Bachelor Thesis Project

Machine Learning in Digital Telerehabilitation
- Telerehabilitation system based on kinect

Author: Artur Vitt
Supervisor: Mauro Caporuscio
Semester: HT 2015
Subject: Computer Science
Abstract

The healthcare as a service is always under pressure and is in great demand. Despite living in a developed world with access to cars, trains, busses and other transportation means, sometimes accessing healthcare can be troublesome and costly. The continuous technological progress provides new means to provide different kind of services, healthcare included. One way of putting technology into good use in field of healthcare is remote rehabilitation.

Remote rehabilitation is a matter of delivering physiotherapy on a distance. The use of remote rehabilitation potentially reduces waiting time for treatment and gives a possibility for people with long traveling distance, to be treated at their locations. The thesis addresses a solution to physiotherapy on distance that utilizes Kinect and machine learning technologies to provide physiotherapy offline. Thesis presents Kinect Digital Rehabilitation Assistant (KiDiRA), which provides simple functions to suffice the needs of a physiotherapist to plan therapeutical treatment and the ability of a patient to get access physiotherapy offline in real-time at home.

More precisely KiDiRA is the system that combines Kinect motion capture device, an interactive graphical interface and a platform to assist with the design of physiotherapeutical exercises and an aid for the patient to execute therapeutic plan on his/her own. The system displays the exercise directives and monitors performance of patient. KiDiRA aims to incorporate science of machine-learning in process of performance evaluation during exercises.
Preface
Cordial thanks to my sister for supporting and encouraging. Also huge thanks to my teacher and my supervisor for good and thorough guidance.
## Contents

1 Introduction ........................................................................................................ 1
   1.1 Aims and Scope ......................................................................................... 2
   1.2 Background .............................................................................................. 2
      1.2.1 Machine learning ............................................................................. 2
         1.2.1.1 Designing learning systems ....................................................... 3
         1.2.1.2 Learning systems background ............................................... 3
      1.2.2 Artificial neural network (ANN) ...................................................... 4
         1.2.2.1 Biological neural networks ..................................................... 4
         1.2.2.2 From biological neural network to artificial neural network .... 7
         1.2.2.3 Training of the ANN ............................................................... 7
      1.2.3 K-nearest neighbors (K-NN) .......................................................... 8
      1.2.4 Random forests (RF) ....................................................................... 8
   1.3 Technical context ....................................................................................... 9
      1.3.1 Glossary ............................................................................................ 10
      1.3.2 Kinect .............................................................................................. 10
         1.3.2.1 Kinect technical details ............................................................ 11
   1.4 Previous research in telerehabilitation ....................................................... 12
   1.5 Problem formulation ................................................................................. 13
   1.6 Motivation ................................................................................................. 13
   1.7 Research Question ................................................................................... 13
   1.8 Scope/Limitation ...................................................................................... 13
   1.9 Target group ............................................................................................. 14
   1.10 Outline ..................................................................................................... 14

2 Review of Kinect Digital Rehabilitation Assistant ........................................... 15
   2.1 Scientific approach .................................................................................. 15
   2.2 Architecture ............................................................................................ 15
      2.2.1 Exercise spatial constraints ............................................................ 15
      2.2.2 KiDiRA use cases ............................................................................ 16
      2.2.3 Components of applications .......................................................... 16
   2.3 Therapy planning .................................................................................... 23
      2.3.1 Creation of new exercises ............................................................... 23
      2.3.2 Exercise monitoring ....................................................................... 24

3 KiDiRA component technical details and validation ....................................... 24
   3.1 Exercise performance classification ....................................................... 25
   3.2 Exercise performance classification method .......................................... 30
      3.2.1 The explicit classification function ................................................. 31
      3.2.2 The implicit classification via machine learning ......................... 31
         3.2.2.1 Creating the training set ......................................................... 32
         3.2.2.2 Training set Generation algorithm ...................................... 32
         3.2.2.3 The implicit classification function via ANN ....................... 38
         3.2.2.4 The implicit classification function via Random Trees .......... 39
         3.2.2.5 The implicit classification function via K-NN ...................... 39
   3.3 Classifier performance measurements and measurements extraction procedure ................................................... 39
   3.4 Reliability and Validity ......................................................................... 42
4 Discussion
4.1 Classification speed .................................. 44
4.2 Classification ratio .................................. 44

5 Conclusion and future work 46
5.1 KiDiRA: Kinect Digital Rehabilitation Assistant .............. 46
5.2 Posture performance algorithm through explicit function and self-learning systems .................................. 46
  5.2.1 Self-learning systems as classification algorithm .......... 46
5.3 Future research .......................................... 47
  5.3.1 Visual features ...................................... 48
  5.3.2 Non-visual features .................................. 48

References 49

A Appendix: KiDiRa architecture A
A.1 KiDiRA activity flow .................................. E
A.2 Interaction diagrams .................................. I
1 Introduction

Current digital technologies provide a large number of computer application opportunities. One particular field of application is telemedicine. Telemedicine is a field concerned with remote medical services. It advocates remote healthcare is able to provide remote medical services without losing the quality. Telemedicine has grown in demand over last decades and became a useful resource in modern healthcare, as digital innovations became easily accessible. Due to higher survivability to diseases and traumas, a new group of patients came about to highly benefit from telemedicine. That group consists of people with physical challenges; patients that can be treated remotely. The use innovations provided by telemedicine gives patients economical and accessibility advantages for healthcare.

Current telemedicine provides methods of remote medical evaluation without the need of a visit to a hospital, which significantly simplifies access to the healthcare. Telerehabilitation is another term for remote physical rehabilitation. It is an alternative to common doctor-patient appointments rehabilitation, which requires patient present at the hospital. Remote rehabilitation concerned with the delivery of medical services over the Internet or other media. This method can be applied to people who cannot travel to the clinic because of their disability or because of travel time. Hence, remote physical rehabilitation allows physiotherapists to engage in a consultation at a distance and can be categorized as physical assessment and physical therapy. Remote physical assessment concerns the evaluation of the body and its functions, whereas remote physical therapy concerns the prescription of or the assistance with specific exercises.

Modalities such as webcams, video conferencing, phone lines, videophones and web pages containing rich Internet applications are commonly used in remote contact. However, a visual nature of remote physical rehabilitation limits the types of rehabilitation services that can be provided. The lack of information (such as spatial data, clinical records, and vital signs) inherent to videoconferences (with no specific medical cues) makes it difficult to provide both the precise evaluation of the body and its functions, as well as a full assistance with exercises.

The usage of noninvasive sensors during remote physical rehabilitation sessions would open new opportunities to engage patients in progressive, personalized therapies with feedback on the performance. Moreover, clinical trials would be able to include remote verification of the integrity of complex physical interventions and compliance with practice. Telerehabilitation systems are constantly under improvement due to continuous emergence of new technologies. Present day telerehabilitation systems support monitoring and physiotherapy treatment of different groups – such as elderly, disabled and sick. They are facilitated by videoconferences and advanced computerized evaluation methods, which are backed up by sensor measurements.

This thesis focuses on the integration of machine learning technology into tele-physiotherapy and presents a prototype of Personal digital assistant that demonstrates the integration of machine learning, which can be a useful resource in healthcare. This computer application aims to provide interfaces to physiotherapist and patient. The interface for physiotherapist is concerned with facilitating feature which would provide control pane and visuals needed to simplify the task of specifications of exercises. The interface for the patient is aimed to provide interactive physiotherapy with feedback and transitions between exercises, which are specified by the physiotherapist.

The application prototype is based on Microsoft Kinect posture tracking system. Kinect system provides a noninvasive posture and skeletal tracking features. Usage of Kinect system offers a great level of control to a therapist and also detailed feedback to the patient.
This can be obtained via incorporating posture tracking features provided by Kinect in an application which implements therapeutic features and a respective graphical interface. A huge benefit of MS Kinect is posture detection without extra wearables.

1.1 Aims and Scope

The aim of the thesis is to compare properties of machine learning algorithms to learn and apply learned knowledge in the task of prediction. The type of learning is limited to Supervised learning. The algorithms which will be applied are *Artificial neural network*, *K - nearest neighbors*, *Random Trees*. The comparison is to be performed in the case of physio telerehabilitation application. The task includes a stage of selection of the feature vector and a target value/vector. The algorithms will be compared in classification speed as frequency, classification ratio as correctly classified from total and training time.

The thesis applies learning algorithm in Kinect Digital Rehabilitation Assistant (KiDiRA). KiDiRA is a tele- physiorehabilitation application. KiDiRA is built for both the physiotherapist and patient, with special emphasis on features that allow offline/smart/direct assessment and feedback during the therapy. KiDiRA bases on Kinect posture tracking features to implement its features of artificial posture guiding assistance. The application provides interface for a physiotherapist to create physiotherapeutic exercises and an interface that allows a patient to undergo a rehabilitation without the presence of medical personnel and use designated exercises supplied by a specialist. The application facilitates patient with a digital assistant which monitors the exercises with the use of Kinect body tracking technology and provides feedback in the real-time. The feedback feature is based on data supplied from Kinect body tracking feature in real-time. KiDiRA aims to bring novel features through automation of therapeutical assistance. The current application brings direct feedback with the description of how well the exercise is performed. The classification problem with which classifiers are tasked is following. The feedback domain consists of "Fail class" where performance is below 40%; the "Pass class" which presents the success rate between 40% and 50% inclusive; the "Second class" which presents success rate of between 50% exclusive and 60% inclusive; the "Third class" which presents success rate of between 60% exclusive and 80% inclusive and the "Outstanding class" which represents the success rate above 80%.

1.2 Background

The section contains a short presentation of machine learning. The presentation contains segments which introduce the design of learning systems, main learning paradigms, and learning algorithms. Each learning algorithm presented is described in general terms which include a learning method and classification mechanism.

1.2.1 Machine learning

Machine learning is a subfield within computer science, which in other words can be denoted as a study of automated knowledge discovery. In this particular case, the machine learning is used as an approach for solving the posture evaluation requirement in the application. If put in more detailed fashion, posture evaluation requirement works as a feedback function for a user that would map comparison between expected posture and given posture to set of label describing completion rate.

---

1 Feature vector is an array of features that represent the object and used to classify it.
2 Target vector or value that is the result of classification or prediction.
This thesis uses learning system to facilitate some of the requirement to given research question. To make a learning system the developer must answer following questions. What is the training experience? What is the target function? How to represent the target function? Which function approximation algorithm is to be applied? This section gives short introduction to design process of learning systems (see for more in the book[1]).

Checkers problem is used to exemplify and describe the mechanism. Development of learning system has four significant moments within its design. First such moment is to select training experience from which the system will learn. Training experience can also be denoted as training data, training data set and so on. The developer when selecting training experience should consider three important properties of the training experience. The first important property of the training experience is if it direct or indirect. For example, let observe a case of learning system for the game of checkers. The direct training examples would consist of individual checkers board states and correct move for each. The indirect training experience, in the example of the checkers learning system, is sequences of moves and results of various played games. In the second case, the experience comes indirectly and requires learning system to infer and assign credit to each move. The credit assignment can be a difficult problem, observe the game can still be lost despite few optimal moves. The credit assignment process assigns the degree of gain or blame to each move given the state of the table. The second important property is the degree to which learner can control the training experience. In case of checkers, the learner either can rely on the teacher to provide informative board states and correct move for these or the learner has complete control over board states and training classifications. The third important attribute of training experience is how well it represents the whole distribution. In the example of checkers learning system that has learned by playing against itself might omit crucial moves which might be used by a human.

The second moment in developing a learning system is the choice of the target function, that is determining what kind of knowledge will be learned. In the example of checkers program, it can generate permitted moves from any board states, but the search algorithm is not known. Thus in case of checkers, learning program must learn to choose a move among permitted once, from any board state.

The third moment in the development of learning system is the selection of representation for the target function. Target function can be described as large tables describing value for each distinct board state, artificial neural network, set of rules, polynomial function or other. The goal is to pick very expressive representation to make approximation be as close to the ideal target function.

The fourth moment of the design is function approximation algorithm. The learning task is simplified to approximation algorithm, which has the aim of searching operational ideal target function. To learn target function, the algorithm requires set of training examples. In case of checkers, it is the examples of board states. In general, it might be very hard to learn such function, the algorithm only expected to produce an approximation to the target function. Learning of the function in other word is function approximation process. [1]

1.2.1.2 Learning systems background
There are three essential paradigms in the machine learning consisting of supervised, unsupervised and reinforcement learning[2].
Learning paradigm | Description
--- | ---
Unsupervised learning | This method is more alike to a biological learning. But this learning method is not suitable for all problems. The classifier is provided with the sample that has input data without any data about on the expected result. Such classifier is set with a task to identify the similarities and classify different patterns in the sample.

Supervised learning | Are such learning methods where the training sample consists of the input examples and the expected output. It requires both the input and the expected output data. For learning the classifier is provided the input and expected output for each input example.

Reinforcement learning | A distinction of a reinforcement learning algorithms, they use logical or real value as a pivot to improve its results.

Online - in online learning, weights are modified directly with each new sample.

Offline learning - in offline learning, it is also called batch training, the batch is pushed through the classifier and the error is accumulated. Then the weights are corrected based on the accumulated error. Such training section of a batch, in offline learning, is also called epoch.[2]

This thesis uses an Artificial Neural network (ANN), K-Nearest Neighbors (K-NN), Random Forests (RF) as learning algorithms. ANN is an algorithm which bases its structure and its mechanism on biological neural networks. ANN has a structure of a directed graph, where edges assigned with weights and each node represents a neuron. Nodes are grouped into an input, output, and hidden layers. Nodes behave in a similar way to neurons. KNN is an algorithm that is very alike to a database. It is called instance-based learner because its learning is consisting storing all the instances and at the classification use training instances to classify new entity. The K represents a constant which determines the number of instances of stored training examples which will be used in voting or another kind of process to determine the answer to a request. RF is an algorithm that uses a collection of decision trees. The learning process consists of the creation of a number of decision trees through a training process from given training data. The forest named random because of the particular method of creation of individual decision tree. Each tree is trained by a subset of training data which consists of randomly picked examples from the original data and has the same size as original training data set.

1.2.2 Artificial neural network (ANN)

Artificial neural networks, in the field of software, are structures and algorithms aimed to represent biological neural networks using digital computational machines. This section describes a simplified version of biological neural networks and its artificial counterpart described by [2].

1.2.2.1 Biological neural networks

A biological neural network consists of billions of similar cells. They have very similar structure and have an enormous amount of connection between each other, which these cells use to continuously communicate. The scientific term for such cell is a neuron, this terminology also used in artificial counterpart. Each biological neuron behaves as a
switch, it either forwards the signal further if conditions meet or stays idle. The tremendous information processing capabilities of biological neural networks are the result of coordinated signals send by a huge number of neurons and immense level of interactivity in between neurons.

Neuron cell is the main processing unit in a biological neural network. Neuron cells act like a switch, when stimuli reach a certain condition neuron propagates signal further on all outgoing connections. Figure 2 presents different components of the neuron. The incoming signals can come from other neurons or cells. The incoming signals are transferred through physical connections which are connected to neuron via synapses. Synapses are points of connection, the connection to neuron occurs through dendrite or directly to soma. Synapses are distinguished into two categories, the electrical and chemical. The main functional distinction between electrical and chemical, in the first case signal, delivered directly to the soma while in the second case the signal is transformed to a chemical signal and then back to electrical and then delivered to soma. Chemical synapse has physical shapes in form of a cleft and this break direct connection to the soma. The cleft also called synaptic cleft. In order for a signal to pass from presynaptic side of the cleft to postsynaptic side of the cleft, the electrical signal is converted to the chemical signal substance at the presynaptic side and converted back to the electrical signal at postsynaptic side. This mechanism is chemical and is able to modulate the signal transferred by excreting different type or quantity of pulse.

![Figure 1.1: Neuron structure](image)

Apart from being able to modulate a signal, the chemical synapse prevents signal moving in other direction. Dendrites are like branches of the neuron cell to which incoming connections are attached. Branching of dendrites is called dendrite tree. The cell body of a neuron called soma, it carries out the function of weighing the sum of all incoming signals and conditional propagation of electrical pulse. The condition required to propagate pulse is activated (excited) state when certain threshold reached. When a cell is activated pulse is sent along all outgoing edges. The nature of electrical signal can have a deactivating effect by inhibiting signals or activating effect through stimulating signal.

A neuron and schwann cell has the following life cycle: resting state, stimulus up to the threshold, depolarization, repolarization, and hyperpolarization. This thesis does not
describe in depth the biological mechanism of signal propagation which occurs between neuron cells. Biological mechanism of a neuron is based on the electrical charge which is built up on the inside and the outside of the cellular membrane (cellular wall), it is also called membrane potential. Sodium and potassium ions are the key components having an important role in the mechanism. There are two forces involved in the process which actively transporting ions in or out and affect the concentration of potassium and sodium ions inside and the outside of the membrane. The osmosis and permeability of cellular membrane work against by continually working to diffuse the concentration of the sodium and potassium ions. In addition neuron structure has sodium-potassium channels which are permanently open and controllable channels to maintain concentration or quickly diffuse concentration of potassium and sodium ions to equilibrium, on the inside and the outside of the membrane. Quick diffusion of ions results in discharge and activation of the neuron. Osmotic force is the force which exists in nature and always tries to arrange elements as uniformly as possible. Initially, the neuron is idle and the concentration of potassium inside of the cell is high. Controllable potassium and sodium channels are closed and potassium actively pumped in while sodium pumped out. Due to a permeability of the cellular wall, potassium and sodium ions continuously slips through the wall. The sodium, in the beginning, is pumped out, it also slips through the membrane but at much slower pace in comparison to potassium. The resulting concentration gradient of sodium on the outside and potassium on the inside gives rise to an electrical gradient. Stimulating impulses open some sodium channels. When a threshold of -55mV is reached, the action potential initiates opening of many sodium channels and sodium pours in and closing of potassium channels which are normally opened. After action potential initiated occurs depolarization because of the change in the intracellular and extracellular concentration. The massive influx of sodium also creates change in charge to approximately +30mV, the created pulse from sodium ion influx is the electrical pulse and action potential. After action potential reached the sodium channels are closed and the potassium channels are opened. The internal and external concentration moves to resting state in result to osmosis and sodium-potassium pumps, the process also called repolarization. Potassium channels are closed slower which results in mild hyperpolarization. The time required for a neuron to process signals again called refractory period, it is in range of 1-2 ms.

Communication between distant neurons or other distant communication to manage energy loses uses special kind of connection. Such neuron is an advanced dendrite which is termed, Axon. An axon can stretch up to one meter. Axon transfers information also to other kinds of cells and provides an arm of control. Axon in vertebrates normally coated with myelin sheath which is composed of schwann cells, myelin acts as an electrical insulator. The pulse is transferred in a saltatory way between schwann cells. Schwann cells are not coated at ends. The gap between schwann cells is 0.1-2mm and goes under name of nodes of Ranvier. At the nodes, polarization and depolarization can occur just as with soma (neuron body). The action potential of one activates action potential of next, from this comes respective name, saltatory conductor.

Information input into the network is done through specifically modified neurons, they go under scientific name receptor. Receptors are sensory cells, they generate action potentials based on external stimulus such as light, temperature, sound and so on. This conversion from external stimuli into membrane potential is called sensory transduction. The signals can be amplified at transduction or with the stimulus-conducting apparatus.

Receptors can be categorized into primary and secondary receptors. Primary receptors have a direct access to the neural system and the stimulus intensity through conversion
defines the action potential. The method of information transfer is alike to amplitude modulation used in analog communication. Example of use primary receptors is a sense of pain. Secondary receptors transmit pulses continuously, the pulse defines which transmitter and amount of neurotransmitter. The stimulus, in this case, is controlling the action potential frequency, that is information carried by the stimulus is encoded through the frequency modulation. Receptors also can form complex sensory organs, such as ears or eyes.

1.2.2.2 From biological neural network to artificial neural network
Technical model is a strong simplification of the biological model. The abstraction of the biological neural network provided as follows: A neural network consists of a large number of entities called neurons, which act as nexus points of a large number of inputs. A function of a neuron is to act as a switch which turns on when certain conditions met. The condition for activation is if the total of incoming stimuli at any time, except for refraction time, reaches a threshold the signal is propagated further. The signal sent in some cases allows amplitude or frequency modulation as a method of encoding information in the pulse. There are modified neurons which have the function of receptors. Receptors are classified between primary with direct access to nervous system and secondary which are processed. In both classes of receptors, the information sent is encoded. Sensory neurons can form complex sensory organs, eyes, ears are such examples. There is a high level of interconnectivity between the neurons, in other words, a dense graph with neurons as nodes. Some connections are unidirectional and some connections are bidirectional. Connections with chemical synapses provide weighting alike process for transmitted pulses. The abstraction above reflects on elements in biological networks which are transferred into the structure of an artificial network. The structure of an artificial network revolves around neuron construct which has input and output a unique switch alike function.

1.2.2.3 Training of the ANN
Training process in case of a classifier within a field of machine learning is a process of learning. A classifier as a learning system performs self-modification by adapting to the changes in the environment, in other words, it gains knowledge and stores by performing the self modification. Specifically for an artificial network, there are 7 methods to store knowledge:

1. Create new connections between the neurons
2. Remove connections between the neurons
3. Modify the weights assigned to each connection
4. Change the activation threshold
5. Select different activation function or propagation function or output function
6. Grow the network by adding new neurons
7. Delete existing neurons

The most common learning procedure only modifies the weights on the connections.[2]
Backpropagation

Backpropagation is one of the training methods that can be used to train artificial neural network (ANN) and belongs to a class of supervised training algorithms. The concept of backpropagation training is based on the propagation of measure classification error on the training sample backward. Backpropagation is used in combination with an optimization algorithm. There are variants of training algorithms that use backpropagation. The core of the concept is based on the adjustment of weights at each layer to reduce the resulting error while propagating the error.

1.2.3 K - nearest neighbors (K-NN)

KNN is another learning algorithm. KNN is an instance-based learner. An instance-based learners stores all training examples and does not create an explicit representation of the target function. The generalization in KNN is postponed until the new instance is needed to be classified. Each example of training data is perceived as a point in multidimensional space. When new examples are classified, K points which are closest to the new example are selected and used to determine the class of the new example. Such kind of algorithms is considered to be “lazy” because processing postponed until the classification is requested and training process does not include construction of an explicit description of the target function. The generalization happens only when an instance must be classified. K-NN also can be used to solve regression problems. K is the constant that defines the number of neighboring entities used in the decision process. In regression problems, any mathematical function can be used to provide a solution for a particular input. In classification problems, neighboring entities can vote on the class that will be used as a predicted result.

During the training process of KNN training examples are stored for later use. Samples are retrieved once classification is necessary. Each training example represents a point in multidimensional space. KNN classified as a lazy learner and it does not create any explicit description of a target function. The generalization takes place during the classification of new instances. The process of classification uses Euclidean distance. Euclidean distance is the length of a straight line between two points. KNN can be applied to discrete, real and symbolic values with respective processes of selection of neighboring examples, which must be created accordingly. In general, distance between two points is calculated as following:

\[
\text{Distance}(p_1, p_2) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}
\]

KNN selects K number of examples which are nearest to the closest points to the tested example. The most common value is returned as a result. When KNN classifier is applied on a continuous real-valued function, instead of returning most common value the mean is returned. [1]

1.2.4 Random forests (RF)

Random forest is the learning algorithm that makes use of decision trees in order to construct and represent the explicit target function from a data sample. A single decision tree consists of nodes and leafs. Each node in the decision tree incorporates a test which is defined during training time to splits training examples. The test is selected in such way
so the split of training set gives subset at each leaf node which contains examples of only one class or set of examples which is less impure than the original set. The leaves represent class labels in case of classification or averages of examples that end in the same leaf. Random forest algorithm during the learning procedure creates a collection of decision trees. During the learning, process algorithm can search or randomly select some or all of the splitting conditions that specify a test. Such splitting condition can specify attributes, thresholds, and values in the test. The great benefit with the decision trees is the simplicity to convert a tree into human readable if statements, which can greatly improve human understandability of underlying target function.

The current decision tree learning algorithm is comprised by ID3, some elements of C4.5 and some of the other recent decision learning algorithms. The strength of ID3 algorithm rests on the selection of an attribute to test for each node. The attribute that benefits process of classification must be selected at each node. ID3 uses a statistical approach to measure the contribution of each attribute as a divider. This statistical property termed with name information gain. Information gain measures how well a given attribute separates training examples. The information gain is calculated on every step of the tree growing, the attribute that has highest information gain is selected in the process. Information gain is computed with the use of entropy. Entropy with other words is impurity of the collection of elements. The entropy is calculated as follows:

\[
\begin{align*}
S &-\text{ collection} \\
c &-\text{ class} \\
Entropy(S) &= \sum_{i=1}^{c} -p_i \log_2(p_i) \\
v &-\text{ value} \\
S_v &-\text{ occurrence of value} \\
InformationGain(S,A) &= Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)
\end{align*}
\]

Tree learning algorithms face issues of tree size, missing attributes values, continuous value attributes, attributes with different costs. A decision tree learning can deal with categorical and continuous attributes. An example method for the continuous attribute is by partitioning continuous attribute into a discrete set of intervals. The continuous attribute can be dynamically converted to a boolean attribute, A into Ac. Ac then is A < c with c being a threshold. The distribution(implementation) of Random Trees(Random Forests) that is used in this thesis selects a random subset of the variable in which it defines a split.

1.3 Technical context

The thesis uses technical means provided by a combination of Microsoft Kinect for Windows system and machine learning technologies as the solution to the research question. Java run-time is used to host and execute the application. Microsoft Kinect system for windows is a natural interaction device, which is developed by MS (Microsoft). Kinect system is a system comprised of different technologies, which provides utility to detect body posture and identify and locate different body joints. The KiDiRA takes use of machine learning technologies to generate prediction model which represents a classification of the relationship between the expected and given posture. In general, the machine learning technologies provide an advanced mechanism to construct models which would describe certain aspects of the data domain.
1.3.1 Glossary

Classifier - in the context of machine learning, it is the implementation of an algorithm that is able to gain experience from sample data in order to perform classification on the data entities. Training set - set of elements which are used as sample data from a larger domain. Upon this set, classifier gains experience and learns the nature of the problem domain.[1]

Learning - learning in the context of learning systems, is a process of self-modification to adapt to a particular task with the aim to produce a better result. Definition: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.[1]

Training data - a dataset that consists of input X examples and sometimes a target examples Y.

Test data - the subset of training data.

Target dataset - is data set representing the target Y to the input X in the training data.

1.3.2 Kinect

Kinect is a natural user interaction system developed by Microsoft Kinect® (Microsoft Corp., Redmond). The Kinect initially was developed for the Microsoft Xbox 360 and was available for purchase November 4, 2010. Kinect provided an interface for voice interaction and body gestures. To realize the functionality the Kinect uses one RGB camera, a 3D depth camera and an array of microphones. The First Kinect for Windows was released February 1, 2012, with the price of $249 and with the special academic price of $149. The business model used is hardware only, this means there is no charge for SDK or runtime. SDK is available free to developers and end users. Microsoft provides free license to use the Kinect for window software, the ongoing software updates and the hardware needed for windows. Kinect for windows supports common Kinect features such as voice and gesture recognition. The Kinect for Windows was extended with a “near mode”. The released Kinect for the Windows was an adapted version of Kinect for Xbox 360. Currently, a Kinect for windows can be purchased for approximately of $150. [3]

V1.0 Kinect for Windows was provided with drivers for MS Windows 7 and non-commercial development kit (SDK). Kinect development capabilities allowed to develop and build applications with C++, C#, or Visual Basic by using MS Visual Studio 2010. SDK exposed features which provide raw data access to the audio and the video streams. SDK also provided access to skeletal tracking, that is tracking of the skeletal image of one or two people moving within a field of Kinect. Another feature provided by SDK was advanced audio capabilities and integration with Windows speech recognition API. Sample code and documentation were also included.

Release V 2.0 included the development app “Kinect Studio” and some features were improved. The voice recognition covered a wider spectrum of languages and accents. The skeletal tracking was extensively improved to track more joints and users. Kinect V 2.0 also included a new sensor. Kinect applications are able to run on windows 8 or windows 10.

Kinect for Windows is extended version of Kinect for Xbox to support applications on Windows platforms. The main feature of Kinect is an interface that recognizes gestures, voice commands, and system objects. The interface offers interaction without any
physical contact (natural, voice and gestures). Fig.1 displays components of Kinect. The displayed Kinect assembly consists of an RGB camera, an infrared-based depth camera, and four microphones. The Kinect software development kit is able to provide 3D information of user’s body joints and also recognize some voice pattern. In addition, the device can provide different depth image resolution. Available resolutions are depending on installed infrared-camera and infrared-projector. The 3D depth data can be used to recreate 3D objects on the scene. The Kinect device is complemented with MS Kinect software framework, which provides functionality to infer the position of body joints in the 3D space from the depth feed.

1.3.2.1 Kinect technical details
Microsoft Kinect device comes with software development kit (SDK) for Windows and Xbox. SDK is comprised of tools and APIs. The main features of Kinect system implemented through advanced software of image and sound processing tools. The feature this thesis interested in is ability to track human body and its joints. Kinect 2 SDK is able to track as many as 6 people and 25 skeletal joints per person. Additionally, the joint tracking feature provided by Kinect system is able to track joints of hand tips, thumbs and shoulder center.[4]

<table>
<thead>
<tr>
<th>Specification</th>
<th>View angle</th>
<th>Vertical tilt range</th>
<th>Frame rate (depth and color stream)</th>
<th>Audio format</th>
<th>Audio input characteristics</th>
<th>Accelerometer characteristics</th>
<th>RGB camera</th>
<th>Infrared depth camera</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viewing angle (kinect2)</td>
<td>60° vertical by 70° horizontal field of view</td>
<td>±27°</td>
<td>30 frames per second (FPS)</td>
<td>16-kHz, 24-bit mono pulse code modulation (PCM)</td>
<td>A four-microphone array with 24-bit analog-to-digital converter (ADC) and Kinect-resident signal processing including acoustic echo cancellation and noise suppression</td>
<td>A 2G/4G/8G accelerometer configured for the 2G range, with a 1° accuracy upper limit.</td>
<td>Stores three channeled data in resolution 1280x960</td>
<td>Stores one channel data in resolution 320x240;640x480;80x60</td>
</tr>
</tbody>
</table>

Table 1.1: Specifications for the Kinect [5]
Table 1.1 provides technical characteristics for the hardware components of the Kinect. The operational range is between 0.8 - 4 meters. The latest novelty is a depth camera with a resolution of 640x480. The best accuracy is achieved by positioning the user within 1.2 - 3.5-meter distance in front of the camera. The depth is measured through a process of triangulation. An infrared projector casts a pattern upon the objects in front of the device, while the infrared camera uses the patterns to evaluate the distance between the point on the object and the device.

1.4 Previous research in telerehabilitation

[7] provides an example that utilizes Kinect to provide telerehabilitation. [7] develops Kinect based telerehabilitation system (KiReS) which is aimed to facilitate needs of physiotherapists in the process of creating, designing, managing, assigning and evaluating physiotherapy protocols and sessions. KiReS also aimed to inspire and provide intuitive exercise interface. Also to provide useful feedback to enhance telerehabilitation process. Prior to our research, we looked into: [8] which is concerned with recovering posture from single 2D images;[9] which concerned with Real-Time Human Pose Recognition from single depth image; [2] Introduction to neural networks. [8] is about extracting body posture 1 from single images or monocular image series with the use of non-linear regression.[8] applies Relevance Vector Machine (RVM), Support Vector Machine (SVM) and other. [9] introduces extraction of body posture from single depth image with use of Random forests. [2] goes through a variety of the classification and regression methods with focus on Artificial Neural networks. [2] As well compares ANN with the biological neural networks and describes details about biological standpoint on bain and its function. [2] mentions the usage of a Multilayer Perceptron to accommodate a more complex pattern. Depth cameras are the best tools to create a 3D view or to detect skeleton using Depth map. Microsoft Kinect created many opportunities in multimedia computing. It is an easy and cheaply available depth camera. Microsoft and other researchers have done a lot of research on depth image using Kinect[10]. In the following article, writer discussed in details about skeletal tracking using Kinect [11].
1.5 Problem formulation

The goal of the research is to generalize machine learning algorithms to evaluate posture performance in real-time. Investigate the suitability of Artificial Neural Networks, Random Forests and K-Nearest Neighbors as a solution. We are using Microsoft Kinect technology with its SDK to detect posture and provide a joint location. Upon this data, we are estimating the posture. Sample postures: forward bend, spinal twist, side stretch, calf.

1.6 Motivation

Development of an application that can provide an offline physical therapy is a potential solution to the drawbacks of wearing sensors. Wearing sensors is uncomfortable for some individuals. The capability of offline physiotherapy expands possibilities for the remote therapy. Action recognition is one of the widely studied fields in the computer vision research group. The research can be commercially applied in surveillance, health, gaming and in many other fields. There are currently many companies working in skeletal detection like LifeSymb in Sweden providing health applications using Kinect. People have used different methods and techniques to detect the posture from Kinect skeletal stream. Our motivation is to compare the performance and accuracy of different machine learning approaches like Artificial Neural Network, KNN and Decision Forest in posture comparison.

1.7 Research Question

<table>
<thead>
<tr>
<th>RQ1</th>
<th>Which of the machine learning methods ANN, Random Forest networks, KNN, SVM is the most accurate for classification of the postures?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1a</td>
<td>What size of training set is required for ANN, Random Forest networks, KNN to provide a ratio of successful classification i.e 95%</td>
</tr>
<tr>
<td>RQ1b</td>
<td>What time is required for a particular ML to classify a depth frame?</td>
</tr>
</tbody>
</table>

1.8 Scope/Limitation

In this thesis, we are limiting us to following machine learning techniques: That is Artificial Neural networks(ANN), Random Forests(RF), and K-Nearest Neighbors(K-NN). The Kinect provides a 3d depth map of the joints in the room, the thesis tests three different types of data upon which it will base its classification. The first type is 38 feature arrays with angles describing the body configuration of two postures, the second is the deviation between two postures in 19 features vector and the third deviations presented as percentages in 19 features array. The application uses angles to describe body posture, which is based on joint data. To simulate exercise, movements were prerecorded and the skeletal information replayed from CSV(comma separated values) as from Kinect. We will devise some structure by which other opensource frameworks can be pluggable to our prototype and instead of skeletal tracking data. However, this thesis restricts test set to normal CSV.
1.9 Target group

Scientists might have used if they are applying machine learning methods such as ANN or RF or KNN to solve a classification problem. Health Sector, because the developed application might provide insight into the estimation of physical condition. Gaming sector, because it allows new types of challenges introduced to the game.

1.10 Outline

This thesis contains the following sections: Introduction, Review of Kinect Digital Rehabilitation Assistant, KiDiRA Component technical details and validation, Discussion, Conclusion. The Introduction section covers details about the problem background that is being researched, who might benefit from the research and be interested in the solution. Review of Kinect Digital Rehabilitation Assistant section describes the architecture of the application and its functionality. KiDiRA Component technical details and validation provides details about implemented design and mechanics of the developed app and models used. The discussion section comprises thoughts and ideas with regards to the solution. Also, discussion contains details, why certain things are done in a certain way and things that could be done differently. Also, discussion section contains personal interpretation with regards to result is given.
2 Review of Kinect Digital Rehabilitation Assistant

Kinect Digital Rehabilitation assistant is the result of this research work. It combines aspects presented in this thesis. This chapter is centered on functionality and architecture of KiDiRA. KiDiRA is a Kinect based telerehabilitation system which covers the creation of exercises and artificial feedback to exercise sessions to enhance rehabilitation.

2.1 Scientific approach

Our research method is based on an inductive research model. We observe the performance of our subjects, which are our machine learning methods. The context, in which they operate, is this particular application problem of offline physiotherapy, that we described earlier. We are measuring: size of training set required to achieve 95% and higher classification rate; time required to train; time required to process 24 frames of features. These metrics are to be analyzed and conclusion would give a tentative hypothesis on our potential candidate for a solution of such problems.

2.2 Architecture

KiDiRA provides graphical interface and features for physiotherapist and patient. Realization of KiDiRA is split into four modules, the graphical interface provides functions define posture through use of controls and preview of the posture, then store and retrieve. The posture can be exported and imported as a file or stored in a database. The implementation of KiDiRA separates graphical user interface for Physiotherapist and the patient.

2.2.1 Exercise spatial constraints

The application is constrained by the field of view supported by the Kinect camera. The blue and light brown sectors originating from the left side of the room depict the field of view of Kinect. The white squire and pointers define distribution from which the samples are artificially are collected, as if the person was there.

Figure 2.3: Kinect field of view
2.2.2 KiDiRA use cases

This section provides overview of use cases and actors. The patient actor is a person who is the receiver of treatment. The therapist is an actor who will be crafting the therapeutic procedures. The use cases have same priority but the main focus was on the "Execute Exercise", the "Define posture" and the "Create Exercise" use cases.

![Class diagram exercise definition component](image)

**Figure 2.4: Class diagram exercise definition component**

2.2.3 Components of applications

The application divides functionality into six components. The Graphical interface component encapsulates implementation of graphical interfaces for physiotherapist and the patient. Posture and exercise persistence functionality is provided by Persistence component, which uses SQLite and provides data management functionality. Additionally, entities can be imported and exported with functionality provided by Entity Export & Import component. Classifier Container component encapsulates classifiers used to evaluate the performance of patient during therapy exercise. Kinect Connector component is symbolic and presents an entry for Kinect device. The component Body Description manages description and specification of posture features 2.5.
The Persistence component provides access to database and CRUD functionality to manage data entities. Entities are persisted on SQLite db file with use of JDBC driver for SQLite. The application stores posture and exercise entities and relation in between them, see entity realization diagram 2.6. The stored values use metric system unit mm.
The persistence of data and manipulation of persisted data occurs through a data access object (DAO), which is implemented within DAO class. DAO class manages specification of relational tables and CRUD functionality. DAO class implements interface IDatabase, which outlines generic methods required to create, read, update and delete entities. DAO object is not safe for multithreaded use. The component exposes functions as described on class diagram 2.7.
Figure 2.7: Class diagram of DAO class and interface

- public Entity create(Entity entity) - create new persistent entity in database as in provided entity, and return newly created entity.
- public Entity read(Entity entity) - return persisted entity with id equals to specified entity
- public Entity update(Entity entity) - update details of entity in database or creates if not found
- public void delete(Entity entity) - deletes entity with provided id from database

The Graphical User Interface (GUI) component serves a function to provide interaction through a graphical interface. Implementation is based on Java AWT, Swing and jogl (Open GL with Java). The GUI component provides role-based separation of function between graphical segments of application. The role of therapist facilitated with graphical assistance in posture and exercise creation. The role of patient facilitated with a visual aid for therapeutic exercises and presentation of other features accessible to the patient. The graphical controls for therapist visualize the managed posture as skeleton and exercise as a vertically aligned selectable list. The user is greeted at entry page which offers an option to proceed to the section for patient or therapist. The GUI is composed of frame class which contains four views. Entry view contains navigation buttons to patient view and therapist view. From the patient view, the user can proceed to exercise view.

The frame class exposes following methods:
- void openTab(TAB_ID id) - open specified TAB, available tabs :PATIENT_TAB, THERAPIST_TAB, ENTRY_TAB, EXERCISE_TAB
- void startExercise(Exercise id) - open EXERCISE_TAB with provided exercise
• void startExercise(Posture id) - open EXERCISE_TAB with provided Posture

Figure 2.8: Partial class diagram of UI

Evaluation component provides implementation of exercise performance classification with use of machine learning and with use of explicit function. The component instantiates classifiers from image which is provided with a file. The image is created with other separate program (Classifier trainer). Evaluation component exposes only one method. The method allows subscription. Through ClassifierCallback interface classifier collects feature vector and updates label on graphical interface.
• `void:subscribe(CLASSIFIER_TYPE type, ClassifierCallbacks callbacks)` - initiates classifier is separate thread, then through callbacks feature vector is collected and respective label is updated on graphical interface.

Exercise definition component bears function of providing the data structures needed to specify and represent the posture and exercise. The component also provides utility to access and handle posture data in a more abstract way by defining a name for each attribute and exposing an interface for access.
Figure 2.10: Class diagram exercise definition component

Posture Import/Export component implements functionality needed to import and export postures and exercises as files. The files are exported as csv files.

- void export(Posture p, String destination) - exports posture to file,
- Posture importFromFile(String from) - imports posture from file
- Posture importFromFile(File from) - imports posture from file
• void export(Posture) - prompts option dialog and saves under provided destination
• Posture importPosture() - prompts file to choose and then imports from provided file

2.3 Therapy planning

In rehabilitation, before therapy procedures can be specified, a therapist performs an assessment of the patients' condition. KiDiRA is a digital platform that provides a very simple facility for a therapist to create exercises and a platform for a patient where a patient can select and run exercises.

![Figure 2.11: Therapist panel](image)

2.3.1 Creation of new exercises

KiDiRA offers an interface for the physiotherapist that provides a simple control panel to specify a posture. The interface is simple with raw inputs. The interface presents skeleton with selectable joints. The therapist is able to position of each joint in relation to each other. The therapist has the option to view skeleton from different angles to create an
exercise physiotherapist needs to specify a location for each joint. The initial posture of
the skeleton is standing. Control panel presents with inputs for X, Y, Z values for joint
locations, joint locations can be browsed through left and right arrows on the keyboard.
Once the joint is selected it starts to flash. Additionally, by selecting the display, mouse
wheel can zoom in and zoom out the skeleton.

2.3.2 Exercise monitoring

The application provides therapy as exercises which are defined by a therapist. An ex-
ercise consists of series of postures which patient should mimic. On screen, the patient
provided with expected posture and his avatar. During the exercise, the patient is being
monitored for real-time feedback and to keep track of progress per posture. The progress
of posture performance is presented as a label on the screen to supply user feedback on
how well the posture is performed. The application also presents patient with success
indicator.

![Exercise panel](image)

Figure 2.12: Exercise panel

3 KiDiRA component technical details and validation

This chapter describes underlying mechanisms and structures in KiDiRA. KiDiRA con-
sists of three main components and two minor components. The first main component
labeled as Exercise Definition. It defines data structure of posture and exercise and pro-
cedure for calculation of the posture features used in exercise performance. The second
component, labeled as Evaluation Module, encapsulates mechanism used in exercise per-
formance. Evaluation Module encapsulates explicit function and alternative methods.
The third main component, labeled as a Graphical User interface, encapsulates graphical interface for patient and therapist.

### 3.1 Exercise performance classification

A prescribed exercise consists of series of postures which patient is expected to perform while being monitored. The application uses 19 metrics to measure the performance of patient repeating given postures. Each metric is an angle, which is a result measure of the angle between body limbs and angles between limbs and plane spanned by the set of joints. All data is extrapolated with Kinect skeletal joint detection feature. Kinect API provides a real-time, with a frequency of 30 fps, detection of joints on 3d depth image. The measurements provided for individual measurement consists of depth measurement z, and x,y which are position on the video frame. The angles are defined as follows:

1. Angle between left shoulder- shoulder center - head
2. Angle between right shoulder- shoulder center - head
3. Angle between head -shoulder center -spine
4. Angle between left shoulder -shoulder center -spine
5. Angle between right shoulder- shoulder center - spine
6. Angle between left shoulder - shoulder center and plane spanned by vectors shoulder center -shoulder right and shoulder center -spine
7. Angle between right shoulder - shoulder center and plane that is spanned by vectors, shoulder center -> left shoulder ; shoulder center -> spine
8. Angle between right hip -spine -shoulder center
9. Angle between spine - shoulder center and plane spanned by vectors: Spine -> left hip, spine -> right hip
10. Angle between left hip - right hip- right knee
11. Angle between R hip -L hip and L hip - L Knee
12. Angle between L hip - left knee and plane spanned by vectors: L hip -> R hip, L hip -> spine
14. Angle between right hip-right knee-right ankle
15. Angle between left hip- left knee - left ankle
16. Angle between L knee - L ankle - L foot
17. Angle between R knee - R ankle - R foot
18. Angle between vectors R foot -> R ankle and R hip -> L hip
19. Angle between vectors L foot -> L ankle and R hip -> L hip
Figure 3.13: Angle 1 and 2, calculation with use of scalar product

Figure 3.14: Angle 3 and 4, calculation with use of scalar product
Figure 3.15: Angle 5 and 6, calculation with use of scalar product

Figure 3.16: Angle 7 and 8, calculation with use of scalar product
Figure 3.17: Angle 9 and 10, calculation with use of scalar product

Angle between spine - shoulder center and plane spanned by vectors: Spine -> left hip, spine -> right hip

Figure 3.18: Angle 11 and 12, calculation with use of scalar product

Angle between R hip - L hip and L hip - L Knee

Angle between L hip - left knee and plane spanned by vectors: L hip -> R hip, L hip -> spine

Figure 3.19: Angle 13 and 14, calculation with use of scalar product

Angle between right hip - right knee - right ankle

Angle between left hip - left knee - left ankle
Figure 3.20: Angle 15 and 16, calculation with use of scalar product

Figure 3.21: Angle 17 and 18, calculation with use of scalar product
3.2 Exercise performance classification method

The application provides four methods to classify posture performance. The first method is a simple use of a function to calculate angles, then to compare angles and to convert difference of angles into percentage and in the end selecting the respective label. The
second method is to use an artificial neural network to produce the mapping between expected, observed vectors and the label or a vector in case of the artificial neural network. The performance classification is based on comparing two postures, the expected posture, which is displayed on the screen, and the posture of a patient attempting to perform expected posture. The application uses as earlier mentioned 19 measurements to describe posture.

3.2.1 The explicit classification function

The user exercise performance is measured as deviation between angles of expected and observed posture. The resulting deviation converted to percentage and further on to label, which presented on the graphical panel. Inner limit vector 
\[ \vec{Z} = (z_1, z_2, z_3, \ldots z_n) \] 
and outer limit \[ ZO = (maxA_1, maxA_2, maxA_3, \ldots maxA_n) \] 
plays role in conversion between deviations and percentage. The values in inner limit vector define a bar which counted as a 100% mismatch. The formula takes two input vectors. The values in \[ \vec{Z} \] calculated with use of arbitrary defined limit on per angle. The limits are presented in table 3.2, and \[ z_n = maxA_n/2. \]

Expected posture descriptor \[ \vec{X} = (x_1, x_2, x_3, \ldots x_n) \] 
Avatar posture descriptor \[ \vec{Y} = (y_1, y_2, y_3, \ldots y_n) \] 
One vector subtracted from another as described in 2. In the following step the reminder is divided by \[ z_n. \] The function \( t() \) trims the result of division if it larger than 1.

\[
t(x) = \begin{cases} 
1 & x > 1 \\
0.5 & x \leq 1 
\end{cases} \quad (1)
\]

\[
f(\vec{X}, \vec{Y}) = 1 - \left( \frac{1}{N} \sum_{n=1}^{N} t\left( \left| \frac{x_n - y_n}{z_n} \right| \right) \right) \quad (2)
\]

The equation 2 calculates and specifies fraction to which two postures matched.

\[
g(f(\vec{X}, \vec{Y})) = \begin{cases} 
\text{Fail class} & f(\vec{X}, \vec{Y}) \leq 0.4 \\
\text{Pass class} & 0.4 < f(\vec{X}, \vec{Y}) \leq 0.6 \\
\text{Second class} & 0.6 < f(\vec{X}, \vec{Y}) \leq 0.8 \\
\text{Outstanding class} & f(\vec{X}, \vec{Y}) \leq 0.8 
\end{cases} \quad (3)
\]

3.2.2 The implicit classification via machine learning

The application provides multiple algorithms to perform tasks of classification. Algorithms share same input training set which is extrapolated with help of explicit function, explained in function above. The input vector consists of percentages which present deviation between expected angle and observed. The measurements are taken continuously while patient is performing a posture in real time. The percentage calculation is based on equation 2.

\[
f : X \rightarrow Y \quad (4)
\]

\[
\vec{X} = \{ \vec{X} \in R^{19}, \vec{Z} \in R^{19} | 0 \leq x_n \leq maxA_n \} \quad (5)
\]
\[ \vec{X} = \{x_1, x_2, x_3, \ldots, x_n\} \]  \hspace{1cm} (6)

For ANN \( Y = \{R^4|y = (0|1)\} \)  \hspace{1cm} (7)

for KNN and Random Trees \( Y = \{R^1|y \in \{1, 2, 3, 4, 5\}\} \)  \hspace{1cm} (8)

### 3.2.2.1 Creating the training set
The classifiers are trained by a separate program, which implements the training set generator and executes the training procedures. The trained classifiers are then stored and loaded into KiDiRA. The training set generator comes with a graphical interface that allows specifying some metrics of the training set and visualization of distribution. The classifier trainer is embedded into the same application as the set generator.

![Figure 3.24: Interface with data statistics. The blue horizontal bars present distribution of classes, the green represents distribution of numbers 1-100](image)

### 3.2.2.2 Training set Generation algorithm
The set generation algorithm produces artificial training data set with the desired number of rows. The generator uses explicit classification function as a base, which explained in section 3.2.1. The explicit function and with specified arbitrary boundaries for the joints we define and limit the domain of hypothetical exercises that can be created by a doctor. The training dataset then contains randomly picked examples of observed, expected vectors. The training examples of a different kind are randomly selected in a sequence. The set generator function is not completely random and constrained to have an equal distribution of examples per class.
The algorithm produces a distribution of classes in the data set, by generating examples of the desired class. The final set contains vectors of the first class, the second class and so forth. The algorithms produce six files, where the three files contain input vectors and three with the target value. Each of the input data file presents vectors of one particular feature vector type. The with type with three different files with respective target values for each classifier. The algorithm takes as input number of training patterns to produce and output destinations. The distribution is presented on figure 3.25.

To produce input data sets algorithm used arbitrarily defined body model. Body model defines maximum and the minimum value in degrees a particular joint can take.

The algorithm starts by selecting a feature vector where each value expresses the percentage of success. In the next step, the algorithms retrieve upper bound and lower bound of the total sum for percentages for a particular class. The upper bound per class is produced according to formula 9. Upper bound is a maximal value of sum of all items in the feature vector. The upper bound used as a pool and then distributed as show in function generateConstrainedRandomPercentageVectorVector().

\[
\text{upper bound} = \begin{cases} 
\text{Fail class} & 1900 \times 0.4 \\
\text{Pass class} & 1900 \times 0.6 \\
\text{Second class} & 1900 \times 0.8 \\
\text{Outstanding class} & 1900 \times 1 
\end{cases}
\]  

(9)

The upper bound used as a pool which is distributed in function generateConstrainedRandomPercentageVectorVector(int[] percentages, int[] bounds,int desiredClass,int base). The function uses standard method, provided by java, to generate random numbers within range of 0 - 100. The first step is to distribute values by using the random function to assign a value within the range to each position in the feature vector without overstepping the upper bound. After successfully assigning a value subtracted the assigned value from the pool(upper bound) and repeat until the subtraction yields value below zero. If after the previous procedure the pool didn’t reach zero, the following routine is used. The pool is attempted to divide evenly between all the cells in the vectors. The pool is then set to zero. If the portion, which is the result of the division, added to a cell makes value greater than the base value(100), the exceeding amount is removed from the cell and returned to the pool. The starting index is randomly selected. The procedure is repeated until the pool has reached zero.

The next part of algorithm uses the generated percentage vector and the body model to produce deviations. The body model provides set of arbitrary selected body metrics which define maximal size of angle that joint can span. The following list specifies maximal value of a joint.

<table>
<thead>
<tr>
<th>maxA1 = 110;</th>
<th>maxA2 = 110;</th>
<th>maxA3 = 180;</th>
<th>maxA4 = 110;</th>
<th>maxA5 = 110;</th>
<th>maxA6 = 20;</th>
<th>maxA7 = 20;</th>
<th>maxA8 = 180;</th>
<th>maxA9 = 45;</th>
<th>maxA10 = 180;</th>
<th>maxA11 = 180;</th>
<th>maxA12 = 180;</th>
<th>maxA13 = 170;</th>
<th>maxA14 = 170;</th>
<th>maxA15 = 110;</th>
<th>maxA16 = 110;</th>
<th>maxA17 = 180;</th>
<th>maxA18 = 180;</th>
<th>maxA19 = 180;</th>
</tr>
</thead>
</table>

Table 3.2: Maximum boundary for angle attributes

The algorithm uses variable sensitivity which reduces provided max to double and makes smaller value to represent a 100% deviation from desired value. The deviation follows the formula described on figure 10.

\[
\text{deviation}_i = \frac{\text{maxA}_i}{\text{sensitivity}} \times \frac{100 - \text{percent}_i}{100.0}
\]  

(10)
The following step in algorithm produces feature vector based on angles of joints. This feature vector will be set with data which represents observed angles and expected angles. The algorithm randomly selects input value from range per joint as specified with use of body model as described above in table 3.2. The lower bound for ranges is zero. The algorithm sets values according to following code:

```
// deviation – array of generated deviations, absolut of expected and observed angle
/* Get body model instance */
BodyModelImpl bm = new BodyModelImpl();
/* Get joint angle upper limits */
List<Integer> list = bm.getUpperLimit();
int[] angles = new int[deviation.length*2];
Random rm = new Random();

for (int i = 0; i<angles.length/2; i++){
    angles[i] = rm.nextInt(list.get(i));
    /* If the sum of angle and deviation is above upper limit for joint
     * the expected angle set by subtracting the deviation from observed angle
     * else set expected as with sum of observed angle and deviation */
    if (angles[i]+deviation[i] <= list.get(i)){
        angles[i+19] = angles[i]+deviation[i];
    }
    else{
        angles[i+19] = angles[i]-deviation[i];
    }
}
```
Figure 3.25: Interface with data statistics. The blue horizontal bars indicate class distribution in the training dataset, with the first line presenting occurrence of 5th class, the second line present first class and so on till 4th class. The green are occurrence of numbers 1-100. The number that is given divided to estimate the size of a single fold. The number of folds is an arbitrary number 3. Each fold is divided to allocate the same size of rows per class.

```java
/**
 * Generates and writes specified number of feature vectors
 * with an equal portion of rows for each class.
 */
public void generatorMethod1()
{
  /* Initiate output writers for angle vectors, deviation vectors, percentage vectors, target vector for KNN, ANN, RTrees*/
  ....

  int[] annTarget = {0.0, 0.0, 0.0};
  int[] kNNLabel = new int[1];
  int[] rTreesLabel = new int[1];
  // Number of classes
  int classCount = 5;
  // Number of folds
  int folds = 3;
  // Size of one fold
  int delta = size/(folds*classCount);
  // bound limiting of generation of entries belonging to current class
```
int bound = delta;
// Current class category to which entities adhere
int desiredClass = 1;
// Maximal permitted value in each bin of percentage vector
int base = 100; /* 0–100 range of percentage value */
float sensitivity = 2;
for (int i = 0; i < size; i++){
  if (bound < i)
    desiredClass = ((desiredClass+1)%classCount)+1;
  bound+= delta;
}
// Generate percentage vector
percentages = generateRandomPercentagesVectorMethod1(percentages, length, desiredClass, base);
// Generate deviation vector from percentages
deviations = generateDeviations(sensitivity, percentages);
// Generate angle vector from deviations
angles = generateAngles(deviations);

kNNLabel[0] = getLabelAsInt(getTotalPercentage(percentages));
rtreesLabel[0] = getLabelAsInt(getTotalPercentage(percentages));
annTarget = getAnnTargVector(getTotalPercentage(percentages));
// Print all the generated input and target arrays
...
```java
int pool = bounds[1];
ext = 0;
Random rm = new Random();
for (int i = 0; i < percentages.length; i++) {
    next = rm.nextInt(depth);
    if (pool - next > 0) {
        percentages[i] = next;
    } else {
        break;
    }
pool-=percentages[i];
}
int portion = 0;
tmp = 0;
while (pool > 0) {
    portion = (int)((double)pool / percentages.length);
pool = 0;
    randomStart = rm.nextInt(percentages.length + 1);
    for (int i = randomStart; i < percentages.length + randomStart; i++) {
        percentages[(i % percentages.length) + randomStart] += portion;
        if (percentages[(i % percentages.length)] > base) {
            tmp = percentages[(i % percentages.length) + randomStart];
            pool+=tmp;
            percentages[(i % percentages.length)] = tmp;
        }
    }
}

/**
 * Creates deviations from percentages
 * @param length
 * @param percents
 * @return
 */
private int[] generateDeviations(int length, int[] percents) {
    BodyModelImpl bm = new BodyModelImpl();
    int[] deviations = new int[length];
    List<Integer> outerLimit = bm.getOuterLimit();
    for (int i = 0; i < length; i++) {
        deviations[i] = (int)(outerLimit.get(i) / (100 - percents[i]) / 100.0);
    }
    return deviations;
}

/**
 * Create array of angles with length as double as long, then of array containing
deviations provided
 * @param deviation
 * @return
 */
private int[] generateAngles(int[] deviation) {
    BodyModelImpl bm = new BodyModelImpl();
    List<Integer> list = bm.getOuterLimit();
```
```java
int[] angles = new int[deviation.length * 2];
Random rm = new Random();
for (int i = 0; i < angles.length / 2; i++) {
    angles[i] = rm.nextInt(list.get(i));
    if (angles[i] + deviation[i] <= list.get(i)) {
        angles[i + 19] = angles[i] + deviation[i];
    } else {
        angles[i + 19] = angles[i] - deviation[i];
    }
}
return angles;
```

3.2.2.3 The implicit classification function via ANN

The Neural network used to consist of 5 layers with input and output included. The network has feedforward topology with layers being completely linked. The input layer consists of 19 neurons followed by the hidden layer with 39 neurons and followed by another hidden layer with 39 neurons and another with the size of 20. The output layer consists of 5 output neurons. The training method which used in training is backpropagation. The training occurs in offline mode. The termination criteria for artificial neural network training procedure is set to a maximum of 1000 permitted iterations and minimal of error change between iteration to 0.01. Neural network uses symmetrical sigmoid function as activation function with parameter1 and parameter2 being set to 1. See equation 11, symmetric sigmoid function.

\[
f(x) = \beta \frac{1 - e^{-\alpha x}}{1 + e^{-\alpha x}}
\] (11)

The ANN classifier is trained to return one of 5 vectors. Each value in the vector represents the expected output of neurons in output layer. The (1,0,0,0,0) stands for ideal output for class 1, (0,1,0,0,0) stands for ideal output for class 2, (0,0,1,0,0) stands for ideal output for class 3, (0,0,0,1,0) stands for ideal output for class 4 and (0,0,0,0,1) stands for ideal output for class 5. The class id is constructed by taking index from a neuron which gives the highest output and then adding one. This is used as class id and further on used as a result of classification.

The used distribution of ANN is ANN_MLP from openCV 3.0. It return array as Matrix 5x1. The array with floats ranging between -1 and 1. The result array of prediction then analyzed, and id of label is extracted as described below. The output neuron with highest value produced by the activation for the given input.

```java
private String getLabel(Mat result) {
    final int rows = result.rows();
    final int cols = result.cols();
    FloatBuffer fb = result.createBuffer();
    float[] arr = new float[cols];
    fb.get(arr);
    int index = 0;
    float max = -2;
    for (int i = 0; i < cols * rows; i++){
        if (max < arr[i]){
            max = arr[i];
            index = i;
        } else {
            index++;
        }
    }
    return (index - 1) / (float) (cols - 1);}
```
3.2.2.4 The implicit classification function via Random Trees  The Random Trees classifier which is used in the application has a common bootstrap procedure. The bootstrap procedure in Random Trees uses a process which randomly selects a subset of training examples with replacement with the size of the training set and then training decision tree on this set. Individual decision tree during training treats the task as regression. The application uses the default method of best split search. The default method sets a fixed number of variables to find the best split at each node. The fixed number is produced according this formula $\sqrt{\text{number of variables}}$. There is also no pruning used. Termination of training procedure occurs when out of the bag (OOB) error gets as low as 0.1 or the maximal number of random trees reaches 50. The Random trees distribution(3.0) from openCV applied regression as the solution to generalization. Classification with RTrees returns mathematical average from voting, which is rounded up or down. The rounded value is treated like class id. The rounding procedure going as follows.

```java
double threshold = 0.5;
...
double result = rTrees.predict(input);
if (raw){
  return result;
} else{
  if (result - (int)result > threshold){
    result = result +1;
  return (int)result;
  } else{
    return (int)result;
  }
}
```

3.2.2.5 The implicit classification function via K-NN  The used K-Nearest Neighbors classifier works in the following manner. The training is a simple procedure to cache all training examples. When the classification is performed, the specified constant K specifies the number of cached entries to be used, which are nearest in the vicinity to the example being classified. The classification occurs through voting. The domain of each variable is continuous data space limited by specified range.

The classifier directly returns class id which is converted to a label for display on working graphical panel. K-NN classifier returns directly value which identifies a label.

3.3 Classifier performance measurements and measurements extraction procedure

The thesis takes four measurements, the training time, the classification ratio on training data, the classification during the execution of exercise and the frequency No real Kinect is used but instead Kinect behavior is implemented through KinectDeviceSimulator class which uses separate thread to read prerecorded posture data file from Kinect and updates posture settings on avatar panel. The updates of the avatar are set to occur with a frequency
of 10 times a second. The collected data is persisted for calculation of average frequency and classification ratio. Frequency is calculated by taking time stamp after and before \( \frac{1}{t_1 - t_0} = f \). Calculation classification ratio achieved by running feature vector through formula and the classifier. The graph 3.26 show training times of classifiers. The names of training measurement on 3.26 should be interpreted as following. The names that contains ex1 are the measurements done from training on feature vectors based on arrays of angles and analogously ex2 but based on deviation between angles of expected posture and observed posture. The entries with ex3 marker are examples are taken from training on feature vectors based on percentages. After the feature vector marker comes values specifying the number of rows in the training set.

![Graph](image)

Figure 3.26: Graph presenting normalized time required to train algorithms on different data sets and classification ration on the trained data
Figure 3.27: Classification ratio and classification frequency during simulation of exercise. ex1 - classifier trained on angles, ex2 - classifier trained on deviations, ex3 - classifier trained on percentages. 1000k, 800000, 100000 specifies size of training set.
3.4 Reliability and Validity

The thesis controls variables which include the training data, the test data, type of execution environment and configuration of each classifier. The training data has fixed sizes, each vector adheres to specification and the specified domain.
The experiments were executed in the environment of Windows 7 home premium operating system, with hardware AMD Phenom quad-core 1.8Ghz and with 8Gbytes of working ram. The application based on java platform of version 1.8.0_05, Java Se Runtime Environment 1.8.0_05-b13, Java HotSpot 64-Bit Server VM<built 25.5-b02, mixed mode>. The training speed is relative and will differ on other hardware setups because the amount of ram and data access time from hard drive has a strong effect on the KNN algorithm and will boost its classification frequency and might also affect classification ratio. As means to increase reliability, this thesis uses different training data sizes and different kind of feature vectors while comparing the classifiers. The variable attributes are the classification frequency, classification ratio, training time which will differ between experiment with a different classifier.

4 Discussion

The task of the thesis was to find the most suitable machine learning algorithm from a specified collection as a solution for a real application problem. The domain classes were clearly outlined from the beginning while the range of input arguments was undefined. The classification problem was to classify two arbitrary postures. It was approached in four different ways. The task of developing the telerehabilitation application wasn’t the main focus and the aim was to develop only the necessary features to a certain extent. The essential features are related with creation and running of the exercise. Selecting proper settings for classifiers seemed like complex task and in order to achieve something tangible more pre-processing of the input was added to reduce the complexity of the classification goal.

The criteria that make a classifier suitable for this application is sufficient classification speed, classification ratio and training time. To investigate which classifier would suit better for the task, this thesis runs generalization of classifiers on different training data sets. This thesis takes measurements on performance during the training and during application execution. The training data, which was used in research, is varying on the length of feature vector and feature vector type and size of the training dataset. This thesis runs and takes measurements on classifiers which were trained on angles, angle deviations, and percentages as feature vectors. Result section describes the use of percentages as feature vectors. Training sets with angles as feature vectors comprised of an array of observed angles in posture and angles from provided posture.

Training data sets were composed through a semi-random sampling of vectors. The training sets have equal or almost equal class distribution within the set. The formula to generate the target data set as described above in 3.2. The training set generator selects values for the feature vector randomly from a domain. The respective target class is then set with help of the explicit target function. Creating a function which would describe a relation seemed simpler in the case of comparing two postures with a result of the operation being a class. Alternative generation of training data could be recording the live sessions of and applying the measurements, but it would be very slow. The current application uses an arbitrary description of body’s limitations through angles and then defines the mapping of performance to classes.

The feature vector of first data set consists of 38 values with 19 first being angles presenting the observed posture followed by 19 values presenting angles defining the expected posture. The second data set was a preprocessed version of the first data; the dimensionality was reduced by double. The feature vector was an array presenting a scalar deviation between the expected value and observed value. The third data set was
preprocessed version of the second set. It had the same dimensionality but each metric presented deviation in percentages. The classifiers were able to learn the training patterns and also performed responsive in the application.

Using a classifier for this particular task to compare postures is less optimal than using formula. The application could use learning technologies to extract the most optimal training procedure by collecting information from exercises. The application can also improve at runtime by modifying exercise by raising the difficulty in the exercises. The application could also increase the recuperation rate by making exercises more interactive by sounds or graphical indicators. The models of classification could have been self-adaptive by performing integration of analysis of patient mood and parametric adaptation to push the patient to limits of its ability to progress its healing course. In addition, the app could integrate additional cosmetic features such as sounds, cheering voices and graphical elements visualizing the progress of the patient. The application also could have been better by centering posture.

The whole training and classification and testing are based on a mathematical formula. The learning systems in this case once they are trained to attempt to reflect the formula. That can be interpreted as if having explicit function minimizes the need for the learning systems. The application didn’t have high focus and doesn’t have a good design. The application implements a simple mechanism where user presents posture and user expected to be performed.

4.1 Classification speed

All algorithms have shown a sufficient classification speed, that can be seen on graph average refresh frequency 3.29. The graph shows refresh frequency during simulated execution of therapy exercise. The separate thread updates posture configuration from prerecorded file. The classification procedure does not pol posture data and if the updates of posture data are slower than the classification speed the classification will run prediction on the same data. The classification speed of K-NN was affected by datasize as expected while classification speed of Random Trees or ANN did not differ on average.

4.2 Classification ratio

The leading positions in the table 3.29 takes ANN and K-NN classifiers. The classifiers scored below 90% mark. Classification speed of K-NN was among the classifiers was the lowest. With the data set increases in size, K-NN takes a longer time to classify, which is expected because K-NN behaves like a database. The best result of K-NN occurred on data which presented a feature vector with deviations, that is the difference between the expected and the observed angle. The Random Trees algorithm performed worst on all feature vectors. Its performance could have been increased by choosing a larger number of decision trees. The Random Trees use the default number of decision trees that are trained. During the experiment, the attempt of running with 70 of trees did not increase the performance of training data. Then the default number of 50 decision was used. Despite feature vector being reduced by half the Random Trees algorithm didn’t show a significant difference, it stayed at the 0.2 ratio mark.

The current function uses an average when classifying which reduces its sensitivity to smaller differences. An improvement to that could be a definition of weight for each angle describing the configuration of the posture. The weight would define the importance of feature during the calculation of the final result. The application could have given the ability for the therapist to outline which joints are in great interests or which should have
more weight when determining the performance degree. Currently, the joints are having arbitrary coloring. The graphical presentation could have been done differently to reflect through color the degree of imitation of posture at joint level by setting different colors as an indicator of the rate of performance.
5 Conclusion and future work

The work performed in this thesis was aimed to compare machine learning algorithms in practice through the development of a telerehabilitation system which would make use of noninvasive sensors and machine learning technologies for remote physical rehabilitation. The application was developed with a focus on personalized, progressive therapies with feedback on progress. Kinect Digital Rehabilitation Assistant (KiDiRA) makes use of capabilities to track skeleton in real time provided by the Kinect MS in order to provide functionality both for physiotherapist and users.

The application identifies two users, the patient, and the therapist user. Application supplies therapist with a platform to specify a custom physical exercise. The patient supplied with an automated guide that assists in the exercises by providing a visual interface which contains time indicators along with expected posture as a skeleton and real-time observed posture as a skeleton, success measurement and tracking of patient performance. The system includes import and export functions which allow to distribute exercises and posture between users and retrieve some details which are tracked with regards to posture and details collected during exercise procedure.

5.1 KiDiRA: Kinect Digital Rehabilitation Assistant

The first result of the thesis is Kinect Digital Rehabilitation Assistant (KiDiRA). The KiDiRA is a simplified telerehabilitation platform for physiotherapists and patients that manages therapies, performance records. The graphical interface shows two 3d avatars, one avatar provides demonstration and the second is a mirror reflection of the patient skeleton which is observed with Kinect. During the session informative indicators which handle depiction of performance and time track. The therapist is given the option to create exercise by creating a number of custom postures.

5.2 Posture performance algorithm through explicit function and self-learning systems

The thesis presents an algorithm for calculation of match between the expected and observed posture from Kinect. The postures described with posture descriptors which are used in the classification algorithm. The match occurs by calculating the distance between descriptors, which is converted to percents and mapped with the appropriate label. Self-learning system uses explicit function as a source of training data samples.

5.2.1 Self-learning systems as classification algorithm

This thesis applies also ANN, K-NN and Random trees classification algorithms on a task of patient posture performance classification during the exercise and in real time. This thesis observes different phases of previously mentioned algorithms. The phase of implementation that is configuration and training. The phase of execution, that is classification during exercise and the rate with which the classification occurs.

Prior to setting up the learning algorithm the mapping model for prediction was defined, input $R^n \rightarrow Y^n$. The $R^n$ is input and is an array of percentages, which is used as input to all learning algorithms. The $Y^n$ differs, the K-NN and the Random trees use a numeric value that later translated to a label and the ANN uses arrays to represent each of the labels.
Application details of the artificial neural network (ANN) at the phase of implementation is presented in the list below.

1. At the phase of implementation, ANN requires a quite extensive setup before it can perform generalization to fit the prediction task. The setup consists of defining mapping method from given feature vector towards a target value with use of input and output neurons.

2. At the phase of implementation layers and sizes for each neuron layer must be determined. In addition, each neuron requires specification of propagation, activation, output function and the type of connection between them. The ANN is created with the use of a feedforward topology. ANN is completely linked which means each neuron of lower layer connected with each neuron of the upper layer.

3. To train ANN there exists multiple training algorithms, in this case, backpropagation is used as a training method. The training algorithms requires termination parameters. For ANN termination criteria consisted of maximum number of iteration and an epsilon value which specified minimum learning rate. The used values are 1000 iteration and 0.01 epsilon value. There are other termination criteria that can be applied, but the used distribution of ANN which is provided by OpenCV did not provide controls.

Application details of Random trees at the phase of implementation is presented in the list below.

1. At implementation phase, the mapping model differs only by $Y^n$ which is a numeric value and represents a label.

2. The Random trees used default setup which uses $\sqrt{\text{number of variables}}$ as fixed size of the subset of randomly selected variables. The distribution also permitted to set the maximal depth of the individual tree, maximum samples size required for node split, maxCategories and termination criteria.

3. The termination criteria used was the default, which specifies a maximum number of trees to 50 and desired accuracy with a minimum of 0.1 error.

Application details of K - nearest neighbors at the phase of implementation is presented in the list below.

1. At implementation phase mapping model used the same as for Random Trees.

2. The algorithm only required selection of arbitrary constant K for the algorithm, which defines the number of points near a point of classification to determine the result.

The KNN algorithm had the lowest frequency in comparison to the Random Trees and the ANN during the simulated therapy execution. During the simulated execution of therapy session, the Random Trees have displayed lower classification ratio in most cases when compared to the ANN and the KNN algorithms. KNN algorithm has outperformed prediction ratio of the Random Trees and the ANN.

5.3 Future research

This thesis was devoted to the integration of self-learning systems into telerehabilitation system.
5.3.1 Visual features

One of the goals of feature work could be the feature which would have focus at adjustments of the skeleton positioning on the screen in order to make exercise more user-friendly by helping the user to identify what is he/she doing wrong. Examples of such are the use of extra arbitrary views of the avatar, augmented reality to displaying graphical elements used the movement of the joint. This also includes indicators and colorings to identify success rate on each of the joint.

5.3.2 Non-visual features

Another step of improvement could be extending the application with feature of facial expression recognition and temperature measurements utilities to measure the stress levels of the patient and adapt the exercise to increase the rehabilitation process. Also, a function which would allow measuring stress levels of the patient through the reading of nonverbal signs might also allow measuring also progress.
References


## Appendix: KiDiRa architecture

This appendix contains descriptions of the interaction diagrams, use cases which are lay ground for functionality of the system. The use cases are described in form of actors, flows and preconditions and complemented with textual description.

<table>
<thead>
<tr>
<th>Use case 1</th>
<th>View exercise list</th>
<th>Use case 2</th>
<th>Load Exercise</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Primary actors:</strong> Patient</td>
<td><strong>Preconditions:</strong> Patient is at entry view</td>
<td><strong>Primary actors:</strong> Patient</td>
<td><strong>Preconditions:</strong> Patient has opened &quot;Exercises&quot; tab</td>
</tr>
<tr>
<td>1. Patient shall navigate to patient panel, by pressing button &quot;Proceed as Patient&quot;</td>
<td></td>
<td>1. Patient shall select exercise by clicking on exercise in exercise list</td>
<td></td>
</tr>
<tr>
<td>2. System shall load patient panel</td>
<td></td>
<td>2. System shall highlight selected exercise</td>
<td></td>
</tr>
<tr>
<td>3. Patient shall select &quot;Exercises&quot; tab</td>
<td></td>
<td>3. Patient shall presses button Load exercise</td>
<td></td>
</tr>
<tr>
<td>4. System shall displays exercise list</td>
<td></td>
<td>4. System shall load exercise view and selected exercise</td>
<td></td>
</tr>
</tbody>
</table>

**Alternative flows:**

1a,2a,3a... Patient exits application

**Postcondition:** System displays all exercises in database as list.

**Complementary:** This use case depicts function which allows the user to view the list of available exercises as a list of selectable names. The user starts the application, he is presented with two choices. The user chooses "Proceed as patient" and transferred to the patient panel with different tabs. The patient then selects tab "Exercises" and system displays list with exercises stored in a database.

1a Database has no stored exercises, system will not proceed or load new exercise after button press of "Load exercise"

**Postcondition:** Patient redirected to exercise view and exercise is loaded.

**Complementary:** This use case is about loading desired exercise. Patient selects exercise, and system redirects user to Exercise view and load the exercise.
Use case 3  |  Execute Exercise
---|---
**Primary actors:** Patient  
**Preconditions:** Exercise panel is displayed and exercise loaded

1. Patient shall press button "BEGIN EXERCISE"

2. System shall hide button "BEGIN EXERCISE"

3. For each posture in exercise  
   (a) System shall display posture to the right on the screen  
   (b) System shall refresh and display posture timer indicator  
   (c) System shall monitor posture of patient and display as avatar on left side on screen  
   (d) Patient shall start performing posture in front of camera  
   (e) System shall display success rate and progress  
   (f) System shall initiate updates of display with maximal frequency of 30

4. System shall stop timers and updates

5. System shall display "BEGIN EXERCISE" button

**Postconditions:** The exercise is started. Next posture is displayed and the classifier is running.  
**Complementary:** This use case depicts execution of the exercise. The user presses "BEGIN EXERCISE" and application runs postures in a sequence. Each posture has duration attribute and last highest score and average. The user is presented with two postures, the left is the avatar and the right is expected posture. The user is presented with time indicator for timer per posture and duration of exercise as a bar. Additionally system provides color bar indicator that presents success rate.

---

Use case 4  |  Create Exercise
---|---
**Primary actors:** Therapist  
**Preconditions:** Therapist panel is displayed

1. Therapist shall press button "Create Exercise"

2. System shall create exercise entity in database

3. System shall refresh display with created exercise loaded

**Postconditions:** The exercise entity is created and loaded. The entity is not stored.  
**Complementary:** This use case depicts procedure of creating the new exercise. The user clicks "Create Exercise" button and the application loads new empty exercise with one new posture active for configuration.

---

Use case 5  |  New Posture
---|---
**Primary actors:** Therapist  
**Preconditions:** Therapist panel is displayed

1. Therapist shall press button "New Posture"

2. System shall create new posture and add to currently active Exercise

3. System shall refresh display

**Postconditions:** New posture entity is created and added to exercise. The display reflects changes.  
**Complementary:** This use case specifies function of addition of new posture to currently managed exercise.
### Use case 6  
**Open Existing Exercise**

**Primary actors:** Therapist  
**Preconditions:** Therapist panel is displayed  

1. Therapist shall press button "Open Exercise DB"  
2. System shall display list of Exercises  
3. Therapist shall select exercise  
4. System shall load exercise and refresh display  
5. Therapist shall click Select  
6. System shall close the list and selected exercise is loaded  

**Alternative flows:**  
5a.1 Therapist shall click cancel  
5a.2 System shall restore back to initial state  

**Post conditions:** Exercise is loaded and display is reflecting the changes.  
**Complementary:** This use case depicts procedure of opening existing exercise. The user is at the therapist panel and presses "Open Exercise DB". System displays list and two options. Selection loads exercise and can be previewed. The user has the option to select or to cancel. If the user chooses "Select" option, the exercise entity is loaded and list closed, else list closed and application returned to initial state.

### Use case 7  
**Delete Exercise**

**Primary actors:** Therapist  
**Preconditions:** Therapist panel is displayed  

1. Therapist shall press button "Open Exercise DB"  
2. System shall display list of Exercises  
3. Therapist shall right click on exercise  
4. System will display pop-up menu with button delete  
5. Therapist shall click delete button in pop-up menu  
6. Therapist shall click cross on close the list  
7. System shall close exercise list  

**Alternative flows:**  
3a1. Therapist clicks on cross  
3a2. System closes list with exercises  

**Post conditions:** Selected exercise is removed.  
**Complementary:** This use case description shows the procedure of deleting exercise from the database. The use case starts when the user at the therapist view. When the user selects button "Open Exercise DB", the application, in turn, displays list with stored exercises. The user then right clicks on desired exercise and system displays a popup menu. The user clicks delete and system removes the entity from the database and refreshes the display. When the user is done, he clicks the cross to close exercise list.
<table>
<thead>
<tr>
<th>Use case 8</th>
<th>Define posture</th>
<th>Use case 9</th>
<th>Export posture</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Primary actors:</strong> Therapist</td>
<td><strong>Primary actors:</strong> Therapist</td>
<td><strong>Primary actors:</strong> Therapist</td>
<td><strong>Primary actors:</strong> Therapist</td>
</tr>
<tr>
<td><strong>Preconditions:</strong> Therapist panel is displayed</td>
<td><strong>Preconditions:</strong> Therapist panel is displayed</td>
<td><strong>Preconditions:</strong> Therapist panel is displayed</td>
<td><strong>Preconditions:</strong> Therapist panel is displayed</td>
</tr>
<tr>
<td><strong>Preconditions:</strong> Therapist panel is displayed</td>
<td><strong>Preconditions:</strong> Therapist panel is displayed</td>
<td><strong>Preconditions:</strong> Therapist panel is displayed</td>
<td><strong>Preconditions:</strong> Therapist panel is displayed</td>
</tr>
<tr>
<td>1. Therapist select desired joint by using arrow keys to navigate between joints</td>
<td>1. Therapist opens desired exercise as in &quot;Open Existing Exercise&quot; use case</td>
<td>1. Therapist opens desired exercise as in &quot;Open Existing Exercise&quot; use case</td>
<td>1. Therapist opens desired exercise as in &quot;Open Existing Exercise&quot; use case</td>
</tr>
<tr>
<td>2. repeat until posture has desired configuration</td>
<td>2. Therapist shall click on the desired posture from list of postures which belong to loaded exercise</td>
<td>2. Therapist shall click on the desired posture from list of postures which belong to loaded exercise</td>
<td>2. Therapist shall click on the desired posture from list of postures which belong to loaded exercise</td>
</tr>
<tr>
<td>(a) System shall indicate the selection, by flashing the square which represents the joint on the skeleton displayed on screen</td>
<td>3. System loads selected posture on to display</td>
<td>3. System loads selected posture on to display</td>
<td>3. System loads selected posture on to display</td>
</tr>
<tr>
<td>(b) Therapist shall set desired coordinates for joint with help of inputs for X, Y, Z provided on the left lower corner of the screen</td>
<td>4. Therapist presses button &quot;Export Posture&quot;</td>
<td>4. Therapist presses button &quot;Export Posture&quot;</td>
<td>4. Therapist presses button &quot;Export Posture&quot;</td>
</tr>
<tr>
<td>(c) System shall adjust joint location accordingly as values are changed</td>
<td>5. System prompts dialog pane with a request to provide the file name.</td>
<td>5. System prompts dialog pane with a request to provide the file name.</td>
<td>5. System prompts dialog pane with a request to provide the file name.</td>
</tr>
<tr>
<td>4. System shall persist exercise and postures</td>
<td>7. Therapist presses button &quot;Save&quot;</td>
<td>7. Therapist presses button &quot;Save&quot;</td>
<td>7. Therapist presses button &quot;Save&quot;</td>
</tr>
<tr>
<td><strong>Alternative flows:</strong></td>
<td><strong>Alternative flows:</strong></td>
<td><strong>Alternative flows:</strong></td>
<td><strong>Alternative flows:</strong></td>
</tr>
<tr>
<td>3a1. Therapist doesn’t click &quot;Save Exercise&quot;</td>
<td>6a1. Therapist presses button cancel</td>
<td>6a1. Therapist presses button cancel</td>
<td>6a1. Therapist presses button cancel</td>
</tr>
<tr>
<td>3a2. System shall not save exercise and changes done to postures</td>
<td>6a2. System shall close dialog pane</td>
<td>6a2. System shall close dialog pane</td>
<td>6a2. System shall close dialog pane</td>
</tr>
<tr>
<td><strong>Postconditions:</strong> The exercise, postures, and modifications are persisted in the database.</td>
<td><strong>Postconditions:</strong> The posture is exported as csv</td>
<td><strong>Postconditions:</strong> The posture is exported as csv</td>
<td><strong>Postconditions:</strong> The posture is exported as csv</td>
</tr>
<tr>
<td><strong>Complementary:</strong> This use case handles creation and modification process of a posture. The user is provided with a graphical representation of the skeleton. The user can select and navigate between the joints with use of left, right arrow keys. When the joint is selected, the user can modify the location of the joint by using inputs for x,y,z values of a joint. The application reacts directly as value is set and updates posture on display. The user has to use the button &quot;Save Exercise&quot; in order to persist changes.</td>
<td><strong>Complementary:</strong> This use case describes the procedure of exporting of a posture. The user uses the option to open stored exercise and selects posture by clicking on posture miniature displayed on right side of the display. Then the user presses button &quot;Export posture&quot; which prompts input for file name and destination. Then the user clicks &quot;Save&quot; and the posture is saved as CSV file.</td>
<td><strong>Complementary:</strong> This use case describes the procedure of exporting of a posture. The user uses the option to open stored exercise and selects posture by clicking on posture miniature displayed on right side of the display. Then the user presses button &quot;Export posture&quot; which prompts input for file name and destination. Then the user clicks &quot;Save&quot; and the posture is saved as CSV file.</td>
<td><strong>Complementary:</strong> This use case describes the procedure of exporting of a posture. The user uses the option to open stored exercise and selects posture by clicking on posture miniature displayed on right side of the display. Then the user presses button &quot;Export posture&quot; which prompts input for file name and destination. Then the user clicks &quot;Save&quot; and the posture is saved as CSV file.</td>
</tr>
</tbody>
</table>
Use case 10  

**Export Exercise**

**Primary actors:** Therapist  

**Preconditions:** Therapist panel is displayed, exercise for export opened

1. Therapist presses button "Export exercise"

2. System shall display dialog pane

3. Therapist types file name for file and selects location

4. Therapist presses button "Save"

5. System saves exercise as XML file with specified name

**Alternative flows:**

4a1 User shall click "Cancel"

4a2 The system shall close dialog panel.

**Complementary:** This use case manages export of exercise. The user has to use a function to load the desired exercise. When exercise loaded, the user uses button "Export Exercise" to initiate exporting procedure. The system displays dialog panel which permits specify destination and file name.

---

A.1 **KiDiRA activity flow**

The KiDiRA is developed to facilitate needs of two actors, the therapist, and the patient. The workflow of each actor occurs in the separate time frame and does intermingle. The application provides facility to create, run exercise and monitor performance with visual feedback. Exercise consists of series of postures. The created exercises are later accessible for the patient to perform as rehabilitation with feedback on the screen. There are two workflows, to manipulate exercises, to open and run exercises located on the local database. The program integrates functionality for both users through shared entry panel which navigates users accordingly to the desired functions and shared the local database.

The activity diagram "Exercise execution" on figure 1.32 depicts an activity during which the patient selects, loads and runs an exercise. On the activity diagram "Exercise execution" there is a swim lane for the patient and the system. The flow of patient consists of two activities. The first activity manages the selection of an exercise. The second activity with the patient is the performance of displayed postures, it starts when the system loads and initiates execution of the exercise. Concurrently with patient performing displayed posture the system monitors and updates displayed user performance classification. Both swim lanes have conditional branching, the branching illustrates loop in activity which repeats for each newly loaded posture. When exercise is complete, that
occurs if time has run out or all postures have been executed, execution of exercise terminates and the patient returned to display with exercise loaded.

The activity diagram "Delete exercise" on figure 1.31 demonstrates workflow which handles the deletion of previously stored exercise. The activity starts with the action of displaying exercise list contained in the database. In the next action, the user selects and confirms exercise for removal. In the next action occurs removal of the exercise. The proceeding action after removal is refreshing the list of exercises. In the following, step user can repeat steps of deletion or move to step of closing the exercise list and refreshing of the therapist panel.

The activity diagram "Create Exercise" on figure 1.32 manages workflow for the creation of an exercise, from the creation of exercise entity to specification of individual posture. The first action which must occur is loading of therapist panel with controls required to create an exercise. During the next step new exercise entity is created. At the next action, new posture entity added to current exercise. During the next step, posture entity is specified. After posture specification, there is a branch which allows either to repeat the procedure of adding and specification of posture or proceeding to the action of saving the created exercise.
The activity diagram "Export Exercise" on figure 1.33 describes procedure of exporting exercise as XML file. As a first step occurs loading of therapist panel, there are
required controls to proceed with the export of exercise. In the next step, the exercise for export is opened and export action triggered. In the next step user expected to specify details, such as file name and destination. The last step exercise is converted to XML and saved as specified.

Figure 1.33: Export exercise

The activity diagram "Export posture" on figure 1.34 describes workflow of exporting of specified posture as CSV file. Activity starts with the action of loading therapist panel which provides controls needed for such activity. The next action is to Open Exercise which contains the posture. The next action is to Select Posture to Export, that is setting a posture from a list as active. The next action is to specify export details. When export details are specified the posture is exported as CSV.

Figure 1.34: Export posture

The activity diagram "Open Exercise" on figure 1.35 describes the flow of opening a previously stored exercise. Manipulation of exercises and postures occurs on view(panel) that is designated for the therapist. The activity starts from interaction with the control
which triggers action to display whole exercise list in the database. The next action is a selection which proceeded with the action of loading of selected exercise. The last action in the activity refreshes the display (Therapist panel).

**A.2 Interaction diagrams**

The sequence diagram "Load exercise" sequence diagram on figure 1.37 represents objects and sequence of calls required to realize case of loading of an exercise into the system. The interaction starts when the user presses button loadExercise, which is located on the patientPanel object. The prominent objects which participate in this case are patientPanel, exerciseTab, mainPanel, exercisePanel. The patientPanel object provides graphical controls and tabs for the patient. When the button is pressed, load exercise button is hidden and selected exercise entity is retrieved from exerciseTab. The exerciseTab contains exercise list which is retrieved from the database. The mainPanel object contains all the views and provides the selection of which view is displayed with function openView(VIEW_ID). When selected exercise retrieved from exerciseTab with use of getSelectedValue(). Next in the sequence is a branch, in case of exercise being a not null entity is delegated and loaded at exercisePanel. The final call on mainPanel is openView(VIEW_ID.EXERCISE_VIEW).

The sequence diagram "Create new exercise" on figure 1.37 demonstrates procedure of creation of new exercise entity. Objects which participate in the process are physicianPanel and posture panel. When button <New Exercise> on physicianPanel, method newExercise() on object postureList is invoked. A new exercise and posture entity is created during the last call. The created exercise is set to current and so is posture. Posture is set to active. When newExercise() task is finished, the new active posture is retrieved from postureList and set as active posture with setActivePosture(posture) call in physicianPanel.

The sequence diagram with label "sd New Posture" on figure 1.38 depicts procedure of creation of new posture. The procedure starts with a press of a button labeled <New Posture>. The listener calls newPosture() function on physicianPanel object. In the next step activePosture retrieved from postureList with getActivePosture() on postureList object. In next step active posture set in object physicianPanel with setActivePosture(activePosture) call.
The sequence diagrams with label "sd Set X value of joint" on figure 1.39 and with
label "sd Select Joint" on figure 1.40 together depict procedure of definition of a posture. The sequence diagram on figure 1.39 presents interaction of objects which involved in setting joint value. The figure 1.39 presents sequence diagram for setting X value of a joint, but the procedure to set Y,Z values is analogous. On graphical control panel exists three inputs, which are dedicated for input of X,Y,Z values. Procedure is triggered when state on xValue, yValue or zValue has changed. During the stateChanged() call postureRenderer is retrieved from posturePanel object with getPostureRenderer(). Then the value held by input set in posture via call setXValueOFSelectedJoint(value). In the setXValueOFSelectedJoint() coordinates are constructed and of attached posture object is updated by setJointLocation(selectedJointID,coords).

The sequence diagram "sd Select Joint" on figure 1.40 displays interaction between object during selection of joint in posture. The procedure started with key event listener, avatarPanel listens and calls selectNextJoint() if VK_LEFT key pressed. If VK_RIGHT pressed, selectPreviousJoint() called on postureRenderer.

![Sequence diagram](image1.png)

**Figure 1.38: New posture**

![Sequence diagram](image2.png)

**Figure 1.39: Set joint value**

K
The sequence diagram "Delete exercise" on figure 1.41 depicts objects which interact during the procedure of the exercise entity deletion. The procedure consists of three steps. The first step is to open the a list of exercises, that is depicted with display list call. During this call list of exercises are retrieved with help of dao object. In next step graphical representation of list of exercises is presented via eList object, this occurs also in displayExerciseList() call. The following calls belong to a loop block which can be repeated by user. In following the user is working with controls on lists graphical representation, he does right click on Exercise entry which displays popup menu and managed via popUpMenu object. Displayed menu has only one option, delete option. The location of courser where right click occurred defines which exercise entity is deleted. In the actionPerformed() of <Delete button> occurs the call delete(exercise) to dao object and then the call getList(entityType) to retrieve updated entity list. In next step eList is updated with new data with call setData(exercises). The loop block ends here, the user finishes sequence by performing a close action on eList object.

The sequence with label "Export exercise" on figure 1.42 depicts the interaction between objects during the procedure of exporting exercise. When the action is performed on Exercise button presented on physician panel, the procedure is initiated. In the actionPerformed() of <Exercise button> occurs the call getExercise() to postureList object with getExercise() call. The returned object is never null. In the next step, the writer object is used with call export(exercise). Writer object displays a dialog in form of filechooser with options and input for filename input. In following steps after dialog appears conditional block which is only entered if the result of the previous dialog was approved_option. If the result from dialog was APPROVED_OPTION, the file name is retrieved from fc dialog object and
called export(exercise, dest) on writer object. If the result of the dialog was other than
approved_opption sequence would end with dialog window closed.

Figure 1.41: Delete exercise
The sequence with the label "Export posture" on figure 1.43 depicts participation and interaction of objects in the procedure of exporting posture as CSV. Sequence starts with a call exportPosture() to physicianPanel. The posture is retrieved from postureList object with call getActivePosture(). In the next step, writer object created and used to export via export(posture) call. In the export function of writer object, the user is presented with fc objects which have properties of file chooser and input. Next in sequence is a conditional block which is entered if the result of the fc object was approved option. In the conditional block in which writer object performs self call export(p, destination). The sequence ends with dialog window is closed.

The sequence with label "Open exercise" on figure 1.44 depicts procedure of opening exercise for editing. The physicianPanel receive displayExerciseList() call from listener of <Retrieve from register> button. In the displayExerciseList() call, list with entities retrieved from dao with getList(entityType) and eList object is instantiated with exercises from dao with a call setData(exercises). Next in procedure is a loop which allows the user to browse through possible exercises. When the exercise in the eList object is selected it is imported into physicianPanel with call importExercise(selectExercise).
Figure 1.43: Export posture
The sequence diagram with label "Import exercise" on figure 1.45 depicts procedure of importing exercise from an xml file. The sequence starts with call `actionPerformed(e)` which is resulted with press of button labeled "Import exercise". In the `actionPerformed(e)` of a listener, reader object is created and `importExercise()` called on it. The reader object starts `fc(filechooser)` object and calls on it to display dialog. Next in sequence is conditional block, it can be reached if resulting value of `fc` object is `APPROVE_OPTION`. The selected file then extracted from `fc` object and parsed in reader object. Exercise entity is returned in response to `importExercise()`. The next call in sequence is `loadExercise(exercise)`, which loads imported exercise into physicianPane. Conditional block ends here. If the `returnVal` from `fc(filechooser)` differs from `APPROVE_OPTION`, execution goes to alternative block where reader returns null. In the end of sequence `fc` object is disposed and no longer displayed.
Figure 1.45: Import exercise