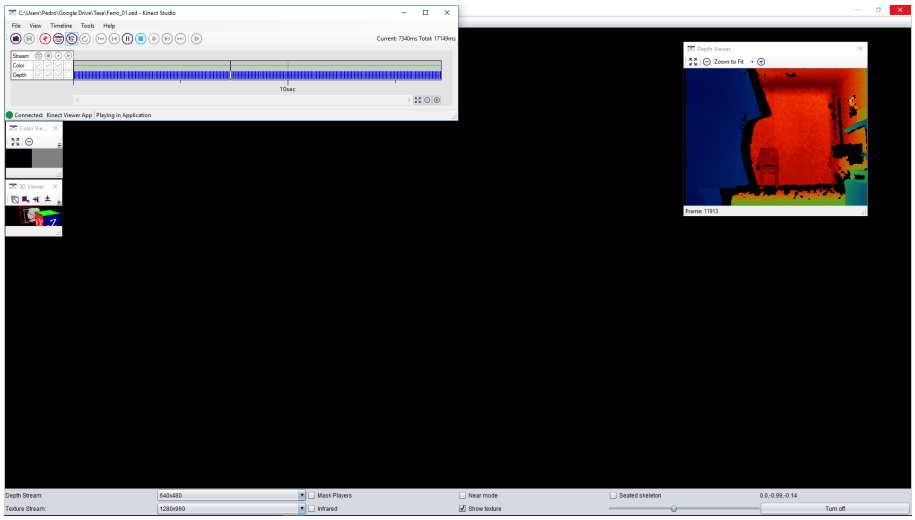

Figure 3.9: Patient skeleton turning around facing the chair

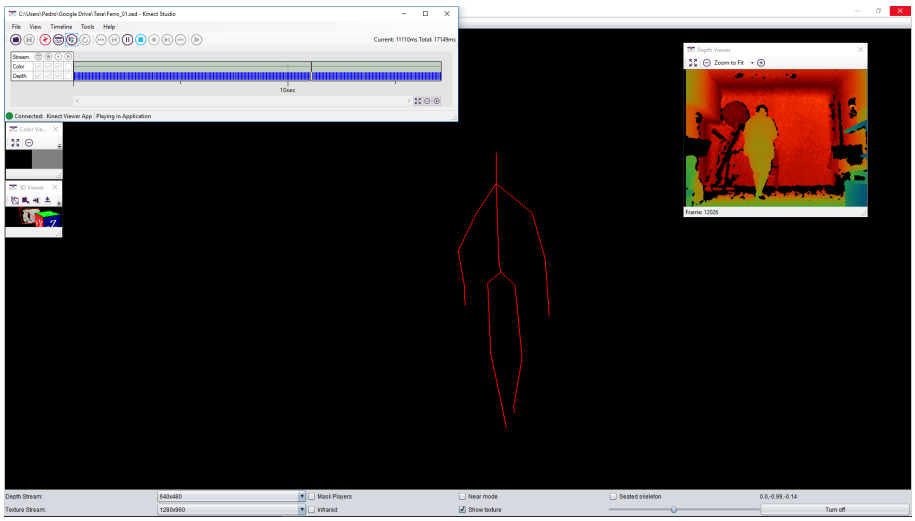

Figure 3.10: Patient skeleton walking back towards the chair

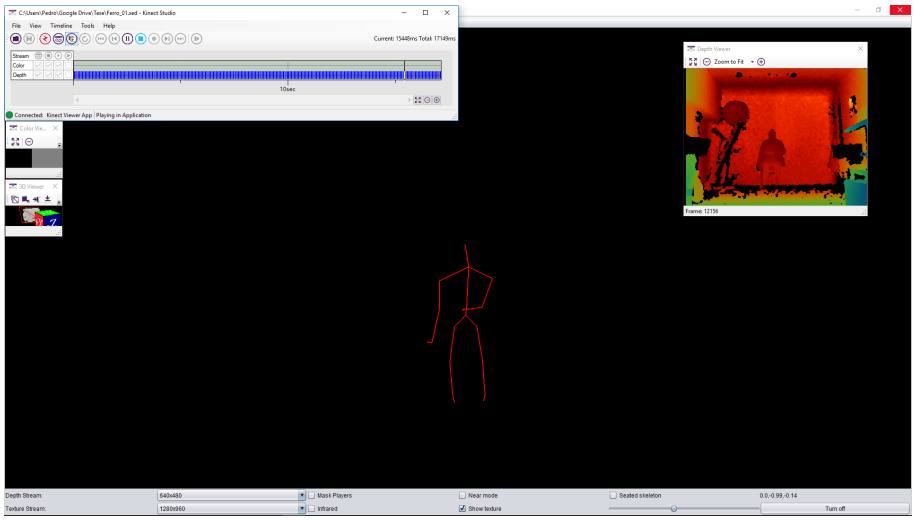

Figure 3.11: Patient skeleton sitting back on the chair

3.2.2 Extracting joint coordinates

In order to facilitate our work and increase its accuracy and achieve successful results, we made use of the J4K Java library which is an open source Java library that implements a Java binding for the Microsoft's Kinect SDK. It communicates with a native Windows library, which handles the depth, color, infrared, and skeleton streams of the Kinect using the Java Native Interface (JNI).

Below following method header is one the existing methods on the Kinect class and is the one responsible to enable us the extraction of the data associated with the skeleton streams. It starts by checking if some skeleton body is detected by the Kinect device, then retrieves for each skeleton, his position, orientation, state, flags and respective joints, and finally writes the skeleton 3D joint coordinates into a log file for later use.

```
1 public void onSkeletonFrameEvent(boolean[] flags, float[] positions, float[]
      orientations, byte[] state) {...}
```

Listing 3.1: Extract skeleton data

The output file format looks something like:

```
BONE |        X        |        Y        |        Z
-----------------------------------------------------
  01 | -0.16948788 | -0.16639492 | 3.4596786
  02 | -0.17006062 | -0.10776921 | 3.5189590
  03 | -0.17665800 |  0.24895142 | 3.5350120
  04 | -0.19132298 |  0.41924775 | 3.4980333
  05 | -0.36282122 |  0.12864287 | 3.5054340
  06 | -0.42800647 | -0.04281822 | 3.4173112
  07 | -0.26290643 | -0.06567054 | 3.2654540
  08 | -0.22328073 | -0.07519613 | 3.2333055
  09 | -0.00193787 |  0.14907743 | 3.5052726
  10 |  0.08717620 | -0.04234611 | 3.4545121
  11 |  0.07774843 | -0.27299222 | 3.4752805
  12 |  0.09540976 | -0.36505340 | 3.4968703
  13 | -0.24931088 | -0.25068653 | 3.4481027
  14 | -0.24318285 | -0.59123003 | 3.4041839
  15 | -0.23263395 | -0.88161606 | 3.3733299
  16 | -0.21569973 | -0.95798370 | 3.3263545
  17 | -0.08691338 | -0.24149096 | 3.4370992
  18 | -0.06345392 | -0.63124865 | 3.4660883
  19 | -0.06795472 | -0.96873450 | 3.5775857
  20 | -0.06833535 | -0.99156760 | 3.5016005
  21 |  0.00000000 |  0.00000000 | 0.0000000
  22 |  0.00000000 |  0.00000000 | 0.0000000
  23 |  0.00000000 |  0.00000000 | 0.0000000
  24 |  0.00000000 |  0.00000000 | 0.0000000
  25 |  0.00000000 |  0.00000000 | 0.0000000
```

Listing 3.2: Output file data format

A list of all twenty-five bones is generated with their corresponding X, Y and Z coordinates, which repeats for the entirety of the movement frames. However, the last five bones can not be tracked and thus they will always be set to the value 0.00000000 on their coordinates.

3.3 Application organization and dataset preparation

In this section will be described the main class of our application responsible for the entire application orchestration, together with an additional class where the dataset files location is specified and handled according to being part of the training set or the final set.

3.3.1 Application orchestration

As in any JAVA application, we had to create a class capable of doing the entire orchestration and coordination of our code, thus the "Application" class was developed which contains a main method responsible for launching our application, one constructor ("Application") and one private method ("loadData"), and two private methods ("readLargerTextFile" and "populateJoints"):

```
[...]

public class Application {

    private ArrayList<double[]> training_inputs;
    private ArrayList<double[]> training_outputs;
    private ArrayList<double[]> classification_inputs;
    private ArrayList<double[]> classification_outputs;

    public Application() {...}

    private void loadData(String data_type) {...}

    public static void main(String[] args) throws IOException {
        Application network = new Application();
    }
}
```

Listing 3.3: Main application class

- 'Application': The class constructor method. Here is specified and initialized the private class variables "training_inputs", "training_outputs", "classification_inputs" and "classification_outputs" respectively by calling the "loadData" method for each situation, plus the "Classification" class initialization and usage by calling its three public methods (the constructor "Classification", the "trainNetwork" and the "finalNetwork" methods). At all times the current operation status is written on the console so the user knows what being done by the application;

- 'loadData': Calls the "ManageData" class in order to prepare a specific dataset (training or final) while orchestrating the calculation of the inputs and outputs for each file of the dataset by using the "CalculateInput" class and saving their values on the class private variables;

- 'main': It is where the JAVA application is started which requires the "IOException" exception in order to rescue and validate what when the application is being started.

3.3.2 Dataset handling

For our application development two datasets were given, each representing the people from a specific eldery healthcare house, in order to handle this situation a class capable of having both datasets file informations was built with the name "ManageData" which contains one constructor ("ManageData"), one public method ("getFilesToUse"), and three private methods ("load", "trainingDataFiles" and "classifyingDataFiles"):

```
[...]

public class ManageData {

    [...]

    public ManageData(String data_type) {...}

    private void load(String[] files) {...}

    private String[] trainingDataFiles() {...}

    private String[] classifyingDataFiles() {...}

    public ArrayList<String[]> getFilesToUse() {...}
}
```

Listing 3.4: Manage data class

- 'ManageData': The class constructor method. Receives the data type as a parameter, the private class variable "files_to_use" is specified and according to the data type parameter calls the "load" while passing the parameter to them;
- 'load': Receives the list of files to be used and their respective information storing it in the "files_to_use" class variable.
- 'trainingDataFiles': Has the list of all files to be used for training purposes together with their associated information;
- 'classifyingDataFiles': Has the list of all files to be used for classification purposes together with their associated information;
- 'getFilesToUse': Returns the class variable "files_to_use";

3.4 Video data analysis and manipulation

With all the important and relevant data saved, it is required to have a mechanism to access it for post preparation and transformation into a usable dataset by our classifier. For such purpose, a principle orchestrating class was developed in order to properly call other relevant classes in their due correct order.

3.4.1 Input calculation

Since there would be lots of iterations in order to read every file of the dataset and process their respective inputs an orchestration class was built for this task. The "CalculateInput" class contains one constructor ("CalculateInput"), one public method ("getInputValues"), and three private methods ("readDataFromFile", "normalizeData", "transformData"):

```
1  [...]
2
3  public class CalculateInput {
4
5      [...]
6
7      public CalculateInput(String[] file_to_use) {...}
8
9      private void readDataFromFile() {...}
10
11     private void normalizeData() {...}
12
13     private void transformData() {...}
14
15     public double[] getInputValues(){...}
16 }
```

Listing 3.5: Calculate input class

- 'CalculateInput': The class constructor method. Receives the file to use information as a parameter, the private class variables "file", "time", "joints" and "normalized_joints" are specified and initialized, and then the three private methods are called in their respective order ("readDataFromFile", "normalizeData" and "transformData");
- 'readDataFromFile': Calls the "ReadTextFile" class passing the class variable "file" as a parameter. Then calls its class method "getJoints" in order to set the class variable "joints".
- 'normalizeData': Calls the "NormalizeData" class passing the class variables "joints" and time" as parameters. Then calls its class method "getNormalization" in order to set the class variable "normalized_joints".
- 'transformData': Calls the "TransformData" class passing the class variables "normalized_joints" and "time" as parameters. Then calls its respective class method "getTransformation" in order to set the class variable "input_values".
- 'getInputValues': Returns the class variable "input_values";

3.4.2 Accessing the stored data

In order to read the previously stored data the class "ReadTextFile" was developed which contains one constructor ("ReadTextFile"), one public method ("getJoints"), and two private methods ("readLargerTextFile" and "populateJoints"):

```
[...]

public class ReadTextFile {

    [...]

    public ReadTextFile(String pathToFile) throws IOException {...}

    private void readLargerTextFile() throws IOException {...}

    private void populateJoints() {...}

    public ArrayList<ArrayList> getJoints() {...}
}
```

Listing 3.6: Read text file class

- 'ReadTextFile': The class constructor method. Here is specified the path to the file, its encoding and the class variables "joints and "coords" are initialized, finishing by calling the 'readLargerTextFile' method;
- 'readLargerTextFile': Reads the specified file storing every X, Y and Z coordinates in the "coords" class variable, lastly calling the "populateJoints" method;
- 'populateJoints': Every X, Y and Z coordinates of all twenty-five bones for each frame are stored in the "joints" variable;
- 'getJoints': A get method which returns the "joints" class variable value.

3.4.3 Data normalization

Having the stored data now loaded into the application it is necessary to verify and normalize it if necessary. This is done by the class "NormalizeData" which contains one constructor ("NormalizeData"), one public method ("getNormalization"), and five private methods ("normalize", "adaptToBoundaries", "findCenter", "calculateSpeed", "calculateAcc" and "checkRepeatedData"):

```
[...]

public class NormalizeData {

    [...]

    public NormalizeData(ArrayList<ArrayList> joints, double time){...}

    private void normalize(){...}
```

```
10
11      private void adaptToBoundaries(){...}
12
13      private Double[] findCenter(ArrayList<String[]> jointList){...}
14
15      private Double calculateSpeed(Double[] Cmi, Double[] Cmf, double
        time_interval){...}
16
17      private Double calculateAcc(Double Va, Double Vb, double Ta, double Tb)
        {...}
18
19      private Boolean checkRepeatedData(int i, Double[] next_data) {...}
20
21      public ArrayList<Double[]> getNormalization(){...}
22  }
```

Listing 3.7: Normalize data class

- 'NormalizeData': The class constructor method. Here is specified the "joints" and "time" variables, the "normalizedJoints" variable is initialized, and the private methods "normalize" and "adaptToBoundaries" are called;
- 'normalize': Its the principal method of the class which calls every other private method in a given order together with additional calculations according to the specific frame in the movement;
- 'adaptToBoundaries': The "normalizedJoints" values are adapted to specific boundaries;
- 'findCenter': "Finds the mass center XYZ coordinates of the skeleton for each frame of the movement;
- 'calculateSpeed': Receives the time interval and the XYZ coordinates between two given frames returning the speed calculated through a mathematic formula;
- 'calculateAcc': Receives the speed, and time of two given frames returning the accelation calculated through a mathematic formula;
- 'checkRepeatedData': Verifies if two followed frames have the same values allowing only those who do not;
- 'getNormalization': A get method which returns the "normalizedJoints" class variable value.

3.4.4 Data transformation

Now that all data has been verified and normalized there is only one final step before sending it to for classification which is the transformation phase. The class "TransformData" is the one responsible for this task which contains one constructor ("TransformData"), one public method ("getTransformation"), and four private methods ("firstClassification", "secondClassification", "thirdClassification" and "fourthClassification"):

```
1  [...]
2
```

```
3  public class TransformData {
4
5      [...]
6
7      public TransformData(ArrayList<Double[]> normalizedJoints, double time)
       {...}
8
9      private void firstClassification() {...}
10
11     private void secondClassification() {...}
12
13     private void thirdClassification() {...}
14
15     private void fourthClassification() {...}
16
17     public double[] getTransformation() {...}
18 }
```

Listing 3.8: Transform data class

- 'TransformData': The class constructor method. Here is specified the "normalizedjoints" and "time" variables, the "transformedData" variable is initialized, and the private methods "firstClassification", "secondClassification", "thirdClassification" and "fourthClassification" are called;
- 'firstClassification': Calculates the acceleration differences counting the number of positives and negatives, returning '1.0' if there are more positives or '0.0' otherwise;
- 'secondClassification': Verifies how many times the speed is superior to the constant value "SPEED" counting the number of positives and negatives, returning '1.0' if there are more positives or '0.0' otherwise;
- 'thirdClassification': Verifies how many times the distance is superior to the constant value "SPACE" counting the number of positives and negatives, returning '1.0' if there are more positives or '0.0' otherwise.
- 'fourthClassification': Verifies the total time of the movement returning 1.0 if greater than twenty seconds, '0.0' in case its between ten seconds (excluded) and twenty seconds (included), or '-1.0' if inferior to 10 seconds (included);
- 'getTransformation': Returns the transformed data array.

3.5 Data classification

The last section of the development process is related to the neural network itself, being one of the most important section of this dissertation. It can be subdivided into two parts: one being the class responsible for calling and handling the neural network implementation, and the other is the three classes which represent the neural network itself.

3.5.1 Handling the neural network

After every dataset has been properly loaded, prepared, normalized and transformed it is now available to be used for training and finally classification purposes. An orchestration class "Classification" was build in order to ease how the dataset should be handled by our neural network implementation, containing one constructor ("Classification") and two public methods ("trainNetwork" and "finalNetwork"):

```
1  [...]
2
3  public class Classification {
4
5      private static final int INPUT_LAYER  = 4;
6      private static final int HIDDEN_LAYER = 3;
7      private static final int OUTPUT_LAYER = 3;
8      private static final int ITERATIONS = 1000000;
9      private Network net;
10     private TrainSet training;
11     private TrainSet classify;
12
13     public Classification(){...}
14
15     public void trainNetwork(ArrayList<double[]> input, ArrayList<double[]>
       output){...}
16
17     public void finalNetwork(ArrayList<double[]> input, ArrayList<double[]>
       output){...}
18 }
```

Listing 3.9: Classification orchestration class

- 'Classification': The class constructor method. Here is initialized the private class variable "net" as an object of the class "Network" using an input layer of four neurons, two hidden layers of three neurons and one output layer with three neurons, "additionally two more private variables (training and classify) are specified as objects of the the class "TrainSet" using four as the input size value and three as the output size value;
- 'trainNetwork': Receives a list of inputs and their corresponding outputs that will be added as data for training purposes and later being used to train the network over a set of one million iterations;
- 'finalNetwork': Receives a list of inputs and their corresponding outputs that will be added as data for classification purposes and later the neural network is asked to classify this inputs and obtain their outputs.

3.5.2 Neural network implementation

The implementation of our neural network [26] [27] was divided into three classes, one being the primary one which has the main functions related to the network, another works as a set of tool methods which can be called as helpers to the main class and the last one is intended to enable the training or classification capabilities of the network on a particular or multiple datasets, it also allows the user to use only a particular part of the dataset instead of the entirety of it (this is called "batch" selection). We will describe only the main Neural Network [28] [29] [30] [31] class since the other two classes are composed of simple helper methods and override for native java methods such as "toString" to better suit our needs in terms of console display, their code is however available in the appendix. The main Neural Network is named "Network" which contains one constructor ("Network"), two public methods ("calculate" and "train") and five private methods ("MSE", "MSE", "train", "backPropagationError", and "updateWeights");

```
[...]

public class Network {
    private final int[] NETWORK_LAYER_SIZES;
    private final int NETWORK_SIZE;
    private final int INPUT_SIZE;
    private final int OUTPUT_SIZE;
    private double[][] output;
    private double[][][][] weights;
    private double[][] bias;
    private double[][] error_signal;
    private double[][] output_derivative;

    public Network(int... NETWORK_LAYER_SIZES) {...}

    public double[] calculate(double... input) {...}

    public void train(TrainSet set, int loops, int batch_size) {...}

    private double MSE(TrainSet set){...}

    private double MSE(double[] input, double[] target) {...}

    private void train(double[] input, double[] target, double eta) {...}

    private void backPropagationError(double[] target) {...}

    private void updateWeights(double eta) {...}
}
```

Listing 3.10: Classification orchestration class

- 'Network': The class constructor method. Receives the network layers disposition and respective sizes as a parameter which is used to initialize the private class variable "NETWORK_LAYER_SIZES". Additionally, the other private class variables are also initialized

("NETWORK_SIZE", "INPUT_SIZE", "OUTPUT_SIZE", "output", "weights", "bias", "error_signal" and "output_derivative");

- 'calculate': Queries the network to calculate the output for a given input that is passed as a parameter;
- 'train 1': Trains the network using a specific dataset, number of loops and enables the user to specify how many entries of the dataset should be used using the variable "batch_size" calling the other method "train" by passing the inputs and outputs of the batch;
- 'MSE 1': Allows the calculation of the mathematics error operation 'Mean Square Error' for a given dataset. The calculations are made by calling the other "MSE" method by passing the set inputs and outputs;
- 'MSE 2': Calculates the 'Mean Square Error' using the inputs and outputs passed as parameters and calling the "calculate" method on the inputs;
- 'train 2': Receives a list of inputs and targets as well as the learning rate as parameters which will be used to call the "calculate" method passing the inputs as a parameter, the "backPropagationError" method passing the targets as a parameter and finally the "updateWeights" method passing the learning rate as a parameter;
- 'backPropagationError': Receives the intended output values as a parameter and then applies a mathematic formula to calculate the back propagation error firstly on the output layers and lastly to the hidden layers.
- 'updateWeights': Receives the learning rate value as a parameter which is used to update the weights of every neuron existing on the network.

3.6 Conclusion

Previously were presented the main classes that were created during our thesis development phase. All of them play their part while complementing each other, plus they were build having mind the possibility of code re-usage and/or easy adaption for other types of projects other than the one of this dissertation. The best example of this is the Neural Network related classes which all have their one main method filled with code capable of testing them as a stand-alone project, which was proven quite useful during the development and debugging phase. All this work, techniques and precautions combined enabled us to achieve our primary objectives for this thesis.

Chapter 4

Results and Conclusions

4.1 Results

On this section will be discussed the application obtained results accompanied by images, graphics, and their respective textual explanation. In order to evaluate the success of our application a total of ten individual and independent runs were undertaken saving the console outputs for each every one of them (the ten console output images are presented in the Appendix section A.3 of this dissertation and go by figures A.1, A.2, A.3, A.4, A.5, A.6, A.7, A.8, A.9 and A.10,). An example of this outputs is shown in the figure 4.1.

```
Loading training DATASET...
Training DATASET successfully loaded !!

Loading classification DATASET...
Classification DATASET successfully loaded !!

Starting the classification training phase...
Mean error value over 1000000: 0.039239559035159195
The classification training phase has completed successfully !!

Starting the classification final phase...
Input: [0.0, 1.0, 0.0, 1.0] => Output: [1.794148235533662E-5, 4.280020476685341E-6, 0.9999957182462021] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [1.3759291124407587E-5, 6.009072274830778E-6, 0.9999939886513286] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [1.3759291124407587E-5, 6.009072274830778E-6, 0.9999939886513286] => Expected: [0.0, 0.0, 1.0]
Input: [1.0, 1.0, 1.0, 0.0] => Output: [6.56112918920319E-5, 0.009292421543713594, 0.9907075767943201] => Expected: [0.0, 0.0, 1.0]

Process finished with exit code 0
```

Figure 4.1: Console Output 1

The console outputs indicate every step that was made by our application when it starts and finishes the loading and preparation phase for both the training and the classification dataset. It also indicates when the neural network training phase is initiated and once it ends the respective mean square error obtained, and lastly the classification phase together with its results (containing the input array, the output array, and the expected/target array).

From the console outputs, we can extract two important information: the mean square error value that was calculated over all the iterations during the network training phase, which can be evaluated in the graph 4.2 that demonstrates the calculated values for each of the ten runs and its variation between them; the other one being the final results of the four selected patients classification which were all successfully classified into their respective conditions, as shown in tables 4.1, 4.2 and 4.3.

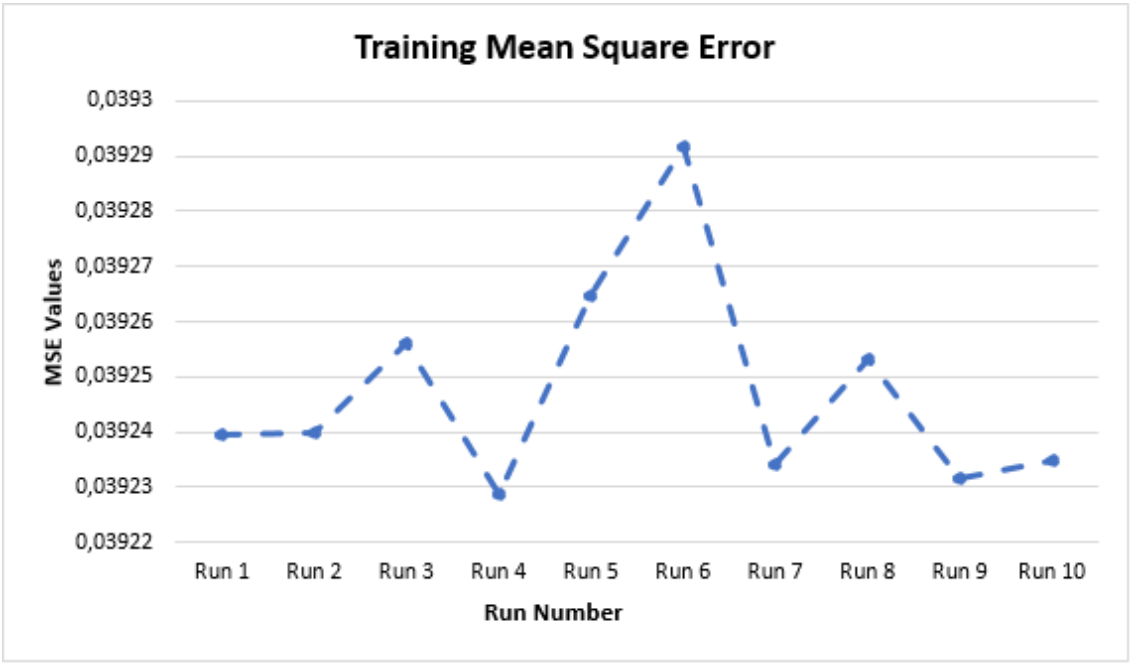

Figure 4.2: Training phase mean square error

Run	Person	Input	Output	Target	Assertion
1	1	[0.0, 1.0, 0.0, 1.0]	[1.794148235533662E-5, 4.280020476685341E-6, 0.9999957182462021]	[0.0, 0.0, 1.0]	Correct
	2	[0.0, 1.0, 0.0, 1.0]	[1.3759291124407587E-5, 6.009072274830778E-6, 0.9999939886513286]	[0.0, 0.0, 1.0]	Correct
	3	[0.0, 1.0, 0.0, 1.0]	[1.3759291124407587E-5, 6.009072274830778E-6, 0.9999939886513286]	[0.0, 0.0, 1.0]	Correct
	4	[0.0, 1.0, 0.0, 1.0]	[6.56112918920319E-5, 0.009292421543713594, 0.9907075767943201]	[0.0, 0.0, 1.0]	Correct
2	1	[0.0, 1.0, 0.0, 1.0]	[2.3166281254593225E-5, 3.0382534894761605E-4, 0.9996961747991125]	[0.0, 0.0, 1.0]	Correct
	2	[0.0, 1.0, 0.0, 1.0]	[1.3374175035206572E-5, 1.1040331924450852E-4, 0.9998895967396797]	[0.0, 0.0, 1.0]	Correct
	3	[0.0, 1.0, 0.0, 1.0]	[1.3374175035206572E-5, 1.1040331924450852E-4, 0.9998895967396797]	[0.0, 0.0, 1.0]	Correct
	4	[0.0, 1.0, 0.0, 1.0]	[4.019351101598086E-5, 0.0028737458916524576, 0.9971262548825077]	[0.0, 0.0, 1.0]	Correct

Table 4.1: Results comparison - Run one and two

Run	Person	Input	Output	Target	Assertion
3	1	[0.0, 1.0, 0.0, 1.0]	[8.692397147286361E-6, 2.250562459199995E-4, 0.9997749439334834]	[0.0, 0.0, 1.0]	Correct
	2	[0.0, 1.0, 0.0, 1.0]	[4.151340806187727E-6, 3.796890184426026E-5, 0.9999620311384434]	[0.0, 0.0, 1.0]	Correct
	3	[0.0, 1.0, 0.0, 1.0]	[4.151340806187727E-6, 3.796890184426026E-5, 0.9999620311384434]	[0.0, 0.0, 1.0]	Correct
	4	[0.0, 1.0, 0.0, 1.0]	[1.8771353518355388E-5, 0.0029391080604733726, 0.9970608929325334]	[0.0, 0.0, 1.0]	Correct
4	1	[0.0, 1.0, 0.0, 1.0]	[1.0528256978821108E-4, 1.386397501830746E-4, 0.9998613602525785]	[0.0, 0.0, 1.0]	Correct
	2	[0.0, 1.0, 0.0, 1.0]	[4.2687069614608084E-5, 8.059852162339667E-6, 0.9999919401481449]	[0.0, 0.0, 1.0]	Correct
	3	[0.0, 1.0, 0.0, 1.0]	[4.2687069614608084E-5, 8.059852162339667E-6, 0.9999919401481449]	[0.0, 0.0, 1.0]	Correct
	4	[0.0, 1.0, 0.0, 1.0]	[4.4180816967724336E-4, 0.013254717914690052, 0.9867452820852]	[0.0, 0.0, 1.0]	Correct
5	1	[0.0, 1.0, 0.0, 1.0]	[1.802401899237266E-4, 5.948085866977581E-4, 0.9994051914133022]	[0.0, 0.0, 1.0]	Correct
	2	[0.0, 1.0, 0.0, 1.0]	[1.8046936236059132E-4, 5.962372003881008E-4, 0.999403762799612]	[0.0, 0.0, 1.0]	Correct
	3	[0.0, 1.0, 0.0, 1.0]	[1.8046936236059132E-4, 5.962372003881008E-4, 0.999403762799612]	[0.0, 0.0, 1.0]	Correct
	4	[0.0, 1.0, 0.0, 1.0]	[1.5941448011733234E-4, 0.00367231086345424, 0.9963276891365457]	[0.0, 0.0, 1.0]	Correct
6	1	[0.0, 1.0, 0.0, 1.0]	[1.9597360963490593E-5, 0.0056902525034688315, 0.9943097474966119]	[0.0, 0.0, 1.0]	Correct
	2	[0.0, 1.0, 0.0, 1.0]	[3.2384962350125657E-6, 3.792436367752014E-6, 0.999996207563631]	[0.0, 0.0, 1.0]	Correct
	3	[0.0, 1.0, 0.0, 1.0]	[3.2384962350125657E-6, 3.792436367752014E-6, 0.999996207563631]	[0.0, 0.0, 1.0]	Correct
	4	[0.0, 1.0, 0.0, 1.0]	[2.5081793701433034E-5, 0.004425233128981999, 0.9955747668708996]	[0.0, 0.0, 1.0]	Correct

Table 4.2: Results comparison - Run three to six

Run	Person	Input	Output	Target	Assertion
7	1	[0.0, 1.0, 0.0, 1.0]	[3.571118091843838E-5, 1.0199549618928155E-4, 0.9998980030829162]	[0.0, 0.0, 1.0]	Correct
	2	[0.0, 1.0, 0.0, 1.0]	[2.8681839679936332E-5, 3.690699647888109E-5, 0.9999630924082283]	[0.0, 0.0, 1.0]	Correct
	3	[0.0, 1.0, 0.0, 1.0]	[2.8681839679936332E-5, 3.690699647888109E-5, 0.9999630924082283]	[0.0, 0.0, 1.0]	Correct
	4	[0.0, 1.0, 0.0, 1.0]	[6.974345519730311E-5, 0.009081429185186346, 0.9909185705607539]	[0.0, 0.0, 1.0]	Correct
8	1	[0.0, 1.0, 0.0, 1.0]	[1.8491360307328446E-4, 9.43060679972735E-4, 0.9990571125906482]	[0.0, 0.0, 1.0]	Correct
	2	[0.0, 1.0, 0.0, 1.0]	[5.231996414149557E-5, 2.3941579209086824E-5, 0.9999760645370442]	[0.0, 0.0, 1.0]	Correct
	3	[0.0, 1.0, 0.0, 1.0]	[5.231996414149557E-5, 2.3941579209086824E-5, 0.9999760645370442]	[0.0, 0.0, 1.0]	Correct
	4	[0.0, 1.0, 0.0, 1.0]	[1.55518442951638E-5, 0.007501954128141977, 0.9924980398583678]	[0.0, 0.0, 1.0]	Correct
9	1	[0.0, 1.0, 0.0, 1.0]	[1.0240703108915565E-5, 6.927109864848383E-6, 0.9999930786973508]	[0.0, 0.0, 1.0]	Correct
	2	[0.0, 1.0, 0.0, 1.0]	[1.035661965675549E-5, 7.162071114991674E-6, 0.9999928439069675]	[0.0, 0.0, 1.0]	Correct
	3	[0.0, 1.0, 0.0, 1.0]	[1.035661965675549E-5, 7.162071114991674E-6, 0.9999928439069675]	[0.0, 0.0, 1.0]	Correct
	4	[0.0, 1.0, 0.0, 1.0]	[1.1727227063032475E-4, 0.004302662952708057, 0.9956980814324169]	[0.0, 0.0, 1.0]	Correct
10	1	[0.0, 1.0, 0.0, 1.0]	[1.0111030104191426E-5, 5.0977333469343E-5, 0.9999490226932375]	[0.0, 0.0, 1.0]	Correct
	2	[0.0, 1.0, 0.0, 1.0]	[1.0069670183713392E-5, 5.014044420395501E-5, 0.9999498595821419]	[0.0, 0.0, 1.0]	Correct
	3	[0.0, 1.0, 0.0, 1.0]	[1.0069670183713392E-5, 5.014044420395501E-5, 0.9999498595821419]	[0.0, 0.0, 1.0]	Correct
	4	[0.0, 1.0, 0.0, 1.0]	[3.836843349483E-5, 0.0034623196198481617, 0.9965376810009974]	[0.0, 0.0, 1.0]	Correct

Table 4.3: Results comparison - Run seven to ten

The tables 4.1, 4.2 and 4.3 have 6 columns which are structured in the following way:

- The 'Run' column displays the run number which goes from one to ten;
- The 'Person' column represents the person associated with the test or in other words the log file of the classification dataset that was used;
- The 'Input' column indicates the input values obtained from the 'TransformData' class represented as an array of four entry values(remembering from the previous chapter, the only possible values are 1.0 or 0.0, the first entry is correlated with the acceleration positive/negative calculation, the second entry is correlated with how many times the speed is superior to a specific constant value, the third entry is correlated with distance difference between two followed frames, and the last entry is correlated with the total movement time);
- The 'Output' column has the output values calculated by the Neural Network in an array of three entries (their values can variate between 0.0 and 1.0 with multiple decimals cases);
- The 'Target' column has the expected result or target output (the only possible values are 1.0 or 0.0, whereas the first entry represents a low mobility deficit, the second entry a medium mobility deficit and the last one a high mobility deficit);
- The 'Assertion' column states if the neural network achieved a successful classification or not.

4.2 Conclusion

Having closer look at the values present on the previous tables. For every run and person, the result was always a successful classification. Mathematically if the array entry values from the 'Output' column are rounded according to being nearer to 1.0 or 0.0, then we would achieve a result array exactly equal to the one in the 'Target' column.

Having this mind the value of the last column 'Assertion' in each of the three tables was set to 'Correct', which means we achieved the best case scenario where our application is doing exactly what it was supposed to do: a correct patient classification according to their mobility difficulties when comparing with previous valid results for the same patients obtained by different methodologies other than Kinect.

Chapter 5

Final Considerations

5.1 Difficulties and future work

During the development of our application, we encountered a lot of problems, such as incorrect/corrupt Kinect video examples, the skeleton joints detection, skeleton isolation (in case of 2 skeletons being present), store their coordinates, mathematics calculations to obtain the instant velocity, acceleration, and distance between two frames and the neural network development.

We managed to overcome said problems with the exception of two particular ones, more specifically the Kinect video problems and the skeleton isolation. In terms of the first one, unfortunately, there was nothing we can do about it since some of the videos had nothing recorded or were incomplete and therefore we had to remove them from our usable datasets (we lost a total of nine videos because of this problem) and one about the existence of two skeletons in the video represented a problem that we could not overcome, there was movement on both skeletons at all times and therefore we could not differentiate between which was the correct one that represented the patient (we lost a total of seven videos because of this problem).
In terms of future work it is important to note the following three cases:

- The most obvious one would be to find a solution for the two skeleton problem;
- Even though we had a classification of type 'Low' none of our videos patients classified for that and so it could be advantageous if future video captures could bring examples of persons within said characteristic for a complete classification experience;
- Create some sort of visual friendly user interface instead of using the console as the standard output of the application;
- And the last one would be to record more videos from different elderly health-care houses in order to collect more data that can be used as different datasets and therefore enhance and further prove our application classification capabilities.

5.2 Conclusions

According to the obtained results together with the mean square error graph, one can easily and safely assume that the overall application performance is amazing and working goes according to our initial objectives and aims.

Therefore it has been proven just like in previous works demonstrated in the State the Art chapter that the Kinect device is very capable and excellent tool to be used in health-care environment applications and if combined with some artificial intelligence and machine learning techniques it has without a doubt a bright future ahead of it.

Our application is hereby approved as a perfect example of this last statement and can be used to evaluate and help patients on other elderly health-care houses aside from the local ones associated with our datasets.

Bibliography

[1] Kinect. https://en.wikipedia.org/wiki/Kinect. Last checked: 21 May 2017. 25, 30

[2] Java for kinect librabry. http://research.dwi.ufl.edu/ufdw/index.php. Last checked: 21 May 2015. 25, 30

[3] Ennio Mingolla Jonathan Chey, Stephen Grossberg. Neural dynamics of motion processing and speed discrimination. Vision Research, 38(18):2769–2786, 1998. 25

[4] World report on disability. http://www.who.int/disabilities/worldreport/2011/accessibleen.pdf. Last checked: 12 August 2017. 29

[5] World report on disability. https://www.unicef.org/protection/Worldreportondisabilityeng.pdf. Last checked: 12 August 2017. 29

[6] Smart-dx. http://www.btsbioengineering.com/products/smart-dx/. Last checked: 21 September 2017. 29

[7] Optoelectronic plethysmography system. http://www.btsbioengineering.com/products/bts-oep-system/. Last checked: 21 September 2017. 29

[8] Optitrack. http://optitrack.com/. Last checked: 21 September 2017. 29

[9] xsens mvn. https://www.xsens.com/products/xsens-mvn/. Last checked: 21 September 2017. 29

[10] Vicon. https://www.vicon.com/. Last checked: 21 September 2017. 29

[11] Autodesk maya. https://knowledge.autodesk.com/support/maya. Last checked: 21 September 2017. 29

[12] Kinect hardware. https://developer.microsoft.com/en-us/windows/kinect/hardware. Last checked: 19 September 2017. 30

[13] Kinect software development kit 2.0. https://www.microsoft.com/en-us/download/details.aspx?id=44561. Last checked: 19 September 2017. 30

[14] Mi Zhang Sebastian Koenig Phil Requejo Noom Somboon Alexander A. Sawchuk Albert A. Rizzo Chien-Yen Chang, Belinda Lange. Towards pervasive physical rehabilitation using microsoft kinect. Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2012 6th International Conference on, page 4, 2012. 30

[15] Sonali Dubey Anil K. Roy, Yash Soni. Enhancing effectiveness of motor rehabilitation using kinect motion sensing technology. Global Humanitarian Technology Conference: South Asia Satellite (GHTC-SAS), 2013 IEEE, page 7, 2013. 30, 31

[16] Roanna Lun Deborah D. Espy M. Ann Reinthal Wenbing Zhao, Hai Feng. A kinect-based rehabilitation exercise monitoring and guidance system. Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on, page 4, 2014. 30, 31

[17] Jessica Coulson Yoann Barnouin Jamie S. McPhee Daniel Leightley, Moi Hoon Yap. Benchmarking human motion analysis using kinect one: An open source dataset. Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2015 Asia-Pacific, page 7, 2015. 30, 32

[18] Amit Konar Pavia Bera, Reshma Kar. Joint pain detection by gait analysis for elderly healthcare. Research in Computational Intelligence and Communication Networks (ICRCICN), 2015 IEEE International Conference on, page 5, 2015. 30, 32

[19] Wenbing Zhao Roanna Lun, Connor Gordon. Tracking the activities of daily lives: An integrated approach. Future Technologies Conference (FTC), page 10, 2016. 30, 33

[20] A Kinect based system for cognitive rehabilitation exercises monitoring. A kinect-based system for cognitive rehabilitation exercises monitoring. Computer Methods and Programs in Biomedicine, page 12, 2014. 30, 33

[21] Jun-Da Huang Yao-Jen Chang, Shu-Fang Chen. A kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities. Research in Developmental Disabilities, page 5, 2011. 30, 33

[22] Florica Moldoveanu Alin Moldoveanu Hesham Alabbasi, Alex Gradinaru. Human motion tracking and evaluation using kinect v2 sensor. E-Health and Bioengineering Conference (EHB), 2015, page 4, 2015. 30, 31

[23] Amit Konar Monalisa Pal, Sriparna Saha. Distance matching based gesture recognition for healthcare using microsoft's kinect sensor. Microelectronics, Computing and Communications (MicroCom), 2016 International Conference on, page 6, 2016. 30, 32

[24] Sanjay Mehrotra Liang Liu. Patient walk detection in hospital room using microsoft kinect v2. Engineering in Medicine and Biology Society (EMBC), 2016 IEEE 38th Annual International Conference of the, page 4, 2016. 30, 32

[25] Jose Antonio Piedra-Fernandez César Bernal Bravo, Juan Jesus Ojeda-Castelo. Art activities with kinect to students with cognitive disabilities: Improving all motor skills. Procedia - Social and Behavioral Sciences, page 4, 2016. 30, 34

[26] University of Belgrade an experiment for Intelligent Systems course. Tijana Jovanovic, Faculty of Organisation Sciences. Neuroph and multilayer perceptron. http://neuroph.sourceforge.net/index.html. Last checked: 13 January 2017. 50

[27] Sigmoid activation function. https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Activation_Functions. Last checked: 13 January 2017. 50

[28] 15 steps to implement a neural net. http://code-spot.co.za/2009/10/08/15-steps-to-implemented-a-neural-net/. Last checked: 23 September 2017. 50

[29] An introduction to neural networks in java. http://www.informit.com/articles/article.aspx?p=30596. Last checked: 23 September 2017. 50

[30] How to build a neural network. https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/. Last checked: 23 September 2017. 50

[31] Neural networks tutorial. https://www.youtube.com/channel/UCaKAU8vQzS-_e5xt7NSK3Xw/videos. Last checked: 23 September 2017. 50

Appendix A

Attachments

This appendix includes all the JAVA functions that were used directly in the development of the final application as well as some auxiliary functions used exclusively for testing purposes.

A.1 JAVA Kinect Functions

Below are written the entire code of each JAVA function used in order to communicate with the Kinect device and extract the skeleton data together with its respective joints.

A.1.1 Kinect Class

```
1  package j4kdemo.kinectviewerapp;
2  import javax.swing.JLabel;
3  import edu.ufl.digitalworlds.j4k.DepthMap;
4  import edu.ufl.digitalworlds.j4k.J4KSDK;
5  import edu.ufl.digitalworlds.j4k.Skeleton;
6  import java.io.File;
7  import java.io.FileOutputStream;
8  import java.io.PrintWriter;
9
10 /*
11  * Copyright 2011-2014, Digital Worlds Institute, University of
12  * Florida, Angelos Barmpoutis.
13  * All rights reserved.
14  *
15  * When this program is used for academic or research purposes,
16  * please cite the following article that introduced this Java library:
17  *
18  * A. Barmpoutis. "Tensor Body: Real-time Reconstruction of the Human Body
19  * and Avatar Synthesis from RGB-D', IEEE Transactions on Cybernetics,
20  * October 2013, Vol. 43(5), Pages: 1347-1356.
21  *
22  * Redistribution and use in source and binary forms, with or without
23  * modification, are permitted provided that the following conditions are
24  * met:
25  *     * Redistributions of source code must retain this copyright
26  * notice, this list of conditions and the following disclaimer.
27  *     * Redistributions in binary form must reproduce this
28  * copyright notice, this list of conditions and the following disclaimer
29  * in the documentation and/or other materials provided with the
30  * distribution.
```

```
31  *
32  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
33  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
34  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
35  * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
36  * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
37  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
38  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
39  * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
40  * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
41  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
42  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
43  */
44
45 public class Kinect extends J4KSDK{
46
47   ViewerPanel3D viewer=null;
48   JLabel label=null;
49   boolean mask_players=false;
50   public void maskPlayers(boolean flag){mask_players=flag;}
51
52   public Kinect()
53   {
54     super();
55   }
56
57   public Kinect(byte type)
58   {
59     super(type);
60   }
61
62   public void setViewer(ViewerPanel3D viewer){this.viewer=viewer;}
63
64   public void setLabel(JLabel l){this.label=l;}
65
66   private boolean use_infrared=false;
67
68   public void updateTextureUsingInfrared(boolean flag)
69   {
70     use_infrared=flag;
71   }
72
73   @Override
74   public void onDepthFrameEvent(short[] depth_frame, byte[] player_index,
       float[] XYZ, float[] UV) {...}
75
76   @Override
```

```
77   public void onSkeletonFrameEvent(boolean[] flags, float[] positions, float
       [] orientations, byte[] state) {...}
78
79   @Override
80   public void onColorFrameEvent(byte[] data) {...}
81
82   @Override
83   public void onInfraredFrameEvent(short[] data) {...}
84 }
```

A.1.2 Kinect method onDepthFrameEvent

```
1  public void onDepthFrameEvent(short[] depth_frame, byte[] player_index,
       float[] XYZ, float[] UV) {
2
3    if(viewer==null || label==null)return;
4    float a[]=getAccelerometerReading();
5    label.setText(((int)(a[0]*100)/100f)+","+((int)(a[1]*100)/100f)+","+((int)
       (a[2]*100)/100f));
6    DepthMap map=new DepthMap(getDepthWidth(),getDepthHeight(),XYZ);
7
8    map.setMaximumAllowedDeltaZ(0.5);
9
10   if(UV!=null && !use_infrared) map.setUV(UV);
11   else if(use_infrared) map.setUVuniform();
12   if(mask_players)
13   {
14     map.setPlayerIndex(depth_frame, player_index);
15     map.maskPlayers();
16   }
17   viewer.map=map;
18 }
```

A.1.3 Kinect method onSkeletonFrameEvent

```
1  public void onSkeletonFrameEvent(boolean[] flags, float[] positions, float[]
       orientations, byte[] state) {
2    if(viewer==null || viewer.skeletons==null)return;
3
4    for(int i=0;i<getSkeletonCountLimit();i++)
5    {
6      viewer.skeletons[i]=Skeleton.getSkeleton(i, flags,positions,
       orientations,state,this);
7      byte[] getJointTrackingStates = viewer.skeletons[i].
       getJointTrackingStates();
8    }
9    try{
10     PrintWriter pw = new PrintWriter(new FileOutputStream(
```

```
        new File("C:\\Users\\Pedro\\Google Drive\\teste.log"), true));
    for(int j = 0; j < 25; j++) {
      float x = viewer.skeletons[0].get3DJointX(j);
      float y = viewer.skeletons[0].get3DJointY(j);
      float z = viewer.skeletons[0].get3DJointZ(j);
      pw.print(j + " | " + x + " | " + y + " | " + z + "\n");
    }
    pw.print("\n");
    pw.close();
  }catch (Exception e) {
    System.out.print("ERROR WRITING TO FILE !!");
  }
}
```

A.1.4 Kinect method onColorFrameEvent

```
public void onColorFrameEvent(byte[] data) {
  if(viewer==null || viewer.videoTexture==null || use_infrared) return;
  viewer.videoTexture.update(getColorWidth(), getColorHeight(), data);
}
```

A.1.5 Kinect method onInfraredFrameEvent

```
public void onInfraredFrameEvent(short[] data) {
  if(viewer==null || viewer.videoTexture==null || !use_infrared) return;
  int sz = getInfraredWidth()*getInfraredHeight();
  byte bgra[] = new byte[sz*4];
  int i, idx = 0, iv = 0;
  short sv = 0;
  byte bv = 0;

  for(i = 0; i < sz; i++)
  {
    sv = data[i];
    iv = sv >= 0 ? sv : 0x10000 + sv;
    bv = (byte)( (iv & 0xfff8)>>6);
    bgra[idx] = bv;
    bgra[idx] = bv;
    bgra[idx] = bv;
    bgra[idx] = 0;
    idx = idx + 4;
  }

  viewer.videoTexture.update(getInfraredWidth(), getInfraredHeight(), bgra);
}
```

A.2 Application Functions

Below are written the entire code of each JAVA function written on our application, composed of the following classes: "Application", "ManageData", "CalculateInput", "ReadTextFile", "NormalizeData", "TransformData", "Classification", "Network", "TrainSet" and "NetworkTools".

A.2.1 Application Class

```
1  package ubi.allab.application;
2
3  import java.io.IOException;
4  import java.util.ArrayList;
5
6  public class Application {
7      private ArrayList<double[]> training_inputs;
8      private ArrayList<double[]> training_outputs;
9      private ArrayList<double[]> classification_inputs;
10     private ArrayList<double[]> classification_outputs;
11
12     public Application() {
13         training_inputs = new ArrayList<double[]>();
14         training_outputs = new ArrayList<double[]>();
15         System.out.println("Loading training DATASET...");
16         loadData("training");
17         System.out.println("Training DATASET successfully loaded !!\n");
18
19         classification_inputs = new ArrayList<double[]>();
20         classification_outputs = new ArrayList<double[]>();
21         System.out.println("Loading classification DATASET...");
22         loadData("classification");
23         System.out.println("Classification DATASET successfully loaded !!\n"
       );
24
25         //Start the classification phase
26         Classification results = new Classification();
27
28         //Training phase
29         System.out.println("Starting the classification training phase...");
30         results.trainNetwork(training_inputs, training_outputs);
31         System.out.println("The classification training phase has completed
       successfully !!\n");
32
33         //Final classification
34         System.out.println("Starting the classification final phase...");
35         results.finalNetwork(classification_inputs, classification_outputs);
36     }
37
```

```
38      private void loadData(String data_type) {
39          int i;
40          double file_inputs[], file_outputs[];
41          String[] file_info;
42          ArrayList<String[]> files_to_use;
43
44          ManageData files = new ManageData(data_type);
45          files_to_use = files.getFilesToUse();
46
47          for (i = 0; i < files_to_use.size(); i++) {
48              file_info = files_to_use.get(i);
49              CalculateInput file = new CalculateInput(file_info);
50              file_inputs = file.getInputValues();
51
52              switch (file_info[2]) {
53                  case "low":
54                      file_outputs = new double[]{1.0, 0.0, 0.0};
55                      break;
56                  case "medium":
57                      file_outputs = new double[]{0.0, 1.0, 0.0};
58                      break;
59                  default:
60                      file_outputs = new double[]{0.0, 0.0, 1.0};
61                      break;
62              }
63
64              if (data_type.equals("training")) {
65                  training_inputs.add(file_inputs);
66                  training_outputs.add(file_outputs);
67              } else {
68                  classification_inputs.add(file_inputs);
69                  classification_outputs.add(file_outputs);
70              }
71          }
72      }
73
74      public static void main(String[] args) throws IOException {
75          Application network = new Application();
76      }
77  }
```

A.2.2 ManageData Class

```
1  package ubi.allab.application;
2
3  import java.util.ArrayList;
4
5  public class ManageData {
```

```
6
7     private static final String PATH = "C:\\Users\\Pedro\\Google Drive\\Tese
      \\";
8     private ArrayList<String[]> files_to_use;
9
10    public ManageData(String data_type) {
11        files_to_use = new ArrayList<String[]>();
12
13        if (data_type.equals("training"))
14            load(trainingDataFiles());
15        else
16            load(classifyingDataFiles());
17    }
18
19    private void load(String[] files) {
20        int i;
21        String[] tmp;
22
23        for (i = 0; i < files.length; i = i + 3) {
24            tmp = new String[3];
25            tmp[0] = files[i];
26            tmp[1] = files[i + 1];
27            tmp[2] = files[i + 2];
28            files_to_use.add(tmp);
29        }
30    }
31
32    private String[] trainingDataFiles() {
33        String full_path = PATH + "Ferro\\";
34
35        return new String[]{
36                full_path + "1.1.log", "17.149", "medium",
37                full_path + "1.2.log", "16.181", "medium",
38                full_path + "1.3.log", "18.664", "medium",
39                full_path + "2.1.log", "15.923", "medium",
40                full_path + "2.2.log", "15.859", "medium",
41                full_path + "2.3.log", "14.213", "medium",
42                full_path + "3.1.log", "17.964", "medium",
43                full_path + "3.2.log", "17.354", "medium",
44                full_path + "3.3.log", "18.348", "medium",
45                full_path + "4.1.log", "25.738", "high",
46                full_path + "4.2.log", "18.770", "high",
47                full_path + "4.3.log", "15.081", "high",
48                full_path + "5.1.log", "27.747", "high",
49                full_path + "6.1.log", "25.359", "high",
50                full_path + "6.2.log", "25.227", "high",
51                full_path + "6.3.log", "25.514", "high",
```

```
                full_path + "7.3.log", "15.694", "high",
                full_path + "9.1.log", "25.293", "high",
                full_path + "9.2.log", "18.957", "high",
                full_path + "9.3.log", "22.018", "high",
                full_path + "10.1.log", "18.910", "high",
                full_path + "10.3.log", "21.956", "high",
                full_path + "11.1.log", "15.679", "medium",
                full_path + "11.2.log", "15.481", "medium",
                full_path + "11.3.log", "16.399", "medium",
                full_path + "12.2.log", "32.620", "high",
                full_path + "12.3.log", "33.142", "high"
        };
    }

    private String[] classifyingDataFiles() {
        String full_path = PATH + "Caria\\";

        return new String[]{
                full_path + "1.2.log", "23.555", "high",
                full_path + "2.2.log", "23.600", "high",
                full_path + "4.3.log", "27.829", "high",
                full_path + "6.3.log", "16.131", "high"
        };
    }

    public ArrayList<String[]> getFilesToUse() {
        return files_to_use;
    }
}
```

A.2.3 CalculateInput Class

```
package ubi.allab.application;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;

public class CalculateInput {

    private ArrayList<ArrayList> joints;
    private ArrayList<Double[]> normalized_joints;
    private String file;
    private double time;
    private double[] input_values;

    public CalculateInput(String[] file_to_use) {
        file = file_to_use[0];
```

```
17          time = Double.parseDouble(file_to_use[1]);
18          joints = new ArrayList<ArrayList>();
19          normalized_joints = new ArrayList<Double[]>();
20          readDataFromFile();
21          normalizeData();
22          transformData();
23      }
24
25      private void readDataFromFile() {
26          try {
27              ReadTextFile data = new ReadTextFile(file);
28              joints = data.getJoints();
29          } catch (FileNotFoundException err){
30              System.out.println("File not found !!");
31          } catch (IOException err) {
32              System.out.println("Error while reading input file !!");
33          }
34      }
35
36      private void normalizeData() {
37          try{
38              NormalizeData normalize = new NormalizeData(joints, time);
39              normalized_joints = normalize.getNormalization();
40          } catch (Exception err){
41              System.out.println("Normalization Error !!");
42              err.printStackTrace();
43          }
44      }
45
46      private void transformData() {
47          try{
48              TransformData transform = new TransformData(normalized_joints,
    time);
49              input_values = transform.getTransformation();
50          } catch (Exception err){
51              System.out.println("Transformation Error !!");
52              err.printStackTrace();
53          }
54      }
55
56      public double[] getInputValues(){
57          return input_values;
58      }
59  }
```

A.2.4 ReadTextFile method

```
1  package ubi.allab.application;
```

```
2
3  import java.io.IOException;
4  import java.nio.charset.Charset;
5  import java.nio.charset.StandardCharsets;
6  import java.nio.file.Path;
7  import java.nio.file.Paths;
8  import java.util.ArrayList;
9  import java.util.Scanner;
10
11 public class ReadTextFile {
12
13     private String pathToFile;
14     private Charset encoding;
15     private ArrayList<ArrayList> joints;
16     private ArrayList<String[]> coords;
17
18     public ReadTextFile(String pathToFile) throws IOException {
19         this.pathToFile = pathToFile;
20         encoding = StandardCharsets.UTF_8;
21         joints = new ArrayList<ArrayList>();
22         coords = new ArrayList<String[]>();
23         readLargerTextFile();
24     }
25
26     private void readLargerTextFile() throws IOException {
27         Path path = Paths.get(pathToFile);
28         int i = 0;
29         String tmp;
30         String[] xyz;
31         try (Scanner scanner = new Scanner(path, encoding.name())) {
32             while (scanner.hasNextLine()) {
33                 tmp = scanner.nextLine();
34                 if (!tmp.isEmpty()) {
35                     xyz = tmp.split(" ");
36                     i++;
37                     coords.add(xyz);
38                 }
39             }
40         }
41         populateJoints();
42     }
43
44     private void populateJoints() {
45         ArrayList<String[]> tmp;
46
47         while (coords.size() != 0) {
48             tmp = new ArrayList<String[]>();
```

```
            for (int x = 0; x < 25; x++) {
                tmp.add(coords.get(0));
                coords.remove(0);
            }
            joints.add(tmp);
        }
    }

    public ArrayList<ArrayList> getJoints() {
        return joints;
    }
}
```

A.2.5 NormalizeData method

```
package ubi.allab.application;

import java.util.ArrayList;

public class NormalizeData {
    private double time;
    private ArrayList<ArrayList> joints;
    private ArrayList<Double[]> normalizedJoints;

    public NormalizeData(ArrayList<ArrayList> joints, double time) {
        normalizedJoints = new ArrayList<Double[]>();
        this.joints = joints;
        this.time = time;
        normalize();
        adaptToBoundaries();
    }

    private void normalize() {
        int i;
        double dist, avgS, avgSi, avgSf, avgA, step_time, difz, tmp;
        Double[] Cm, Cmi, Cmf, data;

        for (i = 0; i < joints.size(); i++) {
            if (i == 0) {
                Cm = findCenter(joints.get(i));
                difz = 0.0;
                avgS = 0.0;
                avgA = 0.0;
                step_time = (time / joints.size()) * i;
            } else if (i == 1) {
                Cm = findCenter(joints.get(i));
                Cmi = findCenter(joints.get(i - 1));
                Cmf = findCenter(joints.get(i));
```

```
34                difz = Math.abs(Cmf[2] - Cmi[2]);
35                avgS = calculateSpeed(Cmi, Cmf, time / joints.size());
36                avgA = calculateAcc(0.0, avgS, (time / joints.size()) * (i -
    1), (time / joints.size()) * i);
37                step_time = (time / joints.size()) * i;
38            } else if (i == joints.size() - 1) {
39                Cm = findCenter(joints.get(i));
40                Cmi = findCenter(joints.get(i - 2));
41                Cmf = findCenter(joints.get(i - 1));
42                difz = Math.abs(Cmf[2] - Cmi[2]);
43                avgS = 0.0;
44                tmp = calculateSpeed(Cmi, Cmf, time / joints.size());
45                avgA = calculateAcc(tmp, avgS, (time / joints.size()) * (i -
    1), (time / joints.size()) * i);
46                step_time = time;
47            } else {
48                Cm = findCenter(joints.get(i));
49                Cmi = findCenter(joints.get(i - 2));
50                Cmf = findCenter(joints.get(i - 1));
51                avgSi = calculateSpeed(Cmi, Cmf, time / joints.size());
52                Cmi = findCenter(joints.get(i - 1));
53                Cmf = findCenter(joints.get(i));
54                difz = Math.abs(Cmf[2] - Cmi[2]);
55                avgSf = calculateSpeed(Cmi, Cmf, time / joints.size());
56                avgS = avgSf;
57                avgA = calculateAcc(avgSi, avgSf, (time / joints.size()) * (
    i - 1), (time / joints.size()) * i);
58                step_time = (time / joints.size()) * i;
59            }
60
61            // use only coordenate Z
62            dist = Cm[2];
63            data = new Double[]{dist, difz, avgS, avgA, step_time};
64
65            //If the current data is equivalent to the previous one it
    should not be saved
66            if (checkRepeatedData(i, data))
67                normalizedJoints.add(data);
68        }
69    }
70
71    private void adaptToBoundaries() {
72        //Distc = [0, +00]  -> 0
73        //Difz = [0, +00]   -> 1
74        //Speed = [0, 1]    -> 2
75        //Accel = [-23, 23] -> 3
76        int i;
```

```
77          for (i = 0; i < normalizedJoints.size(); i++) {
78              if (normalizedJoints.get(i)[2] >= 1.0)
79                  normalizedJoints.get(i)[2] = 1.0;
80              if (normalizedJoints.get(i)[3] >= 23.0)
81                  normalizedJoints.get(i)[3] = 23.0;
82              if (normalizedJoints.get(i)[3] <= -23.0)
83                  normalizedJoints.get(i)[3] = -23.0;
84          }
85      }
86
87      private Double[] findCenter(ArrayList<String[]> jointList) {
88          int i, j;
89          double x, y, z, sum_x = 0.0, sum_y = 0.0, sum_z = 0.0;
90          double Cx, Cy, Cz;
91          String[] tmp;
92          for (i = 0, j = 0; i < jointList.size(); i++) {
93              tmp = (String[]) jointList.get(i);
94              x = Double.parseDouble(tmp[0]);
95              y = Double.parseDouble(tmp[1]);
96              z = Double.parseDouble(tmp[2]);
97              if (x != 0.0 && y != 0.0 && z != 0.0) {
98                  sum_x = sum_x + x;
99                  sum_y = sum_y + y;
100                 sum_z = sum_z + z;
101                 j++;
102             }
103         }
104         Cx = sum_x / j;
105         Cy = sum_y / j;
106         Cz = sum_z / j;
107
108         Double[] Cm = {Cx, Cy, Cz};
109         return Cm;
110     }
111
112     private Double calculateSpeed(Double[] Cmi, Double[] Cmf, double
    time_interval) { //Utilizar so a coordenada Z para calculos
113         //double tmp1 = Math.sqrt(Cmi[0] * Cmi[0] + Cmi[1] * Cmi[1] + Cmi[2]
     * Cmi[2]);
114         //double tmp2 = Math.sqrt(Cmf[0] * Cmf[0] + Cmf[1] * Cmf[1] + Cmf[2]
     * Cmf[2]);
115         double tmp1 = Cmi[2];
116         double tmp2 = Cmf[2];
117         return Math.abs((tmp2 - tmp1) / time_interval);
118     }
119
120     private Double calculateAcc(Double Va, Double Vb, double Ta, double Tb)
```

```
    {
121         return ((Vb - Va) / (Tb - Ta));
122     }
123 
124     private Boolean checkRepeatedData(int i, Double[] next_data) {
125         if (i == 0) return true;
126         if (i > normalizedJoints.size()) return false;
127         int result = 0;
128         Double[] prev_data = normalizedJoints.get(i - 1);
129 
130         result += Math.abs(Double.compare(prev_data[0], next_data[0]));
131         result += Math.abs(Double.compare(prev_data[1], next_data[1]));
132         result += Math.abs(Double.compare(prev_data[2], next_data[2]));
133         result += Math.abs(Double.compare(prev_data[3], next_data[3]));
134 
135         return (result > 3); //Followed frames should not have the same
    values.
136     }
137 
138     public ArrayList<Double[]> getNormalization() {
139         return normalizedJoints;
140     }
141 }
```

A.2.6 TransformData method

```
1  package ubi.allab.application;
2 
3  import java.util.ArrayList;
4 
5  public class TransformData {
6 
7      private static final double SPEED = 0.3;
8      private static final double SPACE = 0.013259536082475;
9 
10     private double time;
11     private ArrayList<Double[]> normalizedJoints;
12     private double transformedData[];
13 
14     public TransformData(ArrayList<Double[]> normalizedJoints, double time)
    {
15         this.time = time;
16         this.normalizedJoints = normalizedJoints;
17         transformedData = new double[4];
18         firstClassification();
19         secondClassification();
20         thirdClassification();
21         fourthClassification();
```

```
22      }
23
24      private void firstClassification() { // Calculates the acceleration
        differences
25          int i;
26          int positives = 0, negatives = 0;
27
28          for (i = 0; i < normalizedJoints.size(); i++) {
29              if (Math.signum(normalizedJoints.get(i)[3]) >= 0)
30                  positives++;
31              else
32                  negatives++;
33          }
34
35          if (positives >= negatives)
36              transformedData[0] = 1.0;
37          else
38              transformedData[0] = 0.0;
39      }
40
41      private void secondClassification() { // Verify how many times the speed
         is superior to "SPEED"
42          int i;
43          int positives = 0, negatives = 0;
44
45          for (i = 0; i < normalizedJoints.size(); i++) {
46              if (normalizedJoints.get(i)[2] >= SPEED)
47                  positives++;
48              else
49                  negatives++;
50          }
51
52          if (positives >= negatives)
53              transformedData[1] = 1.0;
54          else
55              transformedData[1] = 0.0;
56      }
57
58      private void thirdClassification() { // Verifies how many times the
        distance is superior to "SPACE"
59          int i;
60          int positives = 0, negatives = 0;
61
62          for (i = 0; i < normalizedJoints.size(); i++) {
63              if (normalizedJoints.get(i)[1] >= SPACE)
64                  positives++;
65              else
```

```
66                  negatives++;
67          }
68
69          if (positives >= negatives)
70              transformedData[2] = 1.0;
71          else
72              transformedData[2] = 0.0;
73      }
74
75      private void fourthClassification() { // Verify the total time of the
        movement
76          if(time > 20.0)
77              transformedData[3] = 1.0;
78          else if(time > 10.0 && time <= 20.0)
79              transformedData[3] = 0.0;
80          else{
81              transformedData[3] = -1.0;
82          }
83      }
84
85      public double[] getTransformation() {
86          return transformedData;
87      }
88  }
```

A.2.7 Classification Class

```
1   package ubi.allab.application;
2
3   import ubi.allab.neural_network.Network;
4   import ubi.allab.neural_network.TrainSet;
5
6   import java.util.ArrayList;
7   import java.util.Arrays;
8
9   public class Classification {
10
11      private static final int INPUT_LAYER  = 4;
12      private static final int HIDDEN_LAYER = 3;
13      private static final int OUTPUT_LAYER = 3;
14      private static final int ITERATIONS = 1000000;
15
16      private Network net;
17      private TrainSet training;
18      private TrainSet classify;
19
20      public Classification(){
21          net = new Network(INPUT_LAYER, HIDDEN_LAYER, HIDDEN_LAYER,
```

```
    OUTPUT_LAYER);
22          training = new TrainSet(INPUT_LAYER, OUTPUT_LAYER);
23          classify = new TrainSet(INPUT_LAYER, OUTPUT_LAYER);
24      }
25
26      public void trainNetwork(ArrayList<double[]> input, ArrayList<double[]>
    output){
27          for(int i = 0; i < input.size(); i++){
28              training.addData(input.get(i), output.get(i));
29          }
30          net.train(training, ITERATIONS, input.size());
31      }
32
33      public void finalNetwork(ArrayList<double[]> input, ArrayList<double[]>
    output){
34          for(int i = 0; i < input.size(); i++){
35              classify.addData(input.get(i), output.get(i));
36          }
37
38          //net.train(classify, ITERATIONS/10, input.size());
39
40          for (int i = 0; i < input.size(); i++) {
41              System.out.println(Arrays.toString(net.calculate(classify.
    getInput(i))));
42          }
43      }
44  }
```

A.2.8 Network Class

```
1  package ubi.allab.neural_network;
2
3  import java.util.Arrays;
4
5  public class Network {
6      private final int[] NETWORK_LAYER_SIZES;
7      private final int NETWORK_SIZE;
8      private final int INPUT_SIZE;
9      private final int OUTPUT_SIZE;
10     private double[][] output;
11     private double[][][][] weights;
12     private double[][] bias;
13     private double[][] error_signal;
14     private double[][] output_derivative;
15
16     public Network(int... NETWORK_LAYER_SIZES) {
17         this.NETWORK_LAYER_SIZES = NETWORK_LAYER_SIZES;
18         this.NETWORK_SIZE = NETWORK_LAYER_SIZES.length;
```

```
19          this.INPUT_SIZE = NETWORK_LAYER_SIZES[0];
20          this.OUTPUT_SIZE = NETWORK_LAYER_SIZES[NETWORK_SIZE - 1];
21
22          this.output = new double[NETWORK_SIZE][];
23          this.weights = new double[NETWORK_SIZE][][];
24          this.bias = new double[NETWORK_SIZE][];
25          this.error_signal = new double[NETWORK_SIZE][];
26          this.output_derivative = new double[NETWORK_SIZE][];
27
28          for (int i = 0; i < NETWORK_SIZE; i++) {
29              this.output[i] = new double[NETWORK_LAYER_SIZES[i]];
30              this.error_signal[i] = new double[NETWORK_LAYER_SIZES[i]];
31              this.output_derivative[i] = new double[NETWORK_LAYER_SIZES[i]];
32              this.bias[i] = NetworkTools.createRandomArray(
    NETWORK_LAYER_SIZES[i], 0.3, 0.7);
33              if (i > 0) {
34                  this.weights[i] = NetworkTools.createRandomArray(
    NETWORK_LAYER_SIZES[i], NETWORK_LAYER_SIZES[i - 1], 0.3, 0.7);
35              }
36          }
37      }
38
39      public double[] calculate(double... input) {
40          if (input.length != INPUT_SIZE) return null;
41
42          output[0] = input;
43
44          for (int layer = 1; layer < NETWORK_SIZE; layer++) {
45              for (int neuron = 0; neuron < NETWORK_LAYER_SIZES[layer]; neuron
    ++) {
46                  double sum = bias[layer][neuron];
47                  for (int prevNeuron = 0; prevNeuron < NETWORK_LAYER_SIZES[
    layer - 1]; prevNeuron++) {
48                      sum += output[layer - 1][prevNeuron] * weights[layer][
    neuron][prevNeuron];
49                  }
50                  output[layer][neuron] = NetworkTools.sigmoid(sum);
51                  output_derivative[layer][neuron] = (output[layer][neuron] *
    (1 - output[layer][neuron]));
52              }
53          }
54          return output[NETWORK_SIZE - 1];
55      }
56
57      public void train(TrainSet set, int loops, int batch_size) {
58          if (set.INPUT_SIZE != INPUT_SIZE || set.OUTPUT_SIZE != OUTPUT_SIZE)
    return;
```

```
59          for (int i = 0; i < loops; i++) {
60              TrainSet batch = set.extractBatch(batch_size);
61              for (int y = 0; y < batch_size; y++) {
62                  train(batch.getInput(y), batch.getOutput(y), 0.3);
63              }
64              //System.out.println(MSE(batch));
65          }
66      }
67
68      private double MSE(TrainSet set){
69          double v = 0;
70          for (int i = 0; i < set.size(); i++) {
71              v += MSE(set.getInput(i), set.getOutput(i));
72          }
73          return v / set.size();
74      }
75
76      private double MSE(double[] input, double[] target) {
77          if (input.length != INPUT_SIZE || target.length != OUTPUT_SIZE)
    return 0;
78          calculate(input);
79          double v = 0;
80          for (int i = 0; i < target.length; i++) {
81              v += (target[i] - output[NETWORK_SIZE-1][i]) * (target[i] -
    output[NETWORK_SIZE-1][i]);
82          }
83          return v / (2d * target.length);
84      }
85
86      private void train(double[] input, double[] target, double eta) {
87          if (input.length != INPUT_SIZE || target.length != OUTPUT_SIZE)
    return;
88          calculate(input);
89          backPropagationError(target);
90          updateWeights(eta);
91      }
92
93      private void backPropagationError(double[] target) {
94          for (int neuron = 0; neuron < NETWORK_LAYER_SIZES[NETWORK_SIZE - 1];
     neuron++) {
95              error_signal[NETWORK_SIZE - 1][neuron] = (output[NETWORK_SIZE -
    1][neuron] - target[neuron])
96                      * output_derivative[NETWORK_SIZE - 1][neuron];
97          }
98          for (int layer = NETWORK_SIZE - 2; layer > 0; layer--) {
99              for (int neuron = 0; neuron < NETWORK_LAYER_SIZES[layer]; neuron
    ++) {
```

```
100                 double sum = 0;
101                 for (int nextNeuron = 0; nextNeuron < NETWORK_LAYER_SIZES[
    layer + 1]; nextNeuron++) {
102                     sum += weights[layer + 1][nextNeuron][neuron] *
    error_signal[layer + 1][nextNeuron];
103                 }
104                 error_signal[layer][neuron] = sum * output_derivative[layer
    ][neuron];
105             }
106         }
107     }
108 
109     private void updateWeights(double eta) {
110         for (int layer = 1; layer < NETWORK_SIZE; layer++) {
111             for (int neuron = 0; neuron < NETWORK_LAYER_SIZES[layer]; neuron
    ++) {
112                 double delta = (-1) * eta * error_signal[layer][neuron];
113                 bias[layer][neuron] += delta;
114                 for (int prevNeuron = 0; prevNeuron < NETWORK_LAYER_SIZES[
    layer - 1]; prevNeuron++) {
115                     weights[layer][neuron][prevNeuron] += delta * output[
    layer - 1][prevNeuron];
116                 }
117             }
118         }
119 
120     }
121 
122     public static void main(String[] args) {
123         Network net = new Network(4, 3, 3, 2);
124 
125         TrainSet set = new TrainSet(4, 2);
126         set.addData(new double[]{0.1, 0.2, 0.3, 0.4}, new double[]{0.9,
    0.1});
127         set.addData(new double[]{0.9, 0.8, 0.7, 0.6}, new double[]{0.1,
    0.9});
128         set.addData(new double[]{0.3, 0.8, 0.1, 0.4}, new double[]{0.3,
    0.7});
129         set.addData(new double[]{0.9, 0.8, 0.1, 0.2}, new double[]{0.7,
    0.3});
130 
131         net.train(set, 1000000, 4);
132 
133         for (int i = 0; i < 4; i++) {
134             System.out.println(Arrays.toString(net.calculate(set.getInput(i)
    )));
135         }
```

```
136     }
137 }
```

A.2.9 TrainSet Class

```
1  package ubi.allab.neural_network;
2
3  import java.util.ArrayList;
4  import java.util.Arrays;
5
6  public class TrainSet {
7
8      public final int INPUT_SIZE;
9      public final int OUTPUT_SIZE;
10
11     private ArrayList<double[][]> data = new ArrayList<>();
12
13     public TrainSet(int INPUT_SIZE, int OUTPUT_SIZE) {
14         this.INPUT_SIZE = INPUT_SIZE;
15         this.OUTPUT_SIZE = OUTPUT_SIZE;
16     }
17
18     public void addData(double[] in, double[] expected) {
19         if(in.length != INPUT_SIZE || expected.length != OUTPUT_SIZE) return
       ;
20         data.add(new double[][]{in, expected});
21     }
22
23     public TrainSet extractBatch(int size) {
24         if(size > 0 && size <= this.size()) {
25             TrainSet set = new TrainSet(INPUT_SIZE, OUTPUT_SIZE);
26             Integer[] ids = NetworkTools.randomValues(0,this.size() - 1,
       size);
27             for(Integer i:ids) {
28                 set.addData(this.getInput(i),this.getOutput(i));
29             }
30             return set;
31         }else return this;
32     }
33
34     public String toString() {
35         String s = "TrainSet ["+INPUT_SIZE+ " ; "+OUTPUT_SIZE+"]\n";
36         int index = 0;
37         for(double[][] r:data) {
38             s += index +":   "+Arrays.toString(r[0]) +"  >-||-<  "+Arrays.
       toString(r[1]) +"\n";
39             index++;
40         }
```

```
41          return s;
42      }
43
44      public int size() {
45          return data.size();
46      }
47
48      public double[] getInput(int index) {
49          if(index >= 0 && index < size())
50              return data.get(index)[0];
51          else return null;
52      }
53
54      public double[] getOutput(int index) {
55          if(index >= 0 && index < size())
56              return data.get(index)[1];
57          else return null;
58      }
59
60      public int getINPUT_SIZE() {
61          return INPUT_SIZE;
62      }
63
64      public int getOUTPUT_SIZE() {
65          return OUTPUT_SIZE;
66      }
67
68      public static void main(String[] args) {
69          TrainSet set = new TrainSet(3,2);
70
71          for(int i = 0; i < 8; i++) {
72              double[] a = new double[3];
73              double[] b = new double[2];
74              for(int k = 0; k < 3; k++) {
75                  a[k] = (double)((int)(Math.random() * 10)) / (double)10;
76                  if(k < 2) {
77                      b[k] = (double)((int)(Math.random() * 10)) / (double)10;
78                  }
79              }
80              set.addData(a,b);
81          }
82
83          System.out.println(set);
84          System.out.println(set.extractBatch(3));
85      }
86  }
```

A.2.10 NetworkTools Class

```
1  package ubi.allab.neural_network;
2
3  public class NetworkTools {
4
5      public static double[] createArray(int size, double init_value) {
6          if (size < 1) {
7              return null;
8          }
9          double[] ar = new double[size];
10         for (int i = 0; i < size; i++) {
11             ar[i] = init_value;
12         }
13         return ar;
14     }
15
16     public static double[] createRandomArray(int size, double lower_bound,
       double upper_bound) {
17         if (size < 1) {
18             return null;
19         }
20         double[] ar = new double[size];
21         for (int i = 0; i < size; i++) {
22             ar[i] = randomValue(lower_bound, upper_bound);
23         }
24         return ar;
25     }
26
27     public static double[][] createRandomArray(int sizeX, int sizeY, double
       lower_bound, double upper_bound) {
28         if (sizeX < 1 || sizeY < 1) {
29             return null;
30         }
31         double[][] ar = new double[sizeX][sizeY];
32         for (int i = 0; i < sizeX; i++) {
33             ar[i] = createRandomArray(sizeY, lower_bound, upper_bound);
34         }
35         return ar;
36     }
37
38     public static double randomValue(double lower_bound, double upper_bound)
        {
39         return Math.random() * (upper_bound - lower_bound) + lower_bound;
40     }
41
42     public static Integer[] randomValues(int lowerBound, int upperBound, int
```

```
    amount) {

        lowerBound--;

        if (amount > (upperBound - lowerBound)) {
            return null;
        }

        Integer[] values = new Integer[amount];
        for (int i = 0; i < amount; i++) {
            int n = (int) (Math.random() * (upperBound - lowerBound + 1) +
    lowerBound);
            while (containsValue(values, n)) {
                n = (int) (Math.random() * (upperBound - lowerBound + 1) +
    lowerBound);
            }
            values[i] = n;
        }
        return values;
    }

    public static <T extends Comparable<T>> boolean containsValue(T[] ar, T
    value) {
        for (int i = 0; i < ar.length; i++) {
            if (ar[i] != null) {
                if (value.compareTo(ar[i]) == 0) {
                    return true;
                }
            }
        }
        return false;
    }

    public static int indexOfHighestValue(double[] values) {
        int index = 0;
        for (int i = 1; i < values.length; i++) {
            if (values[i] > values[index]) {
                index = i;
            }
        }
        return index;
    }

    public static double sigmoid(double x) {
        return 1d / (1 + Math.exp(-x));
    }
}
```

A.3 Application Console Outputs

The following images represent the ten application console outputs that were obtained and used during the results analysis in chapter four.

A.3.1 Console Output One

```
Loading training DATASET...
Training DATASET successfully loaded !!

Loading classification DATASET...
Classification DATASET successfully loaded !!

Starting the classification training phase...
Mean error value over 1000000: 0.039239559035159195
The classification training phase has completed successfully !!

Starting the classification final phase...
Input: [0.0, 1.0, 0.0, 1.0] => Output: [1.794148235533662E-5, 4.280020476685341E-6, 0.9999957182462021] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [1.3759291124407587E-5, 6.009072274830778E-6, 0.9999939886513286] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [1.3759291124407587E-5, 6.009072274830778E-6, 0.9999939886513286] => Expected: [0.0, 0.0, 1.0]
Input: [1.0, 1.0, 1.0, 0.0] => Output: [6.56112918920319E-5, 0.009292421543713594, 0.9907075767943201] => Expected: [0.0, 0.0, 1.0]

Process finished with exit code 0
```

Figure A.1: Application console output one

A.3.2 Console Output Two

```
Loading training DATASET...
Training DATASET successfully loaded !!

Loading classification DATASET...
Classification DATASET successfully loaded !!

Starting the classification training phase...
Mean error value over 1000000: 0.039239650428880064
The classification training phase has completed successfully !!

Starting the classification final phase...
Input: [0.0, 1.0, 0.0, 1.0] => Output: [2.3166281254593225E-5, 3.0382534894761605E-4, 0.9996961747991125] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [1.3374175035206572E-5, 1.1040331924450852E-4, 0.9998895967396797] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [1.3374175035206572E-5, 1.1040331924450852E-4, 0.9998895967396797] => Expected: [0.0, 0.0, 1.0]
Input: [1.0, 1.0, 1.0, 0.0] => Output: [4.019351101598086E-5, 0.0028737458916524576, 0.9971262548825077] => Expected: [0.0, 0.0, 1.0]

Process finished with exit code 0
```

Figure A.2: Application console output two

A.3.3 Console Output Three

```
Loading training DATASET...
Training DATASET successfully loaded !!

Loading classification DATASET...
Classification DATASET successfully loaded !!

Starting the classification training phase...
Mean error value over 1000000: 0.039256008949246524
The classification training phase has completed successfully !!

Starting the classification final phase...
Input: [0.0, 1.0, 0.0, 1.0] => Output: [8.692397147286361E-6, 2.250562459199995E-4, 0.9997749439334834] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [4.151340806187727E-6, 3.796890184426026E-5, 0.9999620311384434] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [4.151340806187727E-6, 3.796890184426026E-5, 0.9999620311384434] => Expected: [0.0, 0.0, 1.0]
Input: [1.0, 1.0, 1.0, 0.0] => Output: [1.8771353518355388E-5, 0.0029391080604733726, 0.9970608929325334] => Expected: [0.0, 0.0, 1.0]

Process finished with exit code 0
```

Figure A.3: Application console output three

A.3.4 Console Output Four

```
Loading training DATASET...
Training DATASET successfully loaded !!

Loading classification DATASET...
Classification DATASET successfully loaded !!

Starting the classification training phase...
Mean error value over 1000000: 0.0392286312396033
The classification training phase has completed successfully !!

Starting the classification final phase...
Input: [0.0, 1.0, 0.0, 1.0] => Output: [1.0528256978821108E-4, 1.386397501830746E-4, 0.9998613602525785] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [4.2687069614608084E-5, 8.059852162339667E-6, 0.9999919401481449] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [4.2687069614608084E-5, 8.059852162339667E-6, 0.9999919401481449] => Expected: [0.0, 0.0, 1.0]
Input: [1.0, 1.0, 1.0, 0.0] => Output: [4.4180816967724336E-4, 0.013254717914690052, 0.9867452820852] => Expected: [0.0, 0.0, 1.0]

Process finished with exit code 0
```

Figure A.4: Application console output four

A.3.5 Console Output Five

```
Loading training DATASET...
Training DATASET successfully loaded !!

Loading classification DATASET...
Classification DATASET successfully loaded !!

Starting the classification training phase...
Mean error value over 1000000: 0.039264751844877384
The classification training phase has completed successfully !!

Starting the classification final phase...
Input: [0.0, 1.0, 0.0, 1.0] => Output: [1.802401899237266E-4, 5.948085866977581E-4, 0.9994051914133022] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [1.8046936236059132E-4, 5.962372003881008E-4, 0.999403762799612] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [1.8046936236059132E-4, 5.962372003881008E-4, 0.999403762799612] => Expected: [0.0, 0.0, 1.0]
Input: [1.0, 1.0, 1.0, 0.0] => Output: [1.5941448011733234E-4, 0.00367231086345424, 0.9963276891365457] => Expected: [0.0, 0.0, 1.0]

Process finished with exit code 0
```

Figure A.5: Application console output five

A.3.6 Console Output Six

```
Loading training DATASET...
Training DATASET successfully loaded !!

Loading classification DATASET...
Classification DATASET successfully loaded !!

Starting the classification training phase...
Mean error value over 1000000: 0.039291753803629814
The classification training phase has completed successfully !!

Starting the classification final phase...
Input: [0.0, 1.0, 0.0, 1.0] => Output: [1.9597360963490593E-5, 0.0056902525034688315, 0.9943097474966119] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [3.2384962350125657E-6, 3.7924363677520014E-6, 0.999996207563631] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [3.2384962350125657E-6, 3.7924363677520014E-6, 0.999996207563631] => Expected: [0.0, 0.0, 1.0]
Input: [1.0, 1.0, 1.0, 0.0] => Output: [2.5081793701433034E-5, 0.004425233128981999, 0.9955747668708996] => Expected: [0.0, 0.0, 1.0]

Process finished with exit code 0
```

Figure A.6: Application console output six

A.3.7 Console Output Seven

```
Loading training DATASET...
Training DATASET successfully loaded !!

Loading classification DATASET...
Classification DATASET successfully loaded !!

Starting the classification training phase...
Mean error value over 1000000: 0.03923420278232604
The classification training phase has completed successfully !!

Starting the classification final phase...
Input: [0.0, 1.0, 0.0, 1.0] => Output: [3.571118091843838E-5, 1.0199549618928155E-4, 0.9998980030829162] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [2.8681839679936332E-5, 3.690699647888109E-5, 0.9999630924082283] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [2.8681839679936332E-5, 3.690699647888109E-5, 0.9999630924082283] => Expected: [0.0, 0.0, 1.0]
Input: [1.0, 1.0, 1.0, 0.0] => Output: [6.974345519730311E-5, 0.009081429185186346, 0.9909185705607539] => Expected: [0.0, 0.0, 1.0]

Process finished with exit code 0
```

Figure A.7: Application console output seven

A.3.8 Console Output Eight

```
Loading training DATASET...
Training DATASET successfully loaded !!

Loading classification DATASET...
Classification DATASET successfully loaded !!

Starting the classification training phase...
Mean error value over 1000000: 0.039253223392842936
The classification training phase has completed successfully !!

Starting the classification final phase...
Input: [0.0, 1.0, 0.0, 1.0] => Output: [1.8491360307328446E-4, 9.43060679972735E-4, 0.9990571125906482] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [5.231996414149557E-5, 2.3941579209086824E-5, 0.9999760645370442] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [5.231996414149557E-5, 2.3941579209086824E-5, 0.9999760645370442] => Expected: [0.0, 0.0, 1.0]
Input: [1.0, 1.0, 1.0, 0.0] => Output: [1.55518442951638E-5, 0.007501954128141977, 0.9924980398583678] => Expected: [0.0, 0.0, 1.0]

Process finished with exit code 0
```

Figure A.8: Application console output eight

A.3.9 Console Output Nine

```
Loading training DATASET...
Training DATASET successfully loaded !!

Loading classification DATASET...
Classification DATASET successfully loaded !!

Starting the classification training phase...
Mean error value over 1000000: 0.03923141003234531
The classification training phase has completed successfully !!

Starting the classification final phase...
Input: [0.0, 1.0, 0.0, 1.0] => Output: [1.0240703108915565E-5, 6.927109864848383E-6, 0.9999930786973508] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [1.035661965675549E-5, 7.162071114991674E-6, 0.9999928439069675] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [1.035661965675549E-5, 7.162071114991674E-6, 0.9999928439069675] => Expected: [0.0, 0.0, 1.0]
Input: [1.0, 1.0, 1.0, 0.0] => Output: [1.1727227063032475E-4, 0.004302662952708057, 0.9956980814324169] => Expected: [0.0, 0.0, 1.0]

Process finished with exit code 0
```

Figure A.9: Application console output nine

A.3.10 Console Output Ten

```
Loading training DATASET...
Training DATASET successfully loaded !!

Loading classification DATASET...
Classification DATASET successfully loaded !!

Starting the classification training phase...
Mean error value over 1000000: 0.03923473863936241
The classification training phase has completed successfully !!

Starting the classification final phase...
Input: [0.0, 1.0, 0.0, 1.0] => Output: [1.0111030104191426E-5, 5.0977333469343E-5, 0.9999490226932375] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [1.0069670183713392E-5, 5.014044420395501E-5, 0.9999498595821419] => Expected: [0.0, 0.0, 1.0]
Input: [0.0, 0.0, 0.0, 1.0] => Output: [1.0069670183713392E-5, 5.014044420395501E-5, 0.9999498595821419] => Expected: [0.0, 0.0, 1.0]
Input: [1.0, 1.0, 1.0, 0.0] => Output: [3.836843349483E-5, 0.0034623196198481617, 0.9965376810009974] => Expected: [0.0, 0.0, 1.0]

Process finished with exit code 0
```

Figure A.10: Application console output ten